Hybrid discrete-continuous fluid-flow simulation

Benjamin Melamed *[a], Shuo Pan**[b], Yorai Wardi***[c]
[a]Department of MSIS, Rutgers University; [b]RUTCOR, Rutgers University;
[c]School of Electrical and Computer Engineering, Georgia Institute of Technology

ABSTRACT

The fluid-flow paradigm is an alternative approach to the traditional queueing paradigm, where workload movement in a network is described in terms of a fluid metaphor rather than the familiar customer metaphor. Stochastic fluid network models can considerably speed up the simulation of complex flow networks, such as high-speed packet-based telecommunications networks, where traditional queueing-based simulation routinely gives rise to enormous numbers of events. Furthermore, they are also amenable to fast, nonparametric, and unbiased estimation (unlike traditional queueing) of IPA (Infinitesimal Perturbation Analysis) gradients (derivatives of performance measures with respect to various parameters), which may be used to make decisions on network design, replenishment, management or control. This paper outlines the design of a fluid-oriented hybrid simulator admitting both packet flows and fluid flows. It focuses on the fluid processing design and on efficiency issues.

Keywords: Continuous-Flow Models, CFM, Discrete Event Dynamic Systems, DEDS, Fluid-Flow Models, Fluid-Flow Simulation, Hybrid Simulation, IPA, Stochastic Fluid Models.

1. INTRODUCTION

The stochastic fluid model (SFM) paradigm is an alternative approach to the traditional queueing paradigm, where workload movement in a network is described in terms of a fluid metaphor rather than the familiar customer metaphor. The differences lie in the way workload is processed and moved. Specifically, workload parcels (customers in traditional queueing) and their discrete "all-or-nothing" motion are replaced by fluid-like workload whose continuous flow in the network is governed by (random) rates.

SFM networks have been proposed as an alternative modeling approach to traditional queueing for modeling and simulation of complex flow networks, such as high-speed packet-based telecommunications networks.[1,5,8] Here, a traditional queueing-based simulation routinely gives rise to enormous numbers of events associated with packets. The stochastic fluid paradigm carries out a conceptual aggregation that views workload motion in the network as fluid flow whose constituent molecules are packets, and the flow is described in terms of fluid rates. While the identity of individual packets is lost, the identity of messages is retained. Thus, the fluid-flow approach is suitable for high-level analysis as compared with traditional queueing models.

The fluid-flow approach holds the promise of two advantages. First, SFMs and their variants can considerably speed up the simulation of complex flow networks as compared to traditional queueing models. The fundamental trade-off between the fluid-flow paradigm and the queueing paradigms is a reduced number of simulation events in the former versus computationally "cheap" events in the latter. In practice, this advantage materializes for many models of interest, primarily feed-forward networks.

* melamed@rbs.rutgers.edu; phone 1 732 445-3128; fax 1 732 445-6329; http://rbs.rutgers.edu/~melamed/; Rutgers University, Department of MSIS, 94 Rockafeller Rd., Piscataway, NJ 08854; ** shuopan@rutcor.rutgers.edu; phone 1 732 445-0685; fax 1 732 445-5472; http://rutcor.rutgers.edu/~shuopan/; Rutgers University, RUTCOR, 640 Bartholomew Rd., Piscataway, NJ 08854; *** Yorai.Wardi@ece.gatech.edu; phone 1 404 894-8326; fax 1 404 894-4641; Georgia Institute of Technology, School of Electrical and Computer Engineering, Atlanta, GA 30332

