Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

SIMULATION BASED EXPERIMENTS USING EDNAS: THE EVENT-DRIVEN NETWORK ARCHITECTURE SIMULATOR

Sean Salmon
Hala ElAarag

Stetson University
421 N Woodland Blvd
Deland, FL 32723, USA

ABSTRACT

Computer networks serve billions of users all over the world. Research in this field could be performed by building test beds in labs. However, this approach is very expensive, inflexible and hard to reconfigure. It is also difficult and sometimes impossible to replicate some scenarios with test beds. Network simulation on the other hand overcomes all these difficulties. Network simulation can be easily used to study and debug network protocols, understand their interaction and predict how network changes will affect performance. In this paper, we introduce the Event-Driven Network Architecture Simulator, EDNAS. EDNAS is a general-purpose, portable and scalable simulator. We discuss its architecture and implementation. We demonstrate and analyze the results EDNAS provides using various performance measures that are hard to obtain using analytical models. This makes EDNAS very appealing in the study of communication networks.

1 INTRODUCTION

Computer networks are becoming a ubiquitous part of society. It is hard to imagine any aspect of modern life that computers have not touched, from email for personal and business communication, to social networking, such as Facebook and MySpace. As well as, the basic, intrinsic services we think of as necessities, like our phone system.

Since most networks provide a range of services, and thus host multiple protocols, they can quickly become complex systems where protocols interact in ways that are difficult to anticipate. Small, seemingly innocuous changes in how one protocol behaves can have drastic implications on how well information flows through a network; in the same light, if a particular implementation of a protocol does not follow the rules exactly as it should, it can cause major delays – or worse – among the other members of that network.

Network simulation helps deal with this complexity. Simulators provide a safe, controlled environment where different aspects of a network can be observed in greater detail, i.e., "put under the microscope." Additionally, simulation allows different scenarios to be studied without disrupting real systems and users. It allows network administrators to see the potential effects of adding a new server to their infrastructure or allowing a new service to be used on the corporate network. Along with this, simulation permits software architects implementing a new protocol to see if it performs to their expectations, or if an implementation of an existing protocol follows the rules set forth. Students learning about a protocol are also given the ability to witness each aspect of its action in detail.

In this paper, we introduce the Event-Driven Network Architecture Simulator, or EDNAS, which is useful in all the previously described scenarios. EDNAS is *portable*, since it is written in standards-conformant C++ and uses no platform-specific libraries or privileged facilities. Thus it should be useful in professional or academic circumstances where users might be tied to a certain platform. EDNAS is *event-driven*, meaning that the simulation progresses by way of discrete events occurring and spurring other parts of the simulation to action. This allows an unparalleled level of observation, as users can selectively listen for events in which they are interested in, and ignore those that are deemed irrelevant. Finally, and perhaps most importantly, EDNAS is *accurate* and *detailed*. We have put a great deal of effort into making sure that EDNAS provides results that accurately reflect the real world.

The rest of this paper is organized as follows. Section 2 examines some other popular network simulators and breaks them down into two categories, simulators that create a fully-simulated network, like EDNAS, and those that simulate primarily by artificially shaping traffic on a real network. In Section 3 we discuss the architecture of EDNAS and some of the protocols that have been implemented. We then set up and run a network scenario in section 4 and analyze the results. Finally, section 5 provides the conclusion and future work for this paper.

2 RELATED WORK

Due to the diversity of network implementations and the multitude of protocols that must be tested, a wide range of network simulators exist. In this paper, however, the simulators to be discussed will be broken down into two categories. First, those that fully simulate a network, its associated hosts, links, and data transfer, providing what will be called full simulation. Then, those that integrate into an existing network and alter the flow of traffic in ways designed to elicit desired information – these will be referred to as traffic shapers.

2.1 Full Simulation

Full simulators create virtual hosts (not to be confused with virtual machines, which are a separate concept) that use simulated implementations of the protocols under investigation. This layer of abstraction decreases memory and computational requirements (Kiddle 2004), allowing simulation of large networks with hundreds of thousands of nodes. In order to reconcile results from these types of simulations, it is necessary to compare their results to actual traffic (Bajaj et al. 1999). Some examples of famous network simulators are ns, OMNet, Opnet and Qualnet. However, the learning curve of these simulators is slow.

