Experiences Applying Parallel and Interoperable Network Simulation Techniques in On-line Simulations of Military Networks

Kalyan Perumalla, Richard Fujimoto, Thom McLean and George Riley

kalyan@cc.gatech.edu fujimoto@cc.gatech.edu thom@cc.gatech.edu riley@ece.gatech.edu

College of Computing, Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

We present a case study in which we apply parallel simulation methods and interoperability techniques to network simulations for simulation-based on-line control of military communication networks. The on-line simulations model actual military networks, including wired shipboard sub-networks connected via satellite links, and wireless mobile devices. The modeled scenario depicts the communication requirements of an amphibious landing where a complex network connects troops ashore and naval vessels. The simulations use a heterogeneous set of tools, including ns2 models for shipboard wired networks, and GloMoSim models for the wireless devices. In this paper, we document the challenges we encountered in applying parallel and interoperable simulation methods, and describe our solutions. We describe our experiences in addressing the interoperability problems that naturally arose due to the heterogeneity of scenario models. We also present a preliminary study on the scalability of real-time performance of parallel network simulations, which is crucial for on-line simulations. Salient system characteristics of the subject military network scenarios are described for the benefit of exposure to the modeling and simulation research community. Our exercise not only highlights the relevance of parallel and distributed simulation techniques to an important real-life problem, but also demonstrates the feasibility of applying those techniques in a practical setting.

1. Introduction

As part of some of our recent network modeling and simulation projects[1], we were presented with a challenging problem of delivering parallel and distributed simulation solutions to on-line network management in certain portions of defense networks. The military scenarios of interest involved heterogeneous network models and configurations spanning different network simulators. The objectives of the project called for integrated simulation of a few diverse military network scenarios, executing at least as fast as real-time to aid in simulation-based on-line network control. Interestingly, as the detailed requirements unfolded, it became clear that we were presented with an opportunity to apply parallel and distributed simulation methodology and techniques to realizing the overall objectives. New integration (federating) techniques were needed for the interoperation of the military scenario models written using heterogeneous network simulators. Parallel simulation techniques were called for to achieve faster-than-real-time execution of potentially large network configurations.

When we undertook the work and started developing the integrated execution framework, we were faced with issues concerning interoperability, configuration and performance of the simulated network models for the scenarios at hand. Here we document our experiences along these fronts. First, we present an overview of the simulation-based on-line network control framework and a description of the specific military network configurations and scenarios used in our project. We follow this with an identification of the issues we confronted, along with a description of our solutions, in terms of interoperability, configuration and performance. Finally, we conclude with a summary and status, and identify open issues and future work.

2. Background

We first present the context for simulation-based on-line control in military networks, and describe details of our subject scenarios along with our implementation software framework for their interoperable distributed simulation.

2.1. Military Networks

Modern military operations are becoming increasingly reliant on network communications and connectivity. As the foundation of military command and control architectures, reliable and adaptive communication capabilities can translate to significant operational advantages during actual military engagement. For example, timely and accurate exchange of critical information, such as position updates calls for fire, and medical evacuations can considerably enhance the ability of military personnel to make informed critical decisions, and lessen the probability of "blue-on-blue" (fratricide) incidents.

However, due to the inherent unpredictability and dynamically changing nature of hostile battlefield environments and outcomes during engagement, it is

Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)
1087-4097/02 $17.00 © 2002 IEEE

extremely difficult to provision for, plan and design network operation in advance for reliable and/or efficient operation. For instance, devices can be destroyed or can fail or malfunction. Environmental changes, such as introduction of new obstacles or devices moving out of range of each other, greatly affect the connectivity and quality of network operation. To compensate, military network managers must make real-time decisions about deployment of new assets, or reconfiguration of existing assets during operation. The network managers attempt to maintain the required level of network connectivity and Quality of Service when both the availability of the equipment as well as the load demands on the equipment are changing dynamically.

