Large-Scale Network Simulation Techniques: Examples of TCP and OSPF Models

Garrett R. Yaun, Harshad L. Bhutada, Christopher D. Carothers, Murat Yuksel, Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute, Troy, NY
yaung@cs.rpi.edu, bhutah@rpi.edu, chrisc@cs.rpi.edu, yuksem@cs.rpi.edu, shivkuma@ecse.rpi.edu

Abstract— **Simulation of large-scale networks remains to be a challenge, although various network simulators are in place. In this paper, we identify fundamental issues for large-scale network simulation, and propose new techniques that address them. First, we exploit optimistic parallel simulation techniques to enable fast execution on inexpensive hyper-threaded, multiprocessor systems. Second, we provide a compact, light-weight implementation framework that greatly reduces the amount of state required to simulate large-scale network models. Based on the proposed techniques, we provide sample simulation models for two networking protocols: TCP and OSPF. We implement these models in a simulation environment ROSSNet, which is an extension to the previously developed optimistic simulator ROSS. We perform validation experiments for TCP and OSPF and present performance results of our techniques by simulating OSPF and TCP on a large and realistic topology, such as AT&T's US network based on Rocketfuel data.** ***The end result of these innovations is that we are able to simulate million node network topologies using commercial off-the-shelf hyper-threaded multiprocessor systems costing less than $7000 USD, and consumes less than 1.4 GB of RAM in total.***

Keywords—**Large-Scale Network Simulation, TCP, OSPF, Optimistic synchronization protocol.**

I. Introduction

There is a deliberate need for large-scale simulation of various networking protocols in order to understand their dynamics. For example, there are several issues in routing that needs to be understood, such as cascading failures, inter/intra-domain routing stability, and interactions of policy-based routing with BGP features. One needs to perform large-scale simulations of inter-domain routing protocols along with various traffic engineering extensions, in order to see their dynamics cause or effect various performance problems in the current Internet.

Additionally, simulation of multi-cast protocols consisting of 10,000 to even 100,000 nodes has not been demonstrated despite the fact there are many multi-cast protocols (e.g. [1], [2], [3]) that need validation of their scalability by extensive simulation. Likewise, in order to fully understand the dynamics of new transport protocols there is significant need for large-scale network simulations, particularly, protocols on large sensor networks and large peer-to-peer networks have vast potential for scalability problems.

We address this need using two techniques. First, we leverage an optimistic synchronization protocol to enable efficient execution on a hyper-threaded, multiprocessor system. Here, simulation objects, such as a host or router, are allowed to process events unsynchronized without regard for the underlying topology or timestamp distribution. If an out-of-order event computation is detected, the simulation object is rolled back and re-execute in the correct timestamp order. Unlike previous optimistic protocols, such as Time Warp [4], the rollback mechanism is realized using *reverse computation*. Here, events are literally allowed to execute backward to undo the computation. This approach greatly reduces the amount of state required to support optimistic event processing as well as increases the performance [5].

Next, we devised an extremely light-weight model implementation framework called ROSSNet that is specifically design for large-scale network simulation. If we examine state-of-the-art frameworks, such as Ns[6], SSFNet [7], DaSSF [8] and PDNS [9], we find they are overly detailed almost to the point of being full-protocol network emulators. For example, these frameworks provide support for a single end-host to have multiple interfaces, a full UNIX sockets API for connecting to real applications, and other details that we believe are not relevant for large-scale simulation studies. The end result is that these systems require almost super-computer amounts of memory and processing power to execute large-scale models.

In contrast, our framework poses the question: *what do you really need to model in order to answer a particular protocol dynamics question in a large-scale scenario*. For example, are all layers in a protocol stack really necessary? Can a host just be a TCP sender or just a TCP receiver? Does the simulated host really need to be both? By asking these kinds of questions, our framework enables a single TCP connection state to be realized in just 320 bytes total

This research is supported by the DARPA's Network Modeling and Simulation program, contract #F30602-00-2-0537, and NSF CAREER Award CCR-0133488.

(both sender and receiver) and 64 bytes per each packet-event.

The end result of these innovations is that we are able to simulate million node network topologies using commercial off-the-shelf multiprocessor systems costing less than $7000 USD, and consumes less than 1.4 GB of RAM in total.

The remainder of this article is organized as follows: Section II, provides a description of our simulation framework, ROSSNet, and parallel simulation engine, ROSS. Sections III and IV describe the implmentation of our TCP and OSPF models respectively. The results from our validation study for both models are presented in Section V followed a performance study in Section VI. Section VII describes related work and Section VIII presents the conclusions from this research and future work.

II. ROSS & ROSSNET

ROSS is an acronym for Rensselaer's Optimistic Simulation System. It is a parallel discrete-event simulator that executes on shared-memory multiprocessor systems. ROSS is geared for running large-scale simulation models. Here, the optimistic simulator consists of a collection of *logical processes* or LPs, each modeling a distinct component of the system being modeled, such as a host or router. LPs communicate by exchanging timestamped event messages. Like most existing parallel/distributed simulation protocols, we assume different LPs may not share state variables that are modified during the simulation. The synchronization mechanism must ensure that each LP processes events in timestamp order in order to prevent events in the simulated future from affecting those in the past. The Time Warp [4] mechanism uses a detection-and-recovery protocol to synchronize the computation. For the recovery, we employ a technique called *reverse computation*.

A. Reverse Computation

Under reverse computation, the roll back mechanism in the optimistic simulator is realized not by classic state-saving, but by literally allowing to the greatest possible extent events to be reverse. Thus, as models are developed for parallel execution, both the forward and reverse execution code must be written.

The key property that Reverse Computation exploits is that a majority of the operations that modify the state variables are "constructive" in nature. That is, the undo operation for such operations requires no history. Only the most current values of the variables are required to undo the operation. For example, operators such as $++$, $--$, $+=$, $-=$, $*=$ and $/=$ belong to this category. Note, that the $*=$ and $/=$ operators require special treatment in the case of multiply or divide by zero, and overflow/underflow conditions. More complex operations such as *circular shift* (*swap* being a special case), and certain classes of random number generation also belong here [5].

Operations of the form $a = b$, modulo and bit-wise computations that result in the loss of data, are termed to be *destructive*. Typically these operations can only be restored using conventional state-saving techniques. However, we observe that many of these destructive operations are a consequence of the arrival of data contained within the event being processed. For example, in our TCP model, the last-sent time records the time stamp of the last packet forwarded on a router LP. We use the *swap* operation to make this operation reversible.

B. ROSS Implementation

The ROSS API is kept very simple and lean. Developed in `ANSI C`, the API is based on logical process or LP model. Here, an LP represents a physical object in the model such as a host or router in the case of network simulation. To model packets traveling through the network, LPs will scheduled time stamped events messages. Services are provided to allocate and schedule messages between LPs. A random number generator library is provided based on L'Ecuyer's Combined Linear Congruential Generator[10]. Each LP by default is given a single seed set. All memory is directly managed by the simulation engine. Fossil collection and global virtual time computations are driven by the availability of free event memory. Their frequencies are controlled with tuning parameters and start-up memory allocation. The event-list priority queue can be configured to be either a Calendar Queue[11], Splay Tree [12] or a binary heap.

To reduce fossil collection overheads, ROSS introduces kernel processes (KPs). A KP contains the statistics and process event-list for an collection of LPs. With KPs there are fewer event-list to search through during fossil collection, thereby improving performance, particularly when the number of LPs is large. For the experiments presented here we typically allocate 4 to 8 KPs irrespective of the number of LPs. KPs are similar to DaSSF timelines [8] and USSF cluster [13].

For more information on ROSS and Reverse Computation we refer the interested reader to the ROSS User's Guide [14].

