

Parallel Simulation of Hybrid Network Traffic Models

Jason Liu

Department of Mathematical and Computer Sciences
Colorado School of Mines
Golden, Colorado 80401
Emails: xliu@mines.edu

Abstract

We examine a parallel processing method for simulations of large-scale networks with a hybrid traffic representation combining both a time-stepped fluid model and a discrete-event packet-oriented model. This method benefits from the observation that the time it takes to propagate fluid characteristics along the path taken by the traffic flows has a lower bound equal to the minimum link delay as manifested by the governing ordinary differential equations (ODEs). A better lookahead can thus be used to allow parallel simulation of the hybrid model to run without more synchronization overhead than the corresponding discrete-event packet-oriented model. We derive an analytical model comparing the fluid model and the packet-oriented model both for sequential and parallel simulations. We demonstrate the benefit of the parallel hybrid model through a series of simulation experiments of a large-scale network consisting of over 170,000 hosts and 1.6 million traffic flows on a small parallel cluster.

1. Introduction

The prohibitive computational cost is not the only but arguably the most important limiting factor of large-scale network simulations. In simulation, an event is used to represent a simulated network packet either entering or departing from a network router. Considering a simulation scenario with hundreds of thousands of network entities populated with commensurate amount of traffic flows to characterize the behavior of today's Internet, the number of simulation events awaiting to be processed at each *simulated* second can easily exceed what a traditional computer can process during a *real* second, therefore causing significant slowdowns—sometimes in the excess of two to three orders of magnitude. To mitigate the slowdown and allow enough computing resources (especially memory) to support such large network models, parallel simulation [5] is an obvious

option. Preliminary success in running massive network simulations on large parallel computers is particularly encouraging [3]. In the same spirit, parallel simulators have been developed as candidates for general network modeling tasks (for example, [1, 16, 19]).

It is desirable for large-scale network models to achieve real-time performance. Real-time network simulation allows large-scale network models to run in-sync with the wall-clock time. Thus, real network protocols, real network services, and real distributed applications can run in concert with the simulator enabled with real-time network monitoring and control of the virtual network conducting both the simulated and real network traffic [11]. Although parallel network simulation has demonstrated good scalability and impressive speedup overall, its performance largely depends on the size of the network model and the performance of the parallel computers. The latter is particularly problematic since the solution invariably depends on the availability of supercomputing one can wield against common network modeling problems. Consequently, alternative models aiming to reduce the computing demand of large-scale network simulations have been proposed to replace or, more effectively, combine with the discrete-event packet-oriented simulation. Examples include using the epidemic model to describe the fast propagation of malicious worms on the Internet (e.g., [9]) and using fluid representations to characterize the global network traffic behavior (such as [14] and [8]).

Our starting point is the fluid TCP model developed by Liu et al. [13], which uses a set of ordinary differential equations (ODEs) to model the mean traffic behavior of long-term TCP sessions over a network of RED routers. In particular, the differential equations are used to describe the changes of the the TCP congestion window size, the propagation of packet delays and losses on the network, and subsequently the effect on the network queue length. The set of ordinary differential equations can be solved numerically using the time-stepped Runge-Kutta method. It is shown that the fluid model can achieve a speedup of more than

three orders of magnitude over the corresponding discrete-event packet-oriented simulation.

Solutions have also been proposed to combine the fluid model with the packet-oriented simulation. Gu et al. [6] implemented an integration scheme in the *ns-2* simulator that physically divides the simulated network into two subnetworks, one solved by the fluid model and the other by the packet-oriented simulation. At the interface between the two models, packet flows are converted into fluid flows (by averaging over time) before entering the subnetwork controlled by the fluid model so that they can compete with other fluid flows for network resources. Packet delays and losses calculated by the fluid model are then applied to the packet flows re-emerging from the fluid network. A similar approach yet allowing staggering of multiple fluid models for better response time is proposed by Zhou et al. [20].

Problems arise when packet-level characteristics are desired at each router. For example, one cannot apply `traceroute` or send SNMP queries to the routers in the fluid network, which are merely described by differential equations. This restriction would compromise the accessibility of the real packets generated by the real applications interacting with the real-time network simulator, limiting packet-level interactions to only the part of virtual network regulated by the discrete-event simulator. In an earlier effort, we proposed an integration scheme that mixes fluid and packet flows at each simulated router allowing packet-oriented network traffic to reach any part of the virtual network [10]. In addition, a better precision is expected due to the fine-grained interaction between the fluid and packet flows. More importantly, this scheme allows us to dynamically change the traffic composition by switching between packet flows and fluid flows, therefore enabling the trade-off between accuracy and performance. For example, when the simulation event rate is seen to exceed the available computing budget, the simulator can “scale back” by converting the packet flows into the fluid flows in order to reduce the computing demand at the cost of less accuracy rendered by the fluid model. Similarly, fluid flows can be converted into packet flows for a more accurate network traffic representation in situations where idle cycles are detected during the real-time network simulation.

This paper addresses the issues of parallelizing the above hybrid model. We assume spatial partitioning is applied for parallel network simulations; that is, the virtual network is divided into subnetworks to be simulated on parallel processors. In this case, the fluid ODE solver must also be partitioned accordingly. It is well-known that the performance of conservative simulation synchronization protocols depends on the size of the *lookahead* [4]. Lookahead is defined as the lower bound on the timestamp of events a logical process may execute ahead of other logical processes without introducing causality errors. The lookahead is critical to the

performance of parallel simulation as it prescribes the fundamental asynchrony among the logical processes. In network simulation, it is a common practice that the minimum link delay between the partitioned subnetworks is used as the lookahead. Albeit conservative in most cases, good scalability can be achieved since, for large-scale network models, the synchronization overhead is dominated by the event processing cost [1].

One must be careful with the hybrid traffic model to allow good parallel performance. The differential equations in the fluid model are to be solved using the fix-stepped Runge-Kutta method. In particular, the state of the network queues traversed by the fluid flows is evaluated and updated at each Runge-Kutta time step according to the state of the neighboring network queues at the previous time step. If we chose to use the time step size of the Runge-Kutta method as the lookahead, which is typically more than an order of magnitude smaller than the minimum link delay in order to maintain numerical accuracy of the Runge-Kutta method, one would expect a much higher cost for synchronizing the logical processes. This could very well jeopardize the benefit of parallel simulation.