The second advantage is that the fluid-flow paradigm is amenable to IPA (Infinitesimal Perturbation Analysis).[3] IPA is a technique for computing sample path derivatives of performance metrics with respect to parameters of interest (e.g., the derivative of the loss volume as function of buffer size). More formally, let $L(\theta) : \Theta \to \mathbb{R}$ be a real-valued random variable viewed as a function of a real-valued parameter, θ, varying in a closed and bounded interval, Θ. The IPA derivative is the random variable $\frac{d}{d\theta}L(\theta)$ when it exists. To be of practical value, it is necessary for $L'(\theta)$ to be *unbiased*,[3,2] namely,

$$E[\frac{d}{d\theta}L(\theta)] \;=\; \frac{d}{d\theta}E[L(\theta)],$$

and nonparametric (independent of the details of the underlying probability law, especially distributional assumptions). While this is usually not the case in traditional queueing systems, preliminary work[7,9–11] has shown that this is the case in fluid-flow setting. Furthermore, the computational algorithms for IPA derivatives are fast and simple, allowing them to be computed in real time from actual network observations and not just from simulation sample paths (histories). Thus, IPA gradients have potential applications to decision making on network design, replenishment, management and control.

This paper describes the design of a new discrete-event simulation tool, dubbed HDCF-NS (Hybrid Discrete-Continuous Flow Network Simulator), admitting both packet flows and fluid flows. It outlines the worldview of HDCF-NS and its underlying theoretical model, the so-called Continuous Flow Model (CFM). The main focus is on simulator design, particularly the fluid management scheme, and various efficiency issues.

The rest of the paper is organized as follows. Section 2 describes the modeling scope of HDCF-NS and its theoretical underpinnings. Section 3 outlines the layered architecture of HDCF-NS, while Section 4 discusses some efficiency issues pertinent to its implementation. Finally, Section 5 reports on the current status of HDCF-NS and on future directions.

2. MODELING SCOPE

The modeling scope of HDCF-NS consists of hybrid networks supporting multiple flows. The network consists of a set of interconnected stations (nodes), each housing a buffer of capacity $c \geq 0$, and a server. A station may have flow sources and sinks. Flows arrive from a source as messages with a fluid workload, a priority and a network itinerary. A message traverses the network in accordance with its itinerary, and contends at network nodes for network resources (shared buffer space and shared servers), based on its priority. The service discipline is FIFO within priority classes, so that both buffer space and service rates are allocated equally among messages in the highest priority class; any left-over resources are allocated equally among messages in the next-highest priority class, and so on, until resources run out or all node messages are satisfied. Messages leave the network at a sink when their workload drains out of the network.

The hybrid aspect of HDCF-NS admits both discrete flows and continuous flows. While both types arrive as message transactions at a network source and leave the network at a sink, they differ fundamentally in the way their workload is served and routed, as explained below.

2.1. Continuous flows

Continuous flows behave like incompressible fluid, flowing among storage tanks. Since fluid messages are served at a finite rate and are routed continually, it follows that a continuous message is typically "spread" over its itinerary nodes, in the sense that portions of its workload are present at multiple nodes, simultaneously. A continuous message is considered to have departed the network when its entire workload drains out (necessarily this happens last at the last node of the itinerary). Thus, the corresponding message may be termed a *fluid message*. An important constraint on continuous flows is that they may only defined to have a feed-forward topology. The purpose of this restriction is to simplify and speed up HDCF-NS simulations, as will be explained later.

Continuous flows in HDCF-NS at network stations are based on a simple variant of the CFM class, called Continuous Flow Models (CFMs). A single-node CFM consists of a buffer that holds fluid, and a server that discharges fluid from the buffer. The fluid arrival rate is governed by a stochastic process $\{\alpha(t)\}$. Fluid is discharged by the server whose service rate is a stochastic process $\{\beta(t)\}$. The buffer capacity is generally a stochastic process $\{c(t)\}$, but in HDCF-NS it is a fixed quantity, c. It is assumed that the processes $\{\alpha(t)\}$ and $\{\beta(t)\}$ are only constrained to have piecewise-constant sample paths, but are otherwise quite general; in particular, the processes may be autocorrelated.

