Simulation of Network Traffic at Coarse Time-scales

David M. Nicol, Guanhua Yan
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign

Abstract

Simulation of large-scale networks demands that we model some flows at coarser time-scales than others, simply to keep the execution cost manageable. This paper studies a method for periodically computing traffic at a time-scale larger than that typically used for detailed packet simulations. Applications of this technique include computation of background flows (against which detailed foreground flows are simulated), and simulation of worm propagation in the Internet. Our approach considers aggregated traffic between Internet Points of Presence, and computes the throughput of each POP-to-POP flow through each router on its path. This problem formulation leads to a non-linear system of equations. We develop means of reducing this system to a smaller set of equations, which are solved using fixed point iteration. We study the convergence behavior, as a function of traffic load, on topologies based on Internet backbone networks. We find that the problem reduction method is very effective, and that convergence is achieved rapidly. We also examine the comparative speedup of the method relative to using pure packet simulation for background flows, and observe speedups of exceeding 5000 using an ordinary PC. We also simulate foreground flows interacting with background flows, and compare the foreground behavior using our solution with that of pure packet flows. We find that these flows behave accurately enough in our approach to justify use of the technique in our motivating application.

1. Overview

Simulation of large-scale wireline networks has many applications. In some of these only a small fraction of traffic is of specific interest, e.g., the behavior of flows managed by a new transport protocol, or control traffic between BGP routers. However, the flows of interest are affected by, and may even (to a lesser degree) affect the other “background” flows. There is strong motivation to model background flows with less detail than foreground flows, with the objective of significantly lowering the computational requirements. Control traffic such as BGP announcements and DNS queries need to be modeled at the packet level, yet the network being simulated has hundreds of thousands of devices and (typically) tens of thousands of flows represented at any given time. The only hope we have of meeting real-time constraints is to significantly aggregate our treatment of background traffic. In other applications the traffic of interest can be modeled at a coarse time scale. For example, simulation of worm spread across the Internet can be usefully modeled at a time scale much larger than is usually used in network simulation, and the volume of traffic is so large that this becomes necessary.

Our problem is this : given a description of flow intensities at ingress points, efficiently determine link loads throughout the network interior, and determine flow intensities at egress points. Solution to this problem allows us to represent these flows to a more detailed traffic mode in terms of the demand they make on shared bandwidth (and potentially, buffer space). These demands are periodically recomputed, e.g., every 5 seconds. On large networks this method will compute bandwidth consumption significantly faster than will packet representation of the same flows.

These savings naturally come with a cost. Low resolution background flows may be less responsive to changes in network state, they may exhibit less burstiness, the degree to which a simulation using them differs from one with all flows at the same resolution will be unknown. Nevertheless there are contexts where the trade-offs in favor of computational speed are acceptable; indeed there are contexts where one really has no other option but to use highly abstracted representation of background traffic. Our research group is developing a network simulator that will be used in cyber-defense training exercises where the most important accuracy requirements are that the simulated behavior have a realistic “look and feel”. The simulator computes the detailed effects on particular traffic flows (e.g. financial transactions) as cyber-attacks occur, and as defensive measures are taken. Two defining characteristics are that its application demands real-time performance (e.g., one second of simulation time takes no more than one second of wallclock time to advance), and that its application involves networks with potentially hundreds of thousands of devices.

The simulation problem becomes non-trivial when we attempt to capture the effects of bandwidth sharing among flows across a common link. Our specific formulation targets time-scales larger than end-to-end latency, hence one assumes that flows pass instantaneously through the network. This leads to a system of non-linear equations whose solution gives the desired background flows. We propose a fixed point algorithm for its solution.

We empirically analyze this technique on models of real Internet backbone networks, with synthetic traffic generated using Pareto-Poisson Burst Processes (PPBP) [27]. We examine convergence behavior as a function of traffic load, the speedup it offers, and the accuracy of foreground traffic when it is used in place of packet simulation. Our experiments demonstrate the viability of the approach for our motivating application.

2. Related Work

Our problem is related to the area of network tomography, that likewise seeks to determine some network characteristics (e.g. the traffic matrix—volume of flows between any pair of ingress and egress points) from other traffic measurements (e.g. measured link loads on the network edge). There is a large and fascinating literature on the topic, e.g., see [25, 24, 7, 22, 20, 4, 1, 5]. A key difference between our problem and network tomography is that the latter is driven by difficulties in obtaining measurements of certain important quantities, and so works to use models to infer them from measurements that are available. In the virtual world of simulation we can measure anything we like; our problem is to compute those measurements given the offered load.

Our approach includes ideas found elsewhere. The notion of simulating traffic using simple rate functions was introduced almost ten years ago [8], a formulation that observes that FCFS service at a congested router port can be modeled by scaling the flows in portion to their input rates, which is a central facet of our model. Further development is found in [14]; application to TCP is developed in [16], which treats issues in simultaneous simulation of packet and fluid flows, as do [9], [26], and [15]. Specific application to the global Internet and practical problems therein are discussed in [2].

More detailed fluid models found in the literature include treatments of TCP [17], models containing stochastic elements [13], a time-stepped model [23], and a discussion of trade-offs [12, 11]. Use of fixed point iteration to solve for network measures of interest also appears in [3], a model that focuses much more on TCP and less on link loads. A very abstract rate-based model is discussed in [21], work that does not try to capture interflow dependencies in detail, as we do.

Against this backdrop our contribution is the development, optimized solution, and empirical study of a coarse-grained traffic model that focuses on the impact that bandwidth sharing on congested links has on flow rates throughout the entire network. Our fixed point approach is unique in that each approximated solution has internal consistency that can, in some contexts, allow it to be used even if the solution has not yet converged. The approach is a valuable and necessary component of our real-time cyber-defense training simulator.