2.2. Simulation-based On-line Control

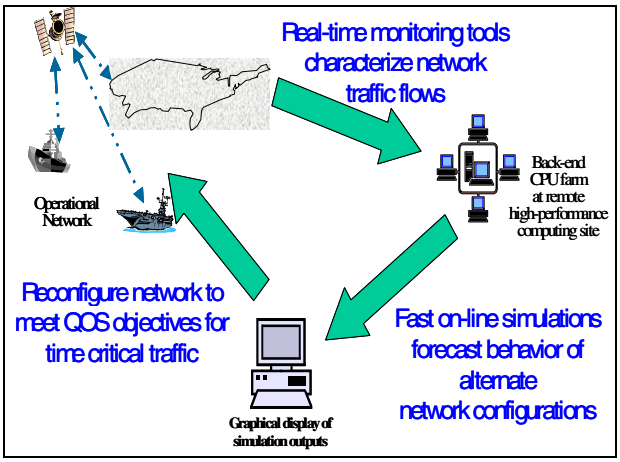

Figure 1: Illustration of simulation-based on-line network control loop.

On-line simulations[5] offer a solution to this problem of dynamic network control. Decisions backed by extensive quantitative analysis can be made by evaluating multiple alternative scenarios and choices concurrently in real-time, and by picking the best alternative from among them. For example, managers can evaluate the outcomes of varying multiple parameters at their disposal, such as introducing new assets, or changing bandwidth or frequency allocations of existing assets, adjusting traffic priority levels, restricting flows from certain classes of traffic, and so on.

The simulation-based on-line network control loop is depicted in Figure 1. Network monitoring tools track the state of the network and its data traffic flows in real-time, and feed that data into fast on-line simulations executed on high performance computing platforms. Multiple concurrent simulations evaluate the effects of multiple 'what-if' scenarios, and produce performance estimates of the reconfigured network scenarios. The predicted estimates are then analyzed, and the optimal configuration is then fed back to the network manager, which in turn initiates the reconfiguration actions on the actual network.

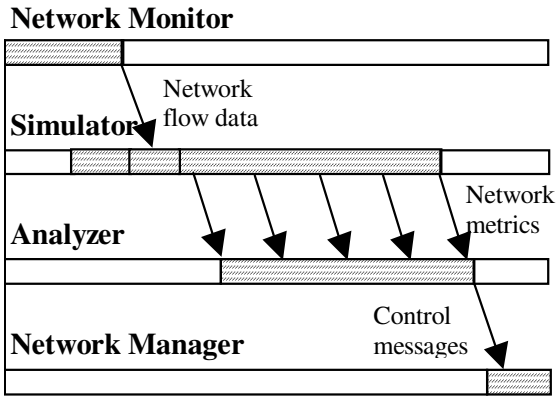

Figure 2: Functional view of the on-line network management loop. Shaded blocks correspond to a single phase in the loop.

In Figure 2, a functional view is shown of the elements and their interactions within the on-line simulation loop. The shaded portions together represent the activity periods of each element corresponding to a single iteration cycle around the loop. The data flow between the activity threads represents dependencies between them. The prefix portion of the simulator thread corresponds to initialization that sometimes can proceed concurrently with network monitoring (e.g. route table computation).

In this research, we investigate the use of parallel and distributed network simulation methods in on-line simulations providing real-time feedback to military network managers. Clearly, to be useful to the manager, the simulation results must be produced at least as fast as real-time, and must represent an accurate picture of the current state of the network. The simulations in our research use actual measured traffic loads for portions of the network, and run a number of different scenarios (such as adjusting priority levels on certain flow classes) that allow the network manager to predict the effect of a set of adjustments and choose the best performing one to adapt to changing network topology and requirements.

2.3. Military Network Scenarios

The integrated simulation scenario that formed the initial subject of our study is illustrated in Figure 3, which shows several ships linked via satellite communicating with amphibious-landing troops.

The sea-based portion of our baseline scenario consisted of 7 ships containing onboard computers. Each ship contains a 100Mbps local area network connecting its onboard computers, and a gateway node on each ship connects the onboard systems to the rest of the world via a 64Kbps satellite link. A variety of application classes are hosted on the onboard systems, including military-

specific applications, as well as conventional applications such as email, HTTP, and chat clients and servers. Within each ship, a significant portion of data traffic stays within the local area network, whereas the rest of the data gets transported over the satellite link on a highly regulated and prioritized basis. Land-based systems and control stations are connected to the ships via their satellite links alone.

