Transaction Level Modeling of Best-Effort Channels for Networked Embedded Devices

Amal Banerjee, Andreas Gerstlauer

Department of Electrical and Computer Engineering, University of Texas at Austin,
Austin, Texas 78712, USA
{abanerj, gerstl}@ece.utexas.edu

Abstract. We use Transaction Level Modeling techniques to specify and validate best-effort channels for networked embedded devices, to integrate the generated specification model in system-level design flow, for prototyping, exploration and validation of design alternatives. A best-effort channel does not provide any guarantees on final data delivery or delivery rate. With more embedded devices existing in networked environments, often sharing a common communication channel, devices compete with each other for all common network resources, e.g., in a wireless sensor network where low power devices share a low bandwidth best-effort channel. To examine such systems, we specify Half-Duplex Ethernet using the SpecC language and Transaction Level Modeling techniques. All models are validated in a multi-station test setup using Ethernet-based network algorithms.

Keywords: Best-effort Channel, Networked Embedded Systems, Transaction Level Model (TLM), System Level Design Language (SLDL).

1 Introduction

Most of today's embedded systems exist in some form of a networked environment. Inspired by Ethernet as the most cost-effective local area network technology in general computing, embedded system designers have adapted it for industrial automation. In order to understand the behavior of these networked embedded devices, for better control and reliable inter-device communication, more design support and robust models are essential. For any communication channel in any network, an arbitration scheme is required to decide which device can use the channel at any given time. Such schemes could either be centralized (e.g., ARM AMBA bus) or distributed (e.g., Ethernet or wireless networks), the latter often in combination with *best-effort* channels. A best-effort communications protocol (e.g., Half-Duplex Ethernet) provides no guarantees on final data delivery or rate of data delivery. Though replaced by Full-Duplex Ethernet and variants like Time Triggered Ethernet in wired networks, best-effort and contention mechanisms resulting in uncertainty and unreliability underlying the Ethernet protocol are also at the heart of wireless embedded networks [19], e.g., mobile device interfaces, wireless local area networks (WLANs) and wireless sensor networks. As such, modeling concepts developed for Ethernet are applicable to a wide variety of networked embedded systems.

In the ISO/OSI network model, the Ethernet [11] sub-layer of the Data Link layer manipulates frames coming in/going out (from/to Physical layer). In the beginning, Ethernet was half-duplex in nature. A *half-duplex* channel allows two way communication, but only in one direction at a time, i.e., the transmitter must stop before the receiver can reply. A Half-Duplex Ethernet channel has active transmitters and passive receivers. It relies on a contention resolution algorithm, Truncated Binary Exponential Back-Off, that allows a failed transmitter to decide how long to wait before the next transmission attempt. While communication channels have been modeled using networks simulators such as ns-2 [22], these do not allow the modeling of complete systems (hardware and software). In contrast, systems are modeled with a System Level Design Language (SLDL), combined with Transaction-Level Modeling (TLM) techniques. To the best of our knowledge, there are currently no network-oriented TLM channel models.

Transaction-Level Modeling (TLM) [6] allows modeling of digital systems with inter-module communication details abstracted and separated from those of the implementation of computation modules. Communication mechanisms such as busses or FIFOs, are modeled as channels, and modules access them via interfaces. Channel models encapsulate low-level details of the information exchange. Transaction requests occur when modules call interface functions of the channel models. The emphasis is on what data is being transferred, rather than how it is being transmitted. Thus, the system designer can experiment, with different bus architectures (supporting a common abstract interface) without re-coding models that interact with any of the buses. Our choice of the SpecC SLDL [21] supports TLM, along with the crucial concept of time with delta cycles, essential for effective hardware/system and network modeling, unlike traditional network simulators such as ns-2 [22], OPNet [23] or OMNet++ [24].

