Published in the Proceedings of the 1999 Western Simulation Multiconference
Conference on Simulation of Computer Networks and Distributed Systems

DISCRETE EVENT SIMULATION OF CSMA/CD LOCAL AREA NETWORKS IN THE NETWORK WORKBENCH

J. Mark Pullen
Department of Computer Science and C^{3}I Center
George Mason University
Fairfax, VA 22030
mpullen@gmu.edu

KEYWORDS: local area networks, media access control, discrete event simulation

ABSTRACT

The Network Workbench is a new software simulation system that implements a 5-layer Internet-like stack with Discrete Event Simulation (DES). The Workbench uses a simple paradigm that it easier to learn and apply than that of the more complex simulation systems in common use today, which makes it very attractive for use in an academic environment for both introductory courses and advanced graduate projects of moderate complexity. This paper reports on the Medium Access Control (MAC) sublayer recently added to the Workbench. The internal structure of the Workbench and its implications for the MAC simulation are described. A critical problem is the inability to "unschedule" and event in the Workbench DES. The mechanisms we have developed to simulate the highly parallel, completely distributed Carrier Sense Multiple Access with Collision Detection (CSMA/CD) under this limitation are explained in detail.

BACKGROUND: NETWORK WORKBENCH

The author, assisted by various George Mason University (GMU) students, has created a collection of C++ modules that facilitate student investigation of network protocols. The Network Workbench consists of executive routines, a Discrete Event Simulation (DES) module, input/output modules, and a protocol stack abstracted from the popular TCP/IP (Internet) protocol family. As with other network simulation systems, the Workbench provides an environment where experiments can take place safely removed from the complications of hardware configurations and other users' network traffic. However, unlike more complex, powerful simulation environments such as the Optimized Network Engineering Tools system (OPNET) (Katzela, 1999) and ns (see http://www-mash.cs.berkeley.edu/ns/ns.html), the Network Workbench is intended as an experimental environment for students. By its design it is easier to learn and apply than the other systems cited.

The Workbench has been used successfully to teach basic protocol algorithms, and as an environment for graduate research involving networks of moderate complexity. Most code modules are available both as pre-compiled binary modules and C++ source code. Modules that are to be programmed for student assignments are available as C++ "stub" code, with correct interfaces but no function, and as precompiled binary modules that work and thus can be used to demonstrate correct network operation. The stub versions have complete linkage, consistent with the header files, and comments outlining the general approach taken by the related TCP/IP protocol, but no code implementing the protocol algorithm. Designing and coding implementations to complete the shell versions is a valuable learning exercise for students, because it requires them to consider the core algorithms of the protocol, implement the algorithms, and understand the result well enough to test its operation. When the Workbench runs, it creates a simulation of a network using the student-programmed protocols, passing the "email" data files provided. Using the Workbench also can be an effective strategy for some graduate research because it requires a much smaller "learning curve" time investment than more sophisticated tools, so long as the complexity of the networks and protocols falls within the capabilities of the Workbench.

The Network Workbench has grown over several years from a research prototype to a useful piece of courseware, and more recently to a sizable piece of software that fills a range of teaching and research needs. Beginning as a research prototype created in the GMU Center for the New Engineer, the Workbench was expanded to working courseware that was useful but not particularly adherent to any software or networking standards, which was used to teach undergraduate Senior and graduate Master's level introductory networking courses. Beginning in Fall 1997 the author rebuilt the Workbench "from the ground up" to enable its use as a

research platform for Internet-related protocols. This re-written, rationalized and object-oriented Workbench has been used successfully by doctoral students to create course projects in areas such as reliable multicast transport, well beyond the introductory projects packaged with the Workbench. The Network Workbench was further expanded by the author in Summer 1998 to support an internetting model, including LANs, that abstracts the key aspects of the Internet Protocol Suite while avoiding much of its complexity. Currently a set of eight student projects is available, in topic areas network topology, error detection, datalink control (DLC), media access control (MAC), routing, reliable transport, multicast, and internet integration. It is common for students to praise the educational value of the Workbench in course-end reports and critiques. The Network Workbench is available for download with no-cost license for academic use at http://bacon.gmu.edu/networkbench.