2.1.1 ns

The *Network Simulator*, or *ns*, is a cross-platform discrete event network simulator. *ns* actually refers to a family of simulators, as there are three different incarnations of the simulator. The original, simply *ns*, was based on the REAL network simulator (Keshav 1988), and was succeeded by *ns*-2 (NS-2 Homepage 2011). It currently coexists with the next generation, *ns-3* (NS-3 Homepage 2011).

ns supports a vast number of protocols. It has multiple implementations of TCP, including ports of the actual TCP implementations in the Linux, OpenBSD, and FreeBSD kernels, which have been validated (Jansen and McGregor 2007). In addition, *ns* can simulate both unicast and multicast transfers. With all this capability, though, comes the disadvantage of complexity; *ns* has a rather large API, and programming for it requires a knowledge of both C++ and OTcl scripting language.

2.1.2 OMNeT++

OMNeT++ (OMNeT Homepage 2011) is a discrete event simulator framework – that is, it is not specifically a network simulator, instead, it provides the tools to ease creation of custom network simulations. While superficially similar to *ns*, *OMNeT++* has a different set of goals: (1) to support very large networks; (2) to facilitate easy tracing and debugging; and (3) to be easily embedded into other applications (Varga 2001). These are accomplished by placing an emphasis on modularity and software component reusability (for points 1 and 2), and being open source (point 3). *OMNeT++* also places a high priority on its user interface; much of the simulation is animated as its takes place, and users can examine live variables "on the fly."

2.2 Traffic Shapers

Traffic shapers can be seen as an intermediate step between the simulators mentioned above and a live network. They do not attempt to model as many characteristics, thus decreasing their abstraction, in exchange for more realistic results. This has the effect of increasing the computational demand and decreasing the scope of the situation that can be modeled.

Shapers often take the form of a kernel module for the host operating system that grants the injector direct access to the systems networking layer. This access allows the traffic shaper to intercept and manipulate incoming and outgoing traffic. Two examples of traffic shapers are Dummynet and NIST Net.

2.2.1 Dummynet

Dummynet is a kernel module originally developed by Luigi Rizzo for FreeBSD (Rizzo 1997), though it is now available for a variety of other UNIX-like operating systems (Carbone and Rizzo 2010). By using the operating system's firewall tables and passing intercepted packets through software, Dummynet can simulate the effects of bandwidth limitations, propagation delays, bounded-size queues, packet losses, and multipath transmissions. This means that while Dummynet simulates far fewer nodes than full simulators like *ns* and *OMNeT++*, it can still provide the stressors necessary to probe a protocol's behavior.

2.2.2 NIST Net

NIST Net (NIST 2011) is a project sponsored by the U.S. National Institute of Science and Technology (NIST) that shares quite a few characteristics with Dummynet. It is also a kernel module, though it is written for Linux. Similarly, it allows users to adjust factors like bandwidth limitation, delay, delay jitter, packet losses, and so forth. NIST Net differentiates itself by adding a graphical interface (usable for on-the-fly tuning) and additional traffic capabilities. Where Dummynet could only modify intercepted traffic, NIST Net provides the ability to define custom packet-handling routines based on packet type, reproduce previously captured traffic through "trace"' files, and even generate realistic responses for live machines via emulated hosts. Due to its ability to emulate the behavior of an entire network in a single hop, its creators refer to it as a "network in a box" (Carbone and Rizzo 2010).

3 EDNAS IMPLEMENTATION

This section is divided into two parts. In the first part, we describe the architecture of the simulator itself and explain how events and timing are implemented. In the second, we present some of the protocols that have been implemented in EDNAS and detail their implementation.

3.1 Event Infrastructure

As mentioned in section 1, simulation in EDNAS progresses by way of events. This method was chosen primarily for two reasons. First, it allows for a great deal of connectivity between parts of the simulation, which makes it possible for much less of the functionality to be "hard-wired" and static. Just like in real life, nodes can make decisions based on a wide variety of factors, not just the small subset conceived of at design time. Secondly, the event-driven paradigm is very intuitive to human beings – cause-effect relationships can be easily grasped. Using an intuitive design makes EDNAS easier to use, design, understand, and believe.

3.1.1 Events

Before discussing events, we first must introduce the concept of an *actor*. An actor is defined (somewhat ephemerally) as any simulation object that can *do* anything. Examples include Nodes, which can send and receive information; Loggers, which records certain aspects of the simulation for later review; even the simulation-wide Scheduler, which is discussed later in this section. Actors define and control events, and other interested actors listen for those events to occur.