3. Model

We are interested in how a network shapes coarse-grained traffic flows between many network points of presence (POPs). We assume that there are n of these, denoted by $P_1, P_2, \ldots, P_n$. Traffic is described by processes $\{T_{ij}(t)\}$, $(1 \leq i,j \leq n)$ that give the rate (bits per second) at which POP T_i injects traffic destined for POP P_j into the network at time t. We discretize time into units of length Δ seconds, and define time $t_k = k \cdot \Delta$, $k = 0,1,2,\ldots$. We make very few assumptions about how $\{T_{ij}(t)\}$ is defined, other than at time t_k the behavior of $P_{ij}(t)$ with $t \in [t_k, t_{k+1}]$ can be predicted. We may use this formalism to reflect an underlying random process, a recorded traffic trace, or even real-time measurements. We will work with "smoothed" values of $\{T_{ij}(t)\}$, as follows. Define $R_{ij}(t_k) = (1/\Delta)\int_{t_k}^{t_{k+1}} T_{ij}(t)\ dt$, and observe that the process that injects traffic at constant rate $R_{ij}(t_k)$ over $[t_k, t_{k+1}]$ injects exactly the same amount of traffic as does $T_{ij}(t)$ over that same period. We will use $\{R_{ij}(t)\}$ to describe injected traffic, in order to simplify the computation. Doing so we may dampen burstiness at scales smaller than Δ, but we will inject the same amount of traffic as the original process.

We model the network itself in terms of routers connected by uni-directional links, understanding that we can model a bi-directional link as a pair of uni-directional ones. The sending endpoint of a link is associated with a router's output port, and associated with each output port is a buffer size. Routing protocols establish forwarding tables that indicate for each router the output port to use for any particular POP destination. We assume that routing decisions are unaltered during a discretized epoch of length Δ. Link latencies are effectively assumed to be zero, which is an appropriate modeling assumption when Δ is large compared to the typical end-to-end latency of a flow.

At a given instant t_k we can examine the sum of the arrival rates to a router's port. Congestion and traffic loss may occur. We modified a congestion model commonly used in fluid traffic simulations (e.g. [8, 14]) for our time-stepped formulation, as follows. Suppose that at time t_k there are m flows mapped to a given port, with arrival rates $\psi_1^{(in)}(t_k), \psi_2^{(in)}(t_k), \ldots, \psi_m^{(in)}(t_k)$, and let $\Psi(t_k) = \sum_{i=1}^{m} \psi_i^{(in)}(t_k)$ be their aggregate arrival rate. Let μ denote the link bandwidth. The rate of the i^{th} flow *out* of the port (denoted $\psi_i^{(out)}(t_k)$) is defined in terms of the relationships between $\Psi(t_k)$ and μ:

$$\begin{aligned}\psi_i^{(out)}(t_k) &= \begin{cases} \psi_i^{(in)}(t_k) \text{ for } \Psi(t_k) \leq \mu \\ \psi_i^{(in)}(t_k)\Big(\mu/\Psi(t_k)\Big) \text{ otherwise} \end{cases} \\ &= \psi_i^{(in)}(t_k) \times \min\{1, \mu/\Psi(t_k)\}. \qquad (1)\end{aligned}$$

In the first case the port can completely serve all of the incoming flow; in the second case every flow's output bandwidth is scaled back by a constant factor, a device that models effect of FCFS service [8].

Equation (1) describes what the simulation needs to do in order to compute a link's utilization over the epoch $[t_k, t_{k+1}]$ as a function of the input flow rates to it at time t_k. However, we will see that this is a deceptively simple description; for, depending on the value of Δ, the flow rates into the port at time t_k may depend on the flow rates *out* of it at time t_k.

4. Algorithm

4.1. Time-step setup

The starting premise of our approach is that we can compute resource consumption of certain types of flows relatively infrequently, and treat that consumption as invariant between updates. These flows may be made sensitive to flows modeled with higher resolution, but only at the fixed update points. Upon reaching an update point we have the opportunity to re-assess the bandwidth allocated on each link. These decisions may be made as a function of observed past behavior. We also have the opportunity to adjust the offered load rate, e.g. , reduce it on flows for which loss has been observed in the last time, in order to model feedback (such as TCP provides). These issues are important, but are not the focal point of the present paper.

When we use this formulation to compute background traffic intensities it is important to capture the competition between foreground and background flows for link bandwidth. We use a simple mechanism for including the next epoch's anticipated foreground flow as we compute flow rates for background flows. At time t_k, for each port p_z, we compute an estimated foreground input rate $\lambda_f^{(in)}(p_z)$ based on recent observations (or even known predictions, if such were available). We model foreground flow passing through the port as though it is injected into the network at this point from a traffic source, crosses the link, and immediately disappears into a traffic sink. As we compute new flow rates, for that port we treat that flow exactly as any other flow. The "fair" contribution of foreground traffic is thus considered as we allocate bandwidth for the background traffic.

Since each flow is recomputed each time-step, we also have the opportunity to alter routing. This is particularly useful in applications where routing changes in response to changes in traffic loads, or where we simulate attacks on the routing infrastructure by disabling routers, eventually causing routing protocols to respond and create new routes. We assume that the routing for the next time step is known before we compute the flow updates.