NETWORK WORKBENCH MECHANISMS

Protocol Stack

Figure 1 shows the general structure of the Workbench. In addition to the header files, "main" module, DES module, and input/output modules, there is an initializing module and a module "interlayer" through which control passes for every invocation of a protocol stack module. The interlayer module contains a series of messages that can be switched on or off by layer, permitting a flow trace of information being passed from layer to layer in the protocol stack. The interlayer module also collects statistics that are printed in summary at end of the simulation. Figure 2 shows the protocol stack used in the Workbench. It is very similar to the 5-layer Internet stack, and includes simplified versions of TCP, UDP, IP, HDLC, and Ethernet.

Internetting

The Workbench models an abstracted version of the Internet architecture. Each node in the Workbench network has two integer attributes: *netnum* (equivalent to the Class A, B or C network number in an Internet address) and *nodenum* (equivalent to the host number in an Internet address). Each interface of each node has three integer attributes: *netnum* and *nodenum* (inherited from the parent node) and *ifacenum* (which uniquely identifies the interface within the node). *ifacenum*=0 is assigned to the node's interface to its LAN (subnet). All nodes on the same LAN have the same value of *netnum*. Other values of *ifacenum* connect to serial links among nodes. By convention, within a LAN *ifacenum*=255 indicates a broadcast address that is received by all nodes on the LAN, *ifacenum*=254 indicates a multicast address and *nodenum*=1 is the router that interconnects the LAN to other internetted LANs. There is no WAN broadcast function, but WAN multicast occurs

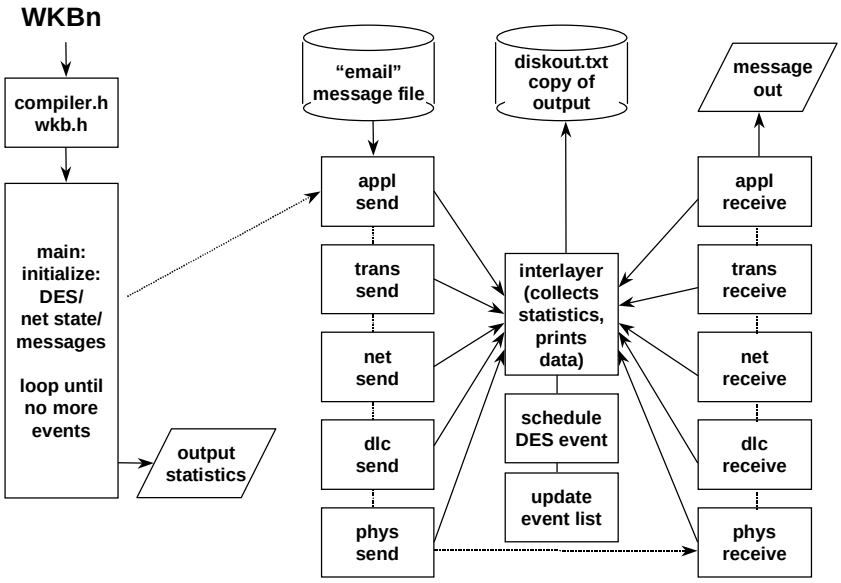

Fig. 1. Network Workbench System Architecture

when *netnum*=254. Each interface in the Workbench has a globally unique eight-bit DLC layer address or "port number", which is used to identify that interface in DLC and MAC frames much like an Ethernet address. The first2**nlinks* addresses identify the ends of WAN links; subsequent addresses are assigned to LAN interfaces. These conventions, and also the fact that addresses are associated with nodes rather than interfaces, are considerable simplifications on the Internet architecture. Nevertheless the author has found that while they simplify the Workbench considerably, they do not cause unacceptable deviations from the basic concepts of the Internet protocols.