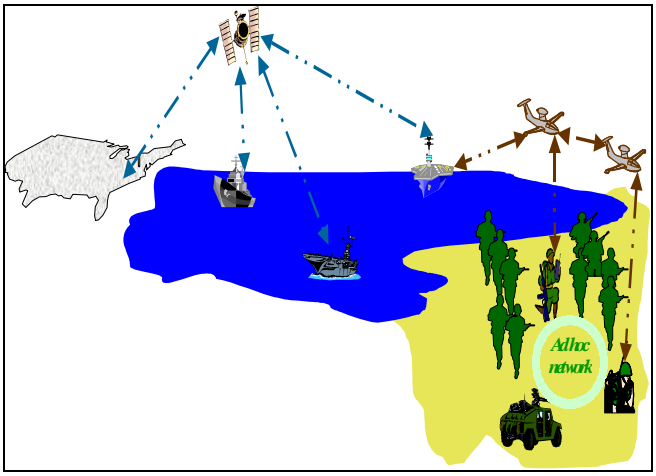

Figure 3: Schematic of the afloat and ground entities communicating in the simulated scenarios.

The amphibious-landing portion of the scenario consisted of 64 mobile entities, each possessing an IEEE 802.11b 11Mbps wireless network interface, operating in ad-hoc network mode. A few of the systems that are afloat are connected to gateway mobile nodes via medium-range (line-of-sight) links. The rest of the mobile nodes communicate with the ships via the gateway mobile nodes.

For simulation purposes, traffic load measurements for the shipboard systems were collected using data collection tools (described later) from an unclassified replica of an operational classified test-bed. The traffic is then summarized and modeled in terms of periodic TCP data generators.

2.4. Simulators and Backplane

The shipboard systems, land-based control station systems and the satellite links were all modeled in the ns-2 simulator[1]. A model of the amphibious landing scenario existed before our project, and was written using the GloMoSim simulator[10] by other groups and validated in projects separate from ours. The ns-2 models were written in a combination of C++ and Tcl languages, while the GloMoSim models were written in C. The GloMoSim scenario included mobility models for movement on the terrain by the mobile entities.

GloMoSim was an appropriate simulator of choice for the mobile portion of the scenario due to its rich set of mobile wireless network models. Similarly, ns-2 accurately fit the requirements of detailed and flexible TCP models for the wired portion of the network.

For integration purposes, we used our dynamic simulation backplane software[5] that facilitates rapid integration of network simulators based on existing network standards. The backplane supports dynamic binding and transformation of network data among heterogeneous simulators.

3. Network Simulator Interoperability

One of the key challenges was the integration of the scenario models written in different simulators, namely, ns-2 and GloMoSim. The shipboard networks and satellite communication models were written in ns-2, while the wireless mobile networks of the amphibious-landing portion were written in GloMoSim. In order to be able to simulate data traffic exchanged across the two models, it became necessary to integrate the two network models so that end points of network connections spanning across the two simulators indeed successfully connect with each other, and are able to exchange data packets between themselves. Thus, TCP sources in GloMoSim needed to connect to TCP sinks in ns-2 and vice versa, and exchange TCP/IP packets between them.

3.1. TCP Model Interoperability

For the TCP endpoints in the shipboard networks, the ns-2 scenario used the *FullTCP* model, which is a fairly detailed implementation of the TCP standard specification. The corresponding TCP endpoints in GloMoSim utilized the built-in TCP model of GloMoSim, which is based on actual BSD UNIX source code for TCP stack processing.

While the TCP models in both ns-2 and GloMoSim are fairly detailed TCP implementations, and mostly faithful to the TCP standards, they possessed sufficiently different properties to make their integration challenging. Resolving each and every disparity is crucial for correct operation – otherwise packets can, at best, mysteriously disappear in either simulator. For example, the TCP input packet-processing code in GloMoSim is especially challenging, containing at least 29 distinct conditions tested on packet contents at different places, any of which can cause an incoming TCP packet to be discarded. In the worst case, the simulators can abruptly fail due to failed assertions or other runtime errors.