C. ROSSNet

By using ROSS as the simulation kernel, we are currently developing a network simulator ROSSNet. Unlike conventional network simulators (e.g. Ns [6], JavaSim [15]) ROSSNet uses the flat programming environment of C rather than an object-oriented paradigm and leverages pointers to functions in the place of "virtual methods". Here, developers set function pointers for both end hosts and routers alike to obtain the desired level of functionality. If a host is to behave like a TCP connection, it will set

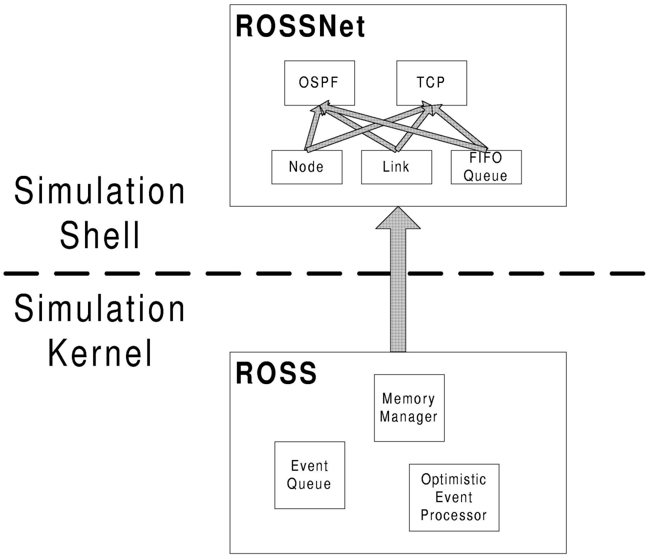

Fig. 1. Structure of ROSSNet.

the event processing function for TCP, likewise if a router is forwarding packets based on either a static routing table or OSPF, it will set its function pointer appropriately. Additionally, ROSSNet attempts to combine or reduce the event population and total number of events processed. For example, in router model, both the forwarding plane and control plane functionality are all realized within the same logical process (LP). Thus, event processing on the control plane side, will immediately effect the forwarding plane without the need for explicit events to be passed between the two planes. ROSSNet will also make use of global data structures. For example, in OSPF, each router maintains a map of the whole network. In simulation, this is not necessary. One can simply keep a global data structure in the simulation such that all the routers can reach it. This way redundant usage of memory is avoided. Last, ROSSNet eliminates unnecessary layers of the protocol stack. For example, if one is interested in simulating behavior of a transport layer protocol, lower layers could be simplified such that they require less resources. This was done in our TCP model configuration.

Figure1 shows the structure of ROSSNet. ROSSNet basically constructs a shell on top of the kernel ROSS. ROSS handles issues related to discrete-event simulation such as maintains event queue, processes events optimistically, manages memory. ROSSNet provides basic components for network simulation such as node, link, FIFO queue. On top of these basic networking components, ROSSNet implements protocols such as OSPF and TCP. We are planning to publicize first version of ROSSNet in Spring 2003. In this paper, we only provide our models for OSPF and TCP simulation.

III. ROSSNET: TCP SIMULATION COMPONENTS

A. TCP Overview

The Internet relies on the TCP/IP protocol suite combined with router mechanisms to perform the necessary traffic management functions. TCP provides reliable transport using a end-to-end window-based control strategy[16]. TCP design is guided by the ”end-to-end” principle which suggests that ”functions placed at the lower levels may be redundant or of little value when compared to the cost of providing them at the lower level” As a consequence, TCP provides several critical functions (reliability, congestion control, session/connection management) because layer four is where these functions can be completely and correctly implemented.

While TCP provides multiplexing/de-multiplexing and error detection using means similar to UDP (e.g.: port numbers, checksum), one fundamental difference between them lies is the fact that TCP is connection oriented and reliable. The connection oriented nature of TCP implies that before a host can start sending data to another host, it has to first setup a connection using a 3-way reliable handshaking mechanism.

The functions of reliability and congestion control are coupled in TCP. The reliability process in TCP works as follows: When TCP sends the segment, it maintains a timer and waits for the receiver to send a acknowledgment on the receipt of the packet. If an acknowledgment is not received at the sender before its timer expires (i.e. a timeout event), the segment is re-transmitted. Another way in which TCP can detect losses during transmission is through duplicate acknowledgments. Duplicate acknowledgments arise due to the cumulative acknowledgment mechanism of TCP wherein if segments are received out of order, TCP sends a acknowledgment for the next byte of data that it is expecting. Duplicate acknowledgments refer to those segments that re-acknowledge a segment for which the sender has already received an earlier acknowledgment. If the TCP sender receives three duplicate acknowledgments for the same data, it assumes that a packet loss has occurred. In this case the sender now re-transmits the missing segment without waiting for its timer to expire. This mode of loss recovery is called ”fast re-transmit”.

TCP flow and congestion control mechanisms work as follows: TCP uses a window that limits the number of packets in flight, (i.e. unacknowledged). TCP congestion control works by modulating this window as a function of the congestion that it estimates. TCP starts with a window size of one segment. As the source receives acknowledgments, it increase the window size by one segment per acknowledgment received (”slow start”), until a packet is lost, or the receiver window (flow control) limit is hit. After this event it decreases its window by a multiplicative factor (one half) and uses the variable `ss_thresh` to denote its current estimate of the network bandwidth-delay product. Beyond `ss_thresh` the window size follows a linear increase. This procedure of additive increase/multiplicative decrease (AIMD) allows TCP to operate in an efficient and fair manner[17].

The various flavors of TCP (TCP Tahoe, Reno, SACK) differ primarily in the details of the congestion control algorithms, though TCP SACK also proposes an efficient selective re-transmit procedure for reliability. In TCP Tahoe, when a packet is lost, it is detected through the fast re-transmit procedure, but the window is set to a value of one and TCP initiates slow start after this. TCP Reno attempts to use the stream of duplicate acknowledgments to infer the correct delivery of future segments, especially for the case of occasional packet loss. It is designed to offer 1/2 RTT of quiet time, followed by transmission of new packets until the acknowledgment for the original lost packet arrives. Unfortunately Reno often times out when a burst of packets in a window are lost. TCP NewReno fixes this problem by limiting TCP's window reduction to at most during a single congestion epoch. TCP SACK enhances NewReno by adding a selective re-transmit procedure where the source can pinpoint blocks of missing data at receivers and can optimize its re-transmission. All versions of TCP would timeout if the window sizes are small (e.g.: small files) and the transfer encounters a packet loss. All versions of TCP implement Jacobson's RTT estimation algorithm (that sets the timeout to the mean RTT plus four times the mean deviation of RTT, rounded up to the nearest multiple of the timer-granularity (e.g.: 500 ms)). A comparative simulation analysis of these versions of TCP was done by Fall and Floyd[18].

B. TCP Optimizations

B.1 TCP Model Data Structures

Our implementation follows the TCP Tahoe specification. There are three main data structures we used. The message, which is the data packet, is sent from host to host via the forwarding plane. The routers LP state maintains the queuing information along with the dropped statistics. Finally the host LP's data structure keeps track of the transferring of data.

A message contains the source and destination address. These addresses are used for forwarding. The message also has the length of the data being transferred which is used to calculate the transfer times at the routers. The acknowledgment number is also included for the sender to observe which packets have been received. The sequence number is another variable which indicates which chuck of data is being transferred.

Now, in our model the actual data transferred is irrelevant and therefore it was not modeled. However in the case that the application was running on top of TCP, such as the Border Gateway Protocol (BGP), such data is required for the correctness of the simulation. We are currently examining solution to this issue.

Now, the router model's state is kept small by exploiting the fact that most of the information is read-only and does not change for the static routing scenarios described in this paper. Inside each router, only queuing information is kept along with a dropped count statistics.

There is a global adjacency list which contains link information. This information is used by the All-Pairs-Shortest-Path algorithm to generate the set global routing tables (one or each router). Each table is initialized during simulation setup and consists only of the next hop/port number for all routers in the network.

Given the port number the router can directly lookup of the next hop address in its entry of the adjacency list. The adjacency list has an entry for each router and each entry contains all the adjacencies for that router. Along with the router neighbor's address, it contains the speed, buffer size, and link delay for that neighbor.

The host has the same data structures for both the sender and receiver sides of the TCP connection. There is also a global adjacency list for the host, however there is only one adjacency per host. In our model, a host is not multi-homed and can only be connected to one router. There is also a read-only global array which contains the sender or receiver host status, and size of the network transfer (which is usually a file of infinite size). The maximum segment size and the advertised window size were also implemented as global variables to cut down on memory requirements.

