To appear in 1999 European Simulation Symposium

Fluid-based Simulation of Communication Networks using SSF*

David Nicol Michael Goldsby Michael Johnson
Dartmouth College Sandia National Laboratories

1 Introduction

Fluid-based models of communication network traffic consider traffic behavior in terms of packet *rates*, rather than packet *instances*. While fluid modeling is commonly used in mathematical approaches, these often require model simplification. Simulation of fluid models may be used in order to retain necessary model detail; it is beginning to receive some attention in the literature [1, 3, 4, 6, 5]. We provide here a cost-benefit study of such techniques, *within a common framework*.

We describe a simulation tool that targets simulation of communication systems, and its use to simulate fluid-based models. Within the same tool we are able to develop and compare packet-based, and fluid-based models, assess any performance benefits and assess inaccuracies introduced. The specific study made here is of networks where traffic is generated as a Modulated Markov Process and is shaped using a Leaky Bucket flow-control mechanism. Switches model FCFS service among all competing flows at output ports. The principle contributions of this paper are to demonstrate fluid simulation in the context of an existing packet-oriented simulator, and to study the performance/accuracy trade-offs between packet-based and fluid-based models.

2 Fluid Simulation in SSF

The Scalable Simulation Framework (SSF)[2] is a Java and C++ based API for the description of network models (see `http://www.ssfnet.org`). Three implementations of this API exist, one in Java and two in C++. The objectives of SSF are to provide a public domain standard of discrete-event simulation of large complex systems. SSF models are compact, flexible, portable, and transparently parallelizable. At the base level the SSF API defines only five base classes: `Entity`, `Process`, `Event`, `InChannel`, and `OutChannel`. An `Entity` is a container of state, of `Processes`, and of communication endpoints. SSF is process-oriented; a `Process` is a bit of C++ code that executes when triggered by a time-out, or the arrival of an `Event` on an `InChannel` it has specified. The process may modify `Entity` state variables, and create `Event`s which are written to `OutChannel`s.

A simple SSF topology might consist of traffic sources, each of which is connected to traffic shapers. Traffic leaving a shaper enters a switch, and is routed to a Host. Hosts in turn might be interconnected through Routers. In an SSF model, each of the fore-mentioned objects would be modeled as an `Entity`, with their inter-connections modeled using `InChannel`s and `OutChannel`s. Entity state variables record statistical information, and maintain a description of traffic as it flows through the system. A `Process` at the traffic source will use time-outs to govern when it wakes up to write more traffic to the shaper; the shaper will have one `Process` to respond to new traffic from a source, and another to pass traffic through to a switch. The two processes may coordinate with each other through state variables in the `Entity` they share. Likewise, a switch `Entity` will have a `Process` to respond to new inputs, and another to route out-going traffic.

The description above applies whether the traffic is a stream of packets, or a fluid flow. This isomorphism is important, as the simulator infrastructure is the same for both cases, indeed, much of the code is identical. For instance, the traffic generator is a Modulated Markov Process (MMP). This means that the traffic generator moves around in "states" of a continuous time Markov chain (CTMC); in state i of this CTMC traffic is considered to be generated at (constant) rate $\lambda(i)$. Our two implementations use the same MMP code, and same "Fluid Leaky Bucket" traffic shaping code. In both cases the output of the Fluid Leaky Bucket is a sequence of time-stamped "Fluid Events"; each such event describes the new output rate of packets. In the fluid-based model these events are fed directly into a fluid-based switch model, whereas in our packet-based model they are fed into a "Fluid-to-Packet" converter which emits packets at the specified rate, packets which are fed to a packet-based switch model. In both models the output of a traffic source defines an identifiable "flow" through the network. At startup each such flow is mapped through the switching

*This research is supported in part by DARPA Contract N66001-96-C-8530, NSF Grant ANI-9808964, and the United States Department of Energy under Contract DE-AC04-94AL85000.

network, where at each switch the input port and output port are specified. Routing is therefore static; given a flow id, a switch knows how to route that flow (the packet simulator knows where to route a new packet and the fluid simulator knows which output queue is affected by a change in the incoming flow rate for the identified flow).