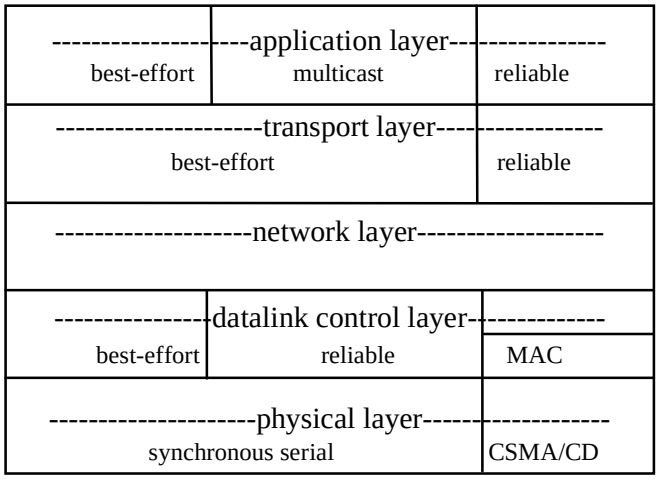

Fig. 2. Network Workbench Protocol Stack

Discrete Event Simulation

In a computer network, there are many asynchronous (unsynchronized) events occurring constantly, yet the whole network must function as a coordinated, distributed system. Discrete Event Simulation (DES) is widely used to study such complex systems by abstracting the key distributed system functions (MacDougall, 1987, Chapter 1). In the Workbench, every simulated event is discretized to a time that is represented by an integer value *sim_timer* that is considered to be a count of some basic *time_step* such as microseconds. Every event happens on a specific value of *sim_timer*. The DES routines are responsible for accounting, in an efficient way, for all events that "have not yet happened". When events "happen" they invoke C++ functions with four parameters: *netnum, nodenum, ifacenum,* and a generic *arguments* which points to a protocol data unit (message, segment, packet, or frame). If for example a transport layer segment is to be sent, invoking function *tl_to_nl(parms,1)* would cause invocation of *function trans_send(parms)* on node 1's host, 10 *sim_timer* ticks from "now" (that is, the current value of *sim_timer* plus 10). Typically the result of an event happening is for one or more new events to be scheduled.

DES Function *next_event( )* causes the function for the next event "waiting to happen" to be invoked. The top level of the Workbench program consists of invoking this function indefinitely, until the event list is empty or a time limit is reached. The DES routines maintain a two-dimensional linked list of events, with the first dimension ordered by "happen time" and second dimension by order of scheduling within a particular "happen time", allowing efficient list search and update. After an event is scheduled, the Workbench DES has no mechanism for "unscheduling" it. This fact has significant implications for simulation of CSMA/CD, as will be seen below.

Stochastic Simulation

The Workbench uses random number generators to create two types of event critical to behavior of networks: message arrival and datalink errors. Each message stream and each datalink has its own independent stream of random events, which are generated by functions that can be replaced by the user. The built-in functions are deterministic (fixed time interval) for best-effort and reliable messages, and Poisson (exponential inter-arrival) for multicast traffic and bit errors. The random number sequences are seeded individually with values that are replicated from run to run, so that the events, while "random," always occur at the same point, to facilitate debugging. To change the distributions, the student needs only to provide generator functions of the same names.

Input/Output

The "email" inputs, WAN topology, number of hosts on each LAN, and multicast topology are contained in files with simple text format that can be edited by students. The interfacing "interlayer" module collects statistics on invocation of the various layers in the Workbench, and also contains a trace function that can be switched on and off at any layer by student-provided modules during execution in order to observe protocol operation. There is a set of user-defined output functions that can be supplemented to provide any desired output when *interlayer( )* is invoked. There also is an interactive mode that limits the text output into chunks small enough to fit on a screen. At end of simulation the inter-layer statistics are automatically displayed to the screen. Text output to the screen also is recorded in file *diskout.txt*. A graphic run-time output display is now under development.

CSMA/CD PROTOCOL

The basic protocol of an Ethernet LAN is Carrier Sense Multiple Access / Collision Detection (CSMA/CD). In this protocol, each LAN interface implements a distributed algorithm intended to assure fair access to the LAN medium without any central coordination. The textbook definition of CSMA/CD is (Stallings, 1997):

1. if the medium is idle, transmit; otherwise go to step 2
2. if the medium is busy, continue to listen until the channel is idle, then transmit immediately
3. if a collision is detected during transmission (that is, two interfaces try to transmit simultaneously), transmit a brief jamming signal to ensure that all stations know there has been a collision, then cease transmission
4. after transmitting the jamming signal, wait a random amount of time, then repeat from step 1

The "random time" is derived by binary exponential backoff (Tanenbaum, 1996):