The receiver contains a "next expected sequence" variable and a buffer for out of order sequence numbers. On the sender side of a connection the following variables are used to complete our TCP model implementation: the round trip timeout (Rto), the measured round trip time (Rtt), the sequence number that is being use to measure the Rtt, the next sequence number, the unacknowledged packet sequence number, the congestion control window (cnwd), the slow-start threshold, and the duplicate acknowledgment count.

For all experiments reported here, the Rto is initialized to 3 second at the beginning of a transfer, along with the slow start threshold being initialized to 65,536. The maximum congestion window size is set to 32 packets. The host, in addition to the variables needed for TCP, has variables for statistic collection. Each host keeps track of the number of packets sent and received, the number of timeouts that occur and its measurement of the transfer's throughput.

B.2 Compressing Router State

As previously indicated, our router design at this point is assumed to have a fixed, static routes. By leveraging this assumption, we set out to reduce the router table state.

Now, a problem encountered with real Internet topologies, such as the AT&T network, is they tend not to have an well defined structure for the purpose of imposing a space-efficient address mapping scheme. Ideally, one would like to impose some hierarchical address mapping scheme on the topology for the purposes of compressing the routing tables. From the model point-of-view, such a compression will not lead to an incorrect simulation of the network so long as flow paths remain the same from the real network to the simulated network. Currently, we are implementing such a scheme.

Our implementation of the routing table just contains the next hop's port number. *Here, the maximum number of ports per routers is 67. Therefore the routing table could be represented in a byte per entry instead of an full integer size address*. In our simulation we have an entry in the routing table for each router. If we had to have an entry for each host, the routing tables would be extremely large. The hosts were addressed in such a way that the router they are connected to can be inferred and therefore a routing table of only routers is acceptable. In the case that it cannot be inferred, we could have a global table of hosts and the routers that they are connected to. This one table is a lot smaller than having a routing table in each router with every host. We note that some topologies are such that a routing table is not needed, such as a hypercube. In these topologies the next hop can be inferred based on current router and the destination.

Last, we assume that routers implement a drop-tail queuing policy. Because of this, routers need not keep a queue of packets to be sent. Instead, the routers schedule packets based on the service rate (i.e., bytes/seconds) and the timestamp of last sent packet. As an example of how this works, let assume we have a buffer size of 2 packets, a service time of 2.0 time units per packet and 4 packets arrive at the following times: 1.0, 2.0, 3.0 and 3.0. Clearly, the last packet will be dropped, but lets see how we can implement this without queuing them. If we keep track of the last send time, we see that the packet at 1.0 will be scheduled at 3.0, following 5.0 and 7.0. Thus, when the last packet arrives, the last sent time is 7.0. If we subtract arrival time of last packet, 3.0 from the last sent time of 7.0, this says there are 4.0 time units worth of data to be sent, which dividing by the service time, yields there are currently 2 packets in the queue. Thus, this packet will be dropped. We are currently examining how this approach could be extended to other queuing policies as well as correct operation under dynamic routing scenarios.

We note that the router state optimizations made to our TCP model may not apply to general purpose network simulation, particularly highly detailed protocol emulation models. It is unclear how much of our performance gains (capacity and speed) are coming from this part of the framework. Our experience suggests that this framework yields tremendous gains even without these router state optimizations.

IV. ROSSNet: OSPF Simulation Components

A. OSPF Overview

Routing protocols in the Internet could be classified into two main groups: link-state routing and distance-vector routing. Typically in the current Internet, distance-vector routing protocols (e.g. BGP) are used for inter-domain routing (i.e. routing among Autonomous Systems (ASs)), while link-state routing protocols (e.g. OSPF, IS-IS) are used for intra-domain routing.

As all the other link-state routing protocols, OSPF maintains a *map* of the network (which typically corresponds to one AS in the Internet) at all routers. Each router collects their local link information and floods the network with that information so that all the routers have a global map of the network.

In OSPF, routers send HELLO packets to their neighbors to check whether they are up or down. HELLO packets are sent periodically at every HelloInterval. If the neighbor does not respond after some period of time, then it is assumed dead. This period of time is called as RouterDeadInterval, and is typically four times the HelloInterval.

Each router maintains LSAs received from other routers. Collection of these LSAs is called as Link-State Database (LS-Database), which in fact shows the global map of the network. The routers run Dijkstra's shortest path algorithm (could also some other Shortest Path First (SPF) algorithms) to find the routes in the network.

When a link goes down or comes up, the routers detects it by HELLO messages. Then, after updating their local LS-Database, they flood an LS-Update message which conveys the change to other routers. Normally, LS-Update messages are sent when a change in the LS-Database occurs. Such a change can happen either because of a local link, or because of an LS-Update message received from elsewhere.

Additional to those above, there is also LS-Refresh messages sent across the OSPF routers. Each OSPF router floods its LS-Database other routers at every LSRefreshInterval, which is typically 45 minutes in the current Internet ASs.

For scalability purposes, OSPF divides an AS into areas and constructs a hierarchical routing among the areas. For each area a corresponding Area Border Router (ABR) is assigned. Additional to ABRs, there are Backbone Routers which are the router nodes among which inter-area routing takes place. Among ABRs and Backbone Routers, one router is assigned as Boundary Router, which is responsible for routing to/from other ASs. All these assignments of routers are typically done manually in the current Internet.

Multi-area routing in OSPF helps scalability. LSAs are flooded only in the area, rather than the whole AS. ABRs flood internal LSAs to other areas as Summary-LSAs. This scales flooding of LSAs. Also, routing among Backbone Routers happen based on address prefixes, which scales routing tables.

More details about OSPF can be found in [19], [20].

B. OSPF Optimizations

We performed several optimizations to the OSPF part of ROSSNet in order to scale the simulation. So far, we have developed models for single-area OSPF simulation.

The OSPF messages can get fairly large (for example, in case of Database Description packet-exchange and subsequent LS Updates in response to the LS Requests). These long messages can be a big impediment in terms of scalability. So, to keep the messages as small as possible, we have used pointers, instead of the actual messages. The router which creates the message, allocates the memory and fills in the required information, and just sends across the pointer. Depending upon the type of message, the memory allocated is freed by the entity receiving the message or the entity originating the message.

In OSPF, HELLO messages take the largest share of event generation. In the simulators widely used, there is generally one event to wake up the "Interface" after every HELLO Interval, and then one event to send the actual HELLO message. This means that two events are required to generate one HELLO message. In our simulator, instead, we schedule just one event to wake up the router, and then the router sends the HELLO messages with some randomization out of every interface. This significantly cuts down the number of events in the simulation.

The LS-Database takes up the largest share of memory, which is stored on every single router. This is the biggest limitation to the scalability (in terms of number of routers) of OSPF simulation. The information required to compare two LSAs, when an LS-Update is received, is stored in the LS-Header. In practice, the link information is replicated at every router and in every LSA. We simulate this by storing only a Link Information Table (LIT) that includes one copy of each link in the topology. So, in our simulation, we store only one copy of the link information (for each link in the topology) globally as shared among all the routers, instead of having a redundant separate copy for each. Besides, we store the LS-Headers locally at each router, so that routers are still able to individually age the LSAs and refresh the self-generated LSAs periodically.

In the case of a link outage, the router connected to that link detects it, and sends out the LS-Update, which consists of the new LS-Header. In the simulation, we reflect this link outage by updating LIT. So the routers receiving the LS-Update can use LIT to run the Dijkstra's algorithm, and calculate its forwarding table. This works perfectly fine for a single link outage before the network converges.

However, the problem with the above strategy arises when there are multiple and frequent link outages or recoveries in the simulated scenario. When there are two simultaneous (or very close time-wise) link changes, nodes in the network will see the various states of the network depending on the arrival order of LS-Updates. For example, assume there are two subsequent link outages that happened for links A and B. It will such that some nodes will hear outage of link A earlier than outage of link B, and some other nodes will hear the other way around. So, if there are multiple link changes within one convergence time, then ways of handling the situation in the simulation is more complicated.

We note here that our OSPF model lacks the reverse execution code path to support optimistic parallel execution. That functionality will be available in the very near future. Consequently, all our OSPF results are based on sequential model execution.

