A VIRTUAL PNNI NETWORK TESTBED

Kalyan Perumalla
College of Computing
Georgia Inst. of Technology
Atlanta, Georgia 30332–0280

Matthew Andrews
Lab. for Computer Science
Massachusetts Inst. of Technology
545 Tech. Square
Cambridge, MA 02139

Sandeep Bhatt
Bellcore, and
DIMACS, Rutgers University
445 South Street
Morristown, NJ 07960

ABSTRACT

We describe our experiences designing and implementing a virtual PNNI network testbed. The network elements and signaling protocols modeled are consistent with the ATM Forum PNNI draft specifications. The models will serve as a high-fidelity testbed of the transport and network layers for simulation-based studies of the scalability and performance of PNNI protocols.

Our models are written in the new network description language TeD which offers two advantages. First, the testbed design is transparent; the model descriptions are developed separately from, and are independent of, the simulation-specific code. Second, TeD is compiled to run with the GTW (Georgia Tech Time Warp) simulation engine which is supported on shared-memory multiprocessors. Therefore, we directly obtain the advantages of parallel simulation.

This is the first complex test of the TeD modeling and simulation software system. The feedback from our experiences resulted in some significant improvements to the simulation software. The resulting PNNI models are truly transparent and the performance of the simulations is encouraging. We give results from preliminary simulations of call admission, set-up and tear-down in sample PNNI networks consisting of two hundred nodes and over three hundred edges. The time to simulate ten thousand call requests decreases significantly with the number of processors; we observe a speedup factor of 5.3 when 8 processors are employed compared to a single processor. Our initial implementations demonstrate the advantages of TeD for parallel simulations of large-scale networks.

1 INTRODUCTION

The Private Network-Network Interface (PNNI) protocol suite (ATM Forum 1996) is an international draft standard proposed by the ATM Forum. The protocol defines a single interface for use between the switches in a private network of ATM (asynchronous transfer mode) switches, as well as between groups of private ATM networks. Corresponding to the network and transport layers of the OSI Reference Model, PNNI supports two categories of protocols: (1) topology discovery and (re)configuration, and (2) dynamic routing via virtual circuit connections.

The main feature of the PNNI protocols is *scalability*, i.e. the complexity of routing does not increase drastically as the size of the network increases. There is an inherent trade-off between scalability versus quality of routing. Given the complex nature of broadband data traffic, simulations are the only feasible avenue to study the scalability of the PNNI protocol and to design and tune strategies for hierarchical addressing, topology aggregation and call admissions.

We are building a virtual PNNI testbed for large-scale simulation studies. Two factors are critical: *performance* and *reusability*. For the network sizes and time scales of interest, the simulations must run on parallel and distributed platforms. Equally important, we insist that the model descriptions be reusable and independent of the simulation-engine code. The reason is straightforward: the value of the testbed to the user lies in the model descriptions, not the simulation engine. We expect to reuse the testbed to study higher-level protocols as well as scenarios to be defined later. Moreover, we do not wish to be tied to a single simulation engine — we prefer to leverage advances in parallel discrete-event simulation technology without having to rewrite the testbed.

Our testbed uses the new network description language TeD (Perumalla and Fujimoto 1996, Perumalla et. al. 1996). TeD is designed with the goals of reuse, transparent modeling, and efficient parallel execution. We have used the C^{++} version of the TeD software system that utilizes the GTW (Georgia Tech Time Warp) library as the underlying parallel simulation engine. Our preliminary simulations run on shared memory multiprocessors.

This paper presents the modeling and simulation aspects of the testbed. Specifically we focus on the issues that arise in (1) separating model code from simulation specifics, and (2) organizing the model for parallel simulation, without interfering with transparency. Ongoing network-oriented studies will be reported in separate, forthcoming publications.

The remainder of this paper is organized as follows. Section 2 presents an overview of the TeD language and the PNNI protocols, and describes the challenges of building transparent PNNI models. Section 3 describes the TeD design of the PNNI testbed, while Section 4 describes some of the implications of the testbed on the simulation engine. Section 5 describes our initial simulation study and presents results of parallel simulations. Finally, Section 6 identifies ongoing and future directions.