The key difference between our fluid-based and packet-based models is the implementation of the switch. FCFS ordering of packets at output ports is easily implemented in the packet-based model. The fluid model is more delicate. If flow i changes its input rate to a FCFS queue (which receives input from multiple flows) at time t and there are the equivalent of p packets already in queue or service at time t, then the effects of that change on allocation of service to flow i "packets" should not be observed until all p packets in the queue at t have been served. In effect, the rate change that occurs at time t is scheduled to change the mix of fluid entering service, at time $t + p/\mu$, where μ is the queue service rate. Therefore, at any instant s each flow i receiving service has an associated input flow rate $\lambda_i(s)$; service rate μ at time s is allocated to the different flows in proportion to their input flow rates at s.

The key difference in computational effort between package-based and fluid-based models is that the number of "events" executed at a switch in the former model is proportional to the number of all packets that enter the switch, whereas in the fluid model it is proportional to the total number of MMP state changes. Thus we can expect computational savings as the number of packets emitted during a MMP burst period is large. However, fluid models are approximations to packet models, and some care must be taken when measuring statistics. We examine that issue next.

3 Statistical Measures

We consider traffic through a switch to be comprised of a number of individual flows, and presume that for each flow one desires to estimate the percentage of lost packets, and the mean packet delay through the switch. We understand a flow to be mapped from one input port to a queue serving an output port. Losses occur when a flow's packets arrive to find the queue full. The delay a packet receives is the sum of service times of all packets ahead of it in queue, plus its own service time. The service delay is the reciprocal of the port's output bandwidth.

The queue is in one of 4 states :

1. empty, with no active flows;

2. non-empty, not full, with flows entering at an aggregate rate different from the output bandwidth;

3. full, with flows entering at an aggregate rate greater than the output bandwidth;

4. flows entering at a positive aggregate rate that is equal to the output bandwidth.

In the fluid simulation, at any point statistics are gathered for a switch model, it will be for a period of simulation time $[s, t)$ over which the arrival rate and the departure rates are constant. This means that any time statistics are gathered, the queue has been in exactly one of the states above, over the entire interval of interest. Now in our fluid model losses are suffered only when the queue is in state (3); the estimated number of packets lost is the product of the length of time the queue remains in state (3) times the difference between the aggregate input rate and the output bandwidth. Each flow is assumed to deliver a volume of packets (excluding losses) in proportion to the rate of the input it contributes over this interval. This estimate is not exact; in a packet-based model, even if the average arrival rate is equal to the average departure rate, variation in arrival times can deliver a packet to a full queue.

Over period $[s, t)$ we compute the sum of waiting delays for all packets that are modeled by the fluid to have arrived in $[s, t)$. If the constant arrival rate is λ over that period and the constant service rate (i.e. bandwidth) is μ, then we define N as the integer part of $(t - s)\lambda$, define Q_s to be the fluid level at time s, estimate the number of packets ahead of an arrival at $u \in [s, t)$ to be $Q_s + (u - s)(\lambda - \mu)$ and so estimate the response time of an arrival at time u as $(Q_s + (u - s)(\lambda - \mu) + 1)/\mu$. Our estimate of the the sum of all such waiting times for all arrivals in $[s, t)$ is

$$\sum_{i=1}^{N} (Q_s + (i/\lambda)(\lambda - \mu) + 1)/\mu.$$

This approximation reflects simplifying assumptions that inter-arrival times are deterministic and equal to the reciprocal of the average arrival rate, and that a full inter-arrival period goes by before the the first arrival.

At the point we measure the volume of delivered packets and average packet delay over an interval $[s, t)$, the arrival and service rates we use are aggregate rates, taken over all individual flows mapped to the switch. We accumulate delivery and delay statistics on a per-flow basis; after computing the aggregate volume of delivered packets and delay, we partition these statistical values to the individual flows in proportion to the flows' arrival rates over $[s, t)$. For example, if the arrival rate of one flow is fully half of the aggregate arrival rate of all flows, then that flow receives credit for one half of the measured packets, and one half of the aggregate sum of packet delays.

4 Experiments

Next we turn to some experiments that seek to estimate the error introduced by fluid approximations, seek out sensitivities in that error, and concurrently look at the potential speedups that might be achieved using the approach.

Intuition suggests to us that at least five factors will be important in our study. Buffer size in a switch is one, for that has direct impact on packet losses. Traffic load is another factor, for that largely determines the fraction of time a switch spends in each of the four states outlined above. A third factor is the size of the network being studied, and a fourth is the number of flows concurrently mapped to a switch. The last factor is the average number of packets generated between transitions of the MMP.

