

A Distributed Simulator for Large-Scale Networks with On-Line Collaborative Simulators

Ji-Fang Zhang, Jingjie Jiang, Boleslaw K. Szymanski
Department of Computer Science
Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Abstract

Management of large-scale heterogeneous networks has become a bottleneck to efficient use of the fast growing global network. The complexity, heterogeneity and speed of the Next Generation Internet (NGI) require new, scalable approaches to network management and control. Towards this end, we are developing a system of collaborative, on-line simulators that can support a suite of distributed network management and control functions. We believe that such a system will improve automatic network management thanks to increased awareness of the global, medium and long range temporal information about network state gathered by each node and processed by the on-line simulators.

In this paper we described a distributed, parallel implementation of a simulator that models a network equipped with the on-line simulator. The essential feature of this simulator is disassociation of the real network simulator from the simulation of the actions of the online simulators. We describe also simulations that we run to demonstrate that on-line experiment design based on on-line simulator can indeed improve network performance.

1 Introduction

Recently, management of large-scale heterogeneous networks has attracted a lot of attention in response to the fast growing global network usage and applications. The complexity, heterogeneity and speed of the Next Generation Internet (NGI) require new, scalable approaches to network management and control. Towards this end, we are developing a system of collaborative, on-line simulators that can support a suite of distributed network management and control functions. Our initial step has been to design and implement a simulator

for collaborative on-line simulators to experiment with different parameters selection in network management. In the future, we plan to use on-line collaborative network simulators for advanced feedback-based traffic management, stabilizing Internet routing, and admission control for premium network traffic classes. We believe that such a system will improve automatic network management thanks to increased awareness of the global, medium and long range temporal information about network state gathered by each node and processed by the on-line simulators. The proposed distributed, collaborative on-line simulation approach has several advantages for the management of large-scale, heterogeneous networks. Using on-line simulation one can achieve on-line parameter sensitivity analysis and tuning for improved stability and performance of distributed network management and control algorithms. Since each simulator represents a local parameterized view of the network, it is simple and can execute very quickly. With on-line simulation the fault tolerance will be improved and if some network nodes are temporarily cut off or out of order from the rest of the simulations the remaining nodes will still be able to produce reasonable simulation results based on the most recent information. The approach and the research to network management developed in this project has the potential to affect the whole spectrum of network management tools and techniques.

The first stage of this large project, which we describe here, uses a standard Network Simulator (NS)[1] software package, the ERICA[2] congestion avoidance algorithm, and a full factorial experiment design[3] to test the assumptions of our approach.

The Network Simulator (NS) is well equipped to build fast and efficient simulation of a network for which we want to test our approach. However, we want to embedded in the real network a set of on-line simulators that will gather information about the traffic flowing through the network, build traffic models and be used to run selection of the traffic parameters. In short, the final simulator must not only simulate real network (let's call it RN) but also on-line network simulators (called ONS) which run experiments with different parameter sets (full factorial experiment design).

The naive solution would be to embed the actions of the on-line network simulator within the real network simulator which in effect will square the complexity of the simulation (potentially, each event in the real network simulation can trigger the network simulator invoked by the experiment run by the online simulator). The experiment triggered simulations need to be coordinated in simulated time with the real network simulation (the delay between the results of experiments produced by the on-line simulator and the current traffic in the real network are important for stability of the network management decisions). Such a design is also likely to require square of the time of the simple real network simulation again because the network simulation is called by the on-line simulators during the real network simulation.

We have found the way of disassociating events of the real-time simulation from experiment design triggered network simulation and reducing coordination between them to the minimum. As a result, the simulations can run on distributed architecture (we used a net-

work of workstations) with large degree of parallelism. Description of the design of the this complex simulation is a subject of this paper.

1.1 Network Simulator Software Package–NS

The Network Simulation(NS)[1] package has been developed by a DARPA-funded research project as a real network simulation tool. NS is a discrete event simulator that provides substantial support for simulation of TCP, routing, and multicast protocol (but lacks, for example ATM support). It grew out of the REAL network simulator designed in 1989 and has evolved substantially since then.

We use the NS version 2 written in C++ that uses OTcl as a command and configuration interface. NS decomposes the more complex object into simpler one for greater flexibility and composability. OTcl, an object oriented version of Tcl is used for defining configuration interface. The interface code to the OTcl interpreter is separate from the main simulator. We choose it as a main tool for transferring network definition between simulators.