Fortunately, by closely inspecting the differential equations of the fluid model, one could make an observation that the propagation of fluid flows actually implies the *same* lookahead as in a pure packet-oriented simulation. The characteristics of a fluid flow—in particular, the flow rate, the cumulative packet delay, and the cumulative packet loss probability—propagate in the same fashion along the flow path as the network packets: changes to the state of a network queue will not affect the state of the subsequent network queue in the flow path until a_l time later, where a_l is the delay of the link in-between. As a result, one can keep using the same minimum link delay as the lookahead in the hybrid model. No extra synchronization overhead is introduced.

In this paper, we provide a detailed account of our implementation of the hybrid model in our parallel network simulator. We describe the data structures used for parallelizing the fluid ODE solver. In particular, we use ghost nodes to simplify the propagation of fluid characteristics across processor boundaries. These data structures allow us to implement the hybrid model without inducing changes to the existing parallel simulation kernel. We also provide a cost analysis to compare the performance of the fluid model with that of the packet-oriented simulation. In both sequential and parallel cases, we show that the performance of the fluid model is proportional to the average traffic intensity and the time step size of the Runge-Kutta method, suggesting that the parallel fluid model can asymptotically achieve the same scalability as the packet-oriented simulation.

We conducted a series of simulation experiments of large-scale networks both on a shared-memory multiprocessor

sors and on a small cluster of 10 dual-processor dual-core AMD machines. The largest network model consisted of more than 170,000 network entities and over 1.6 million (TCP and UDP) traffic flows.

The rest of this paper is organized as follows. Section 2 provides a detailed review of our hybrid model. The parallelization method is described in detail in Section 3. We provide a simple cost analysis in Section 4. Section 5 describes the parallel simulation experiments with the hybrid traffic model. We conclude the paper in Section 6.

2. Hybrid Traffic Model

The hybrid traffic model combines both continuous-time and discrete-event simulation paradigms. The fluid traffic flows are modeled as a set of ordinary differential equations and solved using the time-stepped Runge-Kutta method. The packet-oriented traffic flows are modeled as discrete events representing packets being forwarded hop by hop in the simulated network. To correctly integrate the fluid and packet flows, we must consider the effect of packet flows on fluid flows, and vice versa. In the following description, we inherit most of the math symbols used in the original paper by Liu et al. [13] with changes necessary to describe the integration of packet flows in the equations.

2.1. Modeling Fluid Flows with the Effect of Packet Flows

We divide the set of differential equations into three categories according to their functions as governing the flow rates, the lengths of network queues, and the propagation of packet delays and losses. Fluid flows with the same source and the same destination share the same fluid traffic characteristics and are classified into the same fluid class. The entire fluid network traffic can therefore be viewed as a collection of N fluid classes.

The Flow Rates

If the fluid flows in class i are TCP flows, we can use the following differential equation to model the additive increase and multiplicative decrease behavior of the congestion window size in the TCP congestion avoidance mode:

$$\frac{dW_i(t)}{dt} = \frac{1}{R_i(t)} - \frac{W_i(t)}{2} \cdot \lambda_i(t), \quad (1)$$

where $W_i(t)$ is the size of the congestion window, $R_i(t)$ is the round-trip delay, and $\lambda_i(t)$ is the packet loss rate, at time t . Note that $W_i(t)$ must be in the range between 0 and the maximum congestion window size (such constraint is not described in the equation). The send rate of fluid class i at any given time t can therefore be calculated as $A_i(t) = n_i W_i(t) / R_i(t)$, where n_i is the number of flows

represented in class i . It is obvious that the send rate $A_i(t)$ and consequently the arrival rates to all network queues traversed by fluid class i depend both on the round-trip delay $R_i(\cdot)$ and on the packet loss rate $\lambda_i(\cdot)$.

If the fluid class i consists of constant-rate UDP flows, the send rate of the fluid class is simply a constant.

The Queue Lengths

The lengths of the network queues play an important factor in determining the packet delays and losses. The rate at which the length of a network queue changes depends on the aggregate arrival rate and the departure rate. In particular, the instantaneous length of the network queue associated with the link l at time t can be modeled using the following differential equation:

$$\frac{dq_l(t)}{dt} = \xi_l(t)(1 - p_l(t)) - C_l, \quad (2)$$

where $\xi_l(t)$ is the aggregate arrival rate of both fluid and packet flows, $p_l(t)$ is the packet dropping probability— $\xi_l(t)(1 - p_l(t))$ is therefore the rate of traffic that actually enters the queue—and C_l is the bandwidth of link l . Note that we should limit the queue length to be in the range between 0 and the maximum queue size.

The aggregate arrival rate $\xi_l(t) = A_P^l(t) + \Lambda_l(t)$, where $A_P^l(t)$ is the aggregate arrival rate of all packet flows, and $\Lambda_l(t)$ is the aggregate arrival rate of all fluid classes passing through link l . The aggregate packet arrival rate $A_P^l(t)$ is calculated through averaging—by dividing the number of packet arrivals during the last interval by the length of the interval. Naturally we use the interval between the Runge-Kutta steps as the interval for calculating the average packet arrival rate.

The aggregate fluid arrival rate is the sum of the arrival rates of all N_l fluid classes traversing the link l : $\Lambda_l(t) = \sum_{i \in N_l} A_i^l(t)$. The arrival rate of fluid class i at queue l , $A_i^l(\cdot)$, can be calculated from the propagation of the send rate of class i , $A_i(\cdot)$, subject to delays and losses along the path. The send rate of fluid class i is actually the arrival rate at its first network queue s_i : $A_i^{s_i}(t) = A_i(t)$. For a subsequent queue $g_i(l)$, the arrival rate is simply the departure rate from the upstream queue l after a propagation delay a_l :

$$A_i^{g_i(l)}(t + a_l) = D_i^l(t) \quad (3)$$

Conversely, the departure rate depends on the arrival rate. If the aggregate arrival rate of flows actually entering the queue is less than the service capacity (i.e., the link bandwidth), the departure rate, $D_i^l(\cdot)$, is the same as the arrival rate after a proper queuing delay. Otherwise, the service capacity is shared among all the competing flows arriving at

the queue:

$$D_i^l(t_f) = \begin{cases} A_i^l(t)(1 - p_l(t)) & \text{if } \xi_l(t)(1 - p_l(t)) \leq C_l \\ \frac{A_i^l(t)}{\xi_l(t)} C_l & \text{otherwise} \end{cases} \quad (4)$$

where $t_f = t + q_l(t)/C_l$.