Some of the important disparities in TCP handling between ns-2 and GloMoSim are as follows.

1. GloMoSim computes checksums on all TCP packets it generates. Hence it expects checksums on the

packets it receives, and drops the packets whose checksums are incorrect. Ns-2, on the other hand, does not deal with checksums. For packets generated by GloMoSim and destined to ns-2, it is straightforward to import the packet by just ignoring the checksum portion of the header. However, for packets going in the reverse direction, the packet transformation process is more complex since checksums need to be correctly added to the ns-2 TCP packet in order to convert to a GloMoSim TCP packet. Failing to do so would consistently cause GloMoSim to drop all packets originating from ns-2. An easier, but less satisfactory, solution is to turn off checksum checks in GloMoSim, which is what we adopted, in interest of expediency of implementation. Although turning off checksums has the drawback of not dealing correctly with link models that simulate data corruption, it did not present a problem in our scenario that did not use such link models.

2. To support implementation-specific extensions, TCP includes facilities to append optional field values to the header. While GloMoSim utilizes this feature and adds certain optional header information, ns-2 does not contain support for the same, and hence gets quite confused if the packet header includes optional fields. The solution is to correctly parse the TCP headers in the backplane and strip optional fields in GloMoSim-generated TCP packets before submitting them to ns-2. Since optional fields can by definition be ignored, correct operation is ensured despite discarding those values at the receiver end. Again, a less appealing, but more expedient, solution that we adopted is turning off the generation of optional fields in GloMoSim.

3. Packet fragmentation is another feature in which GloMoSim and ns-2 differ. GloMoSim (partially) supports fragmentation of packets, and sets the IP header values accordingly. On the other hand, ns-2 has little support for packet fragmentation. A comprehensive solution to this incompatibility is inherently complex, since it requires addition of the missing fragmentation feature to ns-2, which can entail significant source code modification. We adopted an ad-hoc solution by ensuring that no fragmented packets were ever sent by GloMoSim to ns-2.

4. While both ns-2 and GloMoSim model sequence and acknowledgement numbers in TCP headers, the numbers are mismatched between the two simulators. Whereas GloMoSim starts its numbers at zero, ns-2 generates packets with sequence and acknowledgement numbers starting with negative one. Although truly compatible TCP implementations should be able to accept and honor any starting values for those fields in the headers, it was necessary to convert them in the backplane to the specific values expected by each simulator for correct operation. Otherwise, connections would succeed, but data exchange would fail on both sides – either packets would be buffered indefinitely or would be discarded promptly, due to mismatched sequence numbers.

5. Window size specifications in TCP headers are handled differently in ns-2 and GloMoSim. While GloMoSim has the capability to accept and properly process receiver-advertised window size setting, ns-2 does not. We addressed this problem by keeping the receiver-advertised window size fixed at a default value. Our solution, although expedient, is sub-optimal in scenarios that require experimenting with window size adjustments at runtime.

The remaining protocol header values are generated and accepted by both simulators in a compatible fashion, adhering to the TCP standard. In both GloMoSim and ns-2, TCP SYN packets are acknowledged with SYN-ACK packets, thus allowing new TCP connection requests to complete correctly. Source and destination port numbers, acknowledgements and retransmissions are also modeled compatibly.

The rest of simulator integration was relatively straightforward, using well-known techniques for event exchange via RTI state update interface, and coordinating simulation time advances via RTI time management services. Similarly, it was straightforward to apply *proxy* and *remote link* techniques[6] for capturing packets originating in one simulator destined for sub-networks in a different simulator, and re-routing them to the proper remote nodes.

The biggest problem with achieving interoperability was not in finding the solution to the problems, but in fact in finding the actual sources of the problems. Although the disparities were easy to fix in retrospect, it was extremely time consuming to trace backwards in execution to find the cause of anomalous simulator operation. Once found, each disparity was relatively straightforward to fix.

3.2. General Model Interoperability

Although our preceding experiences have been specific to TCP model compatibility issues, it is possible to extrapolate to other models as well. Similar semantic compatibility issues are bound to arise with ATM network models, for example.