4.2. Dependency Reduction

It is helpful to imagine the flow update computation in terms of the status of input flow variables. We describe the set of all flow variables at time t_k (including the ingress flow rates) by a vector $\Lambda(t_k)$. At any stage in flow update processing an element of this vector is either *settled* or *unsettled*. We say that a port is *resolved* once all of its input flows are settled, because then Equation (1) specifies its output rates. We say that the port is *transparent* if it is not resolved, but analysis shows that the sum of input rates cannot exceed the port's bandwidth; for such ports every flow's output rate is identical to its input rate. Our aim is to resolve as many flow variables as possible, in a sort of data flow analysis phase. Once finished we necessarily have circular dependencies among some flow variables, and another form of processing is needed. A key component to the flow resolution analysis is the computation and propagation of settled flow values—a settled flow value into a transparent port establishes a settled output value (and hence settled input to the next router in the path). Thus identification of transparent ports is important. Another key component is the observation that we sometimes may determine that a port is transparent if we know *upper bounds* on some of its inputs' as-yet-unsettled variables. A settled flow rate into a transparent but as-yet-unresolvable port produces a settled output rate (which in turn may be used to help make another port either transparent or resolvable.)

We describe the logic of this step in terms of production rules, such as one uses to build expert systems. A rule has a logical expression known as a precondition, and a set of actions to apply whenever the precondition is satisfied. Applied here, the precondition is a Boolean expression about settled/unsettled states of flow variables and the resolvable/transparent state of ports. The post conditions set flow variable values, and change the state of flow variables and ports.

A precise description of the rule set requires notation. We drop the notational dependence on t_k and use λ_i to be the

	Pre-condition	Action
1.	$\sim (\mathcal{S}[p] = \textit{resolved}) \wedge$ $\forall i \in I(p)\ \{\mathcal{S}[\lambda_i] = \textit{settled}\}$	$\mathcal{S}[p] \leftarrow \textit{resolved},$ $\forall i \in I(p)\ \{\mathcal{S}[\lambda_{n_p(i)}] \leftarrow \textit{settled},$ $\lambda_{n_p(i)} \leftarrow \lambda_i \times \min\{1, \mu_p / \sum_{j \in I(p)} \lambda_j\}\}$
2.	$\mathcal{S}[p] = \textit{unresolved} \ \wedge\ \sum_{i \in I(p)} \lambda_i \leq \mu_p \wedge$ $\exists\ i \in I(p)\ s.t.\ \mathcal{S}[\lambda_i] = \textit{bounded}$	$\mathcal{S}[p] \leftarrow \textit{transparent},$ $\forall i \in I(p)\ s.t.\ \mathcal{S}[\lambda_i] = \textit{settled}\ \{\mathcal{S}(\lambda_{n_p(i)}) \leftarrow \textit{settled}\}$
3.	$\mathcal{S}[p] = \textit{transparent} \ \wedge\ \exists i \in I(p)\ s.t.$ $(\mathcal{S}[\lambda_i] = \textit{settled}) \wedge\ \sim (\mathcal{S}[\lambda_{n_p(i)}] = \textit{settled})$	$\mathcal{S}[\lambda_{n_p(i)}] \leftarrow \textit{settled},\ \lambda_{n_p(i)} \leftarrow \lambda_i$
4.	$\phi(p) \geq \mu_p \ \wedge\ \exists i \in I(p)\ s.t.$ $\lambda_{n_p(i)} > (\mu_p/\phi(p))\lambda_i$	$\lambda_{n_p(i)} \leftarrow (\mu_p/\phi(p))\lambda_i$
5.	$\exists i \in I(p)\ s.t.\ \lambda_i < \lambda_{n_p(i)}$	$\lambda_{n_p(i)} \leftarrow \lambda_i$

Table 1. Transition Rules for Dependency Reduction

i^{th} flow variable in vector Λ. The state of λ_i, denoted $\mathcal{S}[\lambda_i]$ is either *resolved*, or *bounded*. In the *settled* state λ_i has its final value, and in the *bounded* state we define λ_i to be the bound. As we will see, at initialization every flow variable is placed into one of these two states.

For every port p we denote the state of p by $\mathcal{S}[p]$, defined to be one of *resolved*, *transparent*, or *unresolved*. We also define $I(p)$ to be the set of indices such that if $i \in I(p)$, then λ_i is an input flow variable for p; we similarly define $O(p)$ as the indices of flow variables defined by flows leaving p. For every λ_i we let $f(i)$ denote the logical end-to-end flow associated with λ_i, and note that for every $i \in I(p)$ there exists a unique $j \in O(p)$ such that $f(i) = f(j)$; this j is denoted $n_p(i)$. μ_p denotes the port's bandwidth. We define $\phi(p)$ to be the sum of rates of all settled flows into p. This value necessarily changes as flows into p become settled in the course of the computation.

At initialization the flow variables describing network inputs are given state *settled* and their values set. Every other flow variable is put in the *bounded* state, and given the ingress value of its associated end-to-end flow. Every port is placed in the *unresolved* state. Table 1 gives five rules that are applied repeatedly, until no further pre-conditions are satified. Rule 1 says that if a port is not in the resolved state but all of its input flows are settled, then the port becomes resolved and all of its output flows become settled. Rule 2 says that if by summing the upper bounds on input flow values to an unresolved port we can determine that it will not be congested, then the port becomes transparent. Rule 3 says the if there is a settled input flow to a transparent port whose corresponding output flow is not settled, then that output flow becomes settled and takes the value of the input flow. Rule 4 observes that if the total volume of known settled flows into p exceeds the bandwidth, then the port will definitely be congested and the scale factor by which all input flows will be multiplied can be upper-bounded; thus application of that scale factor to either a settled flow or an upper bound on a flow produces an upper bound on the output flow value. Rule 4 reduces any output flow upper bound that can be so reduced. Finally, Rule 5 says that if the value (or bound) on an input flow to any port is less than the value or bound assigned to its corresponding output flow, then we can lower the value assigned to the output flow. Rules 4 and 5 are powerful in that they may let us push bounding information past an unresolved port. We note in passing that Rule 4 is the only rule in this set that depends fundamentally on the assumed FCFS service. Leaving out Rule 4, we can use this technique other disciplines, such as weighted fair queueing.