The packet dropping probability $p_l(\cdot)$ is used specifically to model the process of selectively dropping packets dictated by the Random Early Detection (RED) queuing discipline. In a RED queue, the dropping probability is commonly defined as a piece-wise linear function of the average queue length: $p_l(t) = p(x_l(t))$, where $x_l(t)$ is the exponentially weighted moving average (EWMA) of the instantaneous queue length $q_l(t)$, and $p(x) =$

$$\begin{cases} 0 & 0 \leq x < q^{min} \\ \frac{x - q^{min}}{q^{max} - q^{min}} p^{max} & q^{min} \leq x < q^{max} \\ \frac{x - q^{max}}{q^{max}} (1 - p^{max}) + p^{max} & q^{max} \leq x \leq 2q^{max} \\ 1 & \text{otherwise.} \end{cases}$$

q^{min} , q^{max} and p^{max} are configurable parameters of the RED queue. Written as a differential equation, the average queue length can be expressed as follows:

$$\frac{dx_l(t)}{dt} = \frac{\ln(1 - \alpha)}{\delta} x_l(t) - \frac{\ln(1 - \alpha)}{\delta} q_l(t),$$

where α is the weight used in EWMA and δ is the step size of the Runge-Kutta method.

In a regular drop-tail queue, packets are dropped only when the queue is full and the aggregate arrival rate is larger than the service capacity, in which case we set $p_l(t) = (\xi_l(t) - C_l)/\xi_l(t)$. Otherwise, $p_l(t) = 0$.

The Packet Delays and Losses

In cases of TCP fluid flows, we need to determine the round-trip delay and packet loss rate since they affect the size of the TCP congestion window. For each fluid class i , the packet delays and losses can be calculated cumulatively. We use $d_l^i(t)$ to denote the cumulative delay of the fluid flows in class i departing from queue l at time t . If queue l is the first network queue traversed by the fluid flows, the cumulative delay is simply the queuing delay. Otherwise, it is the sum of cumulative delay recorded at the predecessor queue $b_i(l)$, the propagation delay, and the queuing delay.

$$d_l^i(t_f) = \begin{cases} \frac{q_l(t)}{C_l} & \text{if } l = s_i \\ d_{b_i(l)}^i(t - a_{b_i(l)}) + a_{b_i(l)} + \frac{q_l(t)}{C_l} & \text{otherwise} \end{cases} \quad (5)$$

Similarly, we use $r_l^i(t)$ to denote the cumulative packet drop rate of fluid flows in class i departing from queue l at time t . The cumulative packet drop rate is the sum of packet drops at each network queue along the path:

$$r_l^i(t_f) = \begin{cases} A_i^l(t)p_l(t) & \text{if } l = s_i \\ r_{b_i(l)}^i(t - a_{b_i(l)}) + A_i^l(t)p_l(t) & \text{otherwise} \end{cases} \quad (6)$$

We assume packet losses and queuing delays are insignificant on the reverse path as TCP acknowledgment packets travel back from the destination to the source. Let π_i be one-way path latency of fluid flows in class i . That is, $\pi_i = \sum_{l \in F_i} a_l$, where F_i is the set of network queues along the forwarding path of class i . Assuming the path latency is symmetrical, we compute the round-trip time and the packet loss rate as follows:

$$R_i(t) = d_{f_i}^i(t - \pi_i) + \pi_i \quad (7)$$

$$\lambda_i(t) = r_{f_i}^i(t - \pi_i)/n_i, \quad (8)$$

where $f_i \in F_i$ is the last queue traversed by fluid class i on the forwarding path.

So far we have a set of nonlinear time-varying ordinary differential equations and we apply the fixed time-step Runge-Kutta method to solve them. In general, the Runge-Kutta method [2] is a fourth-order method commonly used for approximating the solution $y = y(x)$ of the initial value problem: $dy/dx = f(x, y)$, where $y(x_0) = y_0$. The method uses the following iterative formula to compute the approximations y_1, y_2, \dots, y_n of the actual values $y(x_1), y(x_2), \dots, y(x_n)$, where $x_{i+1} = x_i + \delta$ for $i = 0, 1, \dots$:

$$y_{n+1} = y_n + \frac{\delta}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

where

$$\begin{aligned} k_1 &= f(x_n, y_n), \\ k_2 &= f(x_n + \delta/2, y_n + \delta k_1/2), \\ k_3 &= f(x_n + \delta/2, y_n + \delta k_2/2), \\ k_4 &= f(x_{n+1}, y_n + \delta k). \end{aligned}$$

In simulation, this is achieved by having the simulator schedule an event periodically every δ units of simulation time. We chose δ to be at least an order of magnitude smaller than the minimum link delay to maintain the accuracy of the numerical method. At each Runge-Kutta step, we evaluate the differential equations updating the congestion window size of each fluid flow and the length of every network queue that conducts fluid flows.

2.2. Modeling Packet Flows with the Effect of Fluid Flows

Packet flows are modeled as discrete events. We use one simulation event to represent a packet arriving at a router and another to represent the packet departing from the router. As in a pure packet-oriented simulation, a packet departure event is used to mark the instance that a network router is about to transmit a packet after the packet has endured the necessary queuing delay. In processing the packet

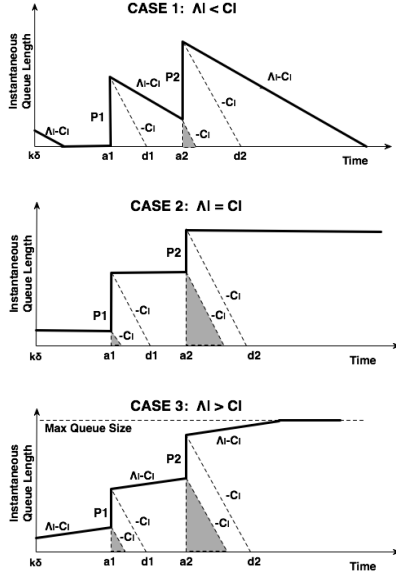


Figure 1. Instantaneous queue length \tilde{q}_l upon hybrid traffic arrivals.

departure event, we compute the time it takes for the packet to reach the next router (i.e., transmission delay and propagation delay) and schedule a packet arrival event.

We pay special attention to the processing of packet arrival events. The admittance of a packet obviously will change the instantaneous queue length, and consequently the average queue length and the packet dropping probability in the case of a RED queue. In the fluid model, the queue lengths and the dropping probability are governed by the differential equations. To reflect the changes that occur between the Runge-Kutta steps upon packet arrivals—in the instantaneous queue length, as well as the average queue length and the packet dropping probability—we create three corresponding shadow variables for each network queue at the beginning of each Runge-Kutta interval. At the k^{th} Runge-Kutta step, we initialize the shadow variables using the values evaluated from the differential equations: $\tilde{q}_l \leftarrow q_l(k\delta)$, $\tilde{x}_l \leftarrow x_l(k\delta)$, and $\tilde{p}_l \leftarrow p_l(k\delta)$. Upon each packet arrival during the k^{th} interval, we readjust the values of these shadow variables and schedule the corresponding packet departure event.