1.1 Related Work

The Ethernet protocol [11] was developed in a very straightforward way. One of the earliest attempts to specify Half-Duplex Ethernet by Weinberg and Zuck [1] uses Henzinger's real-time models and transition diagrams. Bochmann and Sunshine [2] provide an overview of formal methods used in communication protocol design while Schmaltz and Borrione [3] present an ACL2 logic based scheme for the specification of System-on-Chip (SoC) communication architectures. Georges et al. [4] use concepts of network calculus to formulate a mathematical model of industrial Ethernet, while Shalunov et al. [5] study the properties of half and full duplex Ethernet to devise techniques to detect mismatch between the two modes in a given communication channel and its effects on TCP throughput.

With the emergence of TLM techniques, a number of researchers have applied it to specify existing systems. Cai et al. [6] explain the benefits of the use of TLM techniques. Moussa et al. [7] describe VISTA, a new methodology and tool to analyze SoCs. Klingauf et al. [8] present a generic interconnect fabric for TLM. Wieferink et al. [9] use built-in TLM features of SystemC to propose a methodology for exploring SoC multiprocessor systems. Schirner et al. [10] have proposed some novel techniques to address some of the drawbacks of TLM related to efficient communication modeling. In addition, a number of researchers have applied TLM techniques to analyzing the AMBA bus, mostly with SystemC [14-17]. Bombieri et al.

[18] combine SystemC's TLM features with the ns-2 network simulator to analyze voice-over-IP (VOIP) systems using the AMBA bus.

1.2 Goals

Available literature indicates that the focus so far has been on mostly on higher level concepts/theoretical issues and TLM techniques as applied to the analysis of systems using widely used system busses. In contrast, our focus is entirely on best-effort communication channels, which by definition use distributed bus arbitration. Unlike [18], we do not use any network simulator and create our own specification model for a best effort communication channel. Our main goals, based on TLM and SLDL principles, are to combine the two. Specifically:

- Specification and validation of real-world networked embedded systems based on best-effort communication channels, e.g., Ethernet, WLANs and wireless sensor networks.
- Integration of the resulting specification model into the overall system level design process, i.e., a flexible and robust model of networked embedded devices for prototyping of design alternatives, validation of networking effects and rapid, early network-level design space exploration [20].

Networked embedded systems, and wireless networks in particular, often use best-effort communication channels. TLM, with its ability to separate low-level communication details from actual transferred data provides the best means to understand the overall behavior of such a system. In accordance with TLM principles, we design our own abstract Half-Duplex Ethernet channel, including techniques for handling conflicts amongst devices attempting to use the channel simultaneously. The final specification model can be applied as input to system-level design and synthesis tools.

The remainder of this paper is organized as follows. In the next section we introduce our specification/validation model for Half-Duplex Ethernet created with SpecC, with details of how various SpecC language features were used, along with the TLM principles on which the specification model is based. In Section 3 we test the accuracy and validity of our specification model by describing an experiment and its results to analyze the behavior of a widely-used network quality of service (QoS) protocol that operates on Ethernet frames. Finally, we conclude with a brief summary of work performed and future possibilities.

2 Specification and Validation of Half-Duplex Ethernet

The Half-Duplex Ethernet [11] sub-layer of the ISO/OSI Data Link layer is a best-effort protocol, with active senders and passive receivers. Only one of two communicating devices can be sending data at any time. All transmitters share a common channel with maximum specified bandwidth, and transmitters compete with others to gain control of the channel. A transmitter which gains control of the channel has exclusive rights to send data to a receiver of its choice, and can retain control of the channel for as long as it wants. Failed transmitters must wait and use the Truncated Binary Exponential Back-Off algorithm to decide on the duration. Each failed attempt to gain control of the channel is a 'collision'. Each failed transmitter waits for a duration derived from the slot time and the number of failed attempts to