V. Experimental Validations

A. TCP Validation

SSFNet [7] has a set of validation test which shows the basic behavior of TCP. Because of space limitations, we only show how ROSSNet's TCP compares with SSFNet for the Tahoe fast retransmission timeout behavior. This test is configured with a server and a client TCP session with a router in between. The bandwidth is 8 Mb/sec from the server to the router with 5 ms delay and the client to the router had a bandwidth of 800 kilobit per second with a 100 ms delay. The server was transferring a file of 13,000 bytes.

As can be seen from Figures 2 and 3, our implementation with respect sequence number and congestion window behavior performs very similar. The packet drop happens at similar times and so does the fast retransmission.

B. OSPF Validation

In order to validate our OSPF simulation, we experiment on a small topology as shown in Figure 4. There are four routers numbered from 0 to 3, and four end-nodes numbered from 4 to 7. Routers are shown as gray nodes in the Figure 4. Links among the router are all 10Mbps in capacity, while the links connecting routers to end-nodes are all 1Mbps in capacity. In Figure 4, numbers written on each link represents OSPF weight (or cost) for that link. Also, numbers that are written at the beginning of each arrow represents the local enumeration of the link at the node. These enumerations are necessary for forwarding of the packets.

We simulated a scenario where there are two TCP flows in the shown network. One of the TCP flows starts at node 4 and ends at node 7. The other TCP flow starts at node 5 and ends at node 6.

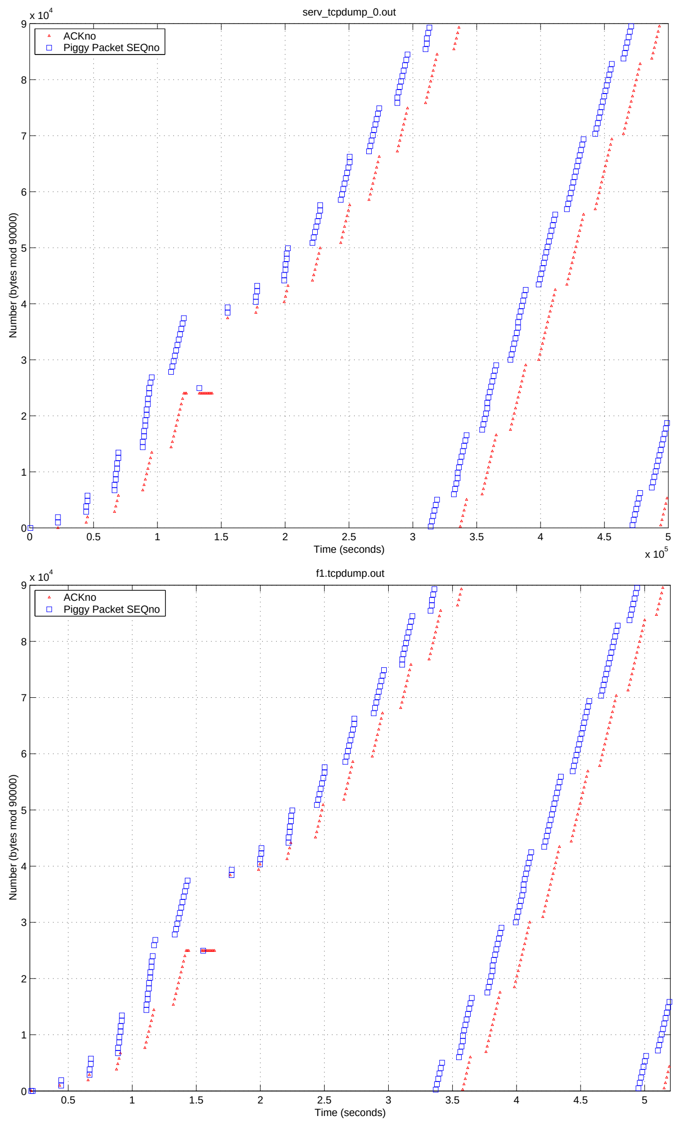

Fig. 2. Comparison of SSFNet and ROSSNet TCP models based on sequence number for TCP Tahoe fast retransmission behavior. Top panel is ROSSNet and bottom panel is SSFNet.

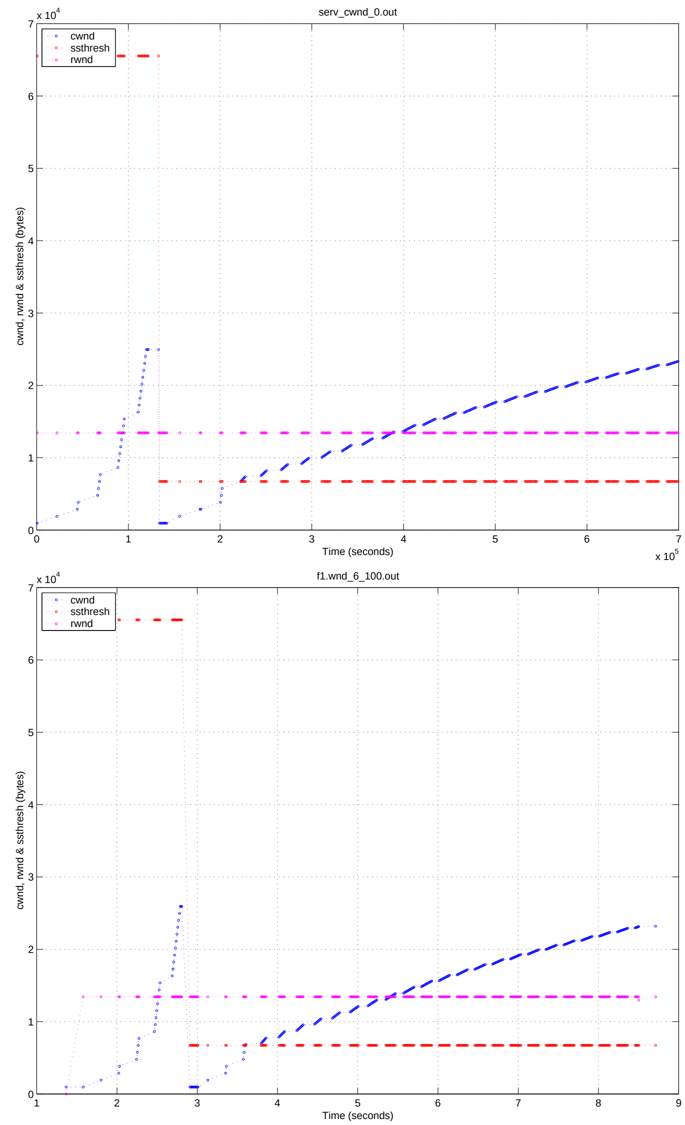

Fig. 3. Comparison of SSFNet and ROSSNet TCP models based on congestion window for TCP Tahoe fast retransmission behavior test. Top panel is ROSSNet and bottom panel is SSFNet.

We simulated a total simulation time of 500 seconds. At time 50, the bi-directional link (i.e. the two one-way links) in between routers 0 and 2 goes down. Later at time 250, it comes back up. We observed the routing tables at the routers and behavior of the two TCP flows.

TableI shows observed routing tables[1] at the four router nodes during the three stages of the simulation. A 255 means the next node is the self. Observe that router nodes adjust themselves to the two link changes properly. We also observed no change in the behavior of TCP flows, because their routes stay the same without getting affected by the link changes.

[1] We did not include entries for end-nodes for simplification.

