A DISCRETE-EVENT SYSTEMS MODEL FOR CONGESTION CONTROL [1]

Kurt R. Rohloff * **Tansu Alpcan** * **Tamer Başar** *

* *Coordinated Science Laboratory*
The University of Illinois
1308 West Main St.
Urbana, IL 61801, USA
{*krohloff,alpcan,tbasar*}*@control.csl.uiuc.edu*

Abstract: This paper presents a discrete-event systems approach to the modeling of a packet-switched communication link shared by multiple users for the purpose of end-to-end congestion control. It introduces a discrete-event system model for the interaction between a shared communication link and the users, which captures the behavior that the link users receive acknowledgments for successfully transmitted packets with a delay that is proportional to the level of congestion in the link. An end-to-end congestion control scheme for this system model is presented that uses the concept of an observer from supervisory control theory. The link model and controllers are implemented in both a Java programming language simulation and the NS2 network simulation software for analysis. *Copyright© 2005 IFAC*

Keywords: Discrete-Event Systems, Queues, Networks, Supervisory Control, Decentralized Control Systems.

1. INTRODUCTION

In packet-switched communication networks, such as the Internet, a user injects packets of data into the network, which are sent over transmission links to a receiver. Some packets may be lost during the transmission, and thus, to keep track of these losses, acknowledgments are returned to the originating sender as incoming data is processed by the receiver,. In order to guarantee that all transmitted data is eventually communicated successfully, a sender will retransmit data until he receives positive acknowledgments for all the packets that need to be sent.

It is common for a communication link to receive packets from users at a faster rate, at least occassionally, than it can retransmit them. When this happens, the packets sent to the link are placed on a processing queue for eventual transmission. Packets sent to a full queue are dropped, indirectly causing a decrease in the link's throughput due to the necessity of retransmitting the dropped packets. A key attribute of the congestion control problem that the model presented herein attempts to capture is that the acknowledgment delay of a transmitted packet is proportional to the level of congestion in the network. By observing delays in acknowledgments for transmitted packets, users could locally diagnose link congestion and therefore take action to decrease their sending rates and lower the overall congestion. This process is called end-to-end congestion control and has been receiving much attention from both the networking and control communities (Kurose and Ross (2005); Srikant (2003)).

[1] This research was supported in part by the NSF Grant CCR 00-85917 ITR.

The most widely used models for the analysis of how users interact with a packet-switched network has been fluid flow and queuing models Although the level of abstraction in these models can be useful for modeling certain aspects of network behavior, these models do not capture the effects of the dependence of a packet's acknowledgment delay on the level of congestion.

With this in mind, the main goal of this paper is to propose a model of the interactions between a shared communication link and its users at a lower level of abstraction that captures the behavior that the acknowledgment delays observed by users are proportional to link congestion. The measure of link congestion used in this paper is the level of the link's processing queue. The discrete-event systems modeling framework (Cassandras and Lafortune (1999)) is natural for this low-level analysis due to the underlying discrete nature of packets. This paper specifically explores the modeling of multiple users sharing single communication links an important special case of general networks.

To the best knowledge of the authors, there is no discrete-event systems model to capture possible variable acknowledgment delay behavior in communication networks. There has been some work in other modeling frameworks to use acknowledgment delay observations in congestion control policies. Among these, Brakmo and Peterson (1995) present the widely used TCP Vegas policy, and Alpcan and Başar (2003) present a game theoretic approach to congestion control.

This paper is structured as follows. The next section gives an overview of the properties possessed by the new link model. Section 3 presents an implementation of the model. An online supervisory control method for the link model is given in Section 4. The paper closes with a discussion of the results in Section 5. Due to the necessary brevity of this paper a comprehensive introduction to discrete-event systems or network congestion control cannot be given, but the reader may consult Cassandras and Lafortune (1999) or Kurose and Ross (2005) for the necessary background.

2. SYSTEM PROPERTIES

The model in this paper assumes a quantized time interval with respect to a global clock tick T such that the link can process and transmit one packet per interval. For reasons of simplicity it is assumed that packets sent over the link are of uniform size, the users' transmission capacities are equal, and one packet can be transmitted per user per time interval. The model should also capture the behavior that there might be processing and transportation delays associated with using the link. It is also assumed that the sum of these delays is an integer multiple d of the time quantization interval T. These assumptions are made in order to develop a relatively simple model that exhibits variable feedback delay behavior.

