TECHNICAL RESEARCH REPORT

Collaborative Simulation for Hybrid Networks

by S. Gupta, J.S. Baras, G.C. Atallah

CSHCN T.R. 96-15
(ISR T.R. 96-80)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Collaborative simulation for hybrid networks *

Sandeep Gupta John S. Baras †

George C. Atallah ‡
Center for Satellite and Hybrid Communication Networks
Institute of Systems Research
University of Maryland, College Park MD

December 10, 1996

Abstract

An experiment for using more than one simulation of sub-networks collaborating as a single large simulation is suggested, based on work with a similar situation. The background and relevant details of the initial solution are described to provide an outline for the solution. Work on this method will help development of simulations for hybrid networks, where newer pieces may be added to the simulation without having to build a large, monolithic and a site-centric development effort.

*This material is based upon work supported in part by the National Science Foundation under Grant No. NSF EEC 94-02384, NASA-GSFC contract number NAS532378, and by the Center for Satellite and Hybrid Communication Networks under NASA contract NAGW-2777, and the State of Maryland. Some of this work was done for a similar situation in an earlier Automatic Network Fault Management project: the time efficiency of the simulations. Eventually, a second approach was suggested there, and the part of the work that did not get used there forms the basis for this note.

†Also with Department of Electrical Engineering.

‡With the Institute of Systems Research during part of this work.

1 Introduction

Simulating the execution of networking code in a discrete event simulation environment is useful for the study of the behavior of the networking protocol implementations and may be useful for the study of the behavior of protocol itself. The level of detail limits the size of the simulated network or the complexity of the applications that need to be studied. This concern can be addressed by using a faster computer, to some extent only.

¿From the point of view of the design of simulations for hybrid networks, a monolithic architecture of the simulation may constrain the development and maintenance of the entire simulation to one location. In a collaborative effort, individual pieces of software and their simulations may be developed at different places. In particular while building a hybrid network, these pieces may be built at different collaborating locations. Also, it may not be possible to visualize the final simulation at the very start as pieces of these hybrid networks, such as wireless, mobile, or high speed links continue to keep evolving.

The individual pieces of the simulation can be quite complex, e.g., an ATM switch, or even a hybrid link, involving high speed interfaces. Even at some of the usual link speeds, the number of events generated can be enormous. Some of the detail can be abstracted, e.g., for the study of the interaction between the protocols and the application at a determined link performance, it may not be useful or even feasible to simulate cell or packet level behavior. Any experiment requiring the simulation of packet level behavior leads to handling a large number of events, unless specifically designed otherwise. Even then, unless such a redesign is done at all the collaborating nodes, the inter-operation of these simulations will be complicated. It will be useful to make a beginning towards getting a set of simulations to work together from the point of view of scalability, and for understanding the inter-operability of simulations running on different machines.

Several simulators come with implementations of popular or otherwise important protocols as modules. The interesting idea would be to use existing modules, without having to rewrite them. Modules for certain complex protocols for ATM, wireless, inter-networking and for complex link and node behavior such as for satellite links, mobile hosts and applications are available with mature simulation products.

Avoiding a rewrite will allow the use of detailed module libraries, and may result in saving the development effort if such an approach works well. This was the motivation behind initial work to resolve a similar situation described here. A key point is that interfaces for natural inter-operation are

available without having to force-fit or redesign existing pieces, even while studying hybrid networks.

2 Available background work

The background for this work is the following method for improving the time efficiency of a WAN traffic simulations by distributing sub-network simulations over hosts. A *proof of principle* minimal system was written and tested.