Figure 1 illustrates an example event setup, where Node *N* defines an event called *OnReceive,* which it "fires" in response to receiving a packet. The scenario depicts three actors listening to the event: *N* itself, likely so it can perform some kind of accounting or send a response if the protocol necessitates, an HTTP server hosted by *N*, which may be waiting for this packet, and the Simulation Log, which accumulates statistical information during the simulation.

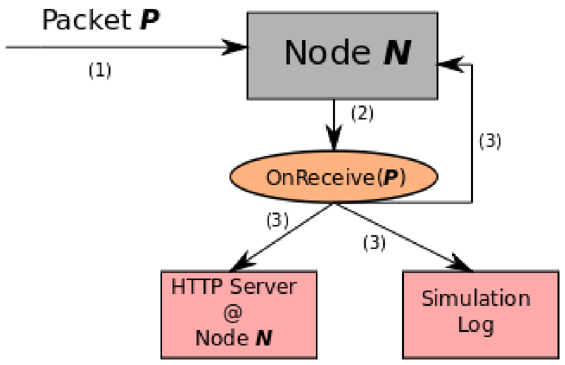

Figure 1: Example Event Firing

3.1.2 Scheduler

The Scheduler provides a means of synchronization for all the simulation components. It controls the progression of time and allows actors to define time-dependent events, such as transfer *timeouts* to decide when a packet has been lost. Additionally, a uniform simulation-wide time is necessary for any simulation to produce accurate and meaningful results and to allow events to occur in the correct sequence. As an implementation detail, we decided to use a 64-bit signed integer to represent time, and to have each distinct value represent a nanosecond. This allows approximately 10^{19} distinct positive time values, giving each simulation a maximum duration of around 292 years, which we believe should be sufficient.

3.2 Protocols

In the interest of ease-of-use and the desire to keep the user from having to re-implement the wheel, we have already implemented a variety of popular network protocols, including Ethernet (IEEE 1985), ARP (Plummer 1982), IPv4 (Postel 1981a), TCP (Postel 1981c), HTTP (Fielding et al. 1999), UDP (Postel 1980), DHCP (Droms 1997), and ICMP (Postel 1981b). In this section, we will discuss our implementation of Ethernet, IPv4, and TCP, as these are the protocols we anticipate being of the most interest to potential users.

3.2.1 Ethernet

Ethernet is a collection of technologies used to transfer information at the data-link layer. It is the base protocol of EDNAS, meaning that all information is ultimately transmitted through Ethernet links.

While it is often necessary to make simplifying assumptions to reduce computational complexity, we have endeavored to provide a very accurate simulation of real-world Ethernet characteristics. For example, we do take into account the 8-byte preamble sent at the beginning of each frame, which is often disregarded as it is not considered part of the frame itself. We also simulate the interframe gap, which is a period of time that Ethernet transmitters must wait between transmissions.

For our simplifying assumptions, we assume that each Ethernet frame has a constant overhead of 304 bits (which includes the size of the preamble). This seemed to be a fair assumption, as the only time that it does not hold true is when the optional Virtual LAN Bridging (IEEE 2005) is enabled. Additionally, Ethernet is simulated only in its switched, full-duplex form. This cuts down greatly on the complexity of implementing the collision detection and retransmission algorithms needed for true multiple-access network media or half-duplex links. This assumption should also be fair, since most modern Ethernet networks make use of switches, which provide full-duplex and limit collision domains. Our transmitters are connected to others by links that have configurable bandwidth and delay (separately configurable for upstream *and* downstream), and each has an outbound buffer that can be set to any desired size.

3.2.2 IPv4

IP's main function is to allow routing. IP defines the addresses that allow computers on the internet to be uniquely identified. It allows messages that are too large for a given link to be broken down into smaller parts via *fragmentation*, and it dictates how a message should be transmitted to reach a host. The IP portion of the packet itself usually contains little or no application-specific data. Instead, IP is used as a container protocol for other protocols like ICMP, TCP, and UDP. Due to this, IP is implemented into EDNAS mainly as a "demultiplexer" to send or receive a packet in these higher level protocols.