1. after the first collision, wait either 0 or 1 time slots (with equal probability) to transmit
2. after the second collision, wait 0,1,2 or 3 time slots
3. after the nth collision wait 0 to 2^{n-1} time slots.
4. limit n to 10 (0 to 1023 time slots)
5. after 16 collisions, report failure to the sending process

SIMULATING CSMA/CD

Using DES, it is simple to simulate the behavior of a deterministic or stochastic process, the outcome of which is be uniquely determined by data known at the time the process is started. For example, the transmission of a data frame over a point-to-point serial link requires only a calculation of the time the frame will require to transit the link, the sum of speed-of-light (propagation) delay plus the frame length divided by the link capacity. For propagation velocity *v*, distance *d*, frame length *l*, and link capacity *C* this is:

$$t = d/v + l/C$$

Thus a DES event can be scheduled for time t with assurance that the behavior or the serial link will be modeled correctly. If a noisy link is being simulated, an appropriate sequence of random numbers can be generated and used to corrupt bits in the frame at transmit time so they "arrive" corrupted.

Simulation of CSMA/CD is more difficult because events which may happen during frame transmission are not known when the process starts. Each MAC interface has processing capability, and any of them might at any time sense a collision and send a jamming signal which would interrupt the current transmission of all other connected interfaces. Thus simulation of CSMA/CD requires that the model for each interface check the function of all other interfaces whenever any event related to transmission occurs. In preparing to add a LAN simulation to the Workbench, two leading network simulation systems were checked to determine how they achieve this. OPNET models CSMA/CD by literally modeling its physical process: each LAN frame is presented in turn to each MAC interface, to determine if a collision will occur. ns on the other hand, while it is an admirable network simulation system, turns out not to have a LAN model yet. (Our experience in adding a LAN model to the Workbench revealed difficulties in simulating CSMA/CD which may explain why ns has not yet included this function.)

While noting that the approach taken by OPNET is effective and produces correct results, we also sought a model of CSMA./CD ("Ethernet") that would take less time to simulate while remaining correct within the scope of the Workbench's abstractions. We used a scheme similar to that of (MacDougall, 1987, chapter 6) but with significant adaptation to account for the fact that our CSMA/CD must function as part of a working network simulator (MacDougall's *ether* simulation is limited to producing statistics for a generic Ethernet LAN). We were able to create a workable CSMA/CD simulation in the Workbench by maintaining two global variables for each LAN, *frame_start_time[ ]* and *frame_finish_time[ ]*,each indexed by *netnum*. When an interface prepares to send a LAN frame, the MAC process consults *frame_finish_time* and arrives at one of three courses of action:

1. *frame_finish_time* is already past, so transmit the frame, updating the LAN state variables *frame_start_time* and *frame_finish_time*, and interface state *mac_buffer[portnum].current_frame*
2. *frame_start_time* is within one propagation delay on the LAN, so a collision may occur
3. *frame_start_time* has passed by more than one propagation delay, and *frame_finish_time* has not yet arrived; so invoke the DES to return to the sending process at *frame_finish_time*

Within case 2, some means of determining whether a collision actually occurs is needed. The options are:

a. assume a collision will always occur (this is equivalent to assuming the station being checked is at the opposite end of the LAN medium from all other stations)

b. treat collisions as stochastic events and generate a uniformly distributed random number between zero and one propagation delay to determine whether a collision occurs

c. assign a physical position on the LAN to each interface, and calculate the collision interval exactly; if the interface being checked is separated from the interface which sent first by length *d*, and the propagation velocity in the medium is *v*, a collision occurs when for

$$lan\_delay = (d / v) * time\_step$$

the following holds:

$$lan\_delay < sim\_timer - frame\_start\_time[netnum]$$

The current workbench implementation follows the first option for simplicity. (This has the effect of generating a maximum number of collisions, which is useful in a student exercise, but it is less than completely accurate. However options b and c also are viable, and would not be complex to implement.)