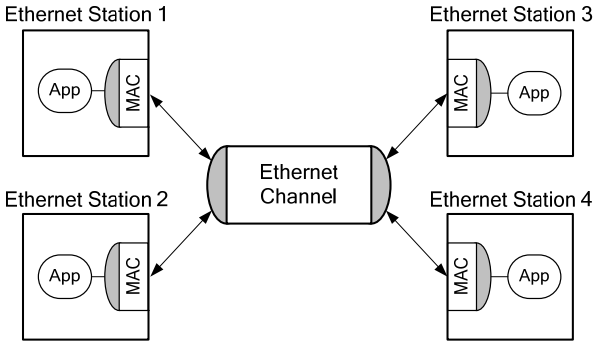

Fig. 1. Basic Ethernet test bench. Four Ethernet stations share a common half duplex Ethernet channel. A media access layer (MAC) in each station implements contention resolution and network access. 'App' is an application exchanging data over the network.

retransmit. After i collisions, a random number of slot times between 0 and $2^i - 1$ is chosen. For the first collision, each transmitter might wait 0 or 1 slot times, and after the second collision, each failed transmitter might wait 0, 1, 2, or 3 slot times. The term 'truncated' indicates that the retransmission timeout has a strict upper bound, e.g., for a ceiling of i=10, the maximum delay is 1023 slot times. A slot time is the round-trip time interval for one ASCII character and is set at 51.2 µs. As transmission delays can cause transmitting stations to collide, a busy network might have hundreds of senders caught in a single collision set. Because of this, after 16 attempts at transmission of one particular frame, the process is aborted.

Our test bench is shown in Fig. 1. Each station can be configured as sender or receiver, and the test setup consists of two senders talking to two receivers. Fig. 2 shows our implementation of the core Ethernet channel. The challenges in modeling the Half-Duplex Ethernet channel are:

- The channel must correctly detect and handle collisions.
- An Ethernet station that has failed to send a frame in the current attempt has to be able to choose the correct wait duration, depending on the total number of failed attempts so far to send this frame (provided that the total number of failed transmission attempts so far do not exceed a pre-defined limit).

The Truncated Binary Exponential Back-Off algorithm is implemented in each station. Ethernet frames are 128 bytes long, with 64 byte header and 64 byte payload.

2.1 Half-Duplex Ethernet Channel

As per SpecC design principles, the `Ethernet` channel implements the `EthernetInterface` interface. Ethernet receivers are passive devices, and each waits for a *dataready* event to read data from the channel. Most of the activity on the channel is when a transmitter tries to send a frame.

We only consider collisions occurring during sending of the Ethernet header. Once an Ethernet station has successfully transmitted the frame header, it gains control of the channel and does not have to check for frame collisions while sending the payload.

```
const unsigned int INTER_FRAME_INTERVAL = 10;
const unsigned int SLOT_TIME = 52;
const unsigned int FRAME_PAYLOAD_INTERVAL  = 3328;

interface EthernetInterface
{
   bool send_frame(unsigned char *, unsigned int);
   void recv_frame(unsigned char *);
};

channel EthernetChannel implements EthernetInterface
{
   unsigned int busy;
   unsigned char localbuffer[128];
   unsigned int i;
   bool collision;
   event dataready;

   bool send_frame(unsigned char *frame,
                               unsigned int stationID)
   {
      while(busy == 2)
         waitfor(SLOT_TIME*64 + INTER_FRAME_INTERVAL);
      if(busy == 1) {
         collision = true;
         return false;
      }
      busy = 1;
      for(i = 0; i < 64; i++) {
         waitfor(SLOT_TIME);
         if(collision) {
           collision = false;
           busy      = 0;
           return false;
         }
      }
      busy = 2;
      waitfor(FRAME_PAYLOAD_INTERVAL);
      memcpy(localbuffer, frame, 128);
      busy = 0;
      collision = false;
      notify(dataready);
      return true;
   }

   void recv_frame(unsigned char *recvframe)
   {
      wait(dataready);
      memcpy(recvframe, localbuffer, 128);
   }
};
```

Fig. 2. SpecC behavior implementing the basic Half-Duplex Ethernet channel.