Simulation of heavy traffic conditions on large packet switched subnetworks using discrete event simulators can be limited in performance by the complexity of code as well as by the rate of events the simulator can handle. During a particular set of simulations it was found that even while simulating code as complex as transport or networking protocols running on several nodes, the limiting factor was the host computers capability to handle the rate of events that were generated in the simulation. Small simulations designed for a particular project were taking a long time, making the initial arrangement impractical for its application. (Later, a second solution was suggested and used for resolving that situation.) The number of events in this simulation were dependent on the number of simulated end-systems and the number of applications they ran. The simulation was intended to study both network and application behavior and this in turn limited the size of the subnetwork that would be used in a particular run. It was required to generate more data from the simulations and yet make it more realistic by adding more complexity to the network, so that certain network management algorithms may be tested. In the following, a set of routines for running several autonomous simulations of subnetworks with a loosely synchronized clock is described. A framework with placeholder routines was built and tested in anticipation of using it to scale X.25 protocol based network simulations. The target simulations did not get to the point where these routines would be used. It uses a technique from parallel programming called barrier synchronization, which can be adapted for use in other such applications. These routines are written with Parallel Virtual Machine [1] programming environment using its barrier synchronization functions. They did not get tested with the entire simulation environment and are documented only in the interest of further work. Some of the places that may require further work are also listed. A technique like this may be particularly useful in simulating a wide area network where otherwise disconnected subnetworks are connected by satellite links.

3 Description of the Constraints

In general there may be two problems in distributing parts of a network simulation over hosts. Operation of the individual components together such that they behave as a large simulation requires that clocks of these simulated *sub-simulations* be synchronized and there be a timely exchange of information that needs to travel between them. If the simulation were to be re-designed from the start, it may be possible to keep the above requirements in mind. In this case, the simulation used a lot of library modules for the protocols, and re-working them in detail did not seem like an elegant or then feasible solution. Instead some of the characteristics of the application could be recognized to design the solution in this particular case. The output information used from the simulation were the status and performance averages from the nodes and links at fixed intervals in seconds, tens of seconds, and summaries. The biggest problem was that the simulations were running several times (an order of magnitude) slower than real time, and this wasn't providing sufficient complexity for the simulation that the project group needed.

The simulator used is run as an application process on the host, and did not have mechanisms to interact with invocations on different hosts. It was also not possible to use standard reliable networking calls for interaction among different simulator invocations as calls with the same names are used in the simulator for other routines. While it may be possible to work around those mechanisms, it would require working with proprietary details of the simulator. All work done with the simulator was done with the proper interfaces and modules provided with the simulator.

The two interfaces that were required and were provided are read-write with files on the system, and the interface to generate the topology and other information on the configuration of the simulation. The first interface is using standard Unix system calls, and the second is a built in capability to provide a readable description of the simulation configuration in terms of the individual components, their attributes and their connections or "links" with other components.

The implementaion of parallelism was done using a Parallel Virtual Machine (PVM). It is a programming environment that provides support for the development and execution of programs that may use task and intra-task parallelism on a set of hosts running the `pvmd` background process. Both task and and intra-task parallelism are used in this method, along with the ability of PVM to broadcast data to the set of tasks that are members of a configurable group. The fact that pvmd background process is required

on the set of hosts that are to run a distributed program indirectly adds a useful check by restricting the locations where the simulation may be run.

Given that the simulation was running several times an order of magnitude slower than real time, and the parameters observed from the simulation were collected over simulated time in seconds, and looking inside the modules for the purpose of redesigning was not an option, a solution had to be devised to be able to scale this simulation. The standard reliable network IPC was not available for use with the simulation binaries. The basic approach around which the routines were developed is as follows. The simulation is to be divided over several hosts, and these individual sub-simulations should be synchronized as well as it should be possible to interchange data among them. This hosts may not all be identical as the same simulator may typically run over several platforms on the LAN, and this also means the individual simulated times at each host may proceed at a different rate. A coordination routine using PVM is used to provide the necessary synchronization.