When viewed at a higher level, the interoperability issues we encountered with GloMoSim and ns-2 models can be abstracted in the context of the general problem of

integrating different protocol models. The grand goal of interoperable simulations is to enable the user to pick and choose models from different simulators, and be able to mix and match them as needed by the user. At the heart of the issue is the fact that different models incorporate different amount of detail for the same protocol (e.g. checksum vs. no checksum, fragmentation vs. no fragmentation). Even in the case of matching amount of detail, often, different models have disparate implementations for the same features of the same protocol (e.g., different initial values for TCP sequence numbers, different treatment of TCP window sizes).

Solution Approaches

One approach to deal with this heterogeneity is to require that all models that need to be interoperated fully conform to network standard specifications. For example, we can require that both ns-2 and GloMoSim fully conform to TCP standards. Complete conformance to standards by definition ensures interoperability. However, this can be quite constraining for achieving interoperability. Modifying the models might not be straightforward, and can entail significant development and testing. Moreover, this approach curtails potentially optimized models that abstract away minor operational details of protocols for runtime and memory efficiency of simulation.

Another alternative could be to let the runtime infrastructure such as our backplane implement and supply the missing features, and provide the necessary additions and conversions among the different model variants. This enables plugging in different models without fear of missing functionality. However, a major drawback of this approach is that the integrating infrastructure itself can end up becoming model-heavy, with a large repository of fallback protocol features incorporated to accommodate all possible combinations of models. The infrastructure thus asymptotically approaches becoming a model-rich simulator all by itself.

For any given protocol in general, it appears that the best approach would not be evident until it is attempted to integrate some representative or prevalent models.

4. Simulated Network Topology and Data Specification

Another problem that we faced was that we had to contend with multiple data formats for input and output specification of both network node configuration and network traffic data. The ns-2 portion of the scenario (the shipboard assets, satellite links, and corresponding wired ground equipment) was coded in the Otcl language, as are all ns-2 scripts. The wireless portion of the scenario was given in a Scenario Description File (SDF) format[12] which specifies fixed and mobile wireless communications devices, device mobility, and the data exchanged among the devices.

In order to describe an integrated simulation across differing simulators, there must be some method for each simulator to be aware of and be able to refer to devices defined in the other simulators. Connection endpoints that are on different, and possibly dissimilar, simulators must be known to each other to specify endpoint addresses. Secondly, the specification of the simulation in two different languages puts an undue burden on the user, requiring detailed knowledge of multiple simulation environments.

Our solution to these problems was twofold – one was to develop a unified specification interface, and the second was to develop conversion tools to convert individual formats to the unified interface.

First, we developed a unified interface to both ns-2 and GloMoSim scenarios, by extending the ns-2 configuration to be a union of ns-2 and GloMoSim configuration parameters. These extensions include specification of all information required in the GloMoSim scenario configuration files. The single ns-2 interface to both ns-2 and GloMoSim enabled the user to describe and execute a heterogeneous simulation using a single description file, containing both ns-2 and GloMoSim information in a common format. The drawback of this approach is that it is not simulator-neutral, and favors one simulator as the master over another. A more satisfactory approach would be to adopt a more standardized method for configuring network simulators (say, an XML-based standard topology, traffic and mobility specification).

Secondly, we implemented an SDF to Otcl conversion tool, which reads the GloMoSim-specific SDF file, and produces an ns-2/Otcl format file containing network configuration and data flow descriptions matching those in the SDF. The tool helped us avoid manual conversion of the rather voluminous data from the SDF file to the Otcl format. Although this effort is work in progress, the current implementation possesses sufficient functionality to produce a reasonable subset of the total information found in the SDF.

5. Network Monitoring, Analysis and Reconfiguration

As described previously, the on-line simulation loop consists of network monitoring tools feeding into analysis engines that in turn spawn on-line simulations to evaluate and pick from multiple alternative scenarios, and the best configuration is effected on the live network via a network management/reconfiguration tool. In our particular military network scenario, we utilized a data collection tool called CoralReef developed by CAIDA

[14] for sniffing and filtering the relevant data traffic on the local area networks. The summarized traces obtained from CoralReef were used to feed into multiple simulation instances. For an initial proof-of-concept capability, we employed manual observation and manual scenario generation for the analysis portion of the control loop. For the reconfiguration phase, we were unable to effectuate actual reconfiguration of the network, primarily due to constraints on access to the network test-bed hardware. However, we do not foresee any significant hurdles in actually employing the reconfiguration recommendations if sufficient privileges are available to operate on an actual operating network.

