# Traffic Management and Network Control Using Collaborative On-line Simulation

Tao Ye[1], David Harrison[2], Bin Mo[2], Shivkumar Kalyanaraman[1], Boleslaw Szymanski[2], Ken Vastola[1], Biplab Sikdar[1] and Hema Tahilramani Kaur[1]
[1]Department of Electrical, Computer and System Engineering
[2]Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180
{yet3, shivkuma, bsikdar, hema, vastola}@networks.ecse.rpi.edu
{harrisod, mob, szymansk}@cs.rpi.edu

*Abstract*— **The complexity and dynamics of the Internet is driving the demand for scalable and effective network control. This paper proposes a collaborative on-line simulation architecture to provide pro-active and automated control functions for networks. The general model includes autonomous on-line simulators which continuously monitor/model the network conditions and execute a search in the parameter state space for better settings of protocol parameters. The protocol parameters are then tuned by the online simulation system. In this paper, we describe the building blocks of this architecture and investigate the implementation challenges in the areas of network modeling, on-line simulation and parameter search. We also discuss the applicability of this system and present the simulation and test results of a preliminary implementation.**

## I. Introduction

With the Internet expanding at an explosive speed, there is a need for scalable, automated network management functionality. We propose a collaborative on-line simulation architecture to address this objective, albeit in a limited scope. The basic idea of the architecture is illustrated in Fig. 1.

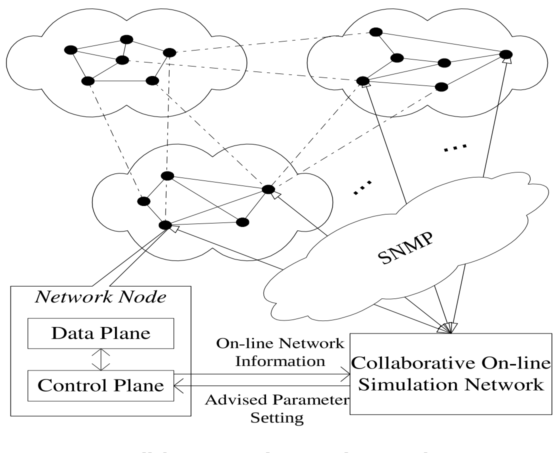

SNMP
Network Node
Data Plane
Control Plane
On-line Network Information
Advised Parameter Setting
Collaborative On-line Simulation Network

Fig. 1. Collaborative on-line simulation architecture

The collaborative on-line simulation architecture operates in the the *management plane* and interfaces with the *control plane* of the network. In particular, it does not interfere with the packet-by-packet data-plane operation of the network. We term this as *"second-order" control* over network functions. The architecture is mainly composed of autonomous on-line simulators which continuously monitor and model the network conditions and topology. Based upon the on-line model of traffic and topology, the simulators can execute simulations to evaluate the performance of the network for a given set of protocol parameters. The assumption is that network control protocols (e.g.: traffic management, routing protocols) are sensitive to traffic loads and a subset of their parameters. The goal then is to have the online simulation system use sophisticated parameter search methods to search for better parameter settings applicable to the current traffic and topology mix. In other words, the simulation system can support continuous tuning of the network based upon the online modeling, parameter search and simulation capabilities. The on-line simulation scheme uses a best-effort parameter search strategy whose emphasis is not on "full" optimization, but on continuously and increasingly moving the system towards a "better" operating point. And in this sense, on-line simulation equips the network management infrastructure with pro-active, dynamic and automated management capabilities.

Now, we will discuss the problems faced in the on-line simulation scheme, such as, network modeling, on-line simulation and parameter search, and investigate possible solutions. In Section 2, we describe the basic structure of on-line simulator. Section 3 introduces our work in on-line modeling, especially the generation of realistic network traffic. Section 4 describes our approach to the efficient parameter search. Section 5 discusses two ways to speed up the network simulation. In particular, our approach for topology decomposition shows a very high speed-up in large-scale simulations. Section 6 and 7 present the simulation results under *ns* and the validation experiment results under Linux. Section 8 discusses the applicability of on-line simulation to routing algorithms. Section 9 concludes this paper and points out the areas for further research. In particular, the collaborative aspects between multiple simulation components will be discussed in future papers.