2 TRANSPARENT PNNI MODELS

TeD (Telecommunications Description) is a language for developing transparent model descriptions which are (i) not tied to any specific simulation engine, and (ii) not cluttered with references to simulation code. While TeD was conceived for telecommunication network models, its basic constructs permit more general use.

Besides developing a virtual PNNI testbed, our modeling experience also served as the first complex test of the TeD modeling and simulation software system. The feedback from our experiences resulted in some significant improvements to the simulation software. The end result is that the PNNI models are truly transparent and the performance of the simulations is encouraging.

The remainder of this section gives high-level overviews of TeD, PNNI, and the challenges of developing transparent models.

2.1 TeD Overview

TeD supports a parsimonious set of concepts: *event*, *channel*, *entity*, *architecture*, *process* and *component*. Entities, the basic building block for TeD models, are object-oriented encapsulations. These abstractions are in the spirit as the VHDL language (Bhasker 1995) for hardware description.

An entity specifies the abstract interface of a network element or protocol. Entity interfaces are defined in terms of channels through which events flow; mappings among channels interconnect entities. The behavior of an entity is specified by its architecture which consists of processes and components (subentities). Inheritance between entities and architectures allow entity (architecture) types to inherit from other entity (architecture) types. Entity and architecture type definitions can be compiled to create a model database. Instantiations and compositions among database objects establish desired network configurations. Models can be customized via parameters that are assigned user-defined values after compilation.

TeD constructs facilitate the development of transparent network models. At the same time, the disciplined (and restricted) ways to compose entities makes it possible to compile models for parallel execution. By enforcing the discipline for composition within the language itself, the dual concerns of model specification and execution are effectively separated. This separation allows the development of complex models which can exploit state-of-the-art technologies for parallel and distributed simulation.

2.2 PNNI Overview

The PNNI protocol is a complex set of ATM routing and signaling mechanisms proposed in the draft specifications v1.0 (ATM Forum 1996). It represents one of the most sophisticated signaling protocols devised to date, aimed at supporting QoS-based routing along with unprecedented levels of scalability.

The PNNI specifications are based on a hierarchical addressing scheme which is used for maintaining network topology information and for call routing. In a global network it is infeasible for each node to contain a complete description of the entire network — the costs of storing and updating this information are prohibitive. On the other hand, a hierarchical scheme allows nodes to have reduced "views" of the network; thus, changes to the network need not be propagated to every node.

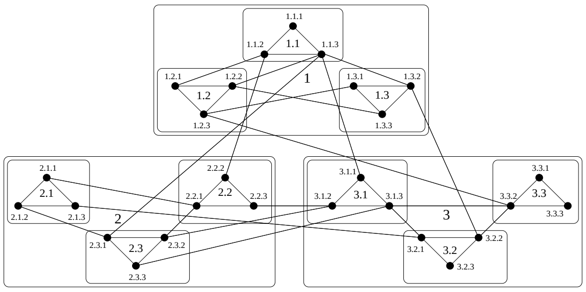


Figure 1: An example PNNI network.

The PNNI hierarchy organizes network nodes (at level-0) into level-1 peer groups; level-1 groups are grouped together into level-2 peer groups, and so on. Figure 1 gives an example of a PNNI hierarchy with 4 levels. The level-0 nodes have addresses of length 3, level-1 groups have addresses of length 2, and so on (there are no constraints on the *number* of nodes in each peer group). The entire network is a level-4 group (with an empty address).

Within every peer group, one member is elected as the peer group leader (PGL). Peer group leaders play a prominent role in establishing and maintaining the topology views for network nodes; they are not used for call routing. The role of each PGL is played by a network node, which we call the role playing physical node (RPPN). The RPPN of a PGL can be determined by recursively following PGLs down the hierarchy to a network node. All interactions conceptually within the hierarchy are carried out among the RPPN nodes in the network.

According to the PNNI specifications, each network node must have complete knowledge of the nodes and edges within its level-1 peer group. This is achieved by having each node flood its information within its level-1 peer group. For higher levels, $\ell > 0$, level-ℓ RPPNs of PGLs that are part of the same level-$(\ell+1)$ peer group exchange aggregated information via routing control channels (RCCs) established by the call setup mechanisms.