The buffer contents (workload) at time t is denoted by $x(t)$ and given by

$$\frac{dx(t)}{dt^+} = \begin{cases} 0, & \text{if } x(t) = 0 \text{ and} \\ & \alpha(t) \leq \beta(t) \\ \dot{c}(t), & \text{if } x(t) = c(t) \text{ and} \\ & \alpha(t) - \beta(t) \geq \dot{c}(t) \\ \alpha(t) - \beta(t), & \text{otherwise} \end{cases} \tag{1}$$

where $\dot{c}(t)$ denotes the time derivative of $c(t)$. The fluid outflow (departure) rate from the server at time t is denoted by $\delta(t)$ and given by

$$\delta(t) = \begin{cases} \min\{\alpha(t), \beta(t)\}, & \text{if } x(t) = 0 \\ \beta(t), & \text{if } x(t) > 0 \end{cases} \tag{2}$$

Finally, the spillover (loss) rate due to finite buffer space is denoted by $\gamma(t)$ and given by

$$\gamma(t) = \begin{cases} \alpha(t) - \beta(t) - \dot{c}(t), & \text{if } x(t) = c(t) \text{ and} \\ & \alpha(t) - \beta(t) \geq \dot{c}(t) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

CFM nodes may be combined to model multiple flows at a station, and may be interconnected to form complex networks. The resulting networks always constitute discrete-event dynamic systems (DEDS),[3] and consequently, can be readily fitted into a discrete-event simulation setting.

Continuous flows should be normally used to model long messages, which in reality are made up of large numbers of packets. As such,they provide a vehicle for reducing the computational burden engendered by packet-based simulations.

2.2. Discrete flows

Discrete flows (packets) follow the familiar conventions of standard queueing models, and therefore merit only a brief description.

A packet is served at a station at some allocated rate (like any other message), but while it is being served, no workload leaves the station. Rather, when the message is fully served, the message with its entire workload is routed immediately to the next node on its itinerary, and so on until it leaves the network. Note that unlike a continuous (fluid) message, a discrete message is present at exactly one station at any given time.

However, some behavioral aspects of packets in buffered and bufferless stations are different. In a buffered station, a packet succeeds or fails to enter the station according as there is sufficient buffer space to accommodate its workload. In a bufferless station, however, the situation is more complex. Here, we define the *nominal arrival rate* of a packet on arrival at a station as either (1) the service rate allocated to it at the time of its service completion in the preceding station, or (2) the prevailing injection rate at its arrival time from the preceding source. The bufferless station accommodates the arriving packet only if its *available service rate* (the residual service rate left over after service-rate allocation to equal or higher-priority fluid flows) is no less than the nominal arrival rate of the packet. If an arrival-rate change of a higher-priority fluid flow exhausts the available service rate, then the packet is preempted and lost. Since a packet always absorbs all available service rate, it will effectively "block" any other packet from entering the station (in other words, a bufferless station can accommodate at most one packet at a time).

Discrete flows can have an arbitrary topology, and unlike continuous flows, they are not constrained to a feed-forward topologies, and this is a consequence of their postulated "all at once" movement. Discrete-flow simulation models always constitute the DEDS, so that hybrid models consisting of continuous and discrete flows can be accommodated in the discrete-event simulation paradigm. Such flows can be used to model packet streams, but should typically be used to model single packets, especially those serving a signaling (control) function. In particular, the admissibility of arbitrary topologies for discrete flows allows us to simulate flow control and congestion notification schemes required by extant protocols or newly proposed ones.

Although HDCF-NS is a hybrid simulation tool, the underlying worldview is a fluid-flow paradigm. In fact, discrete flows are readily fitted into the fluid-flow worldview with only minor modifications. In particular, one can extend the notion of arrival and departure rates to discrete flows in a natural way by taking the corresponding sample paths to be not step functions, but Dirac functions, thus capturing the "instantaneous" (discrete) transitions (routing) of packets among network nodes. The implementation of hybrid simulated discrete and continuous flows will be discussed in the sequel.