3.2.3 TCP

The Transmission Control Protocol, or TCP, addresses IP's best-effort delivery model by providing its own mechanisms for ensuring that data not only reaches its destination, but can be put in the correct order and is free from corruption. This makes it the *de facto* protocol for transferring all but the most time-critical (as TCP does have some overhead) or inconsequential data.

TCP was designed with the intent that it would be resilient to network congestion and unreliable connections (Postel 1981c), and as such utilizes some rather complex machinery. It provides *guaranteed delivery* and *congestion avoidance.*

3.2.4 Guaranteed Delivery

Though IP defines the details necessary to transmit data to another host, it makes no assurance that anything sent will actually make it to its destination. It is an unreliable protocol, meaning that its packet delivery takes a "best-effort" approach, and that lost packets should not be considered an error or even unusual. One way that TCP improves upon IP is by guaranteeing delivery, more specifically, it guarantees that packets are delivered to their destination, that they are put into the correct order, and that the sender is informed that the information was received. While there are multiple strategies on how best to accomplish this (Fall and Floyd 1996), EDNAS currently uses a technique called *Cumulative Acknowledgment*, where acknowledgement of a packet implies receipt of all previous data.

3.2.5 Congestion Avoidance

The second major feature of TCP is its congestion avoidance behaviors. These provide a way for a TCP node to dynamically adjust the amount of data it sends to suit network conditions and avoid increasing network congestion. Though a multitude of competing algorithms exist for this behavior (Ahn et al. 1995; Stevens, Allman, and Paxson 1999; Hengartner, Bolliger, and Gross 2000), their basic functionality consists of an early period known as *slow start*, where the amount of data a sender can have “in flight” (that is, waiting for acknowledgment) increases at a nearly exponential rate.

This growth continues until the sender either reaches a predefined *window* size, at which point it grows at a slower *congestion avoidance* rate, or a packet times out without acknowledgment, which causes the sender to adjust its maximum window size and retry the slow start phase again. EDNAS uses a slightly more advanced form of this known as TCP Reno.

3.2.6 Retransmission Timer Selection

Four variables are used to set the retransmission timer (Fall and Floyd 1996): *rtt, srtt*, *rttvar, and backoff.* Roundtrip time sample arrives with new ACKs. The *rtt* sample is computed as the difference between the current time and a time field in the ACK packet, which is equal to the time the packet was issued at the source. When the first sample is taken, its value is used as the initial value for *srtt.* Half the first sample is used as the initial value for *rttvar.* For subsequent samples, the values are updated as follows:

$$srtt = \frac{7}{8} \times srtt + \frac{1}{8} \times rtt \tag{1}$$

$$rtt\,\mathrm{var} = \frac{3}{4} \times rtt\,\mathrm{var} + \frac{1}{4} \times \left| rtt - srtt \right| \tag{2}$$

The *backoff* is initially 1. The retransmission timer is set to the current time plus

$$backoff \times (srtt + 4 \times rtt\,\mathrm{var} + 1) \tag{3}$$

The *backoff* factor doubles each time a timeout occur to a maximum of 64 (Karn's exponential timer backoff).

4 PERFORMANCE

In this section we demonstrate and analyze the results obtained from the simulator.

4.1 Metrics

In this paper we focus on the three most important metrics in studying communication networks: average throughput, round trip time, and round trip time variation.

4.1.1 Average Throughput

Throughput measures the amount of data sent via a connection with respect to time. It can be measured using the equation

$$R = \frac{\sum si}{T_0 - T_f} \tag{4}$$

Where, *R* is the average throughput, S_i is the size of packet *i,* T_0 and T_f are the times the connection was opened and closed, respectively. While throughput can be interpreted to include *all* data sent, we calculate it based solely on data that successfully reaches its intended recipient.

4.1.2 Round Trip Time

Round trip time, or RTT, is the time that it takes for a given packet to travel from its sender to its receiver, and then for the sender to receive acknowledgment of the successful transfer. For a connection consisting of a single link, we can calculate a minimum for the RTT using the equation:

$$RTT_{MIN} = D_{up} + \frac{S_{data}}{B_{up}} + D_{down} + \frac{S_{ack}}{B_{down}} \tag{5}$$

Where, D_{up} and D_{down} are the upstream and downstream propagation delays for the link, S_{data} and S_{ack} are the size of the data packet and acknowledgment packet, B_{up} and B_{down} are the upstream and downstream bandwidth of the link. This is easily extended to connections of multiple links by performing the above calculation for each link and then summing the results.