We have compared the performance of fluid-based and packet-based SSF models that share a great deal of common code and common infrastructure, *except* for the generation and handling of events. We first report on studies at one switch, and follow-up looking at networks of up to hundreds of switches and thousands of flows. In these studies we look at relative error in the number of delivered packets, and in the mean packet response time, measured as $(f_m - p_m)/p_m$, where f_m and p_m are the fluid and packet measures of interest, respectively. Figure 1 illustrates the relative error in response time and in packet delivery for a single-switch, where there is a single traffic source the arrival and service rates are set to create approximately 90% packet delivery. The MMP is an on-off process. The independent variable is the switch buffer size, measurements are taken at $4, 16, 64$ and 128 packets. Curves for four different "time-scales" are presented. In time scale X (for $X = 1, 10, 100, 1000$) the average number of packets generated by one "on" burst is close to 10X. We see that response time errors are largely independent of time-scale, are quite small in the worst case (less than 10%) and quite good for large buffers. The relative error in fraction of packets delivered is insignificant. It is interesting to note that the magnitude of these errors does not seem to depend on the traffic intensity.

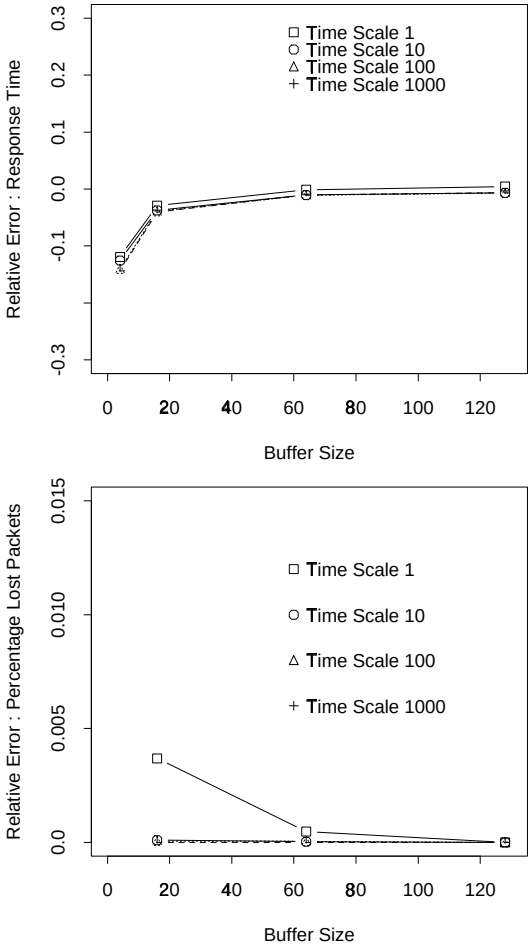

Figure 1: Relative Errors for Single Source Queue

The performance benefit of fluid-simulation should clearly depend on the time scale, for changing the time-scale does not much change the number of events simulated in the fluid model, but very much changes the number of events simulated in the packet model. For the runs described in Figure 1, with a buffer length of 64, the accelerations of fluid based simulation versus packet based simulation are 4.6 for a time-scale of 1, 33 for a time-scale of 10, 112 for a time-scale of 100, and 444 for a time-scale of 1000. Clearly *very* significant performance benefits are possible with relatively small error.

Next we turn to consider how increasing the number of flows into the switch affects the per-flow errors. Here we fix the size of the switch queue at 64, and adjust the relationship between traffic generation rates and service rates to create approximately 95% packet delivery rates. Figure 2 shows the relative errors found as a function of varying the number of flows among 1,2,4,8,16. Relative error in response time is taken as the per-flow relative error. It is again relatively small, in these experiments less than 4% in magnitude. The data does suggest some sensitivity in the number of flows however, with increasing error with increasing numbers of flows. Average relative error in packet loss is small, grows in the number of flows, and is sensitive to time-scale.