Figure 1 shows the changes to the instantaneous queue length upon packet arrivals. When a packet arrives at time a_1 , for example, we need to first adjust the instantaneous queue length due to the constant arrival of fluid flows. The change depends on the difference between the service rate C_l and the aggregate arrival rate of fluid that actually enters the queue $\Lambda_l = \Lambda_l(k\delta)(1 - p_l(k\delta))$. Obviously, the instantaneous queue length should be bounded between 0 and the maximum queue size Q_l . That is,

$\tilde{q}_l \leftarrow \min\{Q_l, \max\{0, \tilde{q}_l + (a_1 - k\delta)(\Lambda_l - C_l)\}\}$. Three cases are shown Figure 1 as the change of the instantaneous queue length depends on whether the in-flow rate is smaller than, equal to, or larger than the out-flow rate. In any case, upon a packet arrival, if the queue has no space to admit the packet or if the packet is chosen intentionally to be dropped by the RED policy (in accordance with the packet dropping probability \tilde{p}_l), we discard the packet. Otherwise, we add the packet to the queue. After the packet has been added to the queue, a corresponding packet departure event must be scheduled. As mentioned before, processing the departure event allows us to schedule the packet arrival event at the next router on the flow path. It is straightforward that the packet arrived at time t can only complete its departure when all the content of the queue at time t is “drained” from the queue at a rate equivalent to the service rate C_l . For example, the first packet arrived at time a_1 in Figure 1 will prompt the simulator to schedule a packet departure event at time $d_1 = a_1 + \tilde{q}_l/C_l$.

3. Parallelization

The METIS graph partitioner [7] is used by the network simulator to spatially divide the simulated network into subnetworks of approximately equal size and at the same time maximize the communication latencies (thereby better lookahead) between the subnetworks. More sophisticated graph partitioning scheme attempting to maximize the latencies and minimize the traffic load between processors can also be applied (for example, see [12]). It is clear that to facilitate router-level integration of the fluid flows and the packet flows, the set of differential equations specified by the fluid model must be solved together with the packet-oriented simulation run in parallel.

3.1. Data Structures

To parallelize the hybrid model—specifically, the ODE solver of the fluid model—we associate the variables to be evaluated by the differential equations with the corresponding modules of discrete-event packet-oriented simulator. For example, the congestion window size of fluid class i , $W_i(t)$, is maintained together with the data structure that represents the simulated host initiating the TCP flows. In this way, as the network is partitioned among the processors, the fluid model is also partitioned accordingly.

A *FluidClass* object is used to represent a class of homogeneous fluid traffic flows (with the same source and destination exhibiting the same traffic characteristics). The fluid class is assigned to the same processor responsible for simulating the source node of the traffic flows. The fluid class contains parameters that specify the type of the fluid flows

(for example, TCP RENO or constant-rate UDP), the number of homogeneous traffic flows represented by this class, as well as the average packet size. We include additional parameters specific to each traffic type. For example, for constant UDP flows, a send rate must be specified. TCP flows must specify the maximum congestion window size (which is the maximum number of packets that can be sent in flight without acknowledgment). Also, for TCP flows, we need to maintain the round-trip time $R_i(\cdot)$, the end-to-end packet loss rate $\lambda_i(\cdot)$, and the current congestion window size $W_i(\cdot)$. These variables are updated at each Runge-Kutta step by the processor responsible for the fluid class.

Each fluid class also maintains a linked list of *FluidHop* objects representing the sequence of network queues traversed by the fluid flows from the source to the destination. Each *FluidClass* object contains two pointers to both the first and the last fluid hops, one representing the traffic initiated at the source node and the other representing the returning traffic ended at the same node. Except for the last hop, there is a one-to-one mapping between the fluid hop and the corresponding network queue traversed by the fluid flows. The last hop is intended to allow the cumulative delay and loss rate to be sent back to the source node where flow state regarding the entire fluid class is calculated and updated (see Equations 7, 8, and 1). For fluid flows spanning multiple processors the linked list is broken into several segments and assign to different processors. We describe the details in the next section.

Each fluid hop contains four variables: the rate of the fluid arriving at the queue $A_i^l(\cdot)$, the rate of fluid leaving the queue $D_i^l(\cdot)$, the cumulative delay $d_i^l(\cdot)$, and the cumulative packet drop rate $r_i^l(\cdot)$. We note that the left-hand side of Equations 3, 4, 5, and 6 all involves assignment of the variables at a future time—if the current time is t , the assignment is expected to happen after the link latency a_l or a certain queuing delay $q_l(t)/C_l$. A data structure must be used to keep the historical record of the assignment. The *FluidSeries* class is basically a linked list of values each associated with a timestamp at which the value is assigned to the variable. The list is sorted in increasing timestamp order. Linear interpolation is applied for the value of the variable with a timestamp lying between two adjacent nodes. Note that the values with timestamps earlier than the current time can be reclaimed to avoid memory overflow. An except is that we should keep the value immediately before or the same as the current time to allow linear interpolation. Figure 2 shows an example of a fluid series with three nodes, one of which has a timestamp smaller than the current time.

The state of each network queue is represented by the *FluidQueue* object, containing parameters related to the configuration of the network queue, such as the maximum queue size, the weight used for calculating the moving av-

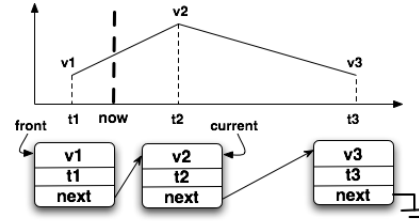


Figure 2. The data structure for maintaining the time-series of $A_i^l(\cdot)$, $D_i^l(\cdot)$, $d_i^l(\cdot)$, and $r_i^l(\cdot)$.

erage of the average queue length from the instantaneous queue length, as well as q_{min} and q_{max} , which are parameters of the RED queuing policy if applicable. It also keeps a list of fluid hops associated with this network queue. State information, such as the instantaneous queue length $q_l(\cdot)$, the average queue length $x_l(\cdot)$, and the packet loss probability $p_l(\cdot)$, is maintained and updated at each Runge-Kutta step. The corresponding shadow variables, \tilde{q}_l , \tilde{x}_l , and \tilde{p}_l , are also updated upon packet arrivals between the Runge-Kutta steps.