Suppose the link's processing queue holds $m < q$ packets where q is the maximum number of packets that can be held. If a user were to then send a packet to be transmitted over the link, in $m + d$ time steps the user should observe an acknowledgment that the packet was received. This captures the behavior that the link has an associated acknowledgment reception delay which is proportional to the congestion level. Once a packet is transmitted over the link, it is assumed that packet acknowledgments are transmitted without loss as the difficulties associated with the queuing of acknowledgments is outside the desired scope of this model. This is a common assumption made in the congestion control literature due to the fact that acknowledgment packets are relatively small and do not generally cause congestion on feedback channels.

For the discrete-event system model of the link and user interaction presented here, all of the system modules have three distinct operating modes which the modules cycle through during the course of operation. Transitions between operating modes progress synchronously in all modules on the occurrence of global *signaling* events.

After initialization, the first mode of operation is the On/Off mode, where each user is allowed to initiate or terminate communication sessions with the link depending on whether the user has packets to send. A user is "On" if it has a sequence of packets that should be sent over the link, while the user is "Off" otherwise. The system enters the local data transmission mode on the occurrence of the global event *Insert*. For this mode of operation the users are allowed to send packets to the link that are to be transmitted. These packets are accepted or dropped by the link depending on the level of the link's processing queue. The global signal *Done* signifies the completion of sending packets, and the model then enters the acknowledgment return mode of operation. During this mode of operation an acknowledgment is sent to a user for a previously transmitted packet if there are any pending. On the occurrence of a global clock tick T, the system returns to the On/Off mode of operation and the process iterates indefinitely.

2.1 Model Operation

The interaction of the users with the link can be thought of as a modified queue of size $q+d$, where when a packet is sent to a link with an empty processing queue, a token is placed on the model queue at level d. Every time interval, if the head of the queue is not empty, a token is pulled from the model queue, returned to the user that had originally sent it, and the contents of the model queue are shifted forward one queue cell. However, if the head of the model queue is empty, all of the cells of the queue containing tokens shift their tokens forward one cell. Therefore, in d time steps, the user will receive an acknowledgment for the packet it had sent. In this manner, if packets sent to the link are modeled as putting tokens on the model queue at the lowest open cell above level d, then when the link processing queue is at level m, the model queue is at level $m + d$. Due to the nature in which tokens are pulled from the queue, users that are modeled as sending packets to the link when the processing queue is at level m are also modeled as receiving acknowledgments in $m + d$ time steps.

A flow diagram of the model operation can be seen in Figure 1 for a system with two users, a processing and transportation delay of 1 and a queue capacity of 2 packets. In the diagram, the queues flow down, and the shaded queue cells correspond to the processing and transportation delay.

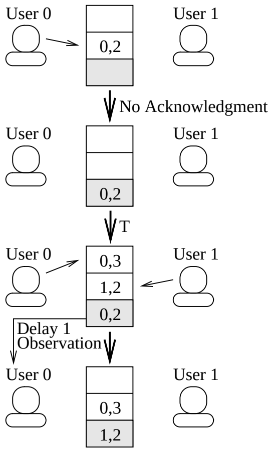

Fig. 1. An overview of the model operations.

In the top sub-diagram of Figure 1, the model is in the local data transmission mode of operation and User 0 sends a packet to the link. A token corresponding to this packet is placed in Cell 1 of the model queue (if the bottom cell of the queue is Cell 0) because this is the first open cell into which tokens can be inserted. Each token retains information about which user originally sent it so acknowledgments can eventually be sent to the appropriate user. Furthermore, the level of the queue when the token was placed in it is also stored with the token so the user will observe the proper acknowledgment delay. By storing acknowledgment delay information on the queue tokens rather than with the users the model avoids the difficulty of having the users retain internal memory of when packets were sent to the link, consequently simplifying the model. In the top sub-diagram of Figure 1, the token placed in the model queue contains the information that it came from User 0 and was originally inserted into the queue at Cell 1.

An acknowledgment return mode of operation can be seen in the transition between the first and second sub-diagrams of Figure 1 where the model enters the acknowledgment return mode of operation, but the head of the queue is empty (that is, the shaded queue cell). Therefore, no acknowledgments are sent to any user during the transition, but the tokens in the queue are passed down one cell.

From the second sub-diagram, the time-step T occurs which signals the beginning of the next cycle of operation in the model. In the third sub-diagram, User 1 first sends a packet to the link, followed by User 0, at levels 1 and 2 respectively. With the transition to the final sub-diagram, the model re-enters the acknowledgment return mode of operation, and User 0 observes that the packet it sent had an acknowledgment delay of 1.