1.2 ERICA–Explicit Rate Indication for Congestion Avoidance

ERICA is an algorithm for congestion avoidance in networks and is used by the on-line simulators in finding the new set of traffic control parameter during experiment design stage. ERICA provides congestion avoidance for Available Bit Rate (ABR) service with the following goals:

1. Maximize link utilization.
2. Equalize share of the network resources (to be exact, “fair share”, based on the resource utilization).
3. Return the system to steady state after any perturbation (if a steady state exists). A system with good transient performance returns quickly to the steady state.
4. Ensure acceptable performance when the system experiences constantly changing capabilities and demands.

The details of ERICA algorithms are available elsewhere [2]

1.3 Experiment Design

A $2^k[3]$ factorial design is used by the ideal on-line simulators in search of the improved traffic parameters. An on-line traffic simulator calculate the statistics of the observed traffic, constructs a traffic model and exchange its model with the neighboring on-line simulators. Once the model is constructed, the on-line simulator is ready to search for the parameters that improve the performance of network. For the Internet routers, there is a limited set of parameters (5 or 6 for traffic management) which can influence the performance of network. Each time OCS will try all extremal combinations (2^m of them for m parameters) for each function OCS want to optimize. By sorting the argument-result table according to the result value, the OCS can find out which parameters should be changed to improve the performance. In simulation, the traffic model is simple copied from the real network simulation. Hence, the detailed procedure is as follows:

1. OCS constructs a model according to the call parameters from the NS simulator. The parameters include topology, traffic model etc.
2. For traffic management, the OCS select five base parameters and determines the range of values for each one.
3. Based on smallest and largest value in the interval of the parameters, the OCS constructs the set of 2^m combinations of parameter values and runs simulation of the network for each combination.
4. The results of all experiments are sorted and the best combination of parameters selected for the new traffic control.

2 The Design and Implementation of Distributed Simulation

The essence of the design idea is the separation of OCS and RN simulators, the operation of OCS does not depend on the Network Simulator (NS) we use. The conceptual and practical separation of these simulators will be very important in later, non-ideal simulations. This separation can be visualized by assuming two connected computers with one running the Network Simulator and other executing simulator OCS. Network Simulator takes a snapshot of the network state, passes this to the simulator OCS. OCS looks into future by simulating the network and providing new set of parameters. And finally the NS takes actions (in proper time instance) of changing the parameters to improve the network performance.

The basic function of simulator OCS is to simulate the network for a given set of traffic parameters, routing and topology information. It searches through the valid parameter space

and identify an optimal parameter set using efficient search methods. This parameter set will be used for routing and network management. Presently, we assume a centralized simulator OCS. The simulator has global information about the entire network. In the next stage of this project, we will introduce collaboration between simulators and limit information of each simulator to a subnetwork. This will require some exchange of information between the simulated OCS's at various nodes or routers. The on-line simulator will collect traffic from the node, characterize it by fitting a suitable model and use this model for simulating the network traffic.

2.0.1 The functionality of OCS

Each time OCS is invoked, it does the following work. First, it extracts topology data from NS simulator by collecting traffic data from the network based on the NS simulation of the real network up to this instance of the simulation time. Next, the OCS characterizes traffic of the real network and constructs its traffic model. To create more realistic simulations, only sources activated by the time of OCS invocation are copied from the NS script. Based on the traffic model, the OCS obtains the current and then recalculates values for traffic management parameters that will influence the future traffic.

The OCS simulator is implemented with a separate code and communicates with NS simulator by a clearly defined interface: a socket for synchronization of NS and OCS simulators and data exchange.

The detailed implementation includes the following changes and additions to the NS simulator.

1. Changed NS source code, to make NS to recognize and process two new events: (1) Start-OCS with time and network status passed, and (2) OCS-return with time and results passed.

The time defines the simulation time of the event.

2. Extended NS script to create a series of invocation of OCS simulator at the predefined instances of the simulation time. The invocations will be made periodically, starting with time zero with periodicity defined by the user. This addition of the script would make NS simulator to place the events in the proper place of the future event heap.
3. Implementation of Start-OCS event as a sequence of the following steps:
 - (a) Create a message from-NS-to-OCS with the topology (incrementally) and the simulation model (incrementally).
 - (b) Sent the message to OCS simulator to start with the start time.