## II. Structure of On-line Simulator

Consider a network with a management interface and an on-line simulation back-end system. Fig. 2 shows the basic structure of such an on-line simulator system.

The simulator is composed of the following function units: *monitor and modeling*, *experiment design*, *management interface* and *experiment execution*.

*Monitor and Modeling* unit continually collects all kinds of information about the local network, such as network

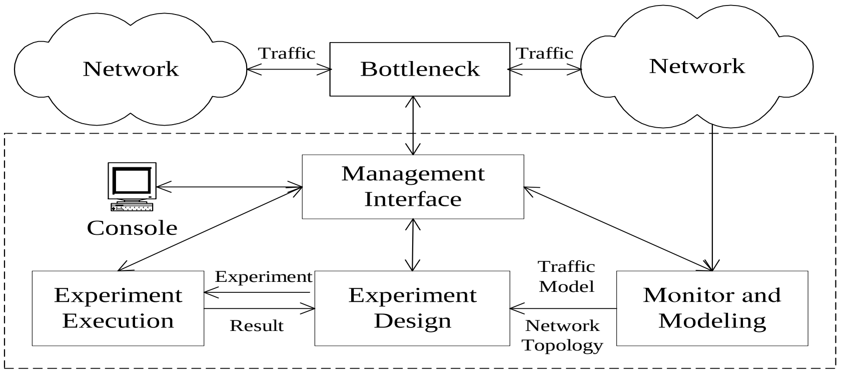


Fig. 2. Structure of on-line simulator

topology, traffic conditions, and tries to build the most updated network model for use by on-line simulation.

*Management Interface* unit is the control center of the on-line simulator. It controls and synchronizes the operation of all the other units. Meanwhile, it is also the interface of the on-line simulator with the outside world. Through this interface, the network administrator can control and monitor the operation of the on-line simulator, such as, choosing search methods and the parameter space to be optimized, examining network status.

*Experiment Design unit* is responsible for setting up *simulation experiments* with all the collected network information and search for the good parameters in the given parameter space with appropriate search techniques.

*Experiment Execution unit* executes the simulation experiments received from *experiment design* unit and returns the results to the designer. The network simulator *ns* [1] developed by UCB/LBNL and the VINT project has been chosen as our current simulation platform for its reliability and wide acceptance. However, this is not the only choice, and any other faster simulation software can be used as execution unit as long as it can provide the efficient and reliable network simulation.

Besides interacting with the local network, the on-line simulator also communicates with other simulators and exchanges the relevant network information, such as network traffic models, good network parameters. Thus, a collaborative and scalable on-line simulation network is formed. Through this, the local simulator acquires a global view of the network and is able to perform better network simulation and control.

## III. On-line Modeling: Workload Generation

On-line modeling is to create traffic models, topology models, protocol models, etc., for use in the simulation to reconstruct network scenarios. On-line modeling, especially traffic modeling, is greatly complicated by the complexity and heterogeneity of the Internet. We will first address the problem of on-line traffic modeling. A presumption of our work is that since we do not understand how to build stationary models of Internet traffic patterns, a reasonable approximation of "current" traffic behavior could be characterized by quasi-stationary models, and such models can be constructed on-line. An important problem in the on-line traffic modeling is how to generate the realistic traffic in the simulation.

The first issue in generating the realistic traffic is to maintain the proper traffic composition in the simulation, which is essential to capturing the behavior of wide area networks. For the Internet, the dominating applications are WWW, Telnet, FTP, SMTP, and NNTP. We use the empirical distributions in [5] to characterize the underlying protocols for these applications. We implement these application-specific traffic generator under *ns* and deploy some measures in the simulation to maintain the proportion of these traffics according to the empirical data from [4].

Another important issue with traffic generation is self-similarity in wide area and Ethernet traffic[5]. Generation of aggregate traffic which is self-similar in nature is of utmost importance in simulation scenarios as Poisson models grossly underestimate the queuing delays and overflow probabilities. We have implemented in *ns* two self-similar traffic sources: Application/Traffic/SupFRP and Application/Traffic/SS, respectively based on the algorithms proposed in [6] and [7].

All the protocol specific, application specific and self-similar traffic generators described above have been validated through extensive simulation and experimentation. The detailed results and analyses are presented in another paper[8].