To illustrate the performance benefits in this case we consider speedup for the 16 source case, as a function of time-scale. Here we observe a speedup of 8 for time-scale 1, 34 for time-scale 10, 219 for time-scale 100, and 2116 for time-scale 1000. Clearly increasing the number

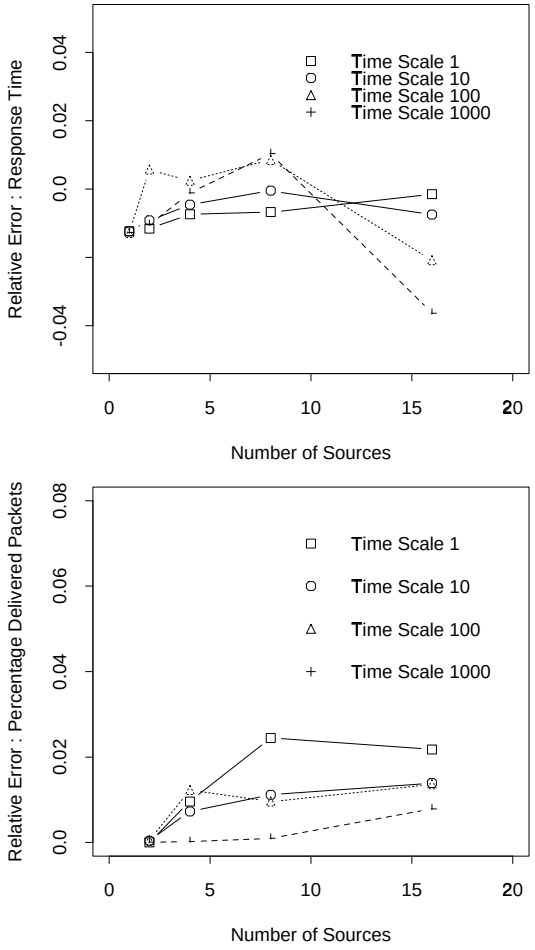

Figure 2: Relative Errors for Multiple Source Queue

of sources had no detrimental effect on the performance potential of fluid simulation.

To conclude this preliminary study we examine the behavior of networks. The switches of each network are inter-connected as a binary hypercube of dimension d, and a number s of flow-shaped MMP sources are attached to each switch, for a total of $s2^d$ flows. Each switch has one output port for each hypercube neighbor (with capacity 64 packets), and one sink port. The flows are routed so that the j^{th} flow originating at switch r is mapped through the network to be received and terminated at switch $(r * s + j) \bmod 2^d$. Standard hypercube routing is used (a flow arriving at switch r, destined for switch f, is routed to the hypercube neighbor of r with least order bit that differs from r's and is identical to f's.) This selection of flow mappings exercises all parts of the network, and creates a variety of congestion conditions flows may experience.

Our experiments cover hypercubes of sizes $4, 8, 16, 32, 64, 128$ and 256 switches. We vary the number of sources assigned to a switch between 2 and 16, and we vary the switch output port service rates to affect flow losses. In the "low loss" case it is set so that on ports where there is loss, it is limited to about 2%; in the "high loss" case overloaded links lose about 40% of the flow. All these experiments use a MMP time-scale of 10 (so that approximately 10 packets are generated per flow burst). As before we compare the average per-flow response time and per-flow loss statistics from fluid-based and packet-based implementations, and compute the relative error. Each flow's response time statistic is computed by adding the relevant components from each switch on its path through the network; its fraction of delivered packets is computed as the product of this fraction at each switch on its path.

Figure 3 shows the relative errors found as a function of network size. The main and important point is that the errors are small. The promise of accuracy observed in the single switch case is delivered for networks, at least for the cases studied here. The errors when there are 2 sources per switch and low loss are miniscule, less than 2% at all network sizes. The speedup of the fluid implementation increases from 34 at the smallest network size, to 46 at the largest. When the low loss case runs with 16 sources per switch, the response time errors increase (but the loss errors do not). The response time errors increase with network size, but even so the errors are less than 13%. The fluid implementation's speedup in this case hovers around 20 for all network sizes, with no discernible trend.