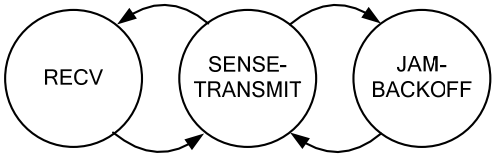

Fig. 3. Ethernet station media access layer state machine.

In SpecC, only behaviors are associated with threads, channels are passive. If a behavior calls a channel method it is a regular function call. Any code in the channel (including waitfor statements) is executed in the context of the calling behavior/thread.

Now, let an Ethernet station *A* want to send a frame. It invokes the *send_frame* function of the Ethernet channel:

1. *A* checks if the channel's *busy* variable has value 2, which indicates that another Ethernet station is sending its payload. *A* waits in a loop until *busy* is not equal to 2. During each iteration of the loop, it waits for a time period equal to that required to send a 64 byte payload, plus the mandatory inter frame gap.
2. If instead *A* finds that the *busy* variable has value 1, then a collision has just occurred. If not, *A* sets the *busy* variable to 1, and starts to send the 64 byte Ethernet header of the current frame, checking for a collision after sending each header byte.
3. When *A* has successfully sent the Ethernet header, it sets the *busy* variable to 2, indicating it has acquired complete control over the channel, and starts sending that Ethernet frame payload. On completion, the frame payload contents are copied into a local channel buffer. *A* resets the status variable *busy* to 0, and sets an event (*dataready*) variable to indicate to all receivers that a frame is available. It is now ready to send/receive any frame to/from any station.

2.2 Media Access Layer

Each Ethernet station is a finite-state machine, with three possible states, JAM_BACKOFF, RECV and SENSE_TRANSMIT, as shown in Fig. 3:

1. Each Ethernet station starts in the SENSE_TRANSMIT state. It invokes the *send_frame* function of the Ethernet channel, which returns a Boolean true if the frame was sent successfully.
2. If the return value is false, the station transitions to the JAM_BACKOFF state. The maximum number of times any station might attempt to re-send a frame is 16. In the JAM_BACKOFF state, the station first waits for a mandatory jam period. It then decides, using the Truncated Binary Exponential Back-Off algorithm and the number of failed attempts so far how long to wait before the next transmission attempt. At the end of the wait period, the Ethernet station transitions to the SENSE_TRANSMIT state and attempts to send that frame again. If the Ethernet station finds in the JAM_BACKOFF state that the maximum number of transmit attempts for the current frame has exceeded the pre-defined maximum limit, it drops the frame and transitions back to the SENSE_TRANSMIT state in order to send or receive the next frame.

3. The Ethernet station transitions between the SENSE_TRANSMIT and RECV states to send the next frame or to receive frames available on the channel, respectively.

Our specification models for both Half-Duplex Ethernet channels and stations adhere strictly to TLM design concepts and make extensive use of SpecC's detailed time construct, which allows time to be simulated in two nested loops, an outer time loop and inner one for events in each simulation step, called the delta cycle. In addition, the Ethernet channel behavior exploits SpecC's event mechanism to notify receivers when data is available for them.

In accordance with TLM principles, each Ethernet station invokes the *send_frame* function of the Ethernet channel when attempting to send a frame. The channel internally tackles the frame collision and only informs the transmitter if one has occurred (by returning a false value). The transmitter in turn can then decide how long to wait before attempting to retransmit again. The Ethernet channel's underlying data transfer mechanism is transparent to the transmitter.

3 Experiments

We devised a set of experiments with increasing levels of complexity to determine if our specification model for Half-Duplex Ethernet meets design goals. We define the average delay for an Ethernet station (transmitter or receiver) as:

- The average delay for a transmitter is the time interval (averaged over 1000 successful frame transmissions) between the station starting to send a frame (in SENSE_TRANSMIT state) and returning to same state to send the next frame.
- The average delay for a receiver is the time interval (averaged over 1000 successful frame receptions) between the station receiving a frame and it returning to the same state (RECV) to receive the next frame.