3. OUTLINE OF ARCHITECTURE

To achieve high efficiency, the fluid-flow worldview of HDCF-NS is implemented in the discrete-event simulation framework, rather than the fixed time increment employed in continuous simulation systems. As mentioned before, this is possible, since the simulated systems are DEDSs. HDCF-NS is driven by events of the following types:

1. **Control events** perform various control functions. Examples include events that reset statistics, update them at prescribed times, and end a simulation run.
2. **Workload transport events** implement fluid-workload transport, driven by rate changes. The latter stem from a variety of causes such as message arrivals, injection-rate modulation, certain transitions among station state, service completions, and routing of discrete messages.
3. **Protocol events** implement protocol specific events such as the firing of a timer.
4. **Statistics events** trigger updates of statistics which are not part of other events. An example is certain IPA derivative computations.

Workload transport events constitute the bulk of processed events. Unlike traditional queueing networks, where the scope of event processing is typically local to a node (for message arrivals) or two (message routing), regardless of topology, the situation in fluid-flow networks is more complicated. Here a rate change event at one node triggers rate updates in all downstream nodes. More specifically, the change of one or more departure rates from a node can change the arrival rates and station state in the set of nodes directly connected to it via continuous flows, which in turn can change their departure rates and so on. Thus, a rate change in a given node can potentially ripple to all downstream nodes. In a similar vein, routing a discrete message from some origination node to some destination node can trigger rate changes downstream both nodes.

The HDCF-NS software is written in Java and is organized in four logical layers:

- **Network Layer:** Maintains the network description and initiates simulation runs.
- **Transport Layer:** Processes fluid workload in a simulation run.
- **Protocol Layer:** Implements a protocol logic in a simulation run (e.g., for telecommunications applications).
- **Statistics Layer:** Estimates station and message statistics in a simulation run, including IPA derivatives.

These are briefly reviewed next.

3.1. Network layer

The network layer is responsible for constructing a fluid-flow network from some description. Network topology for continuous flows is constrained to be feed-forward, but is allowed to be arbitrary for discrete flows. Currently the network is described node-by-node via a text file, but we plan to include a form-based description in the future.

The top object, *HDCFNS*, maintains the network description. Network nodes are modeled by the abstract *Station* class, and its extensions to two concrete classes, *StationBuffered* and *Station-Bufferless*, with the obvious semantics. The latter class is simpler to implement, and can be used to model an optical switch or router in current or emerging technologies.[12]

Each *Station* object has any number of message sources, each modeling a single flow, either continuous or discrete. Accordingly, sources are modeled by the abstract class *Source* and its extensions to two concrete classes, *SourceFluid* and *SourcePacket*. In a similar vein, message stream arriving at each source is modeled by the abstract class *Message* and its extensions to two concrete classes, *MessageFluid* and *MessagePacket*. Each message stream is described by the stochastic process of its interarrival intervals (iid or autocorrelated), the stochastic process of its workload (usually iid), its (fixed) priority and its (fixed) loop-free itinerary of stations to be visited.

HDCF-NS maintains a state variable for each station which reflects the state of the associated buffer and server as follows:

1. State IDLE-EMPTY corresponds to an idle sever and empty buffer. Note that for standard queues this state coincides with the standard idle state. However, this is not the case for fluid-flow as explained in the next item.

2. State BUSY-EMPTY corresponds to a busy sever and empty buffer. This state prevails only in fluid-flow queues when the buffer is empty and the arrival rate does not exceed the service rate.

3. State BUSY-PARTIAL corresponds to a busy server and a buffer which is neither full nor empty.

4. State BUSY-FULL corresponds to a busy sever and full buffer. If the arrival rate exceeds the service rate in this state, some flows will sustain fluid (workload) loss.

Station state tracking is needed to process events correctly, since some state transitions give rise to rate changes. Finally, we have reserved the states IDLE-PARTIAL and IDLE-FULL for possible implementation of traffic blocking.

3.2. Transport layer

The fluid transport layer is responsible for processing fluid workload and collecting all statistics in the course of a simulation run. This layer handles workload arrivals in every message stream, processes message workload at network nodes according to their priorities, and routes the flows according to their itineraries until they drain out of the network. In other words, this layer “pushes” message workload “forward” in the network, from sources to sinks, viewing all workloads as fluid.