In the evaluation of a rule set such as this, at any time there may be a number of ports for which rule pre-conditions are satisfied. Prioritizing selection of which rules to fire is an important component of the solution. Rule 1 should always have highest priority, as the object is to resolve flow variables as quickly as possible. Next most important is Rule 2, for making a port transparent allows propagation of settled values through the port, potentially enabling a Rule 1 firing. Rule 3 has next priority, for the same reason. Rule 4 is next, for by decreasing upper bounds on output flows we may discover that a port is transparent. Rule 5 has lowest priority; Rule 4's lower bounds are sharper (hence it has priority when it applies), but Rule 5 nevertheless may help to identify the transparency of a port.

Efficient rule processing is possible. Our implementation visits each flow variable a small (constant) number of times. There is no ordering, sorting, or searching involved. The computational complexity is thus proportional to the number of flows times the average number of ports through which a flow passes—essentially linear in the size of the problem.

Figure 1 illustrates these concepts with an example topology, and description of the Rules processing that lead to each flow variable becoming settled.

4.3. Circular Dependencies

Unlike the example of Figure 1, it may happen that after dependency reduction is completed some input flow variables remain unsettled. When this occurs, there is a cycle of unresolved ports in the system graph. One small change in this example illustrates the point. If $\lambda_1 + \lambda_9 > \mu$ then p2 does not become transparent by Rule 2; the network is left with a cycle p2 $\rightarrow$ p4 $\rightarrow$ p6 $\rightarrow$ p2 of unresolved ports.

The next step focuses on solving flow variables on such cycles. The variables involved may be a subset of the variables still unsettled. In our running example we have to settle λ_2, λ_6, and λ_{10} on the loop before we can settle all the other unsettled variables. Fortunately it is easy to identify the cycles and restrict our attention to them. The network is a directed graph; the process of resolving ports and settling variables can be thought of as a kind of graph reduction where we remove nodes corresponding to ports, as they are resolved. After dependency reduction we can find cycles by identifying nodes that have out-degree 0 (egress ports), remove them and any edges directed to them. If the edge removal causes another node to have out-degree 0 it too is removed with all its in-edges, and so on. By this process we remove non-cyclical dependents of the cycles whose values we must compute. Once the variables on the cycles are determined we can compute the flow variables for the rest of the system.

We will say that a flow variable is *irreducible* if it is an output of a port whose node remains in the reduced graph. It may happen that the reduced graph has more than one connected component. In the reminder we describe what to do with a given connected component, understanding that different connected components can be solved for independently.

For the sake of notational simplicity, we suppose there are M irreducible flow variables, renamed as $\lambda_1, \lambda_2, \ldots, \lambda_M$; let Λ be the M-vector of these values. Every irreducible flow variable is the output from a port represented in the reduced dependency graph, and is thus expressed by Equation (1) as a function of the port's input flow rates—at least one of which must be irreducible. The port's non-irreducible input flows are known, having been computed either in a previous time-step, during the graph reduction, or as the result of computing flows from a port that will serve only backlogged traffic in the next epoch. From the point of view of Equation (1) applied at this point in the computation, non-irreducible flow variables are constants. Let $f_i(\Lambda)$ be the equation expressing λ_i, let $F_i(\Lambda) = \lambda_i - f_i(\Lambda)$, and let $F(\Lambda)$ express the system of M equations, the i^{th} such being $F_i(\Lambda)$. Our problem is to find a solution to $F(\Lambda) = 0$.

This system of equations is non-linear, because the component equations $f_i(\Lambda)$ are not linear combinations of the variables. Our equations are more complex because the flows from a congested port have a decidedly non-linear response to changes in input flows.

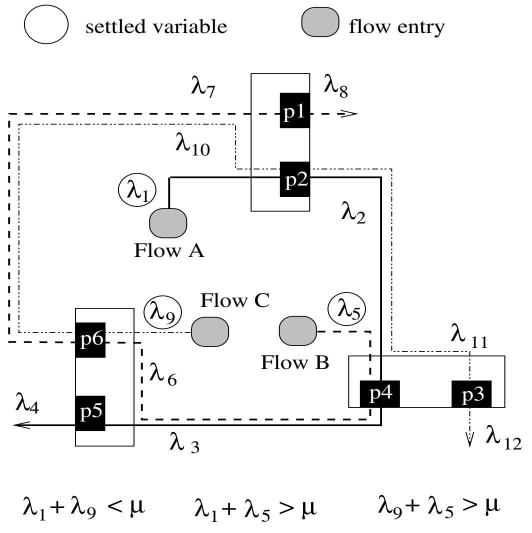
The general problem of solving non-linear systems is known to be very hard. A non-linear system may have between 0 and many solutions, depending on the problem specifics. Solution approaches to non-linear systems are typically iterative, meaning that given an estimated solution Λ_k, one creates another estimate Λ_{k+1} that (hopefully) is closer to a true solution. Typically one iterates until some norm of the difference between successive iterations $||\Lambda_{k+1} - \Lambda_k||$ is less than a tolerance ϵ. Any given iterative algorithm may or may not converge to a solution, depending on the starting point Λ_0.

Some forms of iteration create Λ_{n+1} from Λ_n using derivative information. The widely known Newton-Raphson method [18] computes

$$\Lambda_{n+1} = \Lambda_n - J^{-1}(\Lambda_n)F(\Lambda_n)$$

where $J^{-1}(\Lambda_n)$ is the inverse of F's Jacobian evaluated at Λ_n, and $F(\Lambda_n)$ is the vector resulting from applying Λ_n to each equation F_i. Recall that the ij^{th} entry of F's Jacobi matrix is the value of the partial derivative of F_i with respect to λ_j. Newton's method converges fast on sufficiently well-behaved systems, using sufficiently close starting approximations Λ_0. It has the computational drawback of requiring, *at each iteration*, the inversion of a large dimension sparse matrix. That in itself is a significant computational challenge. It also leaves us with the issue of what traffic matrix to use if it should happen that the Jacobian cannot be inverted or the iterations diverge.