As required for the Truncated Binary Exponential Back-Off algorithm, the wait periods after a collision in each Ethernet station are strictly bound between lower and upper limits.

3.1 Channel Model

To simulate realistic network conditions, our specification model includes bursty traffic generators [13]. Bursty traffic is an infinite sequence of frames with sub-sequences of closely spaced (in time) frames interspersed with sub-sequences of widely spaced (in time) frames, i.e., a plot of frames over time shows peaks and plateaus. Bursty traffic has a long tailed (power law) probability distribution and is typically modeled using a Poisson Pareto Burst Processes with heuristics to enable a close fit to observed data. To circumvent the issue of having to choose correct heuristics, a simple power law distribution is used in our setup.

The effect of adding the power law distributed delays is to increase the average delay in all cases when this delay interval is non-zero, see Fig. 4. This is because the number of collisions increases with the incoming frame rate (bursty traffic). In contrast, when the power law distributed delay interval is zero, the average delay has approximately the same value as if this additional delay is not present at all.

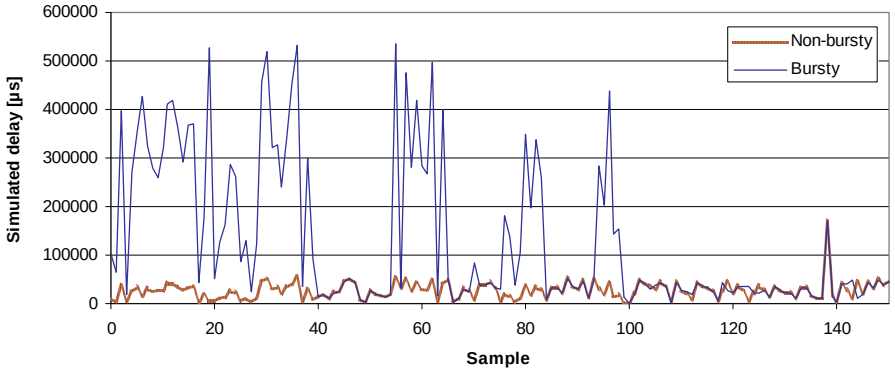

Fig. 4. Bursty and non-bursty average transmitter delays.

3.2 QoS Application

The next phase of our experiments involves creating specification models for a widely used network quality-of-service (QoS) algorithm that works at the Ethernet layer. QoS algorithms share some common characteristics:

1. Always applied to the interfaces of a router, e.g., WAN and LAN interfaces.
2. QoS features work on the producer-consumer model and rely on non-deterministic queues.

Some common QoS operations on network traffic are:

1. Shaping – delay frames/packets to meet a certain rate
2. Scheduling – rearrange frames/packets for output
3. Classifying – separating traffic into queues
4. Policing – measuring and limiting traffic on queues

With these in mind, the simulated QoS architecture is shown in Fig. 5. One sender and one receiver on each side of a router exchange frames with a receiver and sender on the other, respectively. The Token Source/Token Channel pair for each *EthernetHandler* behavior implements the chosen QoS algorithm, as will be explained shortly.

The router interconnects the two networks consisting of Ethernet channels 1 and 2, where behaviors *EthernetHandler1* and *EthernetHandler2* transfer frames between channels 1 and 2 via router interfaces 1 and 2. QoS algorithms/features are imposed on the router via the Token Source/Token Channel combination for each *EthernetHandler*. Internally, each *EthernetHandler* behavior is an Ethernet station with two ports, one dedicated to transferring frames originating in network 1 to network 2 and vice versa.

The Token Channel is a non-deterministic custom queue that, in addition to blocking reads and writes, allows the user to check if it is empty – a feature unavailable for any built-in SpecC queue. We define the average delay for a router interface as:

- The time interval (averaged over 1000 successful attempts) between the router interface receiving a frame successfully, sending it out over the other Ethernet channel, and returning to the state where it can receive the next frame.