3. MODEL OVERVIEW

This section presents an implementation of the discrete-event systems shared link model outlined above for a system with two users, a processing and transportation delay of 1 time step and a link queue capacity of 2 packets. To avoid some of the state explosion difficulties associated with this system, the model is presented in a modular manner where each module is a discrete-event system that captures a specific aspect of the interaction between the link and the users. The system modules can be combined to form a global system model through the use of the parallel composition operation. The parallel composition operation is currently the standard method to model the actions of interacting modules in distributed discrete-event systems.

For the model of the link and user system, there is a module to represent how each user sends packets to the link, seen in Figure 2 for the case of User 0. For User i, the occurrence of the events *Oni* and *Offi* represent that the user is entering the "On" mode or "Off" mode, respectively. The global event *Insert* signals that the users can send a packet to the link. The event *Off_Nothingi* represents that User i has nothing to send to the link because it is off, while the event *Nothingi* represents that User i has nothing to send to the link because it chooses not to. The event *Inserti* represents that User *i* would like to send a packet to the link. The event *Inserti_j* signals that a token is placed from User *i* onto the model queue at level *j* and *Lossi* signals that there is a packet from User *i* that was lost.

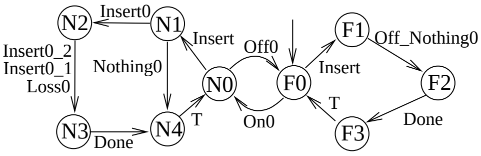

Fig. 2. A model of the interaction of User 0 with the communication link.

Figure 3 shows the system modules that represent each cell of the model queue as discussed for Figure 1 above. Besides the events relevant to the insertion of tokens on the queue described above, there are two classes of events relevant to the operation of the model queue. The *Return* events are the events that occur when a token is pulled from the queue and an acknowledgment needs to be signaled to a user. The event *Returni_j* represents that an acknowledgment is being returned to User *i* with an observed delay of *j*, while *ReturnE* represents that no acknowledgments are returned to any user. The *Pass* events are internal signaling events for the queue operation that are used to coordinate how tokens in the queue are passed from one queue level to the next. For the passing of tokens from queue level 2 to queue level 1, the event *Pass21_i_j* represents that a token corresponding to User *i* is being passed from Cell 2 to Cell 1 with an associated delay of *j*, while *Pass21_E* represents that no token is being passed from Cell 2 to Cell 1.

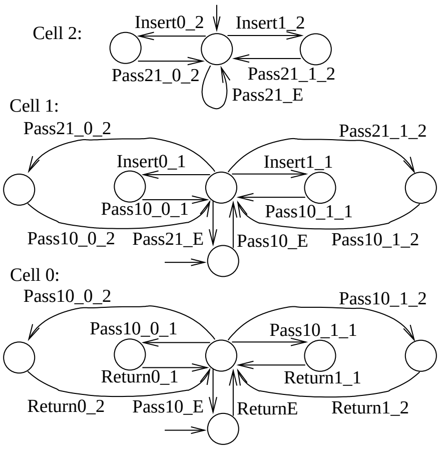

Fig. 3. A model of the queue modules to process variable delays for acknowledgment transmissions.

The module to coordinate the passing of the tokens in the queue cells can be seen in Figure 4. This module ensures that tokens are returned from the queue during the correct mode of operation and the passing of tokens from one queue level to the next occurs in the proper order. That is, when an event is taken from the model queue, a *Return* event occurs first, then a *Pass10* event, then a *Pass21* event and so on.

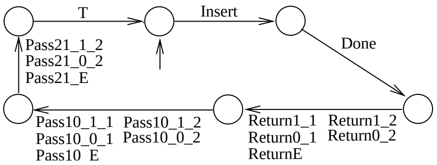

Fig. 4. The module to order queue behavior.

The module to coordinate the level at which tokens are placed on the queue can be seen in Figure 5. This module tracks which of the queue cells should accept a new token. This is also the module that determines which of the events *Inserti_j* or *Lossi* should occur depending on the queue level.

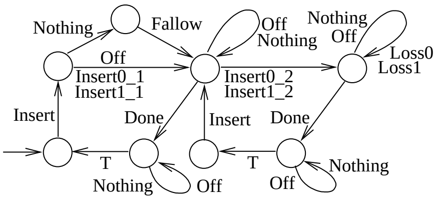

Fig. 5. The module to track occurrences of packet transmissions.

Finally, the module to track if none of the users sends any packets to the link can be seen in Figure 6. The event *Off* occurs if all users are off during the current time interval, while the event *Nothing* occurs if all users that are on decide not to send packets. If *Nothing* occurs, then the event *Fallow* in Figure 5 could occur if the processing queue of the link is empty. This means that the link is being underutilized even though at least one user desires to send packets to the link.