## IV. Experiment Design

Since the network conditions keep changing all the time, the on-line simulation scheme needs the fast experiment design method to quickly find the "good" network parameters before the underlying network information becomes obsolete. The goal of the experiment design is to use the minimum number of experiments to find as good a parameter setting as possible. Note that here the emphasis is *not on seeking the optimum setting*, but to *find a better operating point within the limited time frame*. Based on this principle, an iterative method, which leads the search to the optimum point in a monotonic fashion, would be desirable for the experiment design. Given such a search algorithm, the search can be interrupted at any time and still produce a result likely to be better than the starting point. This allows us the flexibility to design a tradeoff between the goodness of the parameter result and the search time.

The basic procedure of our approach is to first probe the search space roughly and find the important parameters which have greater effects on the network performance. After pruning part of the search space by ignoring less important parameters, we explore the promising search space in more detail. To perform the fast and efficient search in the concerned parameter space, we designed a new hybrid search algorithm which combines the strengths of various search aglrithms and is especially suitable in the context of our problem.

### A. *Probing the Search Space*

As described above, we start our search process with the *rough* probing of the search space. For this purpose, simulations will be conducted *first* to explore the boundaries of the search space. These simulations will be based upon $2^k$ full factorial experiment design[9] that is well known in the area of performance analysis.

Basically, assuming each parameter has a range delimited by extreme values, the $2^k$ full factorial design procedure tries *all possible combinations* of the parameter *extremes*. It then fits the results into a non-linear regression model to analyze the importance of different parameters. However, it is *not a iterative method*, i.e., the meaningful results can only be obtained after finishing all the experiment, which is opposite to our search strategy. To achieve the iterative capability and the goal of monotonically improving results, we designed a experiment scheme to carefully *order* the experiments in a *series of subsets of experiments*. The first subset is generated by considering a $2^{k-p}$ fractional factorial design[9] on the parameter space. Here, $p$ is the minimum integer satisfying $2^{k-p} \geq k$, which is required by the regression analysis described in [9]. $2^{k-p}$ fractional factorial design is a technique which just chooses part of the experiments to execute from $2^k$ full factorial design. With these carefully selecting the experiments, the analysis of the parameter importance can still be accomplished at the expense of some accuracy[9]. After finishing this subset of experiments and analyzing the simulation results, we then step into the next larger subset which is obtained by using $2^{k-p+1}$fractional factorial design, and so on until all $2^k$ experiments are finished. During this process, if the search is interrupted, the analysis result based on the last subset of experiments is returned as the "best-so-far" result.

### B. *Hybrid Search Algorithm*

Once the high level pruning is complete, the next task is to search the remaining parameter space in detail with *general* state space search techniques. Basically, the state space search algorithms include two important components, *exploration* and *exploitation*, and a balance strategy between these two components. *Exploration* encourages the search process to examine unknown regions. In comparison, *exploitation* attempts to converge to a maximum or minimum in the vicinity of a chosen region.

The "No Free Lunch" theorem[13] cautions us that there is no general good search algorithm performing equally efficiently in every problem. And the most efficient search algorithms would aggressively incorporate the maximum domain-specific features into the search strategy at all times. This necessitates that the search algorithm be extremely flexible and adaptive to different kinds of specialized information.

Our hybrid search algorithm is based on the hillclimbing technique[10] with TABU technique[12] included to avoid revisiting the previous region and speed up the exploration process. Different from other algorithms, the hybrid method maintains a dynamic balance between *exploration* and *exploitation* by accepting the bad move with a probability of $\exp(-\textit{current gradient feature}/\textit{average gradient feature})$. The idea here is that in the promising areas whose gradient is much steeper than the average gradient, the algorithm performs *exploitation*, i.e., a hill climbing procedure which quickly converges to the local optimum. Else, depending upon the probability generated by the balance strategy, it performs *exploration*, i.e., random walk. Observe that this balance strategy is different from that of Simulated Annealing(SA)[11]. In SA, the acceptance probability of bad moves decreases gradually according to a predefined cooling scheme, whereas in our algorithm, the acceptance probability is adapted to the gradient features of the current area.