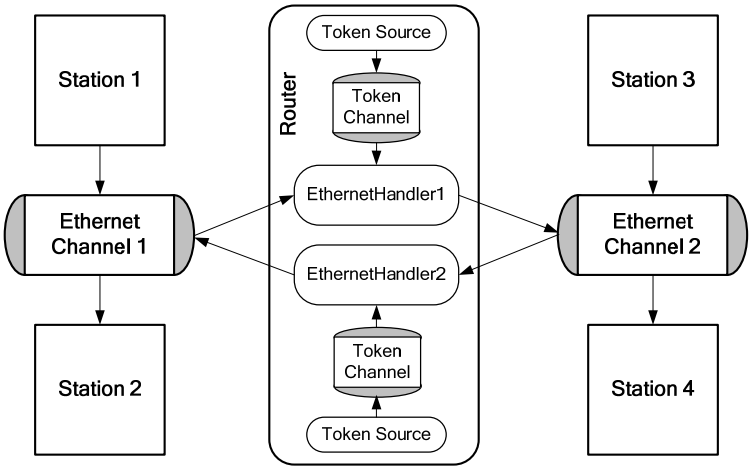

Fig. 5. Simulated Quality of Service architecture.

This is completely independent of the average delay for the basic Half-Duplex Ethernet.

3.2.1 Random Early Detection

Random Early Detection (RED) [12] is used for congestion control and manages queue size intelligently (Fig. 6). Unlike regular queues that drop packets from the tail when they are full, RED does it in a controlled and gradual way.

Once the queue size attains a certain average length, enqueued packets have a finite probability of being marked. A marking probability exceeding a predefined threshold means that the marked packet will be dropped. The marking probability increases linearly with the queue size up to a *maximum dropping probability*. The average queue size used for determining the marking probability is calculated using an Exponential Weighted Moving Average, insensitive to bursts.

When the average queue size is below a preset minimum bound, no packet is marked. When the average queue size exceeds the minimum queue length, the marking probability increases linearly until the average queue size attains the preset maximum queue length. As probability is normally not set to 100%, the queue size might rise above the maximum preset size. Hence, a limit parameter is provided to set a hard maximum for the size of the queue.

3.2.1 Experimental Validation

For the purposes of this experiment, the two Token Source behaviors supply marking probabilities to the two *EthernetHandler* behaviors via the token channels. Both token channels are non-deterministic and non-blocking. Each *EthernetHandler* behavior can thus check if a token (marking probability) is available before trying to extract one. Our implementation of the Token Source/Token Channel pair uses the same functions and parameters as in [12] to generate the marking probabilities. This allows us to compare the results generated by our model (specifically queue length and average queue length) with the original ones in [12].

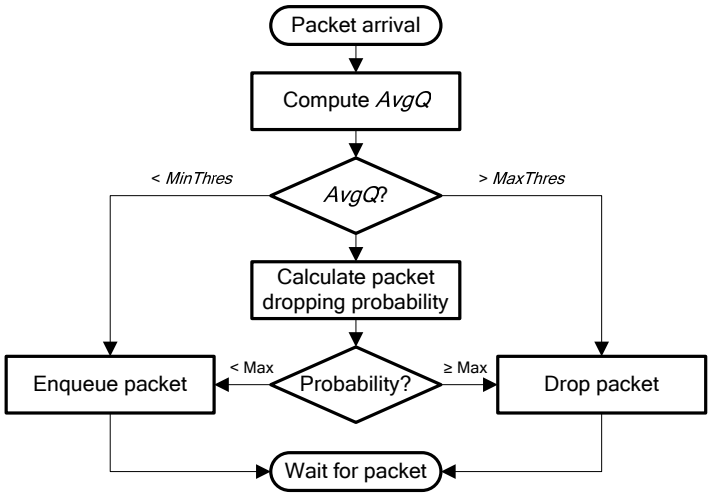

Fig. 6. Random Early Detection (RED) congestion control (*AvgQ*: average queue length, *MaxThres*: maximum queue length threshold, *MinThres*: minimum queue length threshold).