VI. Performance Results

A. Configuration

Our experiments were conducted on a dual Hyper-Threaded Pentium-4 Xeon processor system running at 2.8 GHz. *Hyper-threading* is Intel's name for a simultaneous multithreaded (SMT) architecture [21]. SMT supports the co-scheduling of many threads or processes to fill-up unused instruction slots in the pipeline caused by control or data hazards. Because the system knows that there can be no control or data hazards between threads, all threads or processes that are ready to execute can be simultaneously scheduled. In the case of threads that share data, mutual exclusion is guarded by locks. Consequently, the underlying architecture need not know about shared variables

Simulation Stage	Router 0		Router 1		Router 2		Router 3	
	Destination	Next Node	Destination	Next Node	Destination	Next Node	Destination	Next Node
0-50	0	255	0	0	0	2	0	1
	1	0	1	255	1	0	1	0
	2	2	2	1	2	255	2	0
	3	1	3	0	3	1	3	255
	Destination	Next Node	Destination	Next Node	Destination	Next Node	Destination	Next Node
50-250	0	255	0	0	0	0	0	1
	1	0	1	255	1	0	1	0
	2	1	2	1	2	255	2	0
	3	1	3	0	3	1	3	255
	Destination	Next Node	Destination	Next Node	Destination	Next Node	Destination	Next Node
250-500	0	255	0	0	0	2	0	1
	1	0	1	255	1	0	1	0
	2	2	2	1	2	255	2	0
	3	1	3	0	3	1	3	255

TABLE I

ROUTING TABLES FOR THREE STAGES OF THE SIMULATION FOR OSPF VALIDATION.

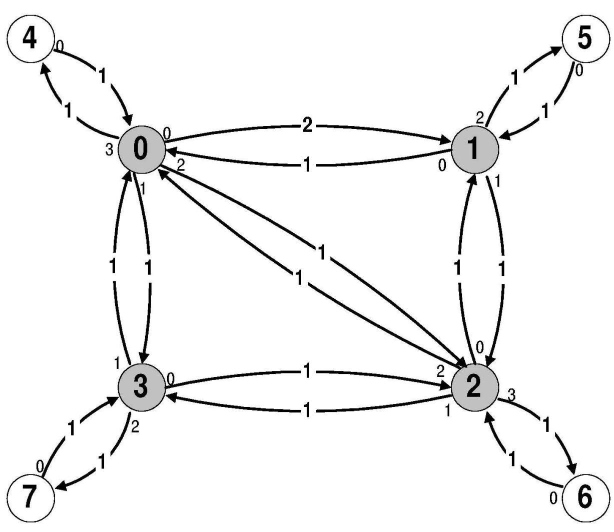

Fig. 4. Topology for experimental validation of OSPF simulation.

or how they are used at the program level. Additionally, because the threads assigned to the same physical processor share the same cache, there is no additional hardware needed to support a cache-coherency mechanism.

Intel's Hyper-Threaded architecture supports two instruction streams per processor core [22]. From the OS scheduling point-of-view, each physical processor appears as if there are two distinct processors. Under this mode of operation, an application must be threaded to take advantage of the additional instruction streams. The dual-processor configuration behaves as if it was a quad processor system. Because of multiple instruction streams per processor, we denote instruction stream (IS) count instead of processor count in our performance study to avoid confusing the issue between physical processor counts and virtual processors or separate instruction streams.

The total amount of physical RAM is 6 GB. The operating system is Linux, version 2.4.18 configured with the 64 GB RAM patch. Here, each process or group of threads (globally sharing data) is limited to a 32 bit address space, where the upper 1 GB is reserved for the Linux kernel. Thus, an application is limited to 3 GB for all code and data (both heap and stack space and thread control data structures).

For all experiments, each TCP connection maintain a consistent configuration. The transfer size was infinite, leading to the transfers running for the duration of the simulation. The maximum segment size was set to 960 bytes. The total size of all headers was 40 bytes. The Initial sequence number was initialized to zero and the slow start thresh was 65536.

All clients and servers were connected in the way that the first half of hosts randomly connected to the second half of hosts. There was a distinct client-server pair for each TCP connection in the simulation. Because of the random nature of connections, there was a high percentage of "long-haul" links that result in a large the number of remote events scheduled between threads.

Last, ROSS is configured with a binary heap for all TCP experiments. However, we have recently implemented a Splay Tree for event-list management and find it produces a 50 to 100% performance improvement over the binary heap. All OSPF experiments have ROSS configured with the faster performing Splay Tree.

B. Synthetic Topology Experiments

The synthetic topography was fully connected at the top and had 4 levels. A router at one level had N lower level routers or hosts connected. The number of nodes was equal to $N^4 + N^3 + N^2 + N$. N was varied between, 4, 8, 16, and 32. The nodes were numbered in such a way that the next hop can be calculated based on the destination at each hop.

The bandwidth, delay and buffer size for the synthetic topology is as follows:

- 2.48 Gb/sec, a delay of 30 ms, and 3 MB buffer,
- 620 Mb/sec, a delay between 10 ms to 30 ms, and 750 KB buffer,
- 155 Mb/sec, a delay of 5 ms, 10ms and 30ms, and 200 KB buffer,
- 45 Mb/sec, a delay of 5 ms, and 60 KB buffer,
- 1.5 Mb/sec, a delay of 5 ms, and 20 KB buffer,
- 500 Kb per second, a delay of 5 ms, and 15 KB buffer

Here, we considered 3 bandwidth scenarios: (i) *high*, which has 2.48 Gb/sec for the top-level router link bandwidths, and each lower level in the network topology uses the next lower bandwidth shown above yielding a host bandwidth of 45 Mb/sec, (ii) *medium*, which starts with 620 Mb/sec and goes down to 1.5 Mb/sec at the end host, and (iii) *low*, which starts with 155 Mb/sec and goes down to 500 Kb/sec at the end host. We note that these bandwidths and link delays are realistic relative to networks in practice.

Our test were run on 1, 2 and 4 instructions streams (IS). The synthetic topography was mapped with each core router and all its children mapped to the same processor.

Table II show the performance results for all synthetic topology scenarios across varying numbers of available instruction streams on the Hyper-Threaded system. For all configurations, we report an extremely high degree of efficiency. The lowest efficiency is 97.4% and to our surprise we observe a large number of zero rollback cases for 2 and 4 instruction streams resulting in 100% simulator efficiency. We observe that the amount of available work per instruction stream (IS) retards the rate of forward progress of the simulation, particularly as N grows and the bandwidth increases. Thus, remote messages arrive ahead of when they need to be processed resulting almost perfect simulator efficiency. This result holds despite inherently small lookahead which is a consequence of link delay and relatively large amount of remote schedule work, which ranges from 7% to 15%. Recall, our link delays range from a small as 5 ms at the low network levels to only about 30 ms at the top router level.

The observed speedup ranges between 1.2 and 1.6 on the dual-hyper-threaded processor system. These speedups are very much in line with what one would expect, particularly given the memory size of the models at hand relative to the small level-2 cache. We note that we were unable to execute the $N = 32$, 45 Mb bandwidth case. This aspect and memory overheads are discussed in the paragraphs below.

The memory footprint of each model is shown as a function of nodes and bandwidth in Table III. We report a steady increase in memory requirements and event-list size as bandwidth and the number of nodes in the network increase. The peak memory usage is almost 1.4 GB of RAM for the $N = 32$, 1.5 Mb bandwidth scenario. The amount of additional memory allocated for optimistic processing is 7000 event buffers which is less than 1 MB. Thus, for 524288 TCP connections, this model only consumes 2.6 KB per connection including event data. By comparison, Nicol [23] reports that Ns consumes 93 KB per connection, SSFNet (Java version) consumes 53 KB, JavaSim consumes 22 KB per connection and SSFNet (C++ version) consumes 18 KB for the "dumbbell" model which contains only two routers.

Last, we find that there is an interplay in how the event population is effected by the network size, topology, bandwidth and buffer space. In examining the memory utilization results, we find that the maximum observed event population differs by only a moderate amount for 1.5 Mb versus 45 Mb case when $N = 16$ despite a rather significant change in network buffer capacity. However, we were unable to execute the 45 Mb scenario when $N = 32$ because it requires more than 17,000,000 events, which is the maximum we can allocate for that scenario without exceeding operating system limits ($\tilde{3}$ GB of RAM). This is because there are many more hosts at a high bandwidth, resulting in much more of the available buffer capacity to be occupied with packets waiting for service. This case results in a 2.5 times increase in the amount of required memory. This suggested, model designers will have to perform some capacity analysis, since networks memory requirements may explode after passing some size, bandwidth or buffer capacity threshold, as happened here.