Finally, the aggregated information received by the RPPN of a level-ℓ PGL is sent to the network nodes below it in the hierarchy. The exchange of topology information is carried out using short `Hello` messages and longer `PTSE` messages. The topology information that is gathered and maintained continually at each node is used in the distributed routing protocols for call admission, call setup and call teardown. This information is periodically updated during network operation. The effect of the hierarchy is that each node has detailed information about nearby nodes but only approximate information about more distant nodes.

2.3 Modeling and Simulation Challenges

The PNNI protocols are challenging, both for modeling and efficient parallel execution. Section 4 describes how the following challenges were met within the TeD – GTW system.

Recursive layer dependencies. The conventional "layered" approach to modeling network element behavior is inadequate to express the complex behavior of the PNNI nodes. For example, establishing the hierarchy requires that RCCs be set up; this requires call routing and admission which, in turn, depends on the hierarchy being set up. This requires that the modeling framework support recursive dependencies.

Complex model behavior. PNNI node behavior involves complex timing, ordering and synchronization of protocol messages. The informal description of the protocol functions in the PNNI draft specification illustrates this complexity, with several timers, decisions and assumptions. The challenge was to formulate these interactions within the TeD discipline, compromising neither the transparency of the TeD model nor the efficiency of the parallel execution.

Complex and large states. PNNI nodes contain large amounts of state information that change in unpredictable ways. The state information itself is complex, and requires modeling support for transparent processing. The large state size requires incremental state saving techniques in the context of optimistic parallel simulation. Indeed, we had to incorporate a transparent incremental state saving facility within the Georgia Time Warp (GTW) system.

Complex communication patterns. In typical PNNI networks, there are several classes of interaction for configuration discovery, topology exchange, call setup, data transfer, and so on. Moreover, the interaction of a node with its neighbors varies with the type of the neighbor and the class of the interaction. The communication patterns are complex and unpredictable. To further aggravate the situation, typical networks (such as the one used in our study) are difficult to partition; this creates hurdles for load balancing for efficient parallel simulation.

Variability in event size and processing. Event sizes vary considerably with type. For example, `Hello` packets are considerably smaller than `PTSE` packets. This variation is important because parallel simulators typically fix event sizes for efficient memory management; forcing all events be the size of the largest event lowers performance because of the overhead of memory copying. Similarly, the computation performed by a PNNI node on receiving a message (event processing) varies with the type of the message. It is important that load balancing mechanisms for parallel simulators not assume uniform costs for event processing.

Complexity of development process. In developing complex simulation models, facilities for automatic checks of model correctness are indispensable. For example, "assertions" are a common technique by which pre-conditions and post-conditions are specified in the models, which are automatically verified at run-time by the simulator, and the simulation halted upon the detection of the first assertion violation. In the context of optimistic parallel simulations, however, *transient* erroeneous conditions may be caused due to optimistic processing, in addition to *non-transient* modeling errors. Generalized correctness checking techniques in the modeling systems are thus needed.

3 PNNI MODELS IN TED

Each PNNI node is modeled as a TeD entity. The entity interface, shown in Figure 2, conists of (i) a parameterized array of channels, one for each ATM link to a neighboring node, and (ii) a `user` channel for the local point of attachment of a user node. The channel types in the array correspond to messages in the PNNI protocol between PNNI nodes. The user channel type includes messages that deal with network services in terms of switched virtual circuit (SVC) setup and teardown. An arbitrary PNNI network configuration can be built by instantiating one entity per network node and connecting channels appropriately.

```
entity PNNINode( int NUM_LINKS )
{
    channels
    {
        inout ATMLink physical[
            $PARAM(NUM_LINKS)$ ];
        inout UserChannel user;
    }
}
```

Figure 2: TeD entity interface of a PNNI node.

As outlined in Section 2, each node participates in establishing the PNNI hierarchy and possibly acting as the PGL for multiple levels within the hierarchy, in periodic exchange of topology information, and in call admission and routing. The corresponding behavioral specification of a PNNI node in TeD partitions nicely into the linear inheritance hierarchy shown in Figure 3.a. Figure 3.b shows the components of the PNNI node architecture and the inheritance among these components.