6. Performance Evaluation

We now present our preliminary performance study aimed at analyzing and optimizing the real-time performance of the scenario execution, and understanding the limits on the scale of larger scenarios that can be simulated in real-time for on-line control.

6.1. Low-latency communication

In order to achieve scalable real-time performance, it was clear that low latency inter-processor communication support was necessary for fast parallel execution. To this end, we chose the commonly available shared-memory symmetric multiprocessor (SMP) platform, and implemented a fast shared-memory communication mechanism. Our communication algorithm was carefully designed to be both portable and efficient, and has been tested on multiple platforms including SGI, Sun, HP and Intel SMPs. On our target platform, which is Linux on Intel processors, round-trip latency is observed to be 5-7 microseconds (for messages up to 4 Kilo bytes) on an Intel 8-way SMP box with 550MHz processors, with a total of 4 GB of memory.

During initialization, the UNIX System V interface is utilized for portably mapping shared memory segments among processes. Since system calls for synchronization are expensive (typically consuming dozens of microseconds per call), we avoided using system calls beyond initialization phase. Instead, we devised and used a pair-wise buffer pool scheme that allows efficient synchronization for message exchanges among processors without incurring system call penalty.

6.2. Simulator Pre-Initialization for Static Network Configuration

Initializing the simulator even before flow data is obtained from network monitors can save simulator initialization time, thus reducing the overall time to simulate the scenarios. If the simulated network is static in nature (i.e., network size and topology doesn't change over time, but only the data traffic changes), then the simulator can proceed with initialization of routing tables and other data structures even before receiving flow data information from the monitors. Since routing table computation can be a time-consuming operation, pre-computing it concurrently with network data monitoring eliminates initialization time from the main control loop, and hence helps reduce overall control loop latency. While, this optimization is possible for static network configurations, the same may not always be possible for networks with dynamically changing connectivity, such as in mobile networks. In the specific scenarios used in our project, since the connectivity did not often change within the ns-2 portion of the network models, we were in fact able to apply this optimization and thus eliminate ns-2's expensive runtime overhead of route computation.

In general, however, to achieve satisfactory real-time performance, it is necessary to address the effect of route computation during initialization. Approaches such as Nix Vector [5] can help compute only the routes that are actually needed during simulation by the specific scenario, thereby avoiding up-front runtime cost. This becomes especially true in the case of large network configurations simulated for relatively short periods of operation.

The GloMoSim portion of the network used dynamic routing models (Bellman-Ford) due to the ad-hoc mobile nature of the wireless network devices which makes network route setup happens inherently at runtime as part of actual simulation, and could not be easily optimized via pre-initialization or on-demand route computation.

6.3. On-line vs. real-time

It is worth noting the difference between on-line simulations and strict real-time simulations. In traditional real-time simulations (e.g. emulation), each and every event is executed when wall clock time reaches the timestamp of that event. In on-line simulations, on the other hand, individual events need not be constrained by wall clock time. Instead, it is sufficient if t seconds of simulated time is simulated in t' seconds of wall clock time where $t' <= t$. This means that, even though bursts in simulation load might violate time constraints in strict real-time simulations, it is possible to even out such bursts over time in on-line simulations. Hence, in our performance study, for the purposes of on-line simulations, we consider a simulation to meet real-time if it completes simulation of t seconds of simulated time in less than or equal to t seconds of wall clock time, without regard to any relation between intermediate points in wall clock and simulated times.

Note that every real-time simulation can always serve as an on-line simulation, but not necessarily vice versa.

6.4. *Scalability of Real-time Execution*

While our initial set of scenarios consisted of relatively smaller network configurations, larger-scale scenarios are envisioned to include multiple instantiations of the smaller scenarios, from regional to world scale. In anticipation of scenarios with larger network configurations, we undertook testing the scalability of the simulation tools. We devised a series of experiments to evaluate scalability limits with increasing network size and data traffic.