3.2. Ghost Hops and Lookahead

A fluid class may traverse multiple subnetworks each assigned to a different processor. In a sequential simulation, the sequence of network queues visited by the flows of a fluid class is represented by a linked list of fluid hops. On parallel machines, we use ghost hops to represent the next fluid hops associated with the network queues assigned to remote processors. The linked list of fluid hops is thus broken into multiple segments assigned to different processors, with the last hop of a previous segment being the ghost hop that mirrors the first hop at the next segment. Using ghost nodes for network simulation is not new. We note that Riley et al. use ghost nodes to partition the network and to simplify communication in the GTNetS simulator [17]. In our case the goal is to facilitate the propagation of fluid characteristics across processor boundaries.

The ghost hop uses the same methods in the *FluidHop* class as a regular fluid hop, namely *setArrivalRate*, *setAccuDelay*, and *setAccuLoss*, for updating $A_i^l(\cdot)$, $d_i^l(\cdot)$, and $r_i^l(\cdot)$, respectively. These methods are invoked upon the next hop when Equations 3, 4, 5, and 6 are evaluated during the Runge-Kutta step. In the case of a ghost hop, the next hop is actually located at a remote processor. Rather than updating the state of the fluid hop, the ghost hop simply sends a message to the remote processor using the standard messaging passing routine provided by the underlying parallel simulation kernel. No simulation kernel modification is necessary. The remote processor receives the message in due time and then updates the variables in the corresponding fluid hop (by invoking the

same set of methods) as if invoked directly from the previous hop. Figure 3 shows an example of the data structures used to represent a class of fluid flows traversing three hops that have been assigned to two processors.

The network simulator is built on a parallel simulation kernel that uses a composite synchronization scheme [15] to regulate the time advancement for all logical processes in the system (the detail of which is beyond the scope of this paper). In particular, the simulator uses the network link delays—specifically, the minimum among the delays of links across the processor boundaries—as the lookahead for synchronizing processors. Suppose that the minimum link delay from logical process LP_A to logical process LP_B is σ . Conservative simulation dictates that the simulation time at LP_B should not be advanced further than the simulation time at LP_A plus σ . It is critical to note that the delays of the channels through which we carry fluid update information are not any smaller than those that carry the simulated packets in the pure packet-oriented simulation. It is important that one should not naively use the time step size, δ , of the Runge-Kutta method to update the variables of the differential equations in the fluid model at a remote processor. The time step size for the Runge-Kutta method is set to be significantly smaller than the link delays, by at least one order of magnitude, to achieve desired numerical accuracy. If we were to use δ , the lookahead that the simulator uses to synchronize the processors would be significantly decreased, causing much more synchronization overhead that would slow down the simulation.

Fortunately, this is not the case as we closely inspect the differential equations of the hybrid model. Equations 3, 5, and 6 specify the propagation of the fluid flow rate, the cumulative delay, and the cumulative loss, respectively, along the flow path. In particular, these equations stipulate that the fluid state at one hop depends on the outcome of the previous one. The fluid rate, the cumulative delay, and the cumulative loss propagate along the flow path, but such propagation takes at least the link delay as the flows travel from one hop to the next. Equation 3 specifies that the departure rate at one hop only becomes the arrival rate at the next hop after the link’s propagation delay a_l . Equation 5 (or Equation 6) states that the cumulative delay (or the cumulative loss rate) at one hop (at the instant of the flows leaving the hop) is used to calculate the cumulative delay (or loss) at the next hop after the sum of the link propagation delay and the queuing delay incurred at the next hop.

As a special case, the round-trip time $R_i(\cdot)$ and the end-to-end packet loss rate $\lambda_i(\cdot)$ in Equations 7 and 8 are calculated from the cumulative delay and loss at the destination after a one-way path latency π_i . (Recall that we do not model the reverse path in the fluid model). That is, the changes made to the cumulative delay and the cumulative loss rate at the fluid hop mapped to the destination node

will not affect the round-trip time and end-to-end packet loss maintained at the source of the fluid flows until π_i units of simulation time later.

In summary, we conclude that the lookahead, which, in essence, is the lower bound on the difference between the time the state at one processor is changed and the time that the change can affect the state of another processor, remains to be the minimum link propagation delay. This means that the synchronization scheme for the packet-oriented simulation can still be used for the parallelized hybrid model without change. That is, the lookahead, which the simulator can leverage to advance the simulation clock of a logical process ahead of other logical processes, remains unchanged. No additional synchronization overhead will be introduced in the parallel hybrid model.

4. Cost Analysis

In this section we use a simple analysis to compare the cost of the fluid model with the packet-oriented model, both run sequentially and in parallel. Note that the purpose of this analysis is not to provide a precise quantitative prediction of either model, but to offer a qualitative comparison of the two models, so that we have a better understanding of the computing cost of the hybrid model. Indeed, our performance analysis indicates that the performance of the hybrid model should scale equally well as the pure packet-oriented simulation on parallel processors.

Let A_i be the mean packet arrival rate for traffic class i . For TCP flows, $A_i = n_i W_i / R_i$, where n_i is the number of homogeneous flows in class i , W_i is the average congestion window size, and R_i is the average round-trip time. For constant-rate UDP flows, A_i is simply the send rate. Assuming there are two simulation events (also called packet events) to represent a packet traversing each hop—one for arrival and the other for departure, the maximum packet-event rate is $\sum_{i=1}^N 2A_i l_i$, where N is the total number of traffic classes and l_i is the number of hops traversed by flows in class i . Here we use the maximum rate since we have not accounted for the packet losses due to network congestion or the RED policy. Suppose that the number of packets sent by class i is but a fraction of the maximum ($0 < \rho_i \leq 1$), and the mean cost of processing each packet event is C_e , the cost of running the packet-oriented simulation for T units of simulation time can be expressed as:

$$T_P = 2TC_e \sum_{i=1}^N \rho_i A_i l_i. \quad (9)$$

The cost of the fluid model is exerted during each Runge-Kutta step. Let C_w be the cost of updating the congestion window size of a fluid class, which includes the calculation of $R_i(\cdot)$, $\lambda_i(\cdot)$, and subsequently $W_i(\cdot)$ for the current