When an event appears on one of the channels in the array, it is processed by the `SVCService` architecture. The `SVCService` architecture models link-to-SVC and SVC-to-link mapping services, so that messages are identified by SVC name rather than link identifiers. Depending on its type, the received message is forwarded to one of three architectures: `TopologyMaintenance`, `RoutingService`, or `UserService`.

The `TopologyMaintenance` architecture inherits from the `SVCService` architecture and models the PNNI topology maintenance protocols using the SVC service. It models PGL instantiation and PGL communications, and builds and maintains the topology database over time for the node as well as for every PGL for which the node is an RPPN.

The `RoutingService` architecture inherits from the `TopologyMaintenance` architecture, thus inheriting the topology database maintenance behavior. It adds the call admission and routing algorithms to service SVC setup requests coming from the neighboring nodes, or from user nodes, or from the `TopologyMaintenance` processes themselves.

The `UserService` architecture inherits from the `RoutingService` architecture, thus inheriting the entire PNNI protocol functionality for a node. To the functionality, it adds the SVC setup, teardown, and data transfer services for user nodes.

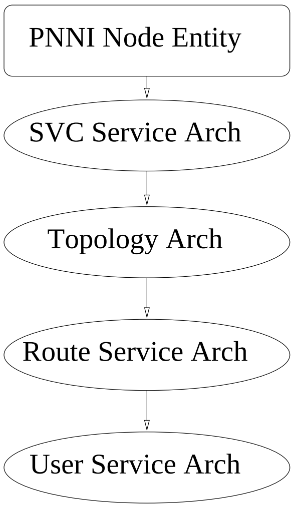
4 EXPERIENCES

In this section, we discuss our approach to the challenges outlined in Section 2.

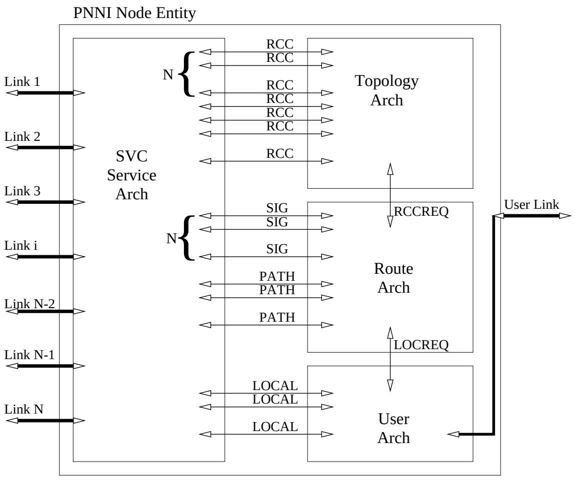
Recursive Layer Dependencies. The mutually recursive dependence between topology maintenance and routing service is resolved by using an internal channel (RCCREQ) in the `TopologyMaintenance` architecture (see Figure 3.b). This channel is used by the `TopologyMaintenance` architecture to forward SVC setup requests. The inheriting architecture, `RoutingService`, caters to these requests coming on the internal channel. Similarly, the routing service architecture caters to SVC setup requests originating from the `UserService` architecture that arrive on another internal channel (LOCREQ) dedicated for that purpose.

Section 2 mentioned the recursive dependence between topology database maintenance and signaling in the PNNI protocol. For *auto-configurability,* the dynamic formation of the peer group hierarchy requires the dynamic set up of SVCs between PGLs. But, at the same time, the signaling support that is required for setting up the SVCs depends on the formation of the peer group hierarchy in the first place.

Two facts underlie this recursion: (1) the SVCs between neighboring nodes at level-0 are permanent virtual circuits that are set up at boot time, and (2) the formation of the hierarchy up to level i is necessary and sufficient for the establishment of SVCs between PGLs at level $i+1$. The first fact is easily modeled by proper initialization of node parameters. The second is used during the hierarchy formation.



a. Entity and architecture hierarchy



b. Internal compositional view of architectures

Figure 3: Illustration of PNNI node model organization in TED

We have addressed the problem of modeling this recursion using a combination of two solutions.