Moving to the high loss cases, with 2 sources per switch we see very low response time errors ($< 1\%$), low delivered traffic errors ($< 5\%$), and significant speedups (increasing from 38 with the smallest network to 66 with the largest). Good error behavior is also observed in the case of 16 sources per switch; response time error remains less than 5%, and the delivered traffic error is less than 13%. The most distinguishing facet of this case is speedup behavior. A speedup of 7.5 is obtained by the 4 switch network, but with increasing network size the speedup drops so much that on the 256 switch network the fluid implementation actually runs twice as slowly! The explanation for the performance decline lies in the accumulating effects of flow modulation. A heavily loaded output queue simultaneously carries a multiplicity of flow descriptors, of flows sharing the output port. Whenever one of the input flow descriptors changes on *any* of these, the output flow descriptors change on *all* of the flows. These changes in turn propagate as changes to the inputs of a multiplicity of queues at the switch connected to the other end of this port, which further propagate. This is the very phe-

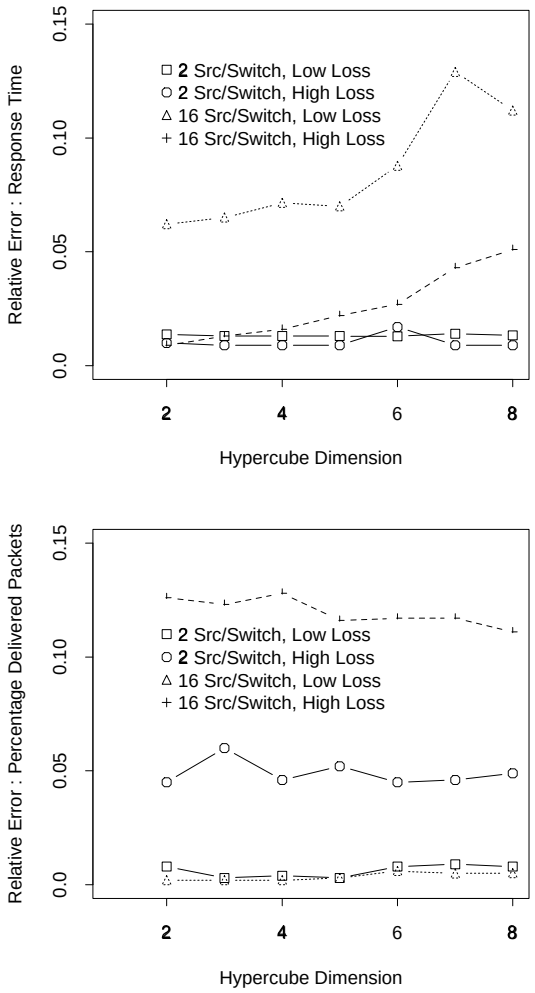

Figure 3: Relative Errors for Hypercube Networks

nomenon analyzed and modeled by [5]. In the 256 switch case the fluid simulation executes nearly as many events as does the packet simulation whereas in the 4 switch case it executes only 1/5 as many; each fluid event is significantly more computationally expensive, hence the slowdown.

This performance degradation is a high priority concern, and we are working on ways of "smoothing" fluid behavior to reduce the propagation effects. The results of those efforts are beyond the scope of this paper.

5 Conclusion

Fluid-based simulation of networks offers great potential for accelerated performance, but questions must be asked about the accuracy of such simulations, and whether the comparisons made to packet-based networks are "fair". We have addressed the second issue by developing fluid modules that "plug-in" into a high-performance simulator of communication networks, making a performance comparison absolutely fair by using as much common framework code as possible. We are beginning to address the question of accuracy, which is essentially an issue of how we gather statistics from a fluid simulation. We describe here methods of gathering per-flow estimates of packet loss and of packet response time, and examine the accuracy of these methods as a function of time-scale, number of sources, and switch buffer length. We find acceptably small errors, and the potential of tremendously large speedups, depending on the nature of the traffic. Performance problems remain with ripple effects, an area we are currently addressing.

References

[1] J.S. Ahn and P.B. Danzig. Packet network simulation: Speedup and accuracy versus timing granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[2] J. Cowie, D. Nicol, and A. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, 1(1):30–38, January 1999.

[3] G. Kesdis, A. Singh, D. Cheung, and W.W. Kwok. Feasibility of fluid-driven simulation for ATM networks. In *Proceedings of IEEE GLOBECOMM*, volume 3, pages 2013–2017, November 1996.

[4] K. Kumaran and D. Mitra. Performance and fluid simulations of a novel shared buffer management system. In *Proceedings of IEEE INFOCOM 98*, March 1998.

[5] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid simulation of large scale networks : Issues and tradeoffs. In *Proceedings of 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, Volume IV, pages 2136-2142, June 1999.

[6] A. Yan and W. Gong. Fluid simulations for high speed networks. *IEEE Trans. on Information Theory*, June 1999.