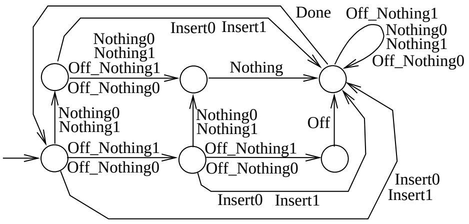

Fig. 6. The module to track if the users do nothing.

4. CONTROL OPERATIONS

Now that the discrete-event systems model for the interaction between a shared communication link and a set of users has been introduced, next properties of performing congestion control operations with the model are discussed in the decentralized supervisory control framework. A set of decentralized controllers make independent observations of event occurrences in a system and use these local observations to calculate a local control action. When combined, the local control actions should achieve a given control objective. For the system outlined above, every User i has a control module S_i to decide if the user should send a packet in a given time interval controlling the events *Inserti* and *Nothingi*. It is assumed that User i can observe the occurrence of *Oni*, *Offi*, *Inserti*, *Nothingi*, *Off_Nothingi* and all packet acknowledgment events *Returni_j*.

The control objective for link model outlined above is, in a sense, to maximize packet throughput on the link such that as few packets as possible are lost, while at the same time the processing queue of the link should never under-run when there are users in the On mode of operation. For the system model introduced above, this means that no event *Lossi* or *Fallow* should occur.

There are several salient properties that put this problem outside the normal scope of the supervisory control framework. For one, it has no physical meaning for a controller to disable the controllable events *Inserti* and *Nothingi* at the same time. On a given time interval, a user may send a packet or not, but not neither. Furthermore, in some situations it may be impossible to totally avoid the occurrence of *Lossi* or *Fallow* events, so the objective of the control system should be to allow as few occurrences of these events as possible. Unfortunately there is no universally accepted notion of controller optimization defined in the supervisory controls framework that would allow for the minimization for event occurrences.

An overriding difficulty associated with controlling the link model is the extremely large state space when the system modules described above are composed. This problem is unavoidable when modeling memory systems such as queues in a discrete-state framework. It was found that even a system with a link processing queue of 5 packets and a processing and transmission delay of 1 time unit has a corresponding composed model with over 20957 states. This realization shows that offline exhaustive precomputation of control strategies for all possible scenarios is prohibitively expensive. A method to avoid the precomputation of control strategies would be to use online methods as in Hadj-Alouane et al. (1996) where control actions are computed on the fly, but may not be optimal.

One such online decentralized control policy is now discussed which is based on a decentralized state estimation policy. Let $G = (X, x_o, \Sigma, \delta)$ be the underlying automaton representation of the model as introduced above. Suppose that Σ^i_{uo} is the set of events unobservable to User i, and $\Gamma^i(s)$ is the set of events that the controller for User i disables after s is observed in the system. The local control action $\Gamma^i(s)$ is commonly a function of the state estimate for User i, $X^i_e(s) \subseteq X$. The state estimate $X^i_e(s)$ is the set of states the User i believes the system could be in after s has been observed.

Define $\Sigma_i(s) = \Sigma_{uoi} \setminus \Gamma^i(s)$, the set of locally unobserved and enabled events. The state estimation $X^i_e(s)$ is defined recursively as follows:

$$X^i_e(s) = \begin{cases} \{\delta(x_0, t) | t \in \Sigma_i(s)^*\} & \text{if } s = \epsilon. \\ \{\delta(x, t) | x \in X^i_e(s'), t \in \Sigma^*_i\} & \text{otherwise.} \end{cases}$$

Note that the state estimator, $X^i_e(s)$, is an exact state estimator in the sense that a state is in $X^i_e(s)$ if and only if the system could be in that state due to the observation of s. The set $X^i_e(s)$ is computationally intensive to find, but is used here to explore the best-case possibilities of a supervisory control approach to congestion control. Also, $X^i_e(s)$ is implicitly a function of the control operation $\Gamma^i(\cdot)$.

Consider a string of observed events $s'\sigma$. The set $X^i_{uc}(s'\sigma)$ represents all states the controller estimates the system could be in if a local packet transmission occurs after σ that is lost.

$$X^i_{uc}(s'\sigma) =$$
$$\{\delta(x, t) | x \in X^i_e(s'), t \in \sigma\Sigma^*_{uoi} Inserti \Sigma^*_{uoi} Lossi\}$$

If $X^i_{uc}(s'\sigma)$ is non-empty, then by enabling *Inserti* after observing σ, the controller might allow *Lossi* to occur for the given state estimate.

Now consider the module in Figure 7, which tracks the packet transmissions and receptions of User i. User i is expecting no acknowledgments if this module is in state *H*. This module is used in the following online control algorithm.

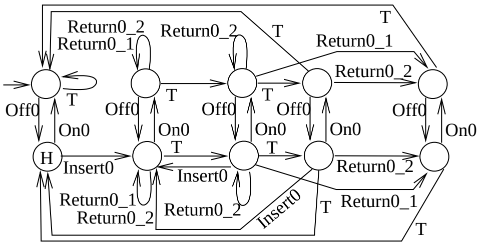

Fig. 7. Module to track outstanding local packets.

Algorithm 1. *O*nline Control Algorithm:

```
Set $s := \epsilon$. Repeat the following indefinitely:
If observable event $\sigma$ occurs:
  Update $s := s\sigma$. Update $X^i_e(s)$, $X^i_{uc}(s)$ and tracker module.
  If tracker module is in state $H$:
    Enable Inserti, disable Nothingi.
  Else:
    If $X^i_{uc}(s'\sigma) = \emptyset$:
      Enable Inserti, disable Nothingi.
```

```
Else:
  Enable $Nothingi$, disable $Inserti$.
```

The underlying idea behind Algorithm 1 is that if the local user is not expecting any packets to be acknowledged, then it should transmit a packet. This probing action ensures the user will always eventually receive some information about the congestion level in the network and maintain a local state estimate. If the local user is expecting a packet to be acknowledged, it should transmit a packet if it does not estimate a loss to be possible by testing for emptiness in $X^i_{uc}(s'\sigma)$.

The system model and controllers just described were implemented using both the Java DES toolkit (Rohloff (2004)) and the standard NS2 simulation tool (UCB et al.). The Java DES toolkit model allows for a simulation of the discrete-event system exactly as outlined in the models above. Due to the computational difficulty of fully exploring the state space of the controlled system, a random walk model was used to simulate the behavior of the link model described above using the online heuristic control policy for a system with two users, a processing and transportation delay of 1 time step and a link queue capacity of 2 packets. One outcome of this simulation was that no packets were lost at the link even though the system has a high potential for congestion due to the fact that the number of users is high compared to the size of the link processing queue. This somewhat surprising result is due to the exact and computationally intensive nature of the state estimation function $X^i_e(\cdot)$ which gives the controller the best possible information about the link's state. Also, the control protocol is conservative in that it avoids packet loss whenever possible.

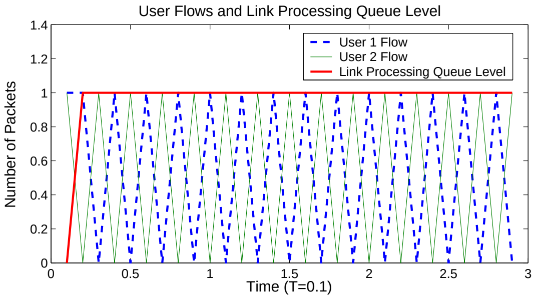

Fig. 8. Results of the NS2 simulation.

The online supervisory controllers outlined above were then translated into the NS2 simulation tool framework (UCB et al.) which assumes a continuous-time system model. NS2 is one of the most widely used packet level network simulation tools. The output of the simulation for the system discussed above can be seen in Figure 8. It was found that the control method is very effective in keeping the queue at a constant level, and no packet losses were observed.

Unfortunately, the nature of the state estimation policy utilized by the online controllers is such that the controllers need full information about all possible behaviors in the system. This means that the controllers should have exact knowledge about the size of the link's processing queue and the round-trip processing and transportation delay. This information is not always available to a congestion controller, but research is ongoing to circumvent these preconditions.

5. DISCUSSION

This paper has introduced a new low-level discrete-event systems model for the interaction of a communication link with multiple users. This model captures the behavior that the acknowledgment delay for the communication link is variable with the level of congestion in the link. An online control scheme is presented for this system that can be used to avoid link congestion and consequently packet loss. One of the messages of this study that supervisory control methods can be useful for the analysis of congestion control.

REFERENCES

T. Alpcan and T. Başar. A utility-based congestion control scheme for internet-style networks with delay. In *Proc. IEEE Infocom*, San Francisco, CA, April 2003. (Also, IEEE Trans. Networking, 2005. To appear.).

L. S. Brakmo and L. L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1999.

N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithms for online synthesis of maximal control policies under partial observation. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 6:379–427, 1996.

J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison Wesley, Boston, second edition, 2005.

K. Rohloff. The discrete-event systems toolkit for Java, 2004. Preprint.

R. Srikant. *The Mathematics of Internet Congestion Control.* Birkhäuser, Boston, first edition, 2003.

UCB, LBNL, and VINT. Network simulator NS (version 2). http://www.isi.edu/nsnam/ns/.