B.1 Hyper-Threaded vs. Multiprocessor System

In this series of experiments we compare a standard quad processor system to our dual, hyper-threaded system in order to better quantify our performance results relative

Number of Nodes, N	End Host Bandwidth	Num IS	Event-Rate	Efficiency	% Remote	Speedup
4	500 Kb	1	441692	NA	NA	NA
4	500 Kb	2	535093	99.388	7.273	1.211
4	500 Kb	4	660693	97.411	14.308	1.495
4	1.5 Mb	1	386416	NA	NA	NA
4	1.5 Mb	2	440591	99.972	7.125	1.140
4	1.5 Mb	4	585270	99.408	14.195	1.516
4	45 Mb	1	402734	NA	NA	NA
4	45 Mb	2	440802	99.445	7.087	1.094
4	45 Mb	4	586010	99.508	14.312	1.612
8	500 Kb	1	210338	NA	NA	NA
8	500 Kb	2	270249	100	7.273	1.284
8	500 Kb	4	331451	99.793	10.746	1.575
8	1.5 Mb	1	177311	NA	NA	NA
8	1.5 Mb	2	237496	100	7.313	1.339
8	1.5 Mb	4	287240	99.993	10.823	1.619
8	45 Mb	1	176405	NA	NA	NA
8	45 Mb	2	221182	99.999	7.259	1.253
8	45 Mb	4	257677	99.996	10.758	1.460
16	500 Kb	1	128509	NA	NA	NA
16	500 Kb	2	172542	100	7.091	1.342
16	500 Kb	4	199282	99.987	10.600	1.550
16	1.5 Mb	1	100980	NA	NA	NA
16	1.5 Mb	2	137493	100	7.092	1.361
16	1.5 Mb	4	153454	99.998	10.626	1.519
16	45 Mb	1	99162	NA	NA	NA
16	45 Mb	2	117312	100	7.102	1.183
16	45 Mb	4	145628	99.999	10.648	1.468
32	500 Kb	1	80210	NA	NA	NA
32	500 Kb	2	108592	100	7.058	1.353
32	500 Kb	4	126284	100	10.586	1.57
32	1.5 Mb	1	75733	NA	NA	NA
32	1.5 Mb	2	90526	100	7.052	1.20

TABLE II

PERFORMANCE RESULTS FOR $N = 4, 8, 16, 32$ SYNTHETIC TOPOLOGY NETWORK FOR LOW (500 KB), MEDIUM (1.5 MB) AND HIGH (45 MB) BANDWIDTH SCENARIOS ON 1, 2 AND 4 INSTRUCTION STREAMS USING A DUAL HYPER-THREADED 2.8 GHZ PENTIUM-4 XEON. EFFICIENCY IS THE NET EVENTS PROCESSED (I.E., EXCLUDES ROLLED EVENTS) DIVIDED BY THE TOTAL NUMBER OF EVENTS. REMOTE IS THE PERCENTAGE OF THE TOTAL EVENTS PROCESSED SENT BETWEEN LPS MAPPED TO DIFFERENT THREADS/INSTRUCTION STREAMS.

to past processor technology. The network topology is the same as previous described with $N = 8$, thus there are 4680 LPs in this simulation. We did however modify the TCP connections such that they are more locally centered. So, in total 87% of all TCP connections were within the same kernel process (KP).

We observe that the dual processor out performs the quad processor system by 16% depsite that the quad processor has 2 times the amount of level-2 cache (each quad processor has 512 KB for a total of 2 MB of cache). The respective speedups relative to their own sequential performance are 3.2 for the quad processor and 1.7 for the dual hyper-threaded system, which is 80 to 85% of the theoretical maximum. If we compare cost-performance, the dual

Number of Nodes, N	Host Bandwidth	Max Event-list Size	Memory Requirements
4	500 Kb	4,792	3 MB
4	1.5 Mb	5,376	3 MB
4	45 Mb	5,376	3 MB
8	500 Kb	45,759	11 MB
8	1.5 Mb	85,685	17 MB
8	45 Mb	86,016	17 MB
16	500 Kb	522,335	102 MB
16	1.5 Mb	1,217,929	202 MB
16	45 Mb	1,380,021	226 MB
32	500 Kb	5,273,847	1,132 MB
32	1.5 Mb	6,876,362	1,364 MB

TABLE III

MEMORY REQUIREMENTS FOR $N = 4, 8, 16, 32$ SYNTHETIC TOPOLOGY NETWORK FOR LOW (500 KB), MEDIUM (1.5 MB) AND HIGH (45 MB) BANDWIDTH SCENARIOS ON 1, 2 AND 4 INSTRUCTION STREAMS USING A DUAL HYPER-THREADED 2.8 GHZ PENTIUM-4 XEON. OPTIMISTIC PROCESSING ONLY REQUIRED 7000 MORE EVENT BUFFERS (140 BYTES EACH) ON AVERAGE WHICH IS LESS 1 MB.

Processor Configuration	Event-Rate	% Efficiency	% Remote	Speedup
1 IS, Hyper-Threaded	220098	NA	NA	NA
2 IS, Hyper-Threaded	313167	100	0.05	1.42
4 IS, Hyper-Threaded	375850	100	0.05	1.71
1 PE, Pentium-III	101333	NA	NA	NA
2 PE, Pentium-III	183778	100	0.05	1.81
4 PE, Pentium-III	324434	100	0.05	3.20

TABLE IV

PERFORMANCE RESULTS FOR $N = 8$ SYNTHETIC TOPOLOGY NETWORK MEDIUM BANDWIDTH ON 1, 2 AND 4 INSTRUCTION STREAMS (DUAL HYPER-THREADED 2.8 GHZ PENTIUM-4 XEON) VS. 1, 2 AND 4 PROCESSORS (QUAD, 500 MHZ PENTIUM-III

hyper-threaded system (˜$7000 USD) is the clear winner over the quad processor system (˜$24,000 USD) by over a factor of three, since it costs less than 1/3 the price at the date of purchase.

Additionally, we observe 100% simulator efficiency for all parallel runs. We attribute this phenomenon to the low remote messages and large amount of work (event population) per unit of simulation time.

C. AT&T Topology Experiments

For our performance study we used AT&T's network topology obtained from Rocketfuel website [24].

As shown in Figure 5, the core US AT&T network topology contains 13173 router nodes and 38164 links. What makes Internet topologies like the AT&T network both interesting and challenging from a modeling prospective is the spareness of connectivity and power-law structure [24]. In the case of AT&T, there are less than 3 links on average. However, at the super core there is a high-degree connectivity. Typically, an Internet service provider's super core will be configured as a fully connected mesh. Consequently, backbone routers will have up to 67 connections to other routers, some of which are other backbone or super core routers and other links to region core routers. Once at the region core level, the number of links per router reduces and thus the connectivity between other region cores is spare. Most of the connectivity is dedicated to connecting local points of presence (PoPs).

In performing a breath-first-search of the AT&T topology, there are distinct eight levels. At the backbone, there are 414 routers. At each successive level yields the following router count : 4861, 5021, 1117, 118, 58, 6 and at the final level there are 5 nodes. There were a number of routers not directly reachable from within this net-