First, we generalized the format of the SVC setup requests by not requiring that the destination of an SVC setup request be a complete ATM address. When the destination address is specified as a partial address, we instead require that it correspond to a peer group address which is then interpreted to be the RPPN corresponding to the PGL of that peer group[1]. This generalized specification of SVC destination enabled us to develop a single model for routing services which services both user-initiated requests (which specify full destination addresses) as well as internally generated requests (which specify partial destination addresses for RCCs between PGLs) for SVC setup.

Second, we modeled the recursive interaction between topology maintenance architecture with the generalized routing service architecture. This was done using TED's *internal channel* feature that allows, among other things, for an *inherited* architecture's processes to send events on an internal channel to be processed by the processes of its *inheriting* architecture. Thus, an internal channel, `RCCREQ`, is used for such communication between `TopologyMaintenance` architecture and `RouteService` architecture. Whenever a PGL at level $i+1$ is elected in a group at level i, the current topology database at the RPPN corresponding to the PGL contains sufficient information to discover the (partial) addresses of the new PGL's neighbors. SVCs must be setup between this PGL and its neighbors. For this, SVC setup requests are forwarded by the `TopologyMaintenance` architecture down the `RCCREQ` internal channel, which are received and serviced by the `RouteService` architecture. For its part, the routing service makes use of the current topology database maintained by the `TopologyMaintenance` architecture (the database is directly accessible because of inheritance). As mentioned previously, the database contains just sufficient information at that point in time to service the current SVC requests to the destination PGLs. Thus, dependency of the `TopologyMaintenance` architecture on the routing service is resolved using an internal channel, whereas the dependency of the `RouteService` architecture on the availability of the appropriate topology database information is resolved by means of access to such information through inheritance.

Complex Model Behavior The complexity of PNNI protocols appeared to demand richness and

[1] A full destination address trivially corresponds to the level-0 PGL of the destination peer group, which is the same as the RPPN address of the destination node.

power from the modeling language, such as arbitrarily nested synchronization points, and timer services. In the simulation community, such expressive power is generally recognized as corresponding to *process-oriented* simulations. Furthermore, it is generally accepted that *event-oriented* simulations, which hold lesser expressive potential, can be more efficiently implemented than process-oriented simulations. TeD attempts to strike a balance between the expressive power and efficiency of execution associated respectively to the two simulation approaches by supporting *quasi processes*. These are processes in which simulation-time can be advanced only in the "main body" of the process. In fact, in the version of TeD we used for PNNI modeling, the restriction was more stringent in that *wait* statements are not allowed to appear under conditional and looping statements, but rather appear only as the "top-level" statements in the processes. The TeD compiler is capable of translating the quasi-process descriptions into event-oriented simulations with zero stack overheads, thus achieving the associated efficiency.

The challenge was to specify the PNNI protocols under the quasi-process model supported by TeD. To our surprise this turned out to be quite natural, and the protocols were modeled in an elegant manner despite the restrictions. This gives us confidence that tradeoffs between expressive power and efficiency can be settled well in many applications.

Large and Complex State Each PNNI node contains a large amount of state information — Topology Database for potentially hundreds of nodes, SVC mapping information for (potentially thousands of) SVC's, link state information for (potentially dozens of) ATM links, and so on. In response to this need we developed the Transparent Incremental State Saving (TISS) facilities of TeD for managing such large state in a transparent manner. As we have witnessed, this makes it possible to develop TeD models without any knowledge of the underlying TISS method or GTW simulation engine.

In essence, we use object-oriented class definitions that perform simulation-specific functions, but in a transparent manner. For example, `NodeAddress` is a class that provides the functionality to manage node addresses. Internally, it is modeled as an array of `INT`, each element of which is a transparent implementation of an incrementally state-saved object. Due to the overloaded assignment operators predefined by TeD over `INT` types, no additional simulation-specific primitives appear in the model code, resulting in transparent model code that is oblivious to implementation details.

Processor Mapping Since, in general, automatic load balancing is very hard, TeD provides facilities for user-defined mappings. We tried three different schemes for mapping PNNI nodes to processors: a simple round-robin scheme, a second based on balancing node degrees, and a third which partitioned the PNNI hierarchy with few cross-edges. As expected, the third scheme performed the best. The first two schemes suffered considerable roll-backs with consequent poor processor utilization.