Under less than ideal conditions, delay can increase well beyond the lower bound given above. Congestion, data corruption, and various other situations negatively affect delay. We calculate delay as the difference in time between the *first* time a packet is sent (in the case of retransmissions) and the first acknowledgment received for that packet.

4.1.3 Round Trip Time Variation

RTT variation – sometimes called jitter – measures the *difference* in RTTs among packets in a given connection. This characteristic is of particular interest in multimedia applications, such as watching videos and listening to music, as large variations make it difficult to predict how much data must be buffered before playback. Since we measure this as the difference in RTT between two consecutive packets, the term ΔRTT is used as shorthand for round trip time variation in this paper.

4.2 Network Topology

While it is easy to setup any network topology using EDNAS, in this section we show one of the most popular yet simplistic network topologies. This topology is very commonly used in studying the performance of TCP on wired and wireless networks. The network topology models a single client requesting and receiving a file from a server. The two are connected via switches and a router as follows. The client is connected via link $\boldsymbol{L_{SC}}$ to client-side switch $\boldsymbol{S_C}$, $\boldsymbol{S_C}$ connects to the router through link $\boldsymbol{L_{RC}}$, the router then connects to the server-side switch $\boldsymbol{S_S}$ using link $\boldsymbol{L_{RS}}$, which then connects to the server via link $\boldsymbol{L_{SS}}$. This is depicted graphically in Figure 2. We use this topology also to act as a simple way to validate the behavior of EDNAS.

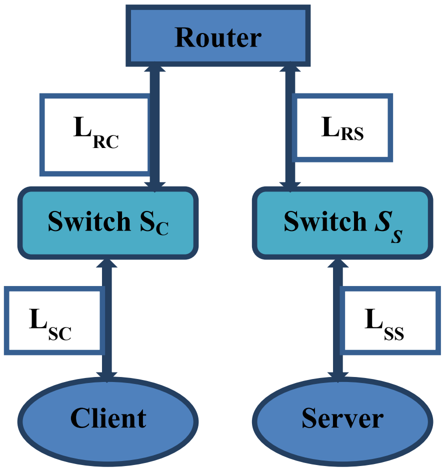

Figure 2: Network Topology

4.2.1 Characteristics

4.2.1.1 Links

Inter-node links have a high degree of configurability. They allow both bandwidth (in units of bits per second) and delay (in units of nanoseconds) to be adjusted independently in each direction, and reliability can be set as a probability between 0 and 1 (inclusive) that any given message will pass. Table 1 shows the values given to the various links.

Table 1: Link Properties

Link	Bandwidth	Delay	Reliability
L_{RC}	1 Gbit/s	1 μs	1.00
L_{RS}	1 Gbit/s	1 μs	1.00
L_{SC}	1 Mbit/s	10 ms	1.00
L_{SS}	10 Mbit/s	10 ms	1.00

4.2.1.2 Buffer Size

Each network interface has an associated outbound buffer. Once a buffer is full, any additional packets that a node attempts to send will be dropped until there is available buffer space. For simplicity's sake, we decided to keep this buffer size uniform across a given node in our simulations, that is, if a node has more than one interface, it will have identically-sized output buffers on each of its interfaces. Table 2 shows the values we assigned to the various nodes. Also, since our simulation uses a uniform maximum segment size (MSS) for TCP packets, we also provided the buffer size in multiples of maximally-sized TCP packets.

Table 2: Node Output Buffer Properties

Node	**Buffer Size**	**TCP Packets**
Router	1 MB	699
Server	1 MB	699
Client	1 MB	699
Switch S_c	64 KB	43
Switch S_s	64 KB	43

4.2.1.3 File Size

These scenarios model one or more clients receiving a file from a server. The client's request is sent as a single message, and the file is sent as a message broken down into as many TCP segments as necessary. The file size used in this example is set to 10MB.

4.2.2 Transfer Progress

We measure the transfer progress by noting the sender's highest acknowledged packet number with respect to time, as shown in Figure 3. This gives an accurate indication of the state of the transfer. As the server can be certain that a packet was received correctly once a packet has been acknowledged.