The transport layer operates differently at the network boundary (sources) and its interior (stations). Starting with the network boundary, the source associated with a message stream queues up each incoming message in an arrival queue (using the traditional queueing paradigm), and funnels its workload to the associated network station (the first node on its itinerary) in a process called *injection*. Injection is different for a *MessagePacket* object (discrete flow) and a *MessageFluid* object (continuous flow). In the former case, the entire discrete message attempts to enter the first itinerary node, and if it cannot be accommodated in its buffer, that message is immediately and completely lost. In the latter case, the injection process is specified as two independent stochastic processes: one describing the sequence of injection rates and the second their durations (each of these processes may be iid or autocorrelated). The source then modulates the injection rate according to this prescribed process, and the injected message may experience partial or complete workload losses. Thus, the injection rate represents the offered rate, while the effective rate is the offered rate minus the loss rate. Finally, when injection is completed, the fully-injected message is moved from the arrival queue to a linked list of messages pending completion, from which it will be removed and discarded in due time, once its sojourn in the network is completed.

The transport layer at a network station is considerably more complex, due to the intrinsic characteristics of continuous flows. Unlike discrete flows (traditional queues), continuous flows require not only the queueing-oriented tracking of fluid messages, but also a detailed tracking and bookkeeping of "historical" arrival rates at each station. To see why, observe that multiple flows into a buffer give rise each to a volume of fluid, called *parcel*; unless served fast enough, each parcel's volume grows at a constant rate. Since flow rates are piecewise-constant, there exists a time interval over which the vector of individual flow rates is also constant. We refer to such time intervals as *durations*. Thus, the total accumulated fluid volume consists of a mixture of parcels (called a *multiparcel*), whose individual parcel volumes are proportional to their flow rates over each duration. Whenever a rate change occurs, the next multiparcel builds up as a different mixture. A parcel associated with a flow is implemented by an object from class *Parcel*, which records its flow affiliation and flow rate. In a similar vein, a set of parcels from different flows over a duration is implemented by an object from class *Multiparcel*, which has a vector of the corresponding flow parcels, and the begin-time and end-time of the common duration. The buffer workload associated with a *Multiparcel* object is inferred (via a Java method) for each constituent flow from the current time (simulation clock), the duration's begin time and end time, and the prevailing service rate allocated to the multiparcel (this service rate is shared among all its constituent flows proportionally to their "historical" arrival rates and coincides with the respective departure rates).

HDCF-NS employs a fluid management scheme, called *parceling*, which organizes incoming fluid into parcels and multiparcels, queues them up for service, and while being served, arranges to discharge each multiparcel constituent flow proportionally to the original arrival rates. To this end, a 2-dimensional linked list is maintained at each station as a variable-size matrix of multiparcels, whose rows correspond to priority queues (sets of flows with the same priority), and each queue consists of multiparcels in the chronological order of their creation. Note that only the multiparcels at the head of each priority queue are served, since they represent the "oldest" fluid in the station buffer. The multiparcel matrix consists of two kinds of (logical) multiparcels: *open* multiparcels and *closed* multiparcels. An open multiparcel captures the current (open-ended) multiparcel in progress, and as such it records a begin-time but no end-time. As soon as one of its flow rates changes, it gets "closed" (in the sense that the current time is assigned as its end-time value), and a new (open) multiparcel is "opened" (in the sense that a new multiparcel is created with an updated vector of parcels and the current time is assigned as its begin-time value). Note that the creation of a new flow is also viewed as a flow-rate change, and if there are no prior flows in the station a singleton open multiparcel is created in the multiparcel matrix. Conversely, when a multiparcel's workload is depleted, it is discarded.

The following observations are straightforward consequences of the parceling management scheme:

1. An open multiparcel may contain fluid or may be empty, but a closed multiparcel cannot be empty.
2. A multiparcel does not mix continuous and discrete flows. It either contains a vector of continuous flows (in which case it may be either open or closed), or a singleton packet (in which case it must be closed).