In general, we observed that the group-based algorithm is far superior to the degree-based algorithm, when the number of processors does not exceed the number of groups. This is expected since nodes communicate with neighbors that are members of the same peer group more often than with other neighbors. Also, the greater the number of groups, the greater the number of processors that can be used while still sustaining a good level of speedup. This leads us to believe that near-linear speedup is possible on larger PNNI networks with hundreds of groups.

Event Size Reduction. The effect of large size of some infrequent events is mitigated using data-compression techniques. The data of the class variables contained in the large events were compressed before sending the event, and uncompressed into class variables of the event upon receiving the event, just before processing it. We observed compression factors between 2 and 3, thus reducing the maximum event size significantly, thereby improving the performance. We found that the occasional compression/uncompression operations do not constitute any significant overheads compared to the average event processing granularity.

Unfortunately, the preceding technique does not constitute a completely satisfactory solution for simulating PNNI networks of very large size. This is because of the possibility that the `PTSE` and/or the `SVCSetupRequest` events can grow linearly with the size of the largest peer group, which implies that event data compression may not be sufficient. We are exploring other solutions, such as splitting the event into smaller-sized events. However, the most elegant solution appears to be one in which the underlying simulator supports variable-sized events, such that the large size of one infrequent event does not affect the performance of managing every single event during the simulation.

Problems with Optimistic Computation To solve the problem of resolving transient and non-transient errors during optimistic parallel simulations, TeD provides a special macro, `POT_ERR()` ("potential error"), that is used in the model to signal a potentially erroneous condition. If the error is a result of optimistic processing, the condition gets rolled back appropriately, and the macro will have null effect. However, if the error was indeed due to a modeling/programming error, an error report is generated by TeD upon the termination of the simulation.

In the process of developing the PNNI models we found that an integrated correctness-checking facility is absolutely essential both to debug as well as to gain confidence in the correctness of the model execution when simulated *in parallel* using a risky and aggresive style of optimistic simulator (such as GTW).

5 SIMULATION STUDY

We have simulated sample PNNI network configurations, with client and server user nodes stochastically generating requests for call setup and teardown. The results show that we can simulate thousands of successful call setup requests per minute of wall-clock time, for a PNNI network containing 200 nodes.

Simulation runs were performed on a large network configuration containing 200 nodes and over 300 edges. A user node, containing an SVC client and an SVC server, is attached to each PNNI node. Each client chooses a server at random and requests an SVC connection to it. The client uses a scheme that favors servers closer to it within the hierarchy, with decreasing bias for nodes further away in the hierarchy.

A call connection can fail for two reasons: if the destination server is unreachable (the network is still "booting up"), or if no path with sufficient capacity is available. When a connection succeeds, the SVC has been setup, and data can be sent over that SVC. Each client uses an exponentially distributed intercall time with mean 1.0, and pareto-distributed hold time with mean 1.0 and α value of 1.2. At the end of a call hold time, the SVC is disconnected, which tears down the SVC across the network. (All constants described above can be customized easily on a per client/server basis.)

The performance results are tabulated in Table 1. The speedup against sequential execution of the model is depicted in Figure 4. The total simulated time is 200 units; of this, the initial network setup time is 150 units, the total number of successful SVC requests is approximately 10,000, and the total number of failed SVC requests is approximately 300, giving a total of about 10,300 SVC requests. The work load corresponds to nearly half-million discrete events that are simulated. The simulations were performed on a 167MHz UltraSparc with more than 10 processors and 1GB main memory.

#PE's	γ	ρ	**Time**	**Speedup**
1	345	100	1352	1.00
2	635	92	714	1.89
3	890	86	499	2.71
4	1132	82	388	3.50
5	1360	78	318	4.25
6	1435	69	284	4.80
7	1513	62	259	5.22
8	1464	53	256	5.30

Table 1: Simulation Results on a 200-node, 307-edge Network. γ is event rate in events/second. ρ is percentage of simulation efficiency. Time is in seconds.