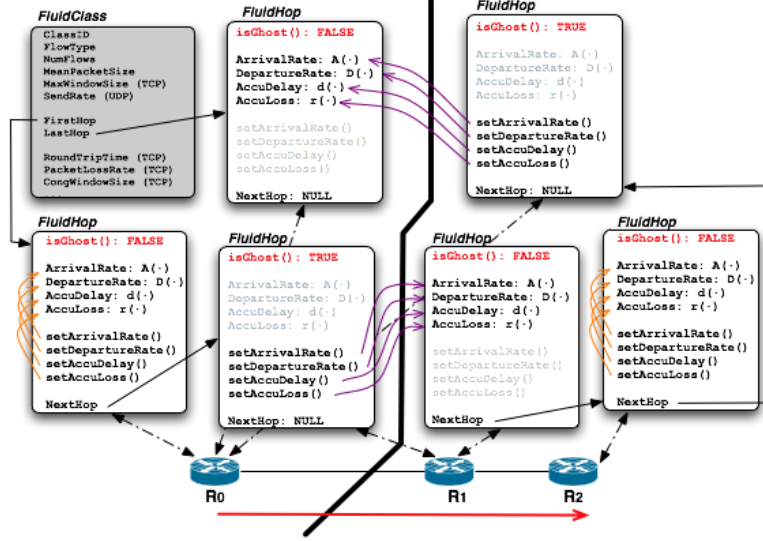


Figure 3. Data structures for a class of fluid flows visiting three routers on two processors.

Runge-Kutta step. Let C_q be the cost of updating the state of a network queue, including $q_l(\cdot)$, $x_l(\cdot)$, and $p_l(\cdot)$. Let C_h be the cost of updating the propagation of fluid rates $A_i^l(\cdot)$ and $D_i^l(\cdot)$, the cumulative delay $d_i^l(\cdot)$, and the cumulative loss $r_i^l(\cdot)$, at each fluid hop. The cost of each Runge-Kutta step can therefore be summarized as $C_w N + C_q m + C_h \sum_{i=1}^N l_i$, where m is the number of fluid network queues (i.e., network queues that conduct fluid traffic). Let δ be the step size of the Runge-Kutta method. The overall cost of the fluid model is:

$$T_F = \frac{T}{\delta} \left(C_w N + C_q m + C_h \sum_{i=1}^N l_i \right).$$

Now if we assume that on average there are γ fluid classes through each fluid queue, i.e., $m\gamma = \sum_{i=1}^N l_i$. Also, considering the cost of updating the congestion window size is comparatively insignificant when compared to the cost of updating the fluid queues and calculating the propagation of fluid characteristics throughout the entire network, we can further assume that $C_w \ll (C_q/\gamma + C_h)l_i$. The cost of the fluid model becomes:

$$\begin{aligned} T_F &= \frac{T}{\delta} \left(C_w N + C_q/\gamma \sum_{i=1}^N l_i + C_h \sum_{i=1}^N l_i \right) \\ &= \frac{T}{\delta} \sum_{i=1}^N (C_w + (C_q/\gamma + C_h)l_i) \\ &\approx \frac{T(C_q/\gamma + C_h)}{\delta} \sum_{i=1}^N l_i. \end{aligned} \quad (10)$$

The following theorem can be obtained directly from Equations 9 and 10:

Theorem 1 *The speedup of the sequential fluid model over the sequential packet-oriented simulation is*

$$\frac{T_P}{T_F} = \frac{2TC_e \sum_{i=1}^N \rho_i A_i l_i}{T(C_q/\gamma + C_h)/\delta \sum_{i=1}^N l_i} = C_\psi A \delta, \quad (11)$$

where $C_\psi = 2C_e/(C_q/\gamma + C_h)$ is a constant and $A = \sum_{i=1}^N \rho_i A_i l_i / \sum_{i=1}^N l_i$ is the average traffic intensity of the network.

Theorem 1 implies that the speedup that can be achieved by the fluid model over the pure packet-oriented simulation is proportional to the network traffic intensity and the time-step size of the Runge-Kutta method. Intuitively, the product $A\delta$ represents the number of packet events to be processed in the pure packet-oriented simulation for each Runge-Kutta time step that are saved instead if using the numerical method.

Now we proceed to estimate the cost of parallelization. For simplicity, we assume that the network model can be divided evenly among the processors. Let $T_P^s(p)$ be the cost of synchronization and $T_P^c(p)$ be the cost of communication for running the packet-oriented simulation on p processors. The cost of the parallel packet-oriented simulation can therefore be expressed as

$$T_P(p) = T_P/p + T_P^s(p) + T_P^c(p). \quad (12)$$

$T_P^s(p)$ depends on the synchronization protocol that the simulator chooses to coordinate the simulation processes running in parallel. For example, in a window-based synchronous conservative algorithm, a global reduction (the cost of which is logarithmic to the number of processors)

is performed every σ units of simulation time, which is the size of the synchronization window. The synchronization cost in this case is $T_P^s(p) = (C_s T / \sigma) \log p$, where C_s is the constant of proportionality.

$T_P^s(p)$ depends on the total number of messages sent between the processors. We assume that the communication is divided evenly among the processors and the cost is proportional to the packet events going across the process boundaries. We use κ_i to denote the number of segments assigned to different processors for traffic flows in class i . That is, traffic flows in class i travel through κ_i processors. The communication cost can thus be calculated as

$$T_P^c(p) = \frac{\mathcal{T} C_m^P}{p} \sum_{i=1}^N \rho_i A_i (\kappa_i - 1), \quad (13)$$

where C_m^P is the average cost of sending a packet event. Since $\rho_i A_i$ is the fraction of packets that are generated and sent along the traffic path, $\rho_i A_i (\kappa_i - 1)$ is the rate of packets that are sent between the processors for flows in class i . The total number of packets should be doubled if we account for the TCP ACK packets on the reverse path, in which case C_m^P can be considered as the average cost of sending two packet events (one for the data packet and the other for the ACK).

For the parallel fluid model, the cost can also be classified similarly into the cost of computation, synchronization, and communication:

$$T_F(p) = T_F/p + T_F^s(p) + T_F^c(p). \quad (14)$$

Since the parallelized fluid model does not alter the lookahead of the pure packet-oriented simulation, the synchronization overhead remains the same: $T_F^s(p) = T_P^s(p)$. Actually, to maintain scalability, the cost of computation must dominate the cost of synchronization. Assuming perfect load balance, the cost of synchronization increases only logarithmically to the number of processors. On the contrary, the cost of processing events grows much faster as the network size increases. For this reason, we can ignore the cost of synchronization in the overall cost of the parallel models.