4 Defining and Coordinating the Sub-simulations

The simulation that initiated this work involved a wide area network with sub-networks. The simulator allows hierarchical building of subnetworks, and links from nodes in the subnetwork can be connected to nodes in other subnetworks. Within one such network, the subnetworks can have both inter-network as well as intra-network traffic. A logical way to define the individual components is to subdivide the simulations using subnetworks. As the simulation needs to run as a single large simulation, the problems that need to be addressed are related to keeping them synchronized. First, they need to proceed in step with each other. Second, the network traffic between the sub-components has to be maintained. Both problems can be addressed, as the simulation had an extremely low efficiency (Simulation Time/Wall Clock time), and the parameters required from the simulation output were averaged over seconds, tens of seconds, etc. The interaction between the subnetworks is in terms of packets addressed to individual nodes in the subnetworks. Reflecting a real situation in networks, the sub-network is identifiable by the destination address in the packet. Sub-dividing the simulation by sub-networks is logical as well as inherently suited by the physical design as within the sub-network the connectivity may be dense. The connectivity to other sub-networks, typically using physical trunk lines, satellite links or other point to point links are likely to be sparser.

The coordination routine starts a given number of tasks and provides a mechanism for keeping the simulated times in step. This routine is written to work with PVM, and assumes pvmd is available on the set of hosts which are to run this *distributed* simulation. During startup, this routine spawns a given number of simulations, and waits till all the instances have read the parameters and are done with initializing the data structures. Once it receives this indication from the simulations, it paces the simulations using barrier synchronization [1].

4.1 Validity of interaction among sub-simulations

The interaction between the sub-simulations can be non-trivial, and ascertaining the validity and accuracy of these interactions is one part that will need to be studied. The key factor that led to this solution as a suggestion is that due to the low simulation efficiency and heavy traffic, a time interval may be chosen over which the traffic may be accumulated and used in the next interval in the adjacent node. It becomes particularly easier if the connections last over several such time intervals, and there isn't a lot of transient behavior. The accuracy and avoidance of deadlock by approximation of the traffic across these intervals is dependent on this time interval. As the coordination is centralized, on a LAN such as Ethernet, it would be easily possible to exchange messages every second for ten to fifteen such simulations, in turn corresponding to less than a tenth of a second in simulated time for each in the original simulation. The packets in the simulator can be time stamped, so the replay on the next sub-network can also have inter-arrival time fidelity. If it is possible to exchange messages more often, say every tenth of a second, the delay may become a non-issue, as it would start approaching the propagation delay of a subnetwork to subnetwork link. Exchanging data from the central coordination routine takes the form of receiving a set of messages from all the processes (simulations) and returning a multi-cast message to the the entire group. The interval chosen also has a bearing on what protocol behavior may or may not be studied in a given simulation, and will require study.

4.2 Designing using the available modules

The first step the above attempt to design the entire simulation was by dividing it by subnetworks over a set of machines, as described above. Synchronizing the simulations becomes a matter of scheduling an event at a fixed simulated-time interval, independent of the rest of the simulation. This can

take the form of an independent clock added to the subnetwork definition, or a periodic event elsewhere in the subnetwork where it will schedule a function corresponding to the barrier synchronization function in the coordination routine. In the test case, for example, the clock was attached to the packet switch that would be part of each subnetwork.

4.3 Resolution of conflicting routines

If it is found that the routine names still conflict in this or a later version of PVM using this arrangement and the problem cannot be solved by separately compiling a library, the PVM part of the code can be split into a separate process communicating using read/write on named pipes or other equivalent mechanisms.

4.4 Resource requirements

Other than the computers and simulators required, as implicitly mentioned here, further work on this activity would require one to two people skilled in using the simulator and should have an interest in developing simulations that will inter-operate. In addition it will require supervision and evaluation of the approach by a researcher who has an overview. It is desirable that one of the two simulator programmers or the researcher have a good understanding of program compilation, linking and execution process.

5 Outline of the solution

This section outlines the test routines and the preliminary solution using one of the network simulators. One piece mentioned above but not included in this discussion is the use of IPC routines. These routines are standard.