Equation (1) itself suggests an iterative method. We define the first approximation Λ_0 by computing the output of each port using as input a flow's ingress flow rate; the ap-



Topology

Step	Rule	Processing
1.	2	$\mathcal{S}[p2] \leftarrow \mathit{transparent}, \mathcal{S}[\lambda_2] \leftarrow \mathit{settled}$
2.	1	$\mathcal{S}[p4] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_6] \leftarrow \mathit{settled},$ $\mathcal{S}[\lambda_3] \leftarrow \mathit{settled}$
3.	1	$\mathcal{S}[p5] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_4] \leftarrow \mathit{settled}$
4.	1	$\mathcal{S}[p6] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_7] \leftarrow \mathit{settled},$ $\mathcal{S}[\lambda_{10}] \leftarrow \mathit{settled}$
5.	1	$\mathcal{S}[p1] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_8] \leftarrow \mathit{settled}$
6.	1	$\mathcal{S}[p2] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_{11}] \leftarrow \mathit{settled}$
7.	1	$\mathcal{S}[p3] \leftarrow \mathit{resolved}, \mathcal{S}[\lambda_{12}] \leftarrow \mathit{settled}$

Dependency reduction processing

Figure 1. Example Topology, and Sequence of Rules Applied

proximation is good if there is relatively little loss. Given approximate solution Λ_n, we compute

$$\Lambda_{n+1} = f(\Lambda_n) = (f_1(\Lambda_n), f_2(\Lambda_n), \ldots, f_m(\Lambda_n)).$$

This formulation is also known as a *fixed-point* computation. The intuition is that if we hold the flow rates into the network constant, over time the flows across each link should stabilize (although as yet we have no formal proof of this). If this were the case then the suggested iterative method is equivalent to pushing the network state along in time until it settles. The converged state is the "fixed-point", in the sense that reapplying the traffic shaping rules expressed by Equation (1) does not alter the flow rates. Naturally, a very significant question to ask is whether the fixed point computation will converge to some solution $\Lambda^* = f(\Lambda^*)$. The literature identifies a general condition under which convergence may be proven. Function $f : \mathbb{R}^n \rightarrow \mathbb{R}^M$ is said to be a *contraction mapping* if there exists $0 < q < 1$ such that for all $\Lambda_1, \Lambda_2 \in \mathbb{R}^M$ and norm $||x||$

$$||f(\Lambda_1) - f(\Lambda_2)|| < q||\Lambda_1 - \Lambda_2||.$$

The Banach fixed point theorem [6] states that there exists a unique fixed point Λ^*, e.g., $\Lambda^* = f(\Lambda^*)$, and that Λ^* may be found by choosing arbitrary Λ_0, and iterate as $\Lambda_n = f(\Lambda_{n-1})$. This sequence converges to the limit Λ^*.

Formally proving convergence, or circumstances under which convergence is achieved remains an important goal of our research. It is a challenging goal though, as it seems that to achieve it we will have to use specific properties of our problem domain. In particular, it seems that the contraction parameter q is dependent on the network being considered, which makes discovery of a general result for our application domain all the more challenging. Nevertheless, Banach's fixed point theorem gives the foundation for our approach. Our experimental section will directly study how convergence behaves on certain motivating networks.

We are in the uncomfortable position of lacking assurance of convergence by Newton's method or the fixed point method suggested by Equation (1). If it should happen that we observe *divergence* after some number of iterations, we have still to construct some representation of background traffic. In our training application context we can justify using a traffic matrix that doesn't satisfy Equation (1), but is in some weaker sense "plausible". Unfortunately intermediate steps of Newton's method may create an estimate Λ_n that is physically unrealizable, in the sense that it may assign outflow rates to a link such that their sum exceeds the link capacity, or in the sense that volume of a given flow that leaves the system exceeds the sum of what is entering plus what was buffered. These things can happen because the Newton method computes the next solution by following a linear projection out from the old solution. No constraints are placed on the solution that is so computed. Thus, if we should find that J is not invertible at some iteration, or that the iterations are diverging, we are left with the problem of using *something* reasonable for the traffic loads, and the intermediate solution Λ_n may not be reasonable.

The fixed point algorithm is a more attractive option with respect to this problem because respect for system constraints is built into Equation (1). We formalize this respect by the definition of a plausible approximation.

Definition 1 *Approximation* Λ_n *is* plausible *if*

1. *every flow rate expressed by Λ_n is non-negative,*
2. *across every link the sum of flow rates assigned by Λ_n to that link does not exceed the link bandwidth,*
3. *for every pair of POPs P_i and P_j, between times t_k and t_{k+1} the volume of P_i-to-P_j traffic leaving the subnetwork represented by the reduced graph is no larger than the sum of the volume of P_i-to-P_j traffic entering the subnetwork, plus the P_i-to-P_j traffic that is buffered along its path through the subnetwork.*

The fixed-point method suggested by Equation (1) ensures plausibility.