The hybrid algorithm also automatically adjusts its *step size* (used for exploration) to suit the current gradient features. When the current gradient is large relative to the average, the step size is reduced to explore this promising area carefully. Otherwise, the step size is increased to quickly get through this area. With the increase of the step size, the rugged microscopic features of search space are smoothed out and the major features are exhibited. The search can take advantage of these macroscopic features to improve the efficiency. A detailed analysis and comparision of candidate search techniques will be reported in future papers.

## V. Speeding Up On-line Simulation

The execution of the simulations is the most time-consuming task in the on-line simulation system. We have designed a few methods to speed up this process.

### A. *Farmer-Worker Infrastructure*

The first method is to parallelize the execution of the simulations. This is achieved by encapsulating each simulation in a thread and distributing the threads across machines. We have developed a farmer-worker infrastructure for this purpose. This infrastructure can be used not only in our on-line simulation, but also as a general performance evaluation tool using multiple simulations or distributed computations. The farmer-worker infrastructure is shown in Fig. 3.

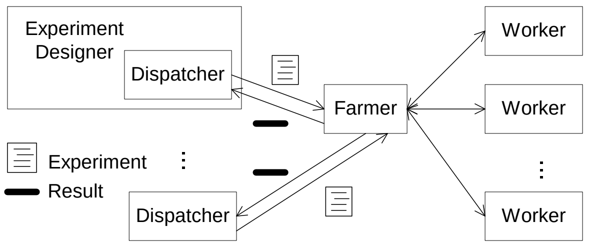


Fig. 3. Farmer-Worker infrastructure

As shown in the figure, the dispatcher is the interface between the distributed simulation executer and the experiment designer.

The latter is the one that decides which simulation experiments to perform (as described in earlier sections). All the experiments have to go through this interface to be distributed among the *workers*. The *farmer* is the center of this infrastructure, which controls and synchronizes the operations of dispatchers and workers. The workers are the actual experiment executers leading to a dramatic speedup. Since each parameter state space point corresponds to a simulation experiment which can be executed by the worker, our online simulation model naturally fits as an application of this infrastructure. All the communication between the components in this scheme is through TCP sockets. Therefore, the dispatchers, farmer and workers can be located anywhere in the network (or a backend computational facility), allowing us to maximize the utilization of computing resources.

## B. Topology Decomposition

The farmer-worker method is effective only when the experiment designer sends out a batch of experiments every time and then waits for the results. Its speedup level is dependent on the size of each batch, or minimally the size of each simulation experiment. The topology decomposition method is used to expedite the execution of *a single simulation experiment* in this model. For example, we have found that the running-time of a simulation is not linearly proportional to the simulated network size. Fig. 4 shows the simulation results on a network with a simple star topology. Using the least squared method to fit the

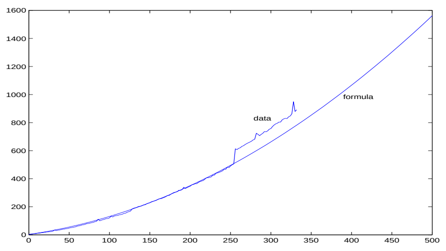


Fig. 4. Execution time vs. simulation size

experiment results of execution time and network size, we can get the following approximate formula:

$$T(n) = 3.49 + 0.8174 \times n + 0.0046 \times n^2 \qquad (1)$$

Here $T$ is the execution time of the simulation, and $n$ is the number of nodes in the simulation. The approximate formula estimation is also shown in Fig. 4, which demonstrates a good match with the experiment data.

From the above, we can see that the execution time of a network simulation may hold a quadratic relationship with the network size. Therefore, it is possible to speed up the network simulation more than linearly by splitting a large simulation into smaller pieces and paralleling the execution of these pieces.

Traditional decomposition only splits up the network topology, but the simulation is still executed as a whole. Therefore, the decomposed parts have to exchange a lot of information to keep them synchronized with each other. Our approach is to first execute these split simulations independently; then repeat each of these split simulations with the output of the other parts as the input; and then repeat again and again until there is no significance difference between the results of two consecutive iterations of split simulations. This approach greatly simplifies the synchronization between parallel parts, and can significantly speed up the simulation of large networks. Some simulations have been done and the results show a very fast convergence. While this approach fits the simulation of simple non-TCP traffic very well, we are investigating issues with respect to TCP traffic.