When a collision occurs, the *mac_send( )* function invokes *jam_csma( )* to broadcast a short "jamming frame," indicated by starting flag 00111110 (good frames start with 01111110). When the jamming frame reaches *mac_receive()* by way of the DES, retransmission is scheduled by creating a new DES event that will invoke *mac_send( )* which will begin to transmit *mac_buffer[send_portnum].current_frame* again. Like several other Workbench protocol functions, *mac_send( )* is organized into multiple sending states structured by a state-transition diagram (figure 3) with a state variable that maintains continuity between invocations. To ensure that the next state associated with *mac_send( )* is not destroyed by intervening events, it is stored as *mac_state_bits* and passed along with the frame through the DES.

Because the Workbench does not support "unscheduling" events, *mac_receive( )* necessarily will be invoked for frames that in fact were destroyed by collisions. Therefore *mac_receive( )* must verify that each frame received is still valid. It does this by consulting *mac_buffer[send_portnum]*. All of this activity is by way of ensuring correct results while efficiently simulating within DES under the Workbench a highly parallel, distributed function.

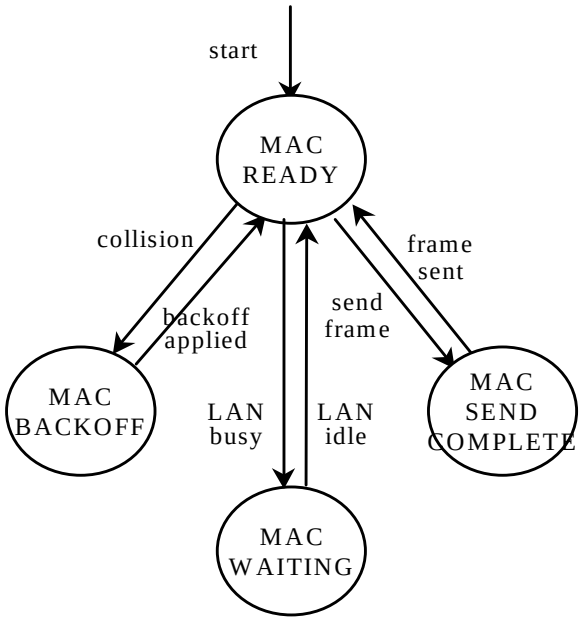

Fig. 3. Workbench MAC Send State-Transitions

CONCLUSIONS AND FUTURE WORK

This paper has described the Network Workbench, a new network simulator intended for student use in introductory courses and also in research projects involving networks of moderate complexity. The Workbench uses DES to implement a five-layer protocol stack abstracted from the Internet architecture, and includes a collection of projects where students program the core algorithms for important protocols such as HDLC, IP, and TCP. The latest additions to the Workbench are an internetting architecture and an accompanying model for Ethernet-like CSMA/CD LANs. Algorithms that implement CSMA/CD efficiently using the Workbench DES are described.

The Network Workbench has been demonstrated to be an extremely useful tool in teaching both introductory network courses at undergraduate and graduate levels, and an advanced graduate-level project-based course. Toward this end the Workbench software will be made available to other teaching faculty on request, under a no-cost license. Future additions to the Workbench are likely to include additional WAN protocols such as a routing protocol, and additional LAN protocols such as token ring, token bus, or Fiber-Distributed Data Interface (FDDI).

REFERENCES

Katzela, I. *Modeling and Simulating Communications Networks*, Prentice-Hall, 1999

MacDougall, M., *Simulating Computer Systems: Techniques and Tools*, MIT Press, 1987

MASH project, University of California-Berkeley, http://www-mash.cs.berkeley.edu/ns/ns.html

Stallings, W, *Data and Computer Communications*, 5th Ed., pp. 404-406, Prentice-Hall, 1997

Tanenbaum, A., *Computer Networks*, pp. 282-283, Prentice-Hall, 1996

ABOUT THE AUTHOR

J. Mark Pullen is an Associate Professor of Computer Science and a member of the C^3I Center at George Mason University, where he heads the Networking and Simulation Laboratory. He holds BSEE and MSEE degrees from West Virginia University, and the Doctor of Science in Computer Science from the George Washington University. He is a licensed Professional Engineer and a Fellow of the IEEE. Dr. Pullen teaches courses in computer networking, and has active research in networking for distributed virtual simulation and networked multimedia tools for distance education. Dr. Pullen recently received the IEEE's Harry Diamond Memorial Award for his work in networking for distributed simulation.