Theorem 1 *Let Λ_n be any approximation with non-negative flow values. Then*
$\Lambda_{n+1} = (f_1(\Lambda_n), f_2(\Lambda_n), \ldots, f_m(\Lambda_n))$ *is a plausible approximation.*

Proof: Given non-negative values for flow variables, it is not possible for Equation (1) to create negative values, thus $f_i(\Lambda_n) \geq 0$ for all i and the first condition for plausibility is satisfied. By construction it is not possible for Equation (1) to assign flows across a link that exceed the link's allocated capacity for background traffic, so the second condition for plausibility is also satisfied. To establish the third condition, choose any flow P_i-to-P_j that passes through the subnetwork, and consider the sequence of ports $p_1, p_2, \ldots, p_z$ it follows through the subnetwork. Now in Equation (1) the value of the expression used in Case 1 is always larger than the value used in Case 2. Therefore an upper bound on the flow value associated with the flow leaving the subnetwork is obtained by assuming that the case 1 expression is used at each port along the path. That upper bound is simply the sum of the rate at which the P_i-to-P_j flow enters the subnetwork, plus $(1/\Delta)$ times the sum of P_i-to-P_j traffic buffered along the path. This establishes the third condition for plausibility. $\Box$

Our experience has yet to show a case where convergence did not occur; future work will look at firming up the theoretical basis for the convergence we observe.

5. Experimental Results

We now consider the results of experiments designed to examine the resource requirements of the algorithm, on four realistic topologies derived from the Internet, and flows across them. Their characteristics are given by the table below:

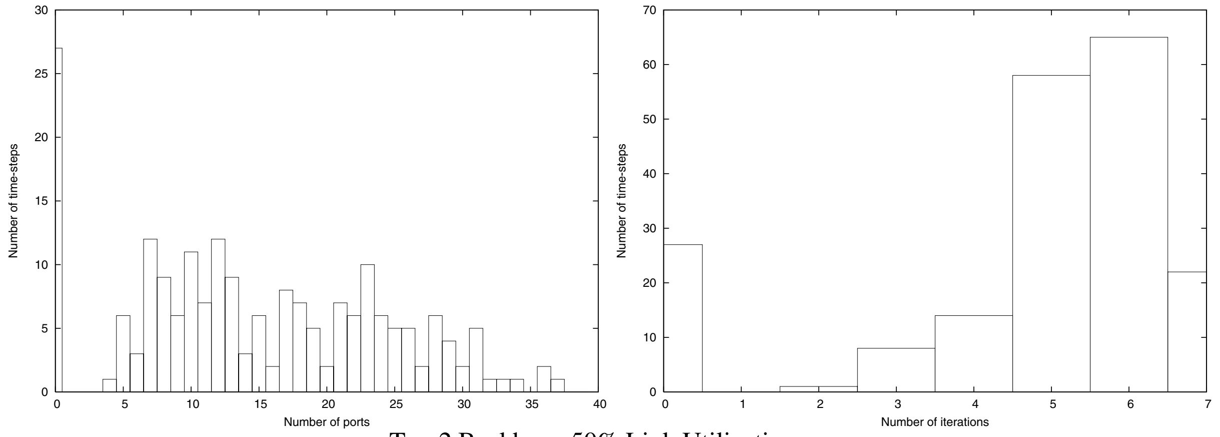
Topology	# routers	# links	# flows	Mbps
Top-1	27	88	702	100
Top-2	244	1080	12200	2488
Top-3	610	3012	61000	2488
Top-4	1126	6238	168900	2488

Top-1 is the a model of the ATT USA backbone, described in [10]. Each node is assumed to be a POP. We create a flow between every possible of the $27 * 26 = 702$ POPs. The link bandwidth (100Mbps) is artificially low to enable us to make a direct comparison of accuracy and performance with equivalent packet-based flows. Top-2, Top-3, and Top-4 are derived from topologies from the Rocketfuel [19] database. Top-2 is the Exodus backbone; every POP directs a flow to 50 others. Top-3 is Top-2 augmented through some peering points with the Above.Net backbone; and Top-4 is Top-3 augmented through some peering points with the Sprint backbone. Shortest-path routing is assumed. In Top-2, Top-3 and Top-4 each POP directs 50 flows to POPs in its own backbone, and 50 to each other backbone in the topology. Thus a Top-2 POP sends 50 flows, a Top-3 POP sends 100 flows, and a Top-4 POP sends 150 flows. Each flow is derived from a Poisson-Pareto-Burst-Process (PPBP) [27]. A Poisson process generates arrivals, each of which contributes a random traffic volume that is sampled from a Pareto distribution. The volumes are additive; one can describe a PPBP at time t in terms of the number of bursts that are active at time t. The PPBP is recognized as a good model of traffic arrival to a backbone. There are three parameters to a PPBP. In our experiments we choose them to create desired average link utilizations, and to a lesser degree, variance. The Hurst parameter for the Pareto distribution is 0.8.