- (c) Wait for the message from OCS simulator with the return time.
 - (d) Create and push on heap event OCS-return and exit the event execution.
4. Implementation of OCS-return event in the following steps:
- (a) Read the data from the from-OCS-to-NS message buffer.
 - (b) Update network model with these data.
 - (c) Exit the event execution.
5. Implementation of the simulation of OCS simulator as the following loop:
- (a) Wait for the message to start with the start time.
 - (b) Read the from-NS-to-OCS message buffer with the network definition.
 - (c) Compute the simulator delay as the function of the OCS simulator speed and complexity of its execution.
 - (d) Compute the time of data return as the sum of the start time plus simulator delay.
 - (e) Send the message to NS simulator with the return time.
 - (f) Execute an algorithm for parameters optimization.
 - (g) Pass the results via the from-OCS-to-NS message.
 - (h) Loop back to the first step.

The most important change in the implementation of the overall system is synchronizing NS and OCS simulators through two new events on NS heap.

2.0.2 The Interface between OCS and NS

At this stage of implementation, we assume that the traffic models etc. are known. When simulator OCS is called, only the parameters for the model and some information about network topology etc. is passed. We use the input script file of NS to pass the topology and traffic information to OCS simulator. This is applicable to the ideal on-line simulator which knows all the necessary information about the whole network. This method should be replaced by some more realistic one for the non-ideal simulator in the future. For the start-OCS command event OCS passes back ΔT , the estimated time needed for OCS ready to send back the results. For the OCS-return command event currently OCS passes back script file to NS.

According to the design specification described above we extended NS source codes to make it recognize and execute two new command: OCS and OCS-return. In the implementation we use of sockets synchronizing OCS with NS and passing the event times data between NS and OCS back and forth. We also implemented a procedure for inserting new parameters into the routing table of the network simulated by NS during NS execution.

2.1 The Performance Issue

After we integrated our codes with the NS simulator code, we have obtained some preliminary results which gave a proof-of-concept for improving traffic management via on-line simulation. However, performing a full experiment design using on-line simulation takes a lot of time in the current design. To improve this time, we designed a scheduler and dispatch mechanisms of having multiple OCS servers communicate with a centralized OCS server directly. This on-line simulation framework uses a farmer-worker model. The essential to this model is a farmer which distributes several experiments to other machines(workers) to run in parallel. By parallelizing worker code (which includes at least two runs of the simulation, so this parallelism is easy to extract) we have achieved a two-level parallelism. As in case of NS-OCS synchronization, we use sockets for the interface communications between the distributor(farmer) and workers. The workers are separate processes running on different machines. The farmer is a distributor function which is compiled together with the central OCS simulator codes. When the OCS start to run experiments it sends TCL script files and parameters through the sockets. The workers write results back through the sockets as soon as experiments complete. A immediate issue is there may be mutual exclusion problems when different experiment results finished trying to write back through socket. We use semaphores to solve this problem and use threads to dispatch experiments.

3 The Implementation Results and Discussions

We conducted an experiment on a simple network topology with eight nodes, full-duplex links of 1Mb bandwidth and 10ms delay and drop-tail queuing (larger example will be included in the final version of this paper). The set of parameters sent to OCS consist in this case of at least two: average interval and activity decay. The NS simulation runs 2.0 seconds. The start-OCS event is invoked at 0.2s. We run the two workers at two additional machines to the one which runs NS and OCS distributor. In simulations, the network experiences the packet dropping before OCS is invoked and packets stops dropping after the OCS sends back the new parameters. It demonstrated the benefit of parameters selection.

The running times used with farmer-worker model and without farmer-worker model are shown in Table 1 (we used time command to record the running time) when two, four or eight parameters are sent to OCS. This Table shows an improvement attributable to the use of the farmer-worker model. The number of experiments run is proportional to the square of the number of parameters sent, so the parallel efficiency quickly raises with this number.

Number of parameters	Sequential run-time	Farmer-worker run-time
2	38.401s	22.971s
4	47.705s	25.422s
8	1m48.916s	25.500s

Table 1: Comparison of running times for the two models: sequential and distributed (farmer-worker).

3.1 Future Improvement and Continuing Work

Presently we assume a centralized simulator OCS, i.e. the simulator has global information about the entire network. However, with the introduction of collaboration, each simulator will have information only about its subnetwork or its own node. In this case, there must be some exchange of information between the simulator OCS at various nodes/routers. The on-line simulator will collect traffic from the node characterize it by fitting a suitable model and use this model for simulating the network traffic.

References

- [1] See <http://www-mash.cs.berkeley.edu/ns/ns.html>.
- [2] A. Charny, D.D. Clark, R. Jain, “Congestion Control with Explicit Rate Indication”, In Proceedings of ICC’95, June 1995.
- [3] Raj Jain, “The Art of Computer Systems Performance Analysis”, John Wiley & Sons, Inc., 1991.