## VI. Performance analysis of the On-line Simulation Architecture

To test the validity of our on-line simulation scheme, we first implemented a simple on-line simulator under *ns* and used it to control some network algorithm. This is indeed a simulation within a simulation! We then observed how this affected the performance of the network which was also being simulated. In the experiment, we adopted Random Early Detection (RED) queueing management algorithm [2] as the underlying network algorithm to be adjusted because of its sensitivity to parameter settings. Another reason is that it has also been indicated that deciding the parameter setting of RED for different network conditions is not a trivial task[2], and the automation of its parameter adjustment will be very meaningful. However, it should be noted that the applicability of the on-line simulation scheme is not limited to RED or other queueing management algorithms. Any other network protocol or algorithm, which is sensitive to the parameter setting, can be tuned with our on-line simulation system.

A simple network topology with one bottleneck link has been used for the purpose of proof-of-concept, which is shown as Fig. 5. The router on the source side is configured with a RED

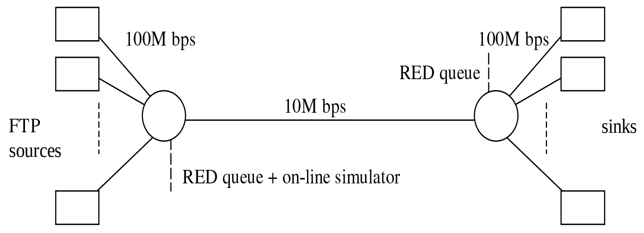


Fig. 5. Test network topology

queue, and an on-line simulator is used here to control the RED queue. The experiment varies the traffic conditions on the bottleneck link: $4$ FTP connections in the first period of $4$ seconds, then $8$ FTP connections in the second period, then $4$ FTP connections again in the third, and finally $16$ FTP connections in the last.

Our objective is to achieve the best throughput. The simulation results are shown in Fig. 6 (without on-line simulation) and Fig. 7 (with on-line simulation). We can see from these figures that when the number of FTP connections changes from $4$ to $8$ and from $4$ to $16$, the on-line simulation is able to tune up the parameters of RED queue, such as maximum threshold and minimum threshold, to reduce oscillation in the queue size and maintain a more stable queueing status. As a result, the utilization is better than the experiment without on-line control, in

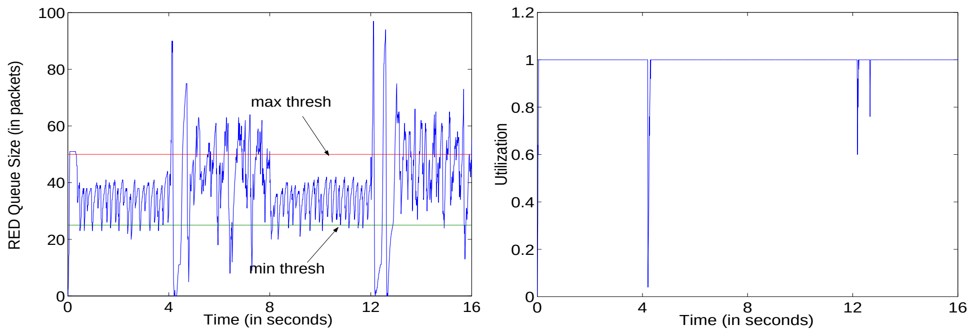


Fig. 6. *ns* simulation: RED average queue size and link utilization (without on-line simulation control)

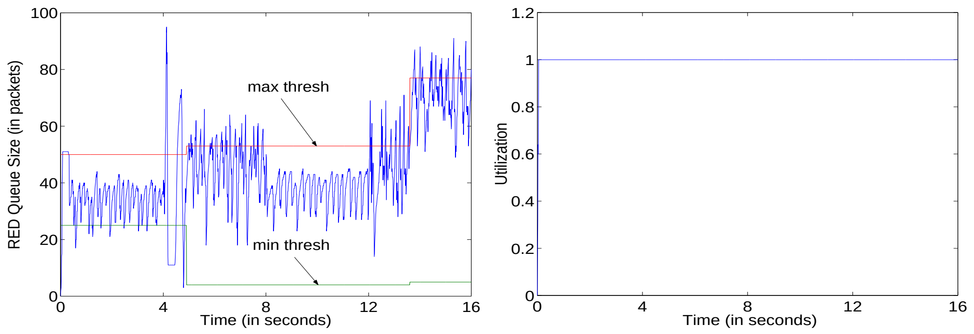


Fig. 7. *ns* simulation: RED average queue size and link utilization (with on-line simulation control)

which we can find that when the traffic conditions change, the utilization drops dramatically for a while when traffic conditions change.