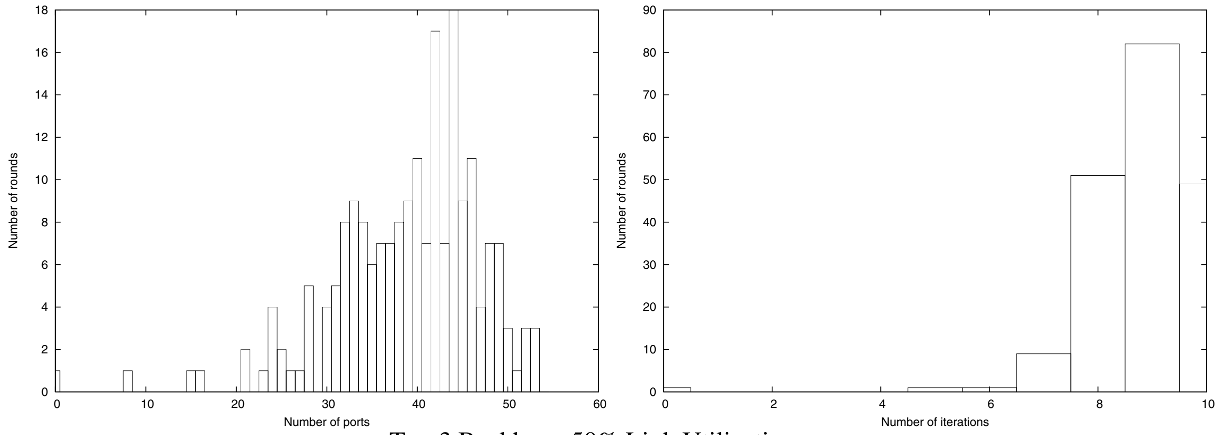
5.1. Convergence Behavior

We first consider how the algorithm behaves in terms of the number of ports on cycles analyzed by the fixed point step, and the number of iterations required to achieve convergence. In all of these experiments we consider a solution to be converged when the maximum relative difference between successive approximations to any flow variable is 0.001, e.g., when $|\lambda_{n+1} - \lambda_n|/\lambda_n \leq 0.001$ for all flow variables λ.

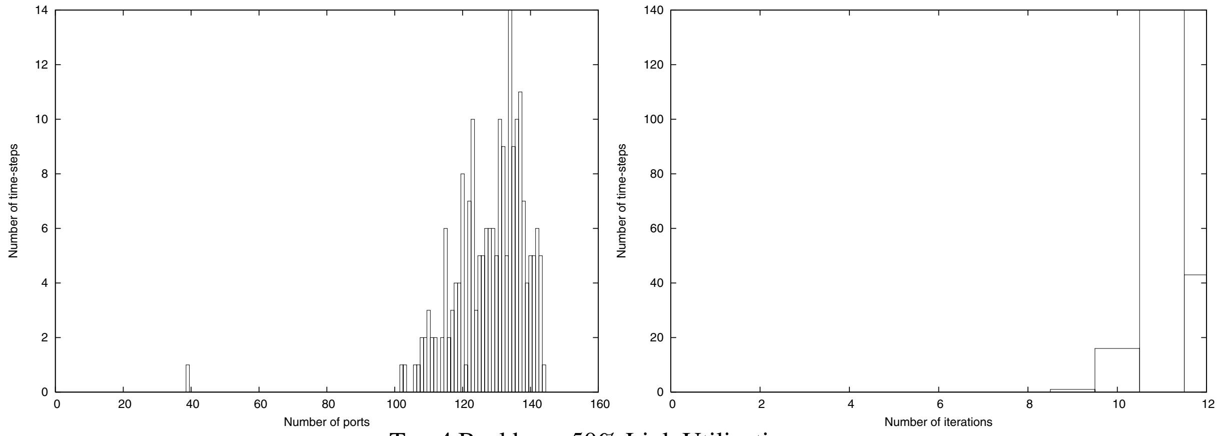
Figure 2 gives histograms of the number of ports on cycles in a time-step, and the number of iterations used on a time-step, on single runs of 200 time-steps. A time-step spans 5 simulation seconds. Top-2, Top-3, and Top-4 topologies are represented. The Top-1 topology very rarely formed cycles, even at high utilitization. In the Top-2 and Top-4 experiments the traffic parameters are selected to create close to 50% link utilization. At these loads, and under these traffic patterns, to a first approximation the number of ports in dependency cycles is proportional to the square root of the number of ports, and the number of iterations is proportional to the square root of the number of ports on cycles. The same measurements at 10% link utilization confirm the intuition that cycles form very infrequently at low utilization.



Top-2 Backbone 50% Link Utilization



Top-3 Backbone 50% Link Utilization



Top-4 Backbone 50% Link Utilization

Figure 2. Histogram of Ports on Cycles, and Iterations per Time-step, for Top-2, Top-3, and Top-4

5.2. Execution Time

Execution speed is a crucial aspect for us, because of our need to execute large network simulations in real-time. The table below gives the average time per time-step for these four topologies, at 20% and 50% link utilizations. The experiments were run using our high-performance simulation engine, on a 1.5GHz PC with 3Gb of memory, taking averages over 10 independent runs. For a given traffic load and topology, variation in run time was quite small.

Topology	secs/time-step (20% link util.)	secs/time-step (50% link util.)
Top-1	0.0026	0.0026
Top-2	0.051	0.051
Top-3	0.283	0.285
Top-4	0.852	0.907

Since a time-step takes 5 seconds, all of these experiments run much faster than real-time on a laptop scale machine. Observe that Top-4 at 50% utilization represents traffic movement equivalent to 970 million packet per second (assuming 1000 byte packets). To move this much traffic in real time on a packet-oriented parallel simulation engine that could scalably deliver 1 million simulated packets per second per CPU would require a supercomputer with 970 CPUs.

We did experiments on Top-1 computing the real speedup delivered by the coarse-grained algorithm against the equivalent packet-oriented flows, using our framework's simulation engine. The table below gives the results, as a function of link utilization. Each data point represents 10 independent experiments, variance was extremely small. Speedup initially increases with link utilization because the amount of work the packet simulator must do increases linearly, while the coarse grained solution's work remains comparatively stable. At higher link utilization we do see the effects though of more cycle resolution, which is more costly.

link util.	speedup	link util.	speedup
10	213	50	3436
20	1665	60	3725
30	2112	70	1023
40	2728	80	1135

5.3. Accuracy

The applications that motivate this work do not require close accuracy. Nevertheless, we ought to develop some sense, if we can, of the impact that using such a coarse-grained flow representation has on foreground traffic. In these experiments the foreground traffic consists of both TCP flows and UDP flows. We attach an end host to each router; the host has a TCP server, TCP client, UDP server, and UDP client. Each client's behavior is modeled by an on-off process. In the on phase the client chooses a server uniformly at random from among the other hosts in the network, and requests a 5Mb data transfer. The on phase ends when the transfer is completed, at which point the client is idle for 5 simulation seconds, and then repeats. On the server side, the inter-packet transmission time is 10 milliseconds. We ran the same foreground traffic against background traffic generated by packet processes, and against background traffic generated by our coarse flow approach. In the coarse grained approach the foreground traffic "sees" the effect of background traffic by a diminished available link capacity.