3. If a priority queue of multiparcels is not empty, it consists of zero or more closed multiparcels, followed by a trailing multiparcel, which may be an open or closed fluid multiparcel, or a packet multiparcel.

4. If a priority queue is allocated a positive service rate, then necessarily all higher-priority queues (if any) consist each of a singleton open empty multiparcel.

3.3. Protocol layer

The protocol layer is designed to furnish protocol details to the generic fluid transport layer. To separate protocols from generic fluid transport, protocols are implemented by concrete classes that extend the abstract *Protocol* class. For example, an ATM flow (connection) is implemented by an object from the concrete class *ProtocolATM*.

It is important to realize that HDCF is not designed to model connectionless transport, nor to handle the details of packetization and packet assembly. Thus, packet loss can be inferred from fluid volume loss, and retransmission can be approximated by increasing the original workload. On the other hand, timer modeling is straightforward while flow control can be readily modeled by discrete (packet) messages that modulate the source's injection rate (recall that packets are not subject to topology constraints). TCP flow control and congestion control are typical examples. The details of approximating other transmission protocols (e.g., multicast routing protocols) in a fluid-flow framework is now being investigated and will be reported in a future paper.

3.4. Statistics layer

The statistics layer estimates and displays standard queueing statistics pertaining to stations and messages. Station statistics include total workload in the buffer, total loss volume at the station, fraction of time in a station state and server utilization. Message statistics include sojourn times and loss volumes in the network and by station.

A novel feature of HDCF-NS is the estimation of IPA derivatives (gradients), as explained in Section 1. We plan to implement IPA derivative of the loss volume at a station, as function of buffer size, arrival rate and service rate, using the formulas derived in.[10] Work on IPA derivatives of buffer contents is currently in progress, with a view to implementation in HDCF-NS.

4. EFFICIENCY ISSUES

Since one of the stated goals of HDCF-NS is to facilitate fluid-flow simulations which would be computationally prohibitive in the traditional queueing paradigm, it is essential that efficiency issues play a central role in its design and implementation. Observe that unlike packet-based simulations, storage complexity is not an issue in fluid-flow simulation. This is the case because the fluid-flow paradigm is specifically designed to avoid the proliferation of packet objects, and the corresponding fluid workloads are simple scalar variables. Consequently the impending discussion addresses primarily speedup issues.

4.1. Event reuse

To save execution time on event allocation, HDCF-NS strives to reuse event objects whenever possible. This device speeds up execution, because it obviates the need to create new event objects and discard old ones.

To this end, a number of event classes were identified, such that at any given (simulation) time, at most one instance of the event can exist in a specific part of the simulation. Examples of such source-related events are message arrival, injection-rate change and injection completion, while examples of station-related events include service-rate change, multiparcel service completion, and change of station state to BUSY-EMPTY or to BUSY-FULL. Accordingly, each *Source* object has a single instance of the associated event classes above as a member, and similarly for each *Station* object, all of which are created statically with their attributes when the network is read in. When scheduling these events in the event list, only the event time needs to be assigned the requisite value. Furthermore, these events can be readily located in the corresponding *Source* or *Station* object (e.g., for removal from the event list or for rescheduling), thereby obviating time-costly searches of the event list.

4.2. Event coverage

We say that simulation event A *covers* simulation event B, if the processing tasks of event A include those of event B. Thus, the concept of *event coverage* is useful for analyzing task commonality for different event classes. When such coverage can be established for two distinct event classes, it may be possible to arrange for the common tasks to be performed exactly once. For example, when the station state changes from BUSY-PARTIAL to IDLE-EMPTY, then the multiparcel service-completion event covers the event that detects an empty buffer. The HDCF-NS implementation contains code fragments that check for event-coverage conditions that save execution time.

4.3. Node order of update

Recall that a (primary) rate change at a network node triggers a sequence of (secondary) node updates that ripple downstream of that node in accordance with the network's feed-forward topology of continuous flows. The proper order of node updates is determined by the topology of the downstream portion, but is not unique. Furthermore, a transition of a packet from one node to another triggers a (primary) rate change in *both* nodes, followed by (secondary) node updates downstream *both* of them. Since the feed-forward topology permits an overlap of the two downstream portions, we should be careful not to update a downstream node more than once.