## VII. Real-world Implementation and Test of On-line Simulation

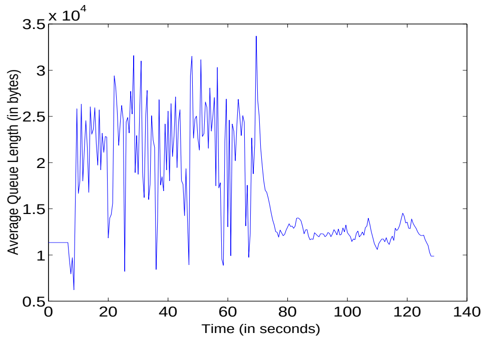
To verify the simulation results of our scheme under real networks, we have built a Linux testbed with 20+ machines and 3 subnets. The network topology is basically like the one shown in Fig. 5. We adopted Linux as our test platform for its open source policy and great popularity. Furthermore, a variety of traffic management algorithms have already been implemented in Linux kernel, such as RED, CBQ. Although these traffic control elements are still in the experimental stage, we found them quite stable in our experiments. We ported our on-line simulator into Linux platform and created an interface so that the simulator can interact with the traffic management algorithm in Linux kernel and adjust the algorithm parameters automatically. The configuration and assumption are almost the same as those in the previous section. We have used *netperf* [15] traffic generator to generate massive TCP traffic from one side of the bottleneck link to the other side and observe the performance of RED on the bottleneck with the on-line simulator applied for dynamic control. For testing purposes, we assume the simulator already has all the information about the network conditions. Our carefully designed traffic generators are applied in the simulation to reproduce the realistic traffic under *ns* and the previously described speedup measures are also used to boost up the speed of the simulations.

The simulation results are shown in Fig. 8. We can see that there is a large oscillation in the average queue length of RED. This is because that we have chosen its parameter at random in the beginning and this parameter setting may not fit the current network conditions at all. Then in the middle of the simulation, we started our on-line simulator to search for the settings with low average queue length. Very quickly, the simulator found a much better setting and applied back to RED queue. This results in a dramatic reduction of the average queue length, as shown in Fig. 8 after the test starts for about 70 seconds, and at the same time, the oscillation looks much smaller than before and the link utilization is not affected.

## VIII. On-line Simulation for Improving Routing Stability

As mentioned before, the application of on-line simulation scheme is not limited to the network management algorithm. For example, a similar approach can also be applied to routing, and an improvement in the end-to-end performance can be achieved by tuning the parameters of routing algorithm [14].

Adaptive routing is known to improve the network performance by increasing throughput and lowering the end-to-end packet delay. But it has been largely abandoned in the Internet due to the problems associated with routing oscillations. We have identified various parameters of an adaptive routing scheme which affect its performance and stability. The simulation based study indicates that these parameters may be tuned using on-line simulation to achieve a stable routing with improved performance. Moreover, the adaptive routing scheme may be deployed in the existing routers using OSPF. The parameters in this algorithm which we consider for tuning are *uFactor*



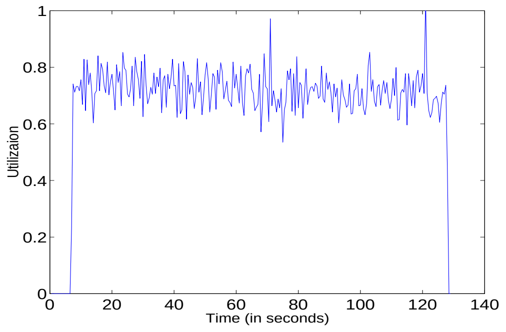


Fig. 8. Real network test: RED average queue length and link utilization

and *bFactor*, which represent the weights associated with the link utilization and buffer utilization when the link cost is given by