RED uses a number of predefined and computed parameters. The predefined parameters are *maximum dropping or marking probability*, *minimum* and *maximum queue lengths* and *queue weight*. The parameters computed per iteration are *count*, *average queue length*, *queue length* and *dropping or marking probability*. Count is the number of frames since the last marked frame. For our simulation, we used the same values for the predefined parameters as in [12]. In addition, the algorithm uses a linear function of time to determine the time interval since the queue was empty. In our case, we use simple difference in measured times to achieve this effect.

As the average queue length varies between the minimum and maximum thresholds, the packet marking probability varies between 0 and the maximum probability. The final marking probability increases linearly as the count since the last marked packet grows. For each frame that is to be sent out over the Ethernet layer, the average queue length is computed as in [12]. Fig. 7 represents results for queue size and average queue size sampled every 1000 successfully transmitted frames for the first 1000 samples at one of the two router interfaces we implemented using SpecC. All together, simulation of more than 3 million frames successfully transmitted over both router interfaces required 15 minutes of simulation time on a 2.8 GHz Linux workstation.

4 Summary and Conclusions

Embedded devices are being increasingly deployed in networked environments, often communicating via best-effort channels, e.g., in wireless sensor networks. Using TLM techniques, we have specified and validated a networked embedded system in which devices communicate via a shared best-effort channel, specifically the Half-Duplex Ethernet sub-layer of the ISO/OSI Data Link layer. Our specification model can be easily integrated into the system level design process, using any appropriate

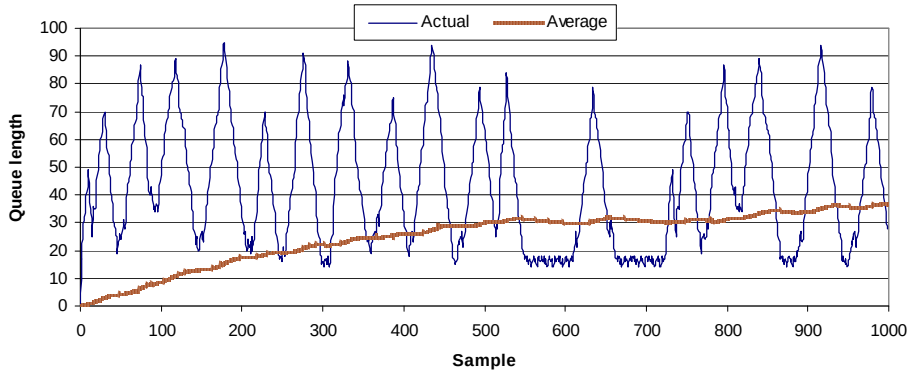

Fig. 7. Queue length and average queue length.

toolset for exploration, prototyping and evaluation of design alternatives. To test if our specification model replicates reality, we have validated it using a multi-station bursty traffic scenario and a widely used network QoS protocol that operates at the Ethernet layer. In the future, we plan to deploy our Ethernet channel for modeling of various realistic, large-scale networked systems. In addition, future directions include analysis, customization and optimization of QoS algorithms for applications in typical resource constrained networked embedded system.