The aim of this study is not so much in investigating parallel simulation performance, but in empirically determining the scale of network sizes and traffic volumes that we can realistically expect to simulate in real-time on off-the-shelf hardware.

For the purposes of this study, we focused on the shipboard networks portion of the scenarios, and used a parallel version of ns-2, called pdns[6]. The shipboard sub-networks were naturally partitioned across processors by allocating all nodes belonging to the same sub-network to the same processor. Such partitioning naturally exposed satellite link latency as the lookahead value across processors (approximately 230 milliseconds), since networks across ships communicated only via satellite links.

The network is scaled by first starting with our baseline scenario of local area networks connected via satellite links, including the data traffic generators from the scenario. These data sources generate sufficient amount of off-ship traffic to almost always keep the low-bandwidth satellite link fully saturated. Additional nodes are added by adding nodes within each local area network. Along with each added node, a new data flow is added with the newly added node as the flow source, and a randomly chosen node as destination node within the same local area network as the added node. Keeping the newly added flow local to the ship is important since any extra remote communication will result in little increase in simulation load, as the satellite link is already fully loaded. Hence, our scaling scheme ensures increasing simulation load with increasing network size, which is a worst-case condition for network scalability. This roughly corresponds to simulating a plausible future scenario in which the number of shipboard systems is increased.

Figure 4 shows the time taken to simulate 300 seconds of the network behavior as the total number of nodes in the network is increased. The experiments were run on an 8-CPU SMP box with each processor being a 550MHz Intel Xeon processor, with 4 GB of main memory.

It is observed that while sequential ns-2 manages better than real-time performance until the number of nodes nears 700, pdns keeps up with real-time up to 4500 nodes and beyond. This shows that while our initial scenario configurations can be easily executed by sequential ns-2, larger-scale scenarios will need, and can indeed benefit from, parallel execution.

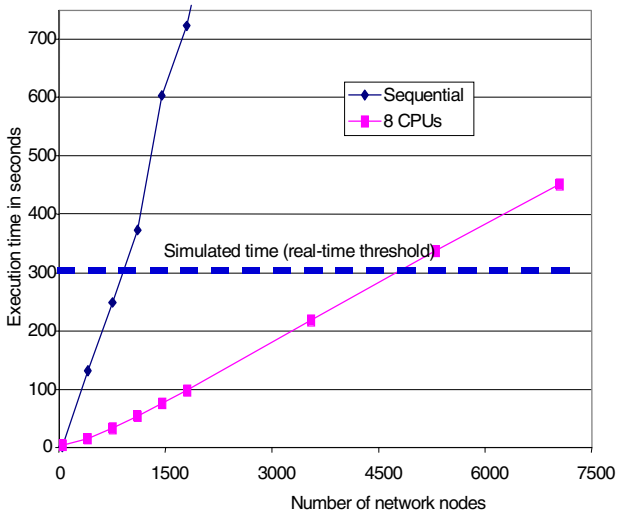

Figure 4: Real-time performance of sequential ns-2 and parallel pdns. Pdns is observed to deliver seven-fold increase in network size compared to sequential ns-2, showing that thousands of network nodes can be simulated in real-time.

7. Related Work

While interoperable simulation techniques have been the subject of study and implementation in the defense community for training and other purposes, little work focused on federated simulation of heterogeneous models of defense networks. More generally, very limited work so far addressed the problem of rapid integration of heterogeneous network models across different network simulators. For example, relatively little is known about how to make different TCP models interoperate with each other seamlessly across simulators.

The work most closely related to this problem is [5][8], in which heterogeneous simulation is performed by transporting network data of a TCP model in one simulator over an IP transport model in a different simulator. Here, we address the problem of directly interfacing disparate TCP models belonging to different simulators. The direct interfacing raises interoperability issues that are in a way different from those in other interoperability scenarios. In previous work, inter-simulator data exchanged across simulators is treated as baggage – the data gets generated in a layer of one simulator and is transported over a lower layer model in another simulator. However, when different models belonging to the same layer are integrated in a peer-to-peer fashion, as in the military network scenarios here, data can no longer be considered baggage, and hence

semantic interoperability for exchanged data needs to be addressed.