Figure 3 also notes the two times that packets timed out during this transfer, which occurred on packets 90 and 1202 at 3.4 and 28.7 seconds, respectively. These two events are examined in greater detail in Figure 4 and Figure 5, which additionally show duplicated acknowledgments and newly-sent packets. Taking note of the difference between the sequence numbers the server is sending and the sequence numbers of recently acknowledged packets, we see that the sender's transfer window has grown quite large, which means that these timeouts are most likely due to switch $\boldsymbol{S_S}$ dropping packets that it can no longer fit into its saturated buffer.

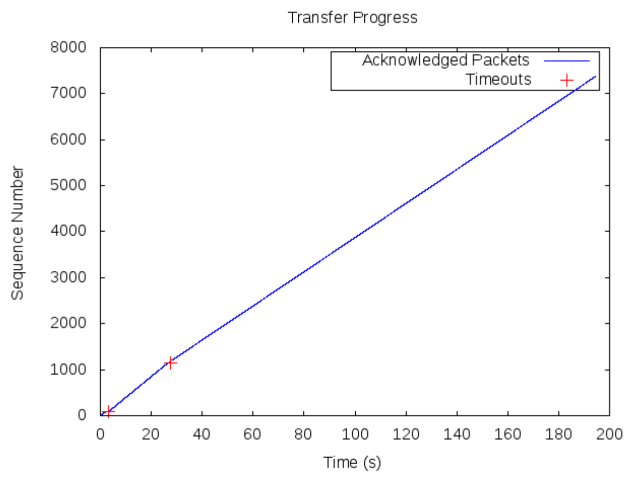

Figure 3: Transfer Progress of Base Scenario

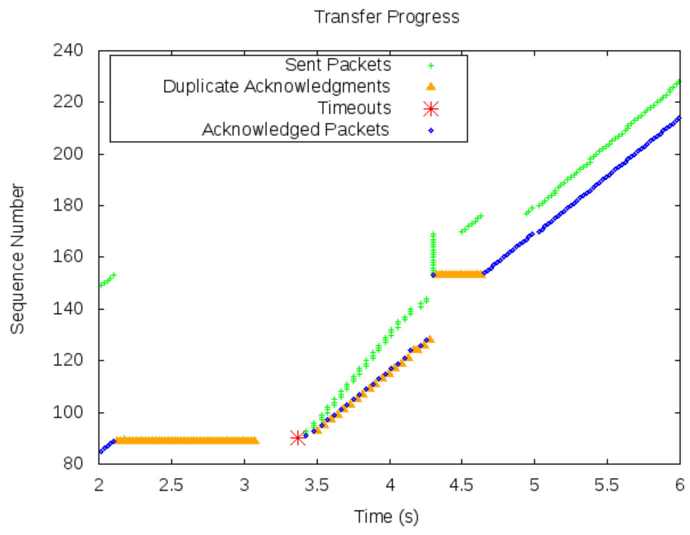

Figure 4: Detailed Transfer Progress Around First Timeout

4.2.3 Round Trip Time

As we stated earlier in section 4.1, we calculate RTT based on the difference in time between sending a packet the *first* time (as opposed to retransmissions) and when the server gets positive acknowledgment that the client received the packet. Figure 6 plots the RTT with respect to packet sequence number, and Table 3 gives a statistical breakdown. Figure 6 shows large fluctuations in the round trip time, centered around the two timeouts. The growth in RTT is caused by the rapidly increasing transfer window size on the server. As the server's window increases, it sends out too many packets to fit in the buffer on switch

$\mathbf{L_{SC}}$ and packet loss occurs. Once the correct window size is reached, the RTT quickly stabilizes and remains at its minimum.