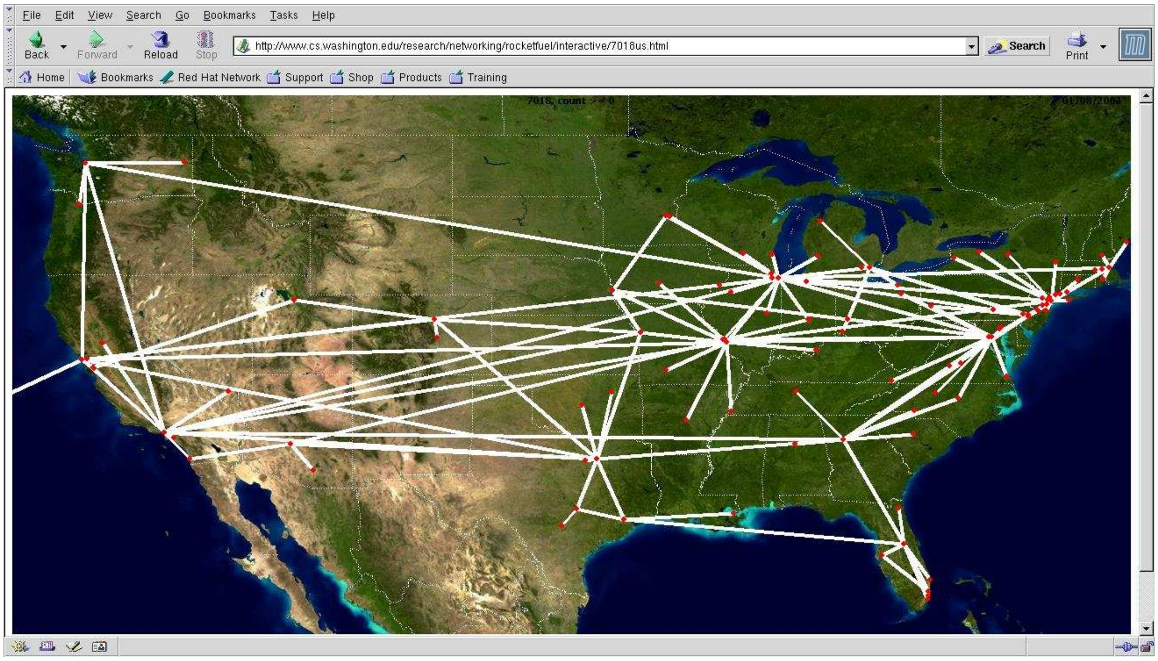

Fig. 5. AT&T Network Topology (AS 7118) from the Rocketfuel data bank for the continental US.

Configuration	Event Rate	% Efficiency	% Remote	Speedup
medium, 1 IS	138546	NA	NA	NA
medium, 2 IS	154989	99.947	52.030	1.12
medium, 4 IS	174400	99.005	78.205	1.25
large, 1 IS	127772	NA	NA	NA
large, 2 IS	143417	99.956	51.976	1.12
large, 4 IS	165197	99.697	78.008	1.29

TABLE V

PERFORMANCE RESULTS FOR AT&T NETWORK TOPOLOGY FOR MEDIUM (96,500 LPS) AND LARGE (266160) ON 1, 2 AND 4 INSTRUCTION STREAMS (IS) USING THE DUAL-HYPER-THREADED SYSTEM.

work. Those routers are most likely transit routers going strictly between autonomous systems (AS). With the transit routers removed, our AT&T network scenario has 11670 routers. Link weights are derived based on the relative bandwidth of the link in comparison to other available links. In this configuration, routing is keep static.

The bandwidth, delay, and buffer size for the AT&T topology is as follows:

- *Level 0 router:* 9.92 Gb/sec, a delay randomly between 10 ms to 30 ms, and 12.4 MB buffer
- *Level 1 router:* 2.48 Gb/sec, a delay randomly between 10 ms to 30 ms, and 3 MB buffer
- *Level 2 router:* 620 Mb/sec, a delay randomly between 10 ms to 30 ms, and 750 KB buffer
- *Level 3 router:* 155 Mb per second, a delay of 5 ms, and 200 KB buffer
- *Level 4 router:* 45 Mb per second, a delay of 5 ms, and 60 KB buffer
- *Level 5 router:* 1.5 Mb/sec, a delay of 5 ms, and 20 KB buffer
- *Level 6 router:* 1.5 Mb per second, a delay of 5 ms, and 20 KB buffer
- *Level 7 router:* 500 Kb per second, a delay of 5 ms, and 5 KB buffer
- *link to all hosts:* 70 Kb per second, a delay of 5 ms, and 5 KB buffer

Hosts are connected in the network at PoP level routers. These routers only have one link to another higher-level router. The first is medium size, with 96,500 nodes or LPs (hosts plus routers) total, and the second is large, with 266160 LPs. In each configuration, the half the host establish a TCP session to a *randomly selected* receiving host. *We observe this configuration is almost pathological for a parallel network simulation because the amount of remote network traffic will be much greater than is typical in practice*. The amount of remote message traffic is much greater than the synthetic network topology because of the networks sparse structure. Our goal is to demonstrate simulator efficiency under high-stress workloads for realistic topologies.

We observe over 99% efficiency for the 2 and 4 IS runs as shown in Table V, yet there is a substantial reduction in the overall obtain speedup. Here, we report speedups for the 4 IS cases of 1.25 for the medium size network and 1.29 for the large. We attribute this reduction to enormous amount of remote messages sent between instruction streams/processors. A parallel simulation using AT&T network topology with a round-robin mapping of LP to processors results 50 to even almost 80% of the all processed events being remotely schedule. We hypothesize that behavior on the part of the model reduce memory locality and results in much higher cache miss rates. Consequently, all instruction streams are spending more time stalled waiting for memory requests to be satisfied. However, we note that more investigation is required to full understand this behavior.

The memory requirements for the AT&T scenario were 269 MB for the medium size network and 328 MB for the large size network, yielding a per TCP connection overhead of 2.8 KB and 1.3 KP respectively. The reason for the reduction per connection in moving from medium to large configuration is because the amount of network buffer space which effects the peak event population did not change, yet the number of connections went up by almost a factor of 3.

D. Initial OSPF Results

Our OSPF experiments use the same AT&T topology configuration as previous described for the medium size network (i.e., 96,500 nodes in the network total). However, we do increase the bandwidth for levels 5, 6 and 7 to 45 Mb/sec. Thus, the amount of traffic generated by the TCP hosts is much greater in this scenario. We also note that we configure all routers in the AT&T network to be inside a *single* OSPF area. Consequently, this results in extremely large OSPF routing tables (i.e., N^2 for N routers in an area) and we are in effect simulating a pathological OSPF scenario as the typical “rule of thumb” for OSPF limits the number of routers per area to 50 [25] with an operational upper bound between 200 to 1000 even with an optimized router. *Our area 12 to 200 times those design limits*. However, despite these modeling extremes we are able to simulate this scenario in conjunction with TCP background traffic, as shown in our performance results (see Table VI).

As shown in Table VI, we observe that the event rate is kept high by the Splay Tree for OSPF without TCP flows, however as we add TCP flows the event population increases by a factor of 12 (150K to 1.8 M). With such a large increase, the event-list management overheads increase by a factor of two which results in a sharp decrease in the event-rate.

The memory utilization is quite large for our models, ranging from 1.9 to 2.3 GB of RAM. We attribute this footprint size to size of the adjacency matrix and routing tables. Recall this model configures OSPF as a single area. While our state compression techniques do in fact reduce memory consumption, this pathological runtime configuration still requires substantial memory requirements. In practice, we anticipate much smaller tables for multi-area OSPF scenarios and significantly less memory.

Overall, we are encouraged by these sequential results and are moving forward on obtaining parallel performance statistics.

VII. Related Work

Much of the current research in parallel simulation for network models is largely based on conservative algorithms. PDNS [9] is parallel/distributed network simulator that leverages HLA-like technology to create a federation of Ns [6] simulators. SSFNet [7], TasKit[26] and GloMoSim [27] all use Critical Channel Traversing (CCT) [26] as the primary synchronization mechanism. DaSSF employs a hybrid technique called *Composite Synchronization*[8], where both the asynchronous CCT algorithm and a barrier synchronization are combined to avoid channel scanning limitations associated CCT while at the same time reducing the frequency a global barrier must by applied.

Recent optimistic simulation systems for network models include TeD [28], which is a process-oriented framework for constructing high-fidelity telecommunication system models. Premore and Nicol [29] implement a TCP model in TeD, however no performance results are given. USSF [13] is an optimistic simulation system that dramatically reduces model run-time state by LP aggregation, and swapping LPs out of core. Additionally, USSF proposes to execute simulation unsynchronized using their NOTIME approach. Based on the results here, a NOTIME synchronization could prove beneficial for large-scale TCP models. Unger et. al. simulate a large-scale ATM network using an optimistic approach [30]. They report speed-ups ranging from 2 to 7 on 16 processors and indicate that optimistic outperforms a conservative protocol on 5 of the 7 tested ATM network scenarios. Finally, a new fixed-point optimistic approach, called Geneis has been proposed by Szymanski et. al.[31]. This approach yields speedups upto 18 on 16 processors for 64 to 256 node TCP models. Super-linear performance is attributed to a reduction in the number of events schedule across machines because of the statistical aggregation of events which is employed by this approach.