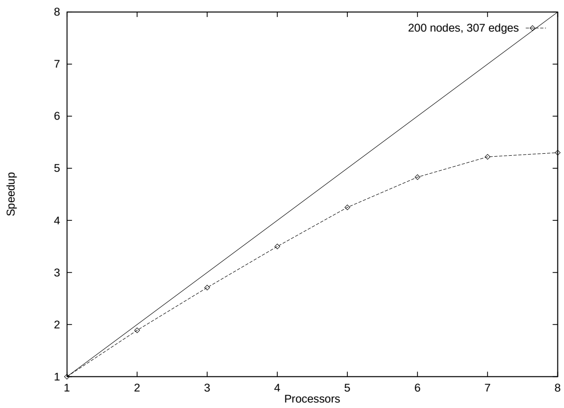


Figure 4: Simulation Speedup on the 200-node Network

In Table 1 the third column is the percentage of events processed that were not the result of a rollback. As the number of processors increases, we observe a rapid increase in the number of rollbacks. We also see in Figure 4 that, as the number of processors increases the speedup is linear at first, and then flattens out. We believe this is caused by the fact that when the number of PNNI nodes per processor goes below a threshold, rollbacks become frequent. We observed this behavior with smaller networks, when the performance flattened out at 3 processors. This leads us to believe that very large networks can be simulated with linear improvements with moderate numbers of processors.

6 ONGOING WORK

We are sufficiently encouraged by our experiences with TeD that we intend to extend the current models to a PNNI testbed which is faithful to the full PNNI specifications. Some of the features we will add include: failure handling, sophisticated PGL election algorithms, and SVC setup crankback algorithms. As the testbed acquires greater functionality, we expect to use it to analyze the scalability of the PNNI protocols and to compare the performance of different admissions and routing strategies.

ACKNOWLEDGMENTS

The authors thank Lisa Zhang for many helpful discussions and Iraj Saniee for supplying the 200-node network for our experiments. This work was carried out, in part, while the first two authors were visiting Bellcore. This work is also supported by NSF Grant NCR-9527163 and DARPA Contract N66001-96-C-8530 to DIMACS, Rutgers University with subcontracts to Georgia Tech. Additional support at MIT was provided by NSF grant CCR-9302476, ARMY grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246.

APPENDIX A: ADDITIONAL FIGURES

```
process #3: PNNINode:TopoMaintenance:hello_send
{\{
    for(int l=0; l<STATE(curr_pgl_level); l++)
    {
        CNodeAddress self;
        get_self_addr( &self, l );
        for(int i = 0; i < MAX_DEGREE; i++)
        {
            int vcc = DCONST(FIRST_RCC_SVC) +
                      l*MAX_DEGREE+i;
            if(is_established(vcc))
                CHANNEL(vcc_out[vcc]) <<
                    EVENT(Hello,(&self));
        }
    }
    double jitt_intvl = DCONST(hello_interval) *
        (0.75+0.5*STATE(hello_jitter_rng).next());
 \}
 wait for $jitt_intvl$;
}
```

Figure 5: A TeD process in the PNNI node model

REFERENCES

ATM Forum, "PNNI Draft Specification v1.0," ATM Forum 94-0471R12, ATM Forum, 1996.

J. Bhasker, "A VHDL Primer," Prentice Hall, 1995

K. Perumalla, A. Ogielski, and R. Fujimoto, "MetaTeD: A Meta Language for Modeling Telecommunication Networks," GIT-CC-96-32, College of Computing, Georgia Institute of Technology, 1996.

K. Perumalla and R. Fujimoto, "A C++ Instance of TeD," GIT-CC-96-33, College of Computing, Georgia Institute of Technology, 1996.

AUTHOR BIOGRAPHIES

KALYAN PERUMALLA is a Research Scientist and a Ph.D. student at the College of Computing at the Georgia Institute of Technology. His research interests include Parallel and Distributed Simulation, Network Modeling and Parallel Combinatorial Optimization.

MATTHEW ANDREWS obtained his PhD in Applied Mathematics from MIT in 1997 and is currently a Member of Technical Staff at Lucent Technologies, Murray Hill, NJ. His interests are in algorithm design and the analysis of communication networks.

SANDEEP BHATT is Senior Research Scientist at Bellcore and member of the research faculty at Rutgers. His research interests include models and algorithms for parallel and distributed computing, graph embeddings, and distributed simulations of many-body systems and communication systems.