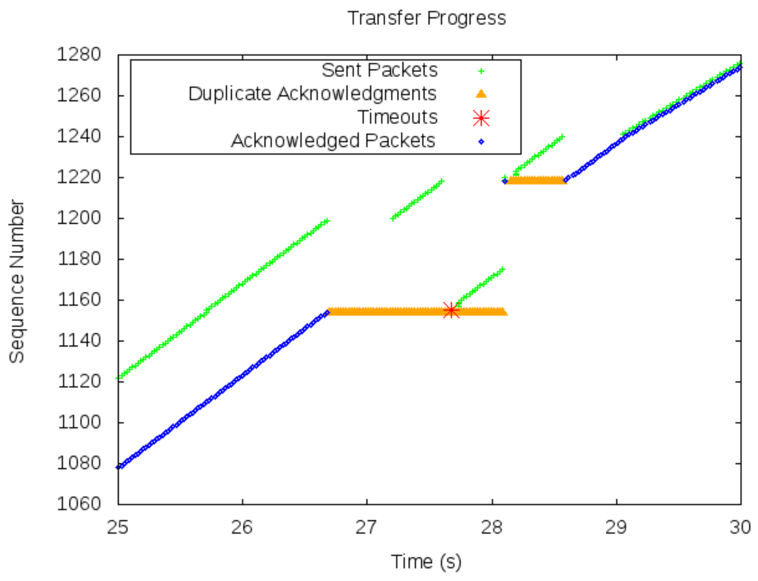

Figure 5: Detailed Progress Around Second Timeout

The ΔRTT, shown below the RTT in the same figure, exhibits similar behavior. We can see that around the two timeouts, the RTT oscillates rather wildly. Once it settles, however, it does not venture far from 0 for the rest of the transfer.

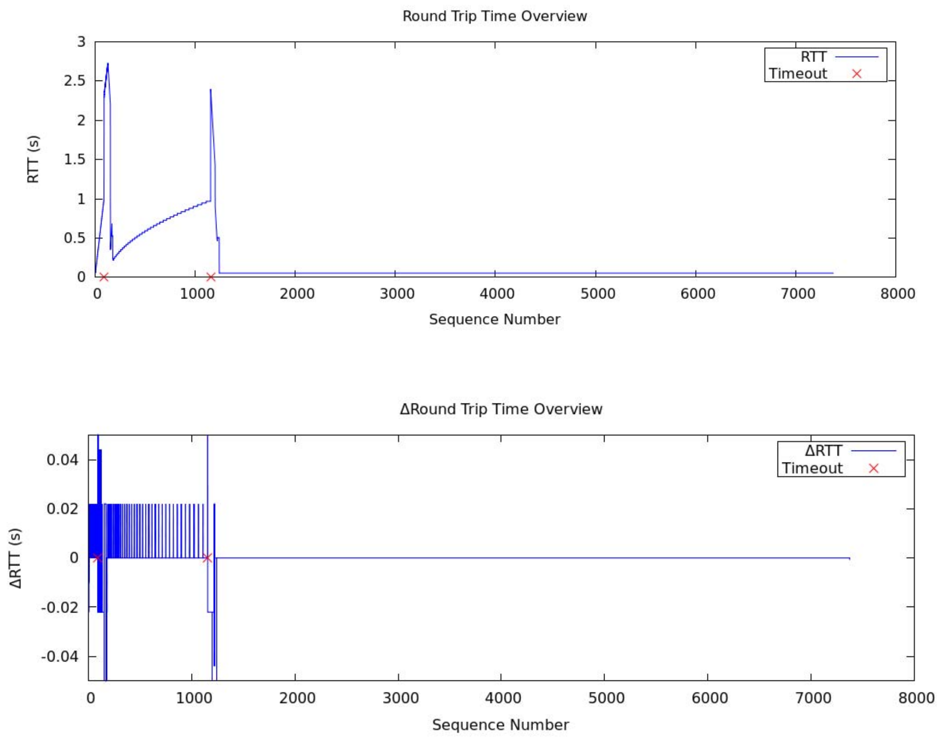

Figure 6: Round Trip Time

Table 3: Round Trip Time Statistical Summary

	Minimum	Median	Maximum	Mean	Std. Dev.
RTT (s)	0.041379	0.053916	2.288691	0.146777	0.23533
ΔRTT (s)	-0.45208	0	0.022	0.000151	0.007035

5 CONCLUSION AND FUTURE WORK

Network simulation is imperative in studying computer networks. Our goal is to provide the network research community with an easy-to-use network simulator that will enable them to study various aspects of performance of communication networks. In this paper we provided the details of the Event-Driven Network Architecture Simulator. In our simulator we have implemented a variety of popular network protocols, including Ethernet, ARP, IPv4, TCP, HTTP, UDP, DHCP, and ICMP. Being written in C++, with an object oriented approach, other researchers could easily extend EDNAS library by implementing other implemented or newly suggested protocols. In this paper, we presented numerous performance measures that one can study using EDNAS.

REFERENCES

Ahn, J., P. Danzig, Z. Liu, and L. Yan. 1995. “Evaluation of TCP Vegas: Emulation and Experiment.” In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 185-195.

Bajaj S., L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. 1999. “Improving Simulation for Network Research." Technical Report 99-702b, University of Southern California.