The communication overhead for the fluid model depends on the number of fluid update events sent by the ghost hops (which is independent of the flow rates). Again, assuming the communication cost is proportional to the number of messages sent between the processors—in this case, fluid events rather than packet events, the communication time can be calculated as

$$T_F^c(p) = \frac{\mathcal{T} C_m^F}{p\delta} \sum_{i=1}^N (\kappa_i - 1). \quad (15)$$

where C_m^F is the average cost of sending a fluid update event. Here, $(\kappa_i - 1)$ is the number of segment crossings (i.e., processor crossings) for fluid class i . At each

Runge-Kutta step, a ghost hop will send a fluid update message to the remote processor carrying the updated fluid information—the flow rates $A_i^l(\cdot)$ and $D_i^l(\cdot)$, the cumulative delay $d_i^l(\cdot)$, and the cumulative loss $r_i^l(\cdot)$ —for the next Runge-Kutta interval.

We can compute the ratio of the communication cost of the packet-oriented model to that of the fluid model:

$$\frac{T_P^c(p)}{T_F^c(p)} = \frac{\mathcal{T} C_m^P \sum_{i=1}^N \rho_i A_i (\kappa_i - 1) / p}{\mathcal{T} C_m^F \sum_{i=1}^N (\kappa_i - 1) / (p\delta)} = \frac{C_m^P}{C_m^F} A' \delta, \quad (16)$$

where $A' = \sum_{i=1}^N \rho_i A_i (\kappa_i - 1) / \sum_{i=1}^N (\kappa_i - 1)$. It is reasonable to assume that the number of segment crossings is proportional to the length of the flow path. That is, $(\kappa_i - 1) = \alpha \cdot l_i$ for some constant α . We have $A' = A$.

Furthermore, we can show that the ratio of the parallel computation cost to the communication cost is also a constant:

$$\begin{aligned} \frac{T_F/p}{T_F^c(p)} &\approx \frac{\mathcal{T}(C_q/\gamma + C_h)/(p\delta) \sum_{i=1}^N l_i}{\mathcal{T} C_m^F / (p\delta) \sum_{i=1}^N (\kappa_i - 1)} \\ &= \frac{\alpha(C_q/\gamma + C_h)}{C_m^F}. \end{aligned} \quad (17)$$

Theorem 2 *If we assume that the models are partitioned evenly among the processors (for both computation and communication) and the number of segment crossings within each traffic class is proportional to the length of the flow path, the speedup of the parallel fluid model over the parallel packet-oriented simulation on p processors is*

$$\frac{T_P(p)}{T_F(p)} \approx C'_\psi A \delta, \quad (18)$$

where C'_ψ is a constant.

Theorem 2 can be derived from Theorem 1 and Equations 12, 14, 16, and 17.

$$\begin{aligned} \frac{T_P(p)}{T_F(p)} &= \frac{T_P/p + T_P^s(p) + T_P^c(p)}{T_F/p + T_F^s(p) + T_F^c(p)} \approx \frac{T_P/p + T_P^c(p)}{T_F/p + T_F^c(p)} \\ &= \frac{C_\psi A \delta T_F/p + T_P^c(p)}{T_F/p + T_F^c(p)} \\ &= \frac{C_\psi A \delta \frac{\alpha(C_q/\gamma + C_h)}{C_m^F} T_F^c(p) + \frac{C_m^P}{C_m^F} A \delta T_F^c(p)}{\frac{\alpha(C_q/\gamma + C_h)}{C_m^F} T_F^c(p) + T_F^c(p)} \\ &= \frac{C_\psi \alpha (C_q/\gamma + C_h) + C_m^P}{\alpha(C_q/\gamma + C_h) + C_m^F} A \delta = C'_\psi A \delta. \end{aligned}$$

Theorem 2 states that the speedup of the parallel fluid model over the parallel packet-oriented simulation remains to be proportional to the product $A\delta$ as in the sequential case. It means that asymptotically the performance benefit of the fluid model over the packet-oriented model can

be extended to the parallel setting. In another word, if the packet-oriented simulation is scalable—i.e., the efficiency of the processors does not deteriorate as one simultaneously increases the size of the network and the number of processors [1], it is reasonable to assume that the fluid model will remain to be scalable (although subject to the simplifying conditions thus made).

5. Experiments

We performed a preliminary validation study of the hybrid model in [10]. Here we focus on the parallel performance of the hybrid approach. For the network model, we chose the baseline NMS challenge topology [18]. The network consists of a variable number of “network clouds” or “campuses”, each with over 500 hosts and routers. The campuses are connected as a ring with additional shortcut connections between the campuses at the top level. The traffic consists of an equal number of TCP and UDP flows. We configured each end host to initiate a total of 10 TCP or UDP connections to servers randomly chosen from either within or outside of the campus requesting large file downloads, which would last far beyond the duration of the test. On average, 50% of the traffic flows were to be between campuses.

The first set of experiments were conducted on an Apple Mac Pro with two 3 GHz dual-core Intel Xeon processors and 9 GB of memory. We measured the time of the simulation both running sequentially and with 4-way parallelism. To obtain a consistent workload for the simulator, we subtracted the simulation ramp-up time, including the time during which the TCP sessions were in the slow-start phase. Figure 4 shows the ratio of real time over simulation time, as we varied the fraction of packet flows. Pure fluid model (i.e., 0% packet flows) ran the fastest. For up to 1% of packet flows, the execution time of the hybrid model, which was dominated by the time-stepped ODE calculations, remained steady. As we introduced more packet flows in the model, the execution time increased as expected. The super-linear increase was due to a larger memory working set and, more importantly, the disproportional increase of the scheduling overhead for the packet events. The pure packet-oriented model (i.e., 100% packet flows) achieves less slowdown than projected because of the removal of the fluid model.

Figure 5 shows the speedup achieved over the pure packet-oriented model of a 32-campus network (17,216 network entities and 161,280 traffic flows) as we varied fraction of packet flows. The majority of the benefit obviously comes from reduced workload from the fluid model.