An approach to on-line network simulation is described in [11]. This work uses an approximate parallel fixed-point computation to simulate the network, as opposed to packet level simulation that is the focus of our work. Simulation interoperability has also been dealt with in the SEAMLSS project [12] where an approach based on the Department of Defense High Level Architecture [13] is used. This work has not addressed the types of interoperability issues (e.g., TCP end nodes modeled in different simulators) addressed in this paper.

8. Conclusion

Our case study serves to demonstrate the relevance and applicability of both parallel network simulation techniques as well as interoperability techniques for heterogeneous network simulators in practical applications. This study also identifies the issues and challenges that arise in the application of parallel and distributed network simulation solutions in a practical setting.

It is interesting to note that the size of networks and intensity of data flow in the scenarios of interest are well within the capabilities of state-of-the-art parallel simulation techniques. In contrast to general Internet research into very large-scale network simulations, the military networks of interest appear to be of manageable size for fast real-time simulation-based on-line control.

However, in general, there are open issues that remain to be resolved to fully enable seamless interoperability and plug-n-play operation of heterogeneous network models (*e.g.* for use in web-based network simulation).

For satisfactory real-time performance, low-latency inter-processor communication mechanism is clearly necessary. Our shared-memory message-passing module enabled us to achieve the desired latency on the shared-memory platform. However, this limits scalability to the number of processors per SMP box. To achieve an additional order-of-magnitude scalability (100 or more processors) we are now developing a combination of shared-memory and Myrinet-based hierarchical low-latency communication support. Although other implementations of hierarchical communication libraries exist, such as certain MPI implementations, our experience showed that they entail unsatisfactory latency overheads for simulation applications. Further work is needed to explore the best solution for scalable low-latency communication for parallel network simulation.

References

[1] Richard Fujimoto, Kalyan Perumalla, George Riley, *Flexible, Efficient Network Emulation – A Backplane Approach* – web page, http://www.cc.gatech.edu/computing/pads/nms.

[2] Thom McLean, Richard Fujimoto, *The Federated-simulation Development Kit: A Source-Available RTI,* Spring Simulation Interoperability Workshop, March 2001.

[3] The Network Simulator – ns2 homepage, http://www.isi.edu/nsnam/ns/.

[4] The OPNET Network Modeler – web page, http://www.mil3.com/products/modeler/home.html.

[5] George Riley, Mostafa Ammar, Richard Fujimoto, Kalyan Perumalla, Donghua Xu, *Distributed Network Simulations using the Dynamic Simulation Backplane*, the International Conference on Distributed Computing Systems, April 2001.

[6] George Riley, Richard Fujimoto, Mostafa Ammar, *A Generic Framework for Parallelization of Network Simulations*, the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, October 1999.

[7] Boleslaw Szymanski, Yu Liu, Anand Sastry, Kiran Madnani, Real-time On-line Network Simulation, Technical Report 04-01, Department of Computer Science, Rensselear Polytechnic Institute, 2001.

[8] Donghua Xu, George Riley, Mostafa Ammar, Richard Fujimoto, *Split Protocol Stack Network Simulations Using the Dynamic Simulation Backplane*, the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, August 2001.

[9] Tao Ye, et al, Network Management and Control Using Collaborative On-line Simulation, the International Conference on Communications, 2001.

[10] Xiang Zeng, Rajive Bagrodia, Mario Gerla, *GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks*, the 12th Workshop on Parallel and Distributed Simulations, May 1998.

[11] B. K. Szymanski, Y. Liu, A.Sastry, and K. Madnani, *Real-Time On-Line Network Simulation*, the 5th Workshop on Distributed Simulation and Real-Time Applications, pp. 22-29, August 2001.

[12] The SEAMLSS Project, http://www.seamlss.com.

[13] High Level Architecture, Defense Modeling and Simulation Office, http://hla.dmso.mil/.

[14] CAIDA, *The CoralReef Software Suite as a Tool for System and Network Administrators*, http://www.caida.org/tools/measurement/coralreef.