TCP behavior is very sensitive to random fluctuations in flows, because a single packet loss triggers a dramatic change in TCP sending behavior. We observed a great deal of variation in many TCP metrics across all our experiments. We have studied relative error in the number of bytes successfully delivered (e.g., the average goodput) for both UDP and TCP flows . While we hesitate to ascribe quantifiable statistical meaning to these results, the general trends are

- UDP traffic is relatively insensitive to the differences in background traffic generation
- TCP behavior is relatively insensitive when link utilization is low (because then its behavior is less driven by packet loss). With higher utilization more differences are observable. Indeed the largest differences appear when TCP behavior is most variant—in the middle range of low loss and high loss. Averaged over all the experiments we did, the relative error in goodput is 17%.

Evidence from this study is that the accuracy of foreground flow behavior against coarse-grained background traffic will be quite adequate for the training application of interest.

6. Conclusions

This paper proposes an approach for simulating network traffic flows at a coarse scale. The approach is motivated by an application where we must deliver real-time performance, and "look-and-feel" accuracy. We develop an optimized means of performing the simulation, and study its behavior on a set of large topologies based on known Internet backbones. We find that it is fast—very very fast relative to packet-based flows—and accurate enough for our purposes.

Acknowledgements

This research was supported in part by DARPA Contract N66001-96-C-8530, and NSF Grant CCR-0209144. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. In addition this project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security.

References

[1] A. Adams, T. Bu, R. Cáceres, N. Duffield, T. Friedman, J. Horowitz, F.L. Presti, S. Moon, V. Paxson, and D. Towsley. The use of end-to-end multicast measurement for characterizing internal network behavior. *IEEE Communications Magazine*, May 2000.

[2] C. Barakat, P. Thiran, G. Iannaccone, C. Diot, and P. Owezarski. A flow-based model for internet backbone traffic. In *Internet Measurement Workshop'02*, San Francisco, CA, November 2002.

[3] T. Bu and D. Towsley. Fixed point approximations for TCP behavior in an AQM network. In *Proceedings of ACM SIGMETRICS 2001*, Cambridge, Massachusetts, June 2001.

[4] J. Cao, D. Davis, S.V. Wiel, and B. Yu. Time-varying network tomography. *J. Americal Statistical Association*, 95:1063–1075, 2000.

[5] M. Coates, A. Hero, R. Nowak, and B. Yu. Internet tomography. *IEEE Signal Processing Magazine*, May 2002.

[6] L. Debnath and P. Mikusinksi. *Introduction to Hilbert Spaces with Applications*. Academic Press, San Diego, CA, 1990.

[7] Anja Feldmann, Albert G. Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational IP networks: methodology and experience. In *SIGCOMM*, pages 257–270, 2000.

[8] G. Kesidis, A. Singh, D. Cheung, and W. W. Kwok. Feasibility of fluid-driven simulation for atm network. In *Proceedings of IEEE Globecom'96*, London, GB, November 1996.

[9] C. Kiddle, R. Simmonds, C. Williamson, and B. Unger. Hybrid packet/fluid flow network simulation. In *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation (PADS'03)*, San Diego, California, June 2003.

[10] M. Liljenstam and A. Andy Ogielski. Tier-1 internet provider networks with multiple peering points. http://www.ssfnet.org/Exchange/gallery/usa/index.html.

[11] B. Liu, D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of IEEE Infocom'01*, Anchorage, Alaska, April 2001.

[12] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, June 1999.

[13] Y. Liu, F. L. Presti, V. Misra, D. Towsley, and Y. Gu. Scalable fluid models and simulations for large-scale ip networks. *ACM Transactions on Modeling and Computer Simulation*, 14(3):305–324, 2004.

[14] D. Nicol, M. Goldsby, and M. Johnson. Fluid-based simulation of communication networks using SSF. In *Proceedings of the 1999 European Simulation Symposium*, Erlangen, Germany, October 1999.

[15] D. Nicol, M. Liljenstam, and J. Liu. Large-scale network simulation using SSF. In *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, LA, December 2003.

[16] D. Nicol and G. Yan. Discrete event fluid modeling of background TCP traffic. *ACM Transactions on Modeling and Computer Simulation*, 14:1–39, July 2004.

[17] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *Proceedings of ACM SIGCOMM'98*, Vancouver, CA, September 1998.

[18] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C++ : The Art of Scientific Computing (2^{nd} Edition)*. Cambridge University Press, 2002.

[19] RocketFuel Project. `http://www.cs.washington.edu/research/networking/rocketfuel/`.

[20] C. Tebaldi and M. West. Bayesian inference on network traffic using link count data. *J. Americal Statistical Association*, 93:557–576, 1998.

[21] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[22] Y. Vardi. Network tomography : Estimating source-destination traffic intensities from link data. *J. Americal Statistical Association*, 91:365–377, 1996.

[23] A. Yan and W. B. Gong. Time-driven fluid simulation for high-speed networks with flow-based routing. In *Proceedings of the Applied Telecommunications Symposium'98*, Boston, MA, April 1998.

[24] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads, 2003.

[25] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation, 2003.

[26] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia. Maya:integrating hybrid network modeling to the physical world. *ACM Trans. on Modeling and Computer Simulation*, 14(2):149–169, April 2004.

[27] M. Zukerman, T. Neame, and R. Addie. Internet traffic modeling and future technology implications, 2003.