For the simulation using this solution it is required that the network model be modified to include a synchronization activity. All protocol modules remain the same, only the overall network (sub-network) simulated has an extra component. This may be added as a separate piece in the network, or as in the case tested, it was placed with the switch module by adding a synchronization event every one second. Such a placement may be useful for subnetworks where the switch is the focal point for traffic exchange. In hybrid networks, the counterparts are easy to find. In this case, for example the model of the WAN packet switching fabric (X.25 cloud) was included to schedule synchronization events every second.

The coordination routine runs several such simulations augmented with the synchronization event. As an example, a routine was written to run several copies of the same simulation. The parts of the routine that may be used in other simulations are included, and the rest is outlined next. The discussion following the code assumes availability of the PVM run time environment configured to use a set of hosts, upto the availability of the PVM command-line. The following use does not need or make any more assumptions about the details of setting up or programming PVM beyond the ability to compile a PVM application and link it with the simulation.[1] The command line to execute the coordination routine with identical simulations (taken for an example only) using PVM may be:

```
spawn syncRun number simulation
```

where `number` denotes the count of simulations to be run and `simulation` denotes the actual command for executing the simulation. The outline of code corresponding to the above synopsis is:

```
/*-------------------------------------------------*/
#include <system include files>
#include <pvm3.h>
main (argc, argv) int argc; char * argv[];
{
int ctid[32];
int howmany, i, w, s, n;
/* group names:                                     */
char *sae = "StartAndEnd"; /* initialization and wrap-up */
char *es  = "EverySecond"; /* synchronization        */
/* Get task id */
i = pvm_mytid();
if (i < 0) {
pvm_perror(argv[0]);
return -1;
}
/* form the synchronization groups */
w = pvm_joingroup(sae);
if (w < 0) {
/* exit as above */
```

[1] The linking may be done by placing the object files in the appropriate place in the compilation / linking process.

```
}
/* similarly */
s = pvm_joingroup(es);
/* if this process is the first to start, */
/* then start the others.                 */
if (w == 0) {
int newArgc;
char *newArgv[];
int c;
/* get the number of tasks from the original */
/* command line.                             */
howmany = atoi (argv[1]);
/*
newArgv, newArgc are assigned from the
commandline, by stripping the count of task
*/
n = pvm_spawn (argv[2], newArgv, PvmTaskDefault, "",
howMany -1, ctid);
/* wait for this to be done, and let others know */
while (pvm_gsize(sae) < howmany);
c = pvm_initsend (PvmDataDefault);
c = pvm_pkint(&howmany, 1, 1);
c = pvm_bcast(sae, 1);
/* error checks after each line skipped */
} else { /* not the first to start */
/* get the total number of tasks. */
pvm_recv (-1, 1);
pvm_upkint (&howmany, 1, 1);
}
/* Initialization and other stuff happens before this */
/* synchronization                                    */
pvm_barrier (sae, howmany);
while (the simulation time lasts)
pvm_barrier (es, howmany);
/* Wrapup */
pvm_barrier (sae, howmany);
.
.
}
```

The condition in the inner while loop, (while the simulation lasts) can be derived in may ways, one example being the broadcast of the simulation duration variable given to the real simulation. To use the coordination routine with a set of simulations the code may be seen in three parts:

- Initializing the routines, and startup. All upto first synchronization call,
- Synchronizing periodically: The while loop with the inner call for barrier synchronization and finally,
- Wrapping up, with the final synchronization call.

In the simulation, once the initial synchronization is complete, the periodic call using barrier synchronization call corresponds to the synchronization call in the simulators. These calls should perform a blocking read/write on the named pipes. The traffic statistics can also be exchanged between the simulations using simple pvm_send() and pvm_recv() calls.

6 Summary

In this note a method for an experiment for collaborative simulation using smaller simulations of hybrid networks is suggested and many relevant details are outlined. A successful test of an application of this work and further study may lead to the development and inter-operation of a group of complex simulations, potentially developed and run at different sites. The suggestion is based on work and tests in a similar situation which is also described.

References

[1] PVM: Parallel Virtual Machine.
Or equivalent programming environment providing the subset of services described above.