$$linkCost = defaultCost \times (1 + uFactor \times Util\_)$$

where, $Util\_$ may be the exponentially averaged link or buffer utilization. These parameters are representative of the adaptiveness of the routing to the congestion. Another parameter that we considered is *Threshold* which reflects the minimum change in the cost to trigger a Link State Advertisement with updated metric value. Also, *Interval* between the routing updates is another parameter that we consider as it represents the tradeoff between the responsiveness of the routing algorithm and the maximum computational and bandwidth overheads associated with frequent route changes. We have demonstrated tunability of the parameters $uFactor$ and $bFactor$ to achieve significantly better end-to-end throughput and delay performance. The tuning of parameter $interval$ was found to increase the throughput at the same time when it minimizes the number of route changes. However, the parameter $threshold$ does not affect the network throughput or the end-to-end delay, but may be tuned to minimize the route changes. This will achieve a TCP-friendly routing as frequent route changes may lead to out-of-order packets, timeouts and result in a poor performance due to the flow control mechanism of TCP. However, testing on experimental testbed network with on-line simulation to tune the routing parameters is a goal for future work.

## IX. Conclusion

In this paper, we describe a collaborative on-line simulation architecture to perform the dynamic, scalable and effective network control. To realize this scheme, the problems faced in the areas of network modeling, network simulation and parameter search are addressed and some solutions are presented. We design and implement various methods to maintain the appropriate mix of traffic composition and generate realistic traffic. We use farmer-worker scheme to distribute experiments and parallel their execution on multiple computing resources. In addition, we also use topology decomposition method to speed up the execution of a single simulation. In parameter search, we use a best-effort, increasingly improving strategy to search for better parameter setting within the limited time frame. We also propose a new hybrid search algorithm, which aggressively takes advantage of the known information of the parameter space to perform highly efficient search.

Various software components have been developed in *ns* and Unix/Linux. Preliminary experiments and simulation were executed on traffic management algorithms for the demonstration of our collaborative on-line simulation concept. These tests produced very promising and encouraging results. All the softwares and results are available on line[16].

The current work is limited to proof-of-concept stage. There is still a lot of work to be done to realize the goal of scalable and effective on-line control in operational networks. Further work needs to address the problem of how to collect data from the network and build a good model from the data. The scalability and cooperativity of the on-line simulation continue to be issues worthy of investigation.

## References

[1] (1997) NS(*network simulator*). http://www-mash.cs.berkeley.edu/ns.

[2] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transaction on Networking*, vol. 1, pp. 397-413, August 1993.

[3] IETF, Differentiated Services(diffserv) working group. http://www.ietf.org/html.charters/diffserv-charter.html.

[4] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder, "Oc3mon: Flexible, affordable, high performance statistics collection", *Proceedings of INET'97*, June 1997.

[5] V. Paxson, and S. Floyd, "Wide area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking'*, 3 (3), 226–244, June 1995.

[6] B. Ryu, "Fractal network traffic: From understanding to implications," *Ph. D. thesis*, Columbia University, New York City, 1996.

[7] A. Andersen, and B. Nielsen, "A markovian approach for modeling packet traffic with long-range dependence," *IEEE Journal on Selected Areas in Communications*, 16 (5), 719–732, June 1998.

[8] M. Yuksel, B. Sikdar, K. S. Vastola and B. Szymanski, "Workload generation for ns Simulations of Wide Area Networks and the Internet," *Proc. of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pp 93-98, San Diego, CA, USA, 2000.

[9] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, 1991.

[10] H.-P. Schwefel, *Evolution and Optimum Seeking*, New York: Wiley, 1995.

[11] S. Kirkpatrick, D.C. Gelatt and M.P. Vechhi, "Optimization by simulated annealing," *Science*, vol. 220, pp.671-680, 1983.

[12] F.Glover, "Tabu Search I," *ORSA J. Comput.*, vol. 1, pp. 190-206, 1989.

[13] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization", *IEEE Trans. On. Evolutionary Comput.*, vol. 1, pp. 67-82, April 1997.

[14] H. T. Kaur and K. Vastola, " The Tunability of Network Routing using Online Simulation," *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems*, July 16-20, 2000 Vancouver B.C. Canada.

[15] *netperf* traffic generator, http://www.netperf.org.

[16] "Network Management and Control Using Collaborative On-line Simulation" website, http://networks.ecse.rpi.edu/˜olsim.