References

1. Weinberg, H. B., and Zuck, L. D.: Timed Ethernet: Real-Time Formal Specification of Ethernet, Lecture Notes in Computer Science, vol. 630, pp. 370-385, 1992.
2. Bochmann, G., and Sunshine, C.: Formal Methods in Communication Protocol Design, IEEE Transactions on Communications, vol.28, no.4, pp. 624-631, April 1980.
3. Schmaltz, J., and Borrione., D.: A Functional Approach to the Formal Specification of Networks on Chip, Lecture Notes in Computer Science, vol. 3312, pp. 52-66, 2004.
4. Georges, J.-P., Rondeau, E. and Divoux, T.: Evaluation of Switched Ethernet in an Industrial Context using Network Calculus, 4th IEEE International Workshop on Factory Communication Systems, Vasteras, Sweden, August 2002.
5. Shalunov, S. and Carlson, R.: Detecting Duplex Mismatch on Ethernet, Lecture Notes in Computer Science, vol. 3431, pp. 135-148, 2005.
6. Cai, L., and Gajski, D.: Transaction Level Modeling: An Overview, Proceedings of the 1st International Conference on Hardware/Software Codesign and System Synthesis, 2003.
7. Moussa, I., Grellier, T. and Nguyen, G.: Exploring SW Performance using SoC Transaction-Level Modeling, Design, Automation and Test in Europe, 2003.
8. Klingauf, W., Günzel, R., Bringmann, O., Partfuntseu, P. and Burton, M.: GreenBus: A Generic Interconnect Framework for Transaction Level Modeling, Design Automation Conference, 2006.
9. Wieferink, A., Kogel, T., Leupers, R., Ascheid, G., Meyr, H., Braun, G. and Nohl, A.: A System Level Processor/Communication Co-Exploration Methodology for Multiprocessor System-on-Chip Platforms, Design, Automation and Test in Europe, 2004.
10. Schirner, G., and Doemer, R.: Fast and Accurate Transaction Level Models using Result Oriented Modeling, International Conference on Computer Aided Design, 2006.
11. Metcalfe, R. M., and Boggs, D.R.: Ethernet: Distributed Packet Switching for Local Computer Networks, Communications of the ACM, vol. 19, no. 7, pp. 395-404, 1976.

12. Floyd, S. and Jacobson, V.: Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transaction on Networking, 1993.
13. Karasaridis, A. and Hatzinakos, D.: Network Heavy Traffic Modeling using Alpha-Stable Self-Similar Processes, IEEE Transactions on Communications, vol. 49, no. 7, pp. 1203-1214, July 2001.
14. Caldari, M., Conti, M., Coppola, M., Curaba, S., Pieralisi, L., and Turchetti, C.: Transaction Level Models for AMBA Bus Architecture Using SystemC, Design, Automation and Test in Europe: Designers' Forum, 2003.
15. Schirner, G., and Doemer, R.: Quantitative Analysis of Transaction Level Models for the AMBA Bus, Design, Automation and Test in Europe, 2006.
16. Pasricha, S., Dutt, N. and Ben-Romdhane, M.: Extending the Transaction Level Modeling Approach for Fast Communicating Architecture Exploration, Design Automation Conference, 2004.
17. Xu, S. And Pollit-Smith, H.: A TLM Platform for System-on-Chip Simulation and Verification, VLSI Design, Automation and Test, April 2005.
18. Bombieri, N., Fummi, F., and Quaglia, D.: TLM/Network Design Space Exploration for Networked Embedded Systems, International Conference on Hardware/Software Codesign and System Synthesis, 2006.
19. Andrews, M., Kumaran, K., Ramanan, K., Stolyar, A., Whiting, P., and Vijaykumar, R.: Providing Quality of Service Over Shared Wireless Link, IEEE Communications, February 2001.
20. Bonivento, A., Carloni, L. and Sangiovanni-Vincentelli, A.: Platform-Based Design for Wireless Sensor Networks, Mobile Networks and Applications, vol. 11, no. 4, August 2006.
21. Gajski, D., Zhu, J., Doemer, R., Gerstlauer, A. and Zhao, S.: SpecC: Specification Language and Methdology, Kluwer, 2000.
22. The Network Simulator ns-2, `http://www.isi.edu/nsnam/ns`.
23. OPNET Technologies, Inc.: OPNET Modeler, `http://www.opnet.com`.
24. OMNet++, `http://www.omnetpp.org`.