We conducted another set of experiments on a small Linux cluster of 10 nodes connected by a gigabit network. Each node contains two ADM dual-core Opteron 270 pro-

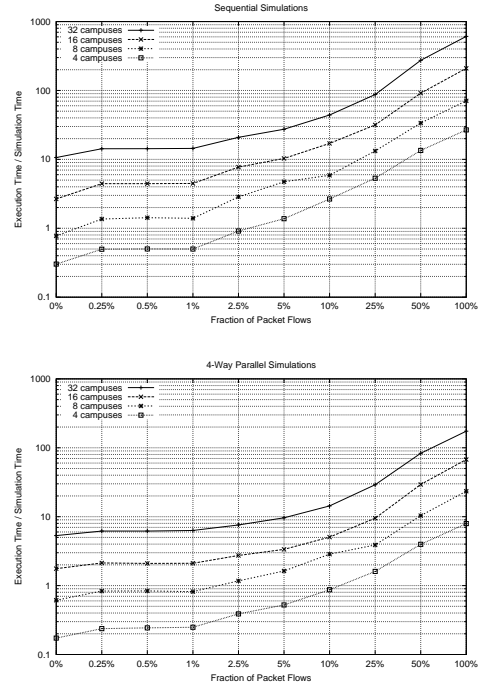


Figure 4. The ratio of execution time to simulation time with varying traffic compositions.

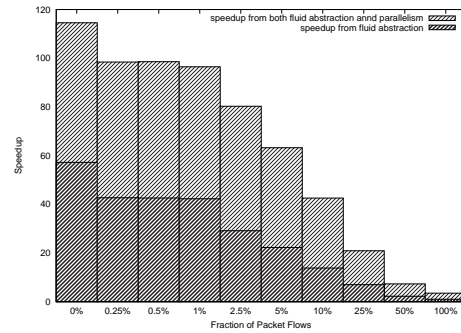


Figure 5. The speedup over the pure packet-oriented simulation of a 32-campus network.

cessors and 8 GB of memory. The simulation ran on up to 8 nodes (i.e., with 32-way parallelism). As we increased the number of nodes, we increased the network size proportionally so that the number of campus networks assigned to each processor core remained at 10. The largest network we simulated thus contained 320 campuses (172,160 hosts/routers and 1,621,800 traffic flows). Figure 6 shows the good scalability of the parallel hybrid model. The effective packet-event rate, i.e., the number of packet events to be processed per second per processor core in the equivalent pure packet-oriented model, remained relatively unchanged over a large span of network sizes. The slight decrease

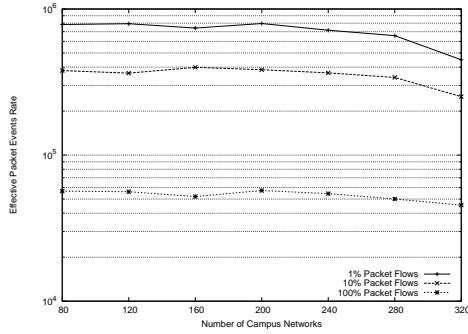


Figure 6. The scalability of the hybrid model.

toward larger network sizes can be attributed to the communication overhead—the increase of cross-campus traffic (recalling that 50% of the traffic generated at a campus is targeted off campus) over a relatively slow interconnection network (a gigabit Ethernet). More prominently than others, due to the reduced workload of the fluid calculations in the hybrid model, the simulation achieved an effective rate of over 800,000 packet-events per second in the case of 1% traffic modeled as packet flows.

6. Conclusions

We proposed a method for parallelizing the hybrid network traffic model combining a time-stepped fluid model with the traditional discrete-event packet-oriented model. It is important to note that this method allows the fluid model to maintain the same lookahead as in the packet-oriented simulation and therefore does not incur any additional synchronization overhead. We provided a simple analytical model to support the claim that the parallelized fluid model maintains the same computational advantage over the packet-oriented model as in the sequential case, which was further supported by the performance results of large network simulation experiments.

References

- [1] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global Internet. *Computing in Science and Engineering*, 1(1):42–50, January 1999.
- [2] H. C. Edwards and D. E. Penney. *Differential Equations: Computing and Modeling, 3rd Edition*. Prentice Hall, 2003.
- [3] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. Large-scale network simulation – How big? How fast? In *Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS)*, October 2003.
- [4] R. M. Fujimoto. Lookahead in parallel discrete event simulation. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 34–41, August 1988.
- [5] R. M. Fujimoto. *Parallel and distributed simulation systems*. John Wiley & Sons, 2000.
- [6] Y. Gu, Y. Liu, and D. Towsley. On integrating fluid models with packet simulation. In *Proceedings of IEEE INFOCOM 2004*, March 2004.
- [7] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. University of Minnesota, September 1998.
- [8] C. Kiddle, R. Simmonds, C. Williamson, and B. Unger. Hybrid packet/fluid flow network simulation. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS'03)*, pages 143–152, June 2003.
- [9] M. Liljenstam, Y. Yuan, B. J. Premore, and D. M. Nicol. A mixed abstraction level simulation model of large-scale Internet worm infestations. In *Proceedings of the 10th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'02)*, 2002.
- [10] J. Liu. Packet-level integration of fluid TCP models in real-time network simulation. *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, December 2006. To appear.
- [11] J. Liu, S. Mann, N. V. Vorst, and K. Hellman. An open and scalable emulation infrastructure for large-scale real-time network simulations. Submitted for publication.
- [12] X. Liu and A. A. Chien. Traffic-based load balance for scalable network emulation. In *Proceedings of the 2003 Supercomputing Conference (SC'2003)*, November 2003.
- [13] Y. Liu, F. L. Presti, V. Misra, D. F. Towsley, and Y. Gu. Scalable fluid models and simulations for large-scale IP networks. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(3):305–324, July 2004.
- [14] D. M. Nicol. Fluid simulation: discrete event fluid modeling of TCP. In *Proceedings of the 33rd Winter simulation Conference*, pages 1291–1299, December 2001.
- [15] D. M. Nicol and J. Liu. Composite synchronization in parallel discrete-event simulation. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):433–446, May 2002.
- [16] G. F. Riley, M. H. Ammar, R. M. Fujimoto, A. Park, K. Perumalla, and D. Xu. A federated approach to distributed network simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(2):116–148, April 2004.
- [17] G. F. Riley, T. M. Jaafar, R. M. Fujimoto, and M. H. Ammar. Space-parallel network simulations using ghosts. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, pages 170–177, May 2004.
- [18] Standard baseline NMS challenge topology. <http://ssfnet.org/Exchange/gallery/index.html>, Last accessed: October 2006.
- [19] G. R. Yaun, D. Bauer, H. L. Bhutada, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. Large-scale network simulation techniques: examples of TCP and OSPF models. *ACM SIGCOMM Computer Communication Review*, 33(3):27–41, 2003.
- [20] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia. MAYA: integrating hybrid network modeling to the physical world. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(2):149–169, April 2004.