HDCF-NS takes advantage of the static topology of the simulated networks to determine and save the order of node updating as part of the preprocessing stage of a network simulation. To this end, all network nodes are completely ordered (sequenced) in compliance with the partial ordering defined by node reachability via continuous flows. Using this sequence, a (static) subsequence of node updates is precomputed for each network node and stored in the corresponding object. During a simulation run, each node experiencing a primary rate change simply uses its sequence to control the order of updates in downstream nodes, thereby saving redundant computations.

4.4. Selective Update

State updating of network objects (stations, flows, statistics, etc.) is generally computationally expensive. Consequently, deferring state updating whenever possible is desirable. The HDCF-NS implementation checks if such deferral is possible and performs only selective update of flows and statistics. To this end, these objects maintain a member that keeps track of the last update time.

5. CONCLUSION

This paper has presented the architectural outline of HDCF-NS. While the details of its design are still evolving, the layered architecture permits us to generate a broad design of each layer with the minimal interaction with the others. Currently, we are concentrating on the design and implementation of the network layer and the fluid transport layer, to be followed by the statistics layer and protocol layer. A more complete description of HDCF-NS and its modeling capabilities will be reported elsewhere.

ACKNOWLEDGMENTS

This work was supported in part by DARPA Agreement F30602-00-2-0556 and NSF Grant DMI-0085659.

REFERENCES

1. D. Anick, D. Mitra, and M.M. Sondhi, "Stochastic Theory of a Data-Handling System with Multiple Sources", *The Bell System Technical Journal* **61**, 1871–1894, 1982.
2. C.G. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associates Publishers, Irwin, Boston, MA, 1993.
3. Y.C. Ho and X.R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*, Kluwer Academic Publishers, Boston, MA, 1991.
4. G. Keisidis, A. Singh, D. Cheung, and W.W. Kwok, "Feasibility of Fluid-Driven Simulation for ATM Network", *Proc. IEEE Globecom* **3**, 2013–2017, November 1996.
5. H. Kobayashi and Q. Ren, "A Mathematical Theory for Transient Analysis of Communications Networks", *IEICE Trans. on Communications* **E75-B**, 1266–1276, 1992.
6. B. Liu, Y. Guo, J. Kurose, D. Towsley, and W.B. Gong, "Fluid Simulation of Large Scale Networks: Issues and Tradeoffs", *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 1999.
7. N. Miyoshi, "Sensitivity Estimation of the Cell-Delay in the Leaky Bucket Traffic Filter With Stationary Gradual Input", *Proc. International Workshop of Discrete Event Systems (WoDES'98)*, 190–195, Cagliari, Italy, August 26-28, 1998.
8. A. Yan and W.B. Gong, "Fluid Simulation for High Speed Networks with Flow-Based Routing", *IEEE Trans. on Information Theory* **45**, 1588-1599, 1999.
9. Y. Wardi and B. Melamed, "Continuous Flow Models: Modeling, Simulation and Continuity Properties", *Proceedings of 38-th IEEE CDC*, 34–39, Phoenix, Arizona, December 7-10, 1999.
10. Y. Wardi and B. Melamed, "Loss Volume in Continuous Flow Models: Fast Simulation and Sensitivity Analysis", *Proceedings of 8-th IEEE Mediterranean Conference on Control and Automation (MED-2000)*, Patras, Greece, July 17-19, 2000.
11. Y. Wardi and B. Melamed, "Variational Bounds and Sensitivity Analysis of Continuous Flow Models", *J. of Discrete Event Dynamic Systems*, 2001. To appear.
12. M. Yoo, C. Qiao and S. Dixit, "Optical Burst Switching for Service Differentiation in the Next-Generation Optical Internet", *IEEE Communications* **39:2**, 98–104, 2001.