Carbone, M., L. Rizzo, and U. di Pisa. 2010. “Dummynet Revisited.” *ACM SIGCOMM Computer Communication Review* 40(2):13-20.

Droms, R. 1997. "Dynamic Host Configuration Protocol." RFC 2131, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc2131.txt.

Fall, K., and S. Floyd. 1996. “Simulation-Based Comparisons of Tahoe, Reno and SACK TCP.” *ACM SIGCOMM Computer Communication Review* 26(3):5-21.

Fielding R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. 1999. "Hypertext Transfer Protocol–HTTP/1.1." RFC 2616, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc2616.txt.

Hengartner, U., J. Bolliger, and T. Gross. 2000. “TCP Vegas Revisited.” In *Proceedings of IEEE Infocom*, 1546-1555.

IEEE 1985. *IEEE 802.3: IEEE Local Area Networks: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) - (ETHERNET)*. IEEE Standards Association. Accessed July 23, 2011. http://standards.ieee.org/getieee802/download/802.3-2008_section1.pdf.

IEEE 2005. IEEE 802.1: 802.1Q - Virtual LANs. IEEE Standards Association. Accessed July 23, 2011. http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf.

Jansen, S., and A. McGregor. 2007. “Validation of Simulated Real World TCP stacks.” In *Proceedings of the 2007 Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 2177–2186. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Keshav, S. 1988. “REAL: A Network Simulator.” Technical Report, University of Berkeley, California. Accessed March 22. www.eecs.berkeley.edu/Pubs/TechRpts/1988/CSD-88-472.pdf.

Kiddle, C., 2004. “Scalable Network Emulation.” Ph.D. Thesis, University of Calgary, Department of Computer Science.

NIST. 2011. “NIST Net Home Page.” National Institute of Science and Technology. Accessed July 23, 2011. http://www-x.antd.nist.gov/nistnet/.

NS-2 Homepage. 2011. “The Network Simulator – ns-2.” Accessed July 23, 2011. http://www.isi.edu/nsnam/ns/.

NS-3 Homepage. 2011. “The ns-3 network simulator.” Accessed July 23, 2011. http://www.nsnam.org/.

OMNet Hompage. 2011. “Omnet++.” Accessed July 23, 2011. http://www.omnetpp.org.

Plummer, B. 1982. "Ethernet Address Resolution Protocol." RFC 826, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc826.txt.

Postel, J. 1980. "User Datagram Protocol." RFC 768, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc768.txt.

Postel, J. 1981a. "Internet Protocol." RFC 791, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc791.txt.

Postel, J. 1981b. "Internet Control Message Protocol." RFC 792, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc792.txt.

Postel, J. 1981c. "Transmission Control Protocol." RFC 793, Internet Engineering Task Force. Accessed July 23, 2011. http://www.ietf.org/rfc/rfc793.txt.

Rizzo, L. 1997. “Dummynet: A Simple Approach to the Evaluation of Network Protocols.” *ACM SIGCOMM Computer Communication Review* 27:31–41.

Stevens, W., M. Allman, and V. Paxson. 1999. “TCP Congestion Control.” Accessed July 23, 2011. http://www.ietf.org/rfc/rfc2581.txt.

Varga, A. 2001. “The OMNeT++ Discrete Event Simulation System.” In *Proceedings of the European Simulation Multiconference (ESM’2001)*, 319–324.

AUTHOR BIOGRAPHIES

SEAN SALMON is a software engineer at Scientific Research Corporation in Charleston, SC. He graduated from Stetson University in May of 2010. His research interests include Peer-to-Peer networking technologies and network simulation. His email address is ssalmon@stetson.edu.

HALA ELAARAG is an associate professor of computer science at Stetson University, DeLand, Florida. She received her Ph.D. degree in Computer Science from the University of Central Florida in 2001, her M.Sc. and B.Sc. from Alexandria University in 1991 and 1989 respectively. Her research interests include computer networks, evolutionary computation, modeling and simulation and performance evaluation. Dr. ElAarag is Program Chair and board member of Consortium of Computing Sciences in Colleges Southeastern Region. She is Vice General Chair and General Chair of Society for Modeling and Simulation International (SCS) Spring Simulation Multiconference 2011 and 2012, respectively. She serves on the technical committee of many international conferences and reviews for multiple journals. Her email address is helaarag@stetson.edu.