Configuration	Event Rate	Max Event-list Size	Events Processed	Memory Requirements
OSPF, no TCP	419286.66	150000	796200468	1.92 GB
OSPF with TCP	197954.02	1800000	1783473402	2.29 GB

TABLE VI
OSPF WITH TCP PERFORMANCE RESULTS FOR AT&T TOPOLOGY (96,500 LPS) SCENARIOS ON 1 INSTRUCTION STREAM USING A DUAL HYPER-THREADED 2.8 GHZ PENTIUM-4 XEON. SIMULATES 100 SECONDS OF NETWORK TRAFFIC.

VIII. CONCLUSIONS AND DISCUSSIONS

In this paper, we investigated fundamental issues for large-scale network simulations. We proposed solutions and techniques for the problem of scaling network simulations to millions of nodes.

Based on the proposed techniques, we developed a scalable simulation models for OSPF routing protocol and TCP transport protocol. We ran simulations of these models on a very large and realistic topology which is AT&T's topology obtained from Rocketfuel [24] website. To date, this capability has not been demonstrated.

With the use of optimistic parallel simulation techniques coupled with reverse computation, speedups of 1.7 for a hyper-threaded dual processor system and 3.2 for a quad processor system are reported. These speedups were achieved with an insignificant amount of additional memory for optimistic processing (i.e., $\tilde{1}$ megabyte in practice).

The parallel TCP model proved to be extremely efficient with very few rollbacks observed. Parallel simulator efficiency ranged between 97 to 100% (i.e., zero rollbacks). This suggests that the model could be executed *unsynchronized* with a negligible amount of error.

The model was implemented as lean as possible which allowed for the million node topology to be executed. We observed model memory requirements between 1.3 KB to 2.8 KB per TCP connection depending on the network configuration (size, topology, bandwidth and buffer capacity).

The hyper-threaded system was able to provide a low cost-performance ratio. What is even more interesting is that these systems blur the lines in terms of sequential versus parallel processing. Here, to obtain higher rates of performance from a single processor, one has to resort to executing the model in parallel. As this technology matures to even high clock rates, we anticipate single processors having many more instruction streams, which will provide an even greater opportunity for parallel simulation tools and techniques.

There have been many ideas that have come about during this work. In the future, we will work on development of a scalable simulation model for BGP and investigate inter-domain routing issues by performing large-scale simulations of them. We will also be working on the implementation of a faster event-list management to reduce priority queue overheads. Also the implementation of TCP functionality such as delayed acknowledgment, ticks for round trip time calculation, and Reno capabilities are in the works. The concept of creating a hierarchical address mapping scheme from a random network topology as well as a better LP to processor mapping scheme to reduce remote events has also been a topic of discussion.

Additionally, as more optimistic models are developed we are learning how they interoperate and how network researchers would like to utilize them. The outcome from this research will be modular software architecture that does not add either memory or computational overheads as compared with its direct implementation counterpart. The architecture should allow for the creation of different applications using the transport protocol level (i.e., TCP), such as Border Gateway Protocol for both inter and intra domain routing and web traffic. In the modular model there should be the ability to turn on and off different layers within the overall protocol stack as well as particular features, such as the need to have data represented in the message. This flexibility will enable the model to be tuned for optimum performance within the constraints placed on its expected operating environment and required level of accuracy.

Finally, in the creation of these models, we leveraged existing models in both the Ns-2 and SSFNet frameworks. We find that "porting" model functionality to our platform is relatively straight forward. In the future, we plan to devise porting guidelines and provide detailed case studies of how we have ported OSPF, TCP, BGP, and multi-cast for use as a reference.

REFERENCES

[1] J.F. Rezende and S. Fdida, "Scalability issues on reliable multicast protocol," in *Proceedings of COST 237 Workshop*, 1999.
[2] Jiang Li and Shivkumar Kalyanaraman, "Mca: A rate-based end-to-end multicast congestion avoidance scheme," in *Proceedings of the Internation Conference on Communications (ICC)*, 2002.
[3] Jiang Li and Shivkumar Kalyanaraman, "Ormcc : A simple and effective single-rate multicast congestion control scheme," in *Submitted*, http://www.cs.rpi.edu/˜ lij6/Research/index.html, 2002.
[4] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, July 1985.
[5] C. D. Carothers, K. Perumalla, and R. M. Fujimoto, "Effi cient

parallel simulation using reverse computation," *ACM Transactions on Modeling and Computer Simulation*, vol. 9, no. 3, pp. 224–253, July 1999.

[6] "UCB/LBLN/VINT network simulator - ns (version 2)," http://www-mash.cs.berkeley.edu/ns, 1997.

[7] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski, "Towards realistic million-node internet simulations," in *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 1999.

[8] D. M. Nicol and J. Liu, "Composite synchronization in parallel discrete-event simulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 5, 2002.

[9] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "A generic framework for parallelization of network simulations," in *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 1999, pp. 128–135.

[10] P. L'Ecuyer and T. H. Andres, "A random number generator based on the four lcgs," *Mathematics and Computers in Simulation*, vol. 44, pp. 99–107, 1997.

[11] R. Brown, "Calendar queues: A fast o(1) priority queue implementation for the simulation event set problem," *Communications of the ACM (CACM)*, vol. 31, pp. 1220–1227, 1988.

[12] R. Ronngren and Rassul Ayani, "A comparative study of parallel and sequential priority queue algorithms," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 2, pp. 157–209, 1997.

[13] D. M. Rao and P. A. Wilsey, "An ultra-large scale simulation framework," *Journal of Parallel and Distributed Computing (in press)*, 2002.

[14] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: Rensselaer's optimistic simulation system user's guide," Tech. Rep. 02-12, Department of Computer Science, Rensselaer Polytechnic Institute, http://www.cs.rpi.edu/tr/02-12.pdf, 2002.

[15] "JavaSim," http://javasim.cs.uiuc.edu, 1999.

[16] V. Jacobson, "Congestion avoidance and control," in *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.

[17] D. M. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *Journal of Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, June 1989.

[18] K. Fall and S. Floyd, "Simulation-based comparison of tahoe, reno, and sack tcp," *Computer Communication Review*, vol. 26, pp. 5–21, 1996.

[19] R. Perlman, *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, Addison-Wesley, 1999.

[20] C. Huitema, *Routing in the Internet*, Prentice Hall, 2000.

[21] J. L. Lo, S. J. Eggers, J. S. Emer, H. M. Levy, R. L. Stamm, and D. M. Tullsen, "Converting thread-level parallelism to instruction parallelism via simultaneous multithreading," *Transactions on Computer Systems*, vol. 15, no. 3, pp. 322–354, 1997.

[22] "Intel Pentium 4 and Xeon Processor Optimization Reference Manual," http://developer.intel.com/design/pentium4/manuals/248966.htm.

[23] D. Nicol, "Scalability of network simulators revisited," in *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS) part of Western Multi-Conference (WMC)*, 2003.

[24] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2002.

[25] D. Kotfila `kotfid@cs.rpi.edu`, "Personal communication," *Director, Cisco Academy, RPI*, 2002.

[26] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary, "Scheduling critical channels in conservative parallel discrete event simulation," in *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, 1999, pp. 20–28.

[27] R. A. Meyer and R. L. Bagrodia, "Path lookahead: a data flow view of pdes models," in *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, 1999, pp. 12–19.

[28] K. Perumalla, A. Ogielski, and R. Fujimoto, "Ted – a language for modeling telecommunication networks," in *Proceedings of ACM SIGMETRICS Performance Evaluation Review*, 1998, vol. 25.

[29] B. J. Premore and D. M. Nicol, "Parallel simulation of tcp/ip using ted," in *Proceedings of the Winter Simulation Conference (WSC)*, 1997, pp. 437–443.

[30] B. Unger, Z. Xiao, J. Cleary, J-J Tsai, and C. Williamson, "Parallel shared-memory simulator performance for large atm networks," *ACM Transactions on Modeling and Computer Simulation*, vol. 10, no. 4, pp. 358–391, 2000.

[31] B. K. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Madnani, "Genesis: A system for large-scale parallel network simulation," in *Proceedings of Workshop on Parallel and Distributed Simulation (PADS '02)*, 2002, pp. 89–96.