Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2007, Article ID 52492, 16 pages
doi:10.1155/2007/52492

Research Article

TCP Traffic Control Evaluation and Reduction over Wireless Networks Using Parallel Sequential Decoding Mechanism

Khalid Darabkh[1] and Ramazan Aygün[2]

[1] *Electrical and Computer Engineering Department, University of Alabama in Huntsville, Huntsville, AL 35899, USA*
[2] *Computer Science Department, University of Alabama in Huntsville, Huntsville, AL 35899, USA*

Received 12 April 2007; Accepted 9 October 2007

Recommended by Sayandev Mukherjee

The assumption of TCP-based protocols that packet error (lost or damaged) is due to network congestion is not true for wireless networks. For wireless networks, it is important to reduce the number of retransmissions to improve the effectiveness of TCP-based protocols. In this paper, we consider improvement at the data link layer for systems that use stop-and-wait ARQ as in IEEE 802.11 standard. We show that increasing the buffer size will not solve the actual problem and moreover it is likely to degrade the quality of delivery (QoD). We firstly study a wireless router system model with a sequential convolutional decoder for error detection and correction in order to investigate QoD of flow and error control. To overcome the problems along with high packet error rate, we propose a wireless router system with parallel sequential decoders. We simulate our systems and provide performance in terms of average buffer occupancy, blocking probability, probability of decoding failure, system throughput, and channel throughput. We have studied these performance metrics for different channel conditions, packet arrival rates, decoding time-out limits, system capacities, and the number of sequential decoders. Our results show that parallel sequential decoders have great impact on the system performance and increase QoD significantly.

1. INTRODUCTION

One of the major advantages of wireless networks over wired networks is the ability to collect data from locations where it is very costly or almost impossible to set up a wired network. Some applications of remote data collection have research interests in wild life monitoring, ecology, astronomy, geophysics, meteorology, oceanography, and structural engineering. In these systems, the data are usually collected by wireless end points (WEPs) having sensors. As the technology improves, WEPs maintain new types of data collection through new sensors. However, the capacity of the wireless system may fail to satisfy the transmission rate from these WEPs. Moreover, the transmission rate is not stable for WEPs in case of interesting events when multiple sensors are activated and the size of data to be transmitted increases significantly. Increasing the transmission rate may aggravate the system and channel throughput because of the significant number of retransmissions due to the high number of packets having errors and getting lost. The traditional TCP-based protocols cannot deal with the bandwidth errors since TCP assumes that packet loss occurs due to network congestion [1]. Whenever a packet is lost, TCP systems decrease the sending rate to half [1, 2] worsening the quality of delivery (QoD). Moreover, the use of stop-and-wait ARQ in IEEE 802.11 [3] standard reduces the throughput for TCP-based protocols. Stop-and-wait ARQ is preferred because of the high error rate and low bandwidth in wireless channels. The TCP is usually improved in two ways, that is, by splitting or end-to-end improvement [1]. In splitting, the hop that connects the wireless network to the wired network establishes TCP from sender to itself and from itself to the receiver. However, this type of splitting is against the semantics of end-to-end TCP [1]. In the alternate end-to-end adjustment, the sender adjusts its sending rate based on error rates and network congestion by probing the network or receiving messages from the receiver.

1.1. Quality of delivery

It is obvious that numerous retransmissions in wireless channels aggravate the performance of TCP significantly. Our major target in this paper is to reduce the number of retransmissions at the data link layer so that the performance of TCP

is improved. The *complete delivery* of email messages, document files, or any arbitrary file *with no errors* and *as fast as possible* is a challenging objective of QoD that needs to be achieved. We define the QoD as the best effort strategy to increase the integrity of service using available bandwidth by customizing data link layer without promising or preallocating resources for the sender (i.e., traffic contract) as in quality of service (QoS). The major goal in QoD is to maximize the quality under given specific resources without any dedication for the sender. Therefore, our strategies enhance the quality of service (or quality of data) obtained at the receiver. When looking into the network architecture or delivery path (source, intermediate hops, channels, and destination), the intermediate hops (like routers) play critical role in achieving the best optimistic QoD. In general, the intermediate hops mainly consist of (a) a queue or buffer to store packets that arrive from the channel and (b) a server to process the arriving packets and deliver them to the next hop. TCP/IP is a connection-oriented suite that consists of communication protocols [4] that offer end-to-endreliability, in-order delivery, and traffic control. Consequently, the in-time delivery is not an accomplished goal. Thus, delivery *with very low number of retransmissions* is so important to overcome in-time delivery problem. It becomes one of the major targets for good QoD. Therefore, the traffic control by reducing the number of retransmissions should be achieved in a way to accomplish the necessary QoD.

1.2. TCP traffic control

Flow, congestion, and error controls [5] are the parts of the traffic control. It is known that flow control protects the recipient from being overwhelmed, while congestion control protects the network from being overwhelmed. Flow control and network congestion affect each other. High system flow may lead to possible network congestion. Consequently, network congestion causes longer delivery time. The automatic repeat request (ARQ) [6] refers to error control that utilizes error detection techniques (e.g., parity bits or cyclic redundancy check (CRC) code), acknowledgments, timers, and retransmissions. In this paper, we focus on stop-and-wait ARQ since it is employed in IEEE 802.11 protocol. This method needs lower system requirements than other protocols (like sliding window) since there is just one packet coming at a time from the transmitter side (i.e., it needs less system capacity since less data need to be retransmitted in case of error as no packets are transmitted until ACK has been received). Actually, the major purpose of using stop-and-wait ARQ is to prevent possible network congestion since multiple simultaneous packets sending (like sliding window approach) over noisy channels may clearly cause network congestion. To reduce the number of retransmissions, error correction is a necessary step to be accomplished at the data link layer especially in wireless networks. Convolutional decoding [7] using sequential decoding [8, 9] algorithms is an error detection and correction technique. In fact, it is widely used in telecommunication environments. Sequential decoding has variable decoding time and is highly adaptive to the channel parameters such as channel signal-to-noise ratio (SNR). This is so important in wireless environments since the packet error rate (PER) is a function of weather conditions, urban obstacles, multipath interference, mobility of end stations, and large moving objects [1]. It is very powerful decoding mechanism since it is able to detect and correct errors extensively. Furthermore, it has a great impact on the router system flow and network congestion since it is able to increase the delivery time by decreasing the unnecessary router system flow and network congestion accordingly.

The finite buffer of router system buffer (system capacity) is used to absorb the variable decoding rate. The size of that finite buffer has an impact on the system throughput and clearly it cannot be chosen arbitrarily. When the size is too small, the probability of dropping (discarding) packets increases. Therefore, the incoming flow increases due to retransmission of lost packets. Consequently, the congestion over the network increases. When the buffer size is not satisfactory, a quick (but ephemeral) remedy is to increase the buffer size. However, large buffer sizes promote the delay for getting service (waiting time) since more packets are in the queue to be served. This may also increase the flow rate and congestion over the network because of the unnecessary retransmissions due to the time-out of sender. Increasing the buffer size may not correspond to more than paying more for worse QoD. Therefore, the buffer size has a significant impact on the flow and congestion controls. The expected advantage of using sliding window approach is its higher system throughput and faster delivery. Unfortunately, it may significantly increase the network congestion and decrease the QoD accordingly if it is employed with sequential decoding. Actually, our results show that we cannot get a high system throughput even for stop-and-wait ARQ when the network is congested. In wireless networks, the damage (packet error rate (PER)) is much larger than in wired networks. Therefore, the decoding time is much larger. Consequently, the system buffer is utilized too fast and early.

1.3. Our approach

To resolve these issues, we propose a router system that effectively works for stop-and-wait ARQ at the link layer while decreasing number of retransmissions using sequential decoding. We propose that if this router system is used as wireless router, access point, or even end point, the number of retransmissions can be reduced significantly, thus increasing the effectiveness of TCP protocols. Our system targets the low bandwidth and high error rate for wireless channels. In this sense, our system is hop-to-hop rather than end-to-end. We claim that the data transmitted from the WEPs should be corrected as early as possible.

To investigate the problem of undesired retransmissions, we study and simulate a router system with sequential convolutional decoding algorithms. We firstly study (single) sequential decoding system with a (very large) finite buffer. We simulate this system using MATLAB and measure the performance in terms of average buffer occupancy, blocking probability, channel throughout, system throughput, and probability of decoding failure. We then design and simulate a router system having parallel sequential decoding

environment. Our system can be considered as type-I hybrid ARQ with sequential decoding. Type-I hybrid ARQ is widely implemented with forward error correction (FEC) [10–13]. Our experiments show that our router system with parallel sequential decoders reacts better to noisy channels and high system flow. Our router system with parallel sequential decoders has yielded low packet waiting time, low loss probability, low network congestion, low packet error rate (PER), and high system throughput. Our simulator for router system having parallel sequential decoders is implemented using Pthreads API package under a symmetric multiprocessors (SMPs) system with 8 processors. Our both simulators are based on stochastic modeling by using discrete-time Markov chain.

The contributions of this paper are as follows:

(1) introduction of a novel wireless router system with *parallel sequential decoding* mechanism that works efficiently with
   (i) *finite reasonable system capacity*;
   (ii) *hop-to-hop system;*
   (iii) *stop-and-wait ARQ*;
   (iv) especially *wireless environments*;
(2) simulation for (singular) sequential decoding algorithms for *finite* buffer systems;
(3) evaluating the average buffer occupancy, blocking probability, system throughput, probability of failure decoding, and channel throughput that represent the major impacts on the number of retransmissions and complete delivery time;
(4) showing the problems caused by large buffer size when operating a sequential decoder;
(5) simulation of novel parallel sequential decoding system for finite buffer systems;
(6) mitigating the congestion and increasing the QoD with *high system and channel throughputs* using parallel sequential decoding system.

This paper is organized as follows. The following section describes the background on channel coding. Section 3 describes the system for a sequential decoder with a finite buffer system. The simulation results for a sequential decoder are discussed in Section 4. Section 5 explains the parallel sequential decoder system and simulation results. The last section concludes our paper.

2. CHANNEL CODING

Channel coding [14, 15] is a process to add redundant bits to the original data bits to immune the system against noise. The most common coding techniques that are used in channel coding are linear block code, CRC codes, and convolutional codes. Figure 1 shows the block diagram of coding. In linear block code, the data stream is divided into several blocks of fixed length k, where each block is encoded into a code word of length $n > k$. This method presents very high code rates, k/n (the overall data rate is $R_{\text{overall}} = (n/k)R_{\text{source}}$), usually above 0.95. This leads to high information content in code words. It has a limitation on error correction capabilities. It is useful for channels with low raw error rate probabilities and less bandwidth. CRC code is one of the most common coding schemes used in digital communications. It is very easy to be implemented in electronic hardware and efficient encoding and decoding schemes, but it supports only error detection. Therefore, it must be concatenated with another code for error correction capabilities.

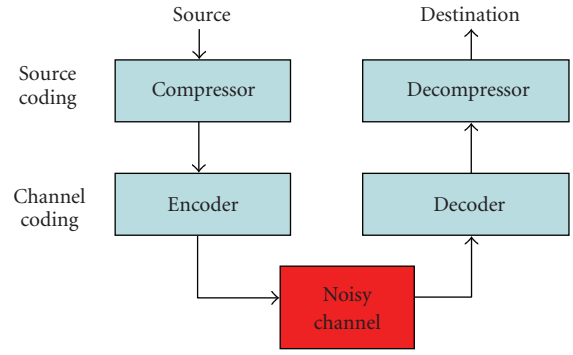

Figure 1: Block diagram of coding.

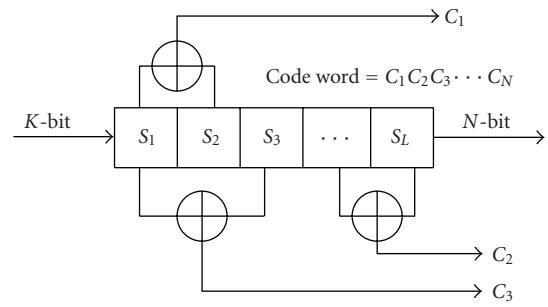

Figure 2: Encoder block (shift register) using convolutional codes.

2.1. Convolutional code

Convolutional code [16] is an advanced coding technique that is designed to mitigate the probability of erroneous transmission over noisy channel. In this method, the entire data stream is encoded into one code word. It presents code rates usually below 0.90, but with very powerful error correction capabilities. It is useful for channels with high raw error rate probabilities, but it needs more bandwidth to achieve similar transmission rate.

A convolutional coder consists of an L-stage shift register and n code word blocks' (see Figure 2) modulo-2 adders (XOR gates). Therefore, it has a constraint length L. Figure 2 shows the encoder side using convolutional codes. The shift register is a finite state machine (FSM). The importance of the FSM is that it can be described by a state diagram (operational map of the machine at each instance of time). The number of state transitions is 2^{L-1} states. Any transition for the coder produces an output depending on a certain input.

2.2. Maximum likelihood decoding and sequential decoding

There are two important decoding algorithms for convolutional codes: the maximum likelihood decoding (Viterbi's algorithm) and sequential decoding. Viterbi decoding [17, 18] was developed by Andrew J. Viterbi, a founder of Qualcomm Corporation. It has a fixed decoding time. It is well suited to hardware decoder implementations. Its computational and storage requirements grow exponentially as a function (2^L) of the constraint length, and they are very attractive for constraint length $L < 10$. To achieve very low error probabilities, longer constraint lengths are required. Thus, Viterbi decoding becomes infeasible for high constraint lengths (therefore, sequential decoding becomes more attractive). Convolutional coding with Viterbi decoding has been the predominant forward error correction (FEC) technique used in space communications, particularly in satellite communication networks such as very small aperture terminal (VSAT) networks.

Sequential decoding was first introduced by Wozencraft for the decoding of convolutional codes [16, 19–21]. Thereafter, Fano developed the sequential decoding algorithm with a milestone improvement in decoding efficiency [7, 22, 23]. The sequential decoding complexity increases linearly rather than exponentially. It has a variable decoding time. A sequential decoder acts much like a driver who occasionally makes a wrong choice at a fork of a road then quickly discovers the error (because of the road signs), goes back, and tries the other path. In contrast to the limitation of the Viterbi algorithm, sequential decoding [24–26] is well known for its computational complexity being independent of the code constraint length. Sequential decoding can achieve a desired bit error probability when a sufficiently large constraint length is taken for the convolutional code. The decoding complexity of a sequential decoder becomes dependent on the noise level [14, 23]. These specific characteristics make the sequential decoding very useful.

The sequential decoder receives a possible code word. According to its state diagram, it compares the received sequence with the possible code word allowed by the decoder. Each sequence consists of groups and each group consists of n digits. It chooses the path whose sequence is at the shortest Hamming distance (HD) from the first n received digits (first group), then it goes to the second group of the n received digits and chooses the path whose sequence is the closest to these received digits. It progresses this way. If it is unlucky enough to have a large number of (cumulative) errors in a certain received group of n digits, it means that it took the wrong way. It goes back and tries another path.

3. DISCRETE-TIME MARKOV CHAIN MODEL

We simulate the router system with sequential decoding as a service mechanism using discrete-time Markov model. In this model, the time axis is portioned into slots of equal length. This slot time is precisely the time to transmit a packet over the channel (i.e., propagation time plus transmission time). We assume that all incoming packets have the same size. This is the case if we send Internet packets (typically of size 1 KB) over a wireless link (where packets have the size of around 300 bytes) or over the so-called ATM networks (in which cells have the size of 52 bytes). Thus, the router system can receive at most one new packet during a slot. In this paper, we use practical assumption that the buffer is of finite length to be close to real environment. Hence, any packet loss happens during transmission due to lack of buffer space; a packet retransmission will occur from the sender side if time-out occurs or negative ACK arrives. This retransmission is based on stop-and-wait ARQ. However, the new packets arrive at the decoder from the channel according to Bernoulli process. A slot carries an arriving packet with probability λ and it is idle (no transmission) with probability $1 - \lambda$.

SNR increases as the signal power gets larger than noise power. This indicates that the channel is getting better (not noisy). Thus, low decoding time may suffice. On the other side, if SNR decreases, this represents that the noise power gets larger than signal power. Therefore, the channel is getting worse (noisy). Consequently, larger decoding time is required. To demonstrate that variable decoding time, we need a distribution with a dominant parameter to represent SNR of the channel such that when it gets higher and higher, the probability density function of that distribution goes to zero earlier and earlier accordingly. On the other hand, when it gets lower and lower, the chance of going to zero is lower and lower. In fact, it can also go to infinity. Thus, there should be a limit employed to prevent that case. Moreover, we need a parameter that determines the minimum value that a random variable can take to represent the minimum decoding time. Actually, the Pareto (heavy-tailed) distribution [27, 28] is the best fit to demonstrate this variable decoding time. Thus, the decoding time follows the Pareto distribution with a parameter β, which is a function of SNR. The buffer size is assumed to be at least one. We make another assumption that the decoding time of a packet is in chunks of equal length to the slot size. That is, the decoder can start and stop decoding only at the end of a slot. This assumption replaces the continuous distribution function of the decoding time by a staircase function that is a pessimistic approximation of the decoding time. This approximation yields an upper bound on the number of packets in the queue [27, 29, 30]. In order to make this protocol consistent with our assumptions, we assume that each slot corresponds to exactly the time to transmit a packet over the channel (propagation time plus transmission time).

It is very important to realize that we cannot let the decoder perform decoding for infinite time. Thus, a decoding time-out limit (T) should be operated with the system. We use similar assumptions as in [27, 30–33]. If a packet requires j slots for decoding ($j \leq T$), it leaves the system at the end of the jth slot after the beginning of its decoding, and the decoding of a new packet starts (if there is a new packet in the decoder's buffer) at the beginning of the following slot. If a packet's decoding needs more than T slots, the decoder stops that packet's decoding after T slots. This packet cannot be decoded and thus a decoding failure results. Therefore, the decoder signals a decoding failure to the transmitter of the packet. The retransmission is based on stop-and-wait

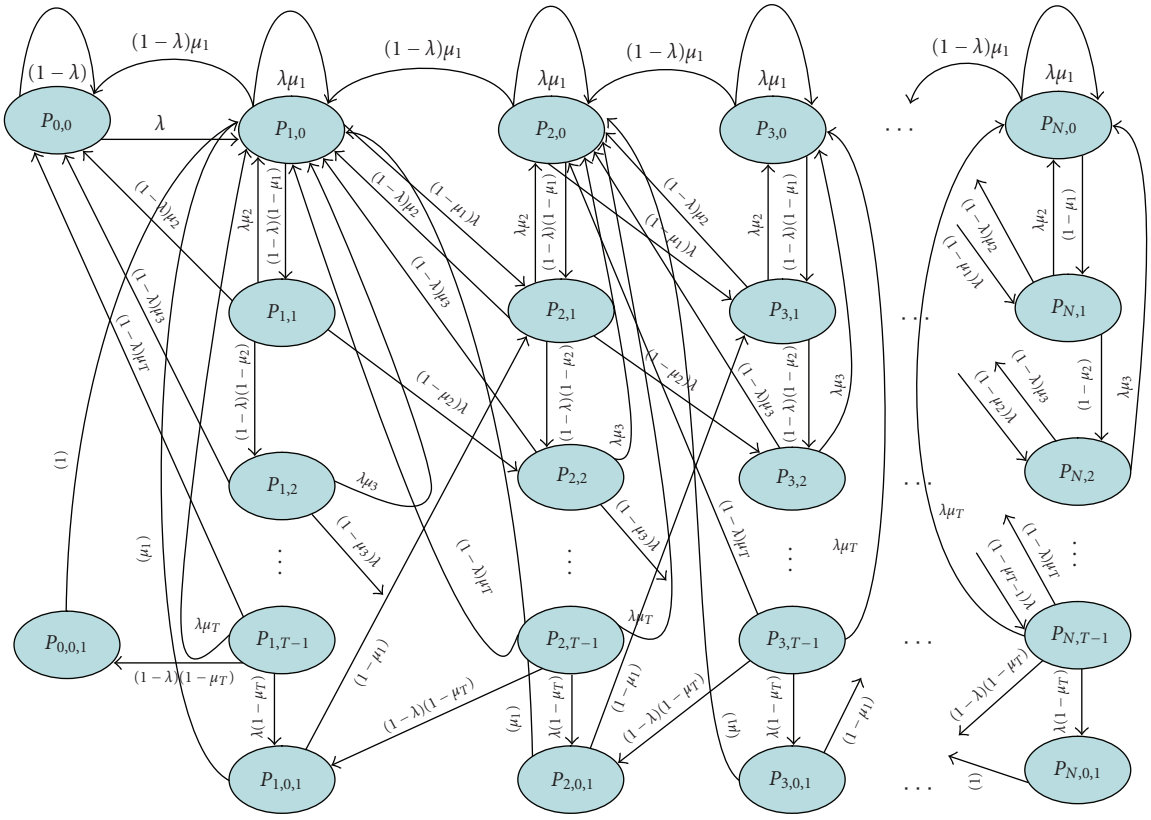

Figure 3: Probability state transitions of the router system with a buffer and a sequential decoder.

ARQ. Therefore, if a decoding failure occurs, the packet is retransmitted at the following slot, while the decoder starts at that slot decoding another packet if there is any in the buffer. Therefore, the channel carries a retransmitted packet during the slot that follows decoding failures. Consequently, new packets cannot arrive in those slots but can be transmitted during all the other slots.

The state of the system with just a sequential decoder can be represented [27, 30] by (n, t, w), where n is the number of packets in the buffer including the packet being decoded, t is the number of slots the decoder has already spent on the packet that is currently being decoded, and w is the number of packets to be retransmitted. Since the system has a finite system capacity, the value of n must be limited between 0 and the maximum permitted system capacity ($0 \le n \le N$). If the decoder needs more than t slots to be completely decoded, then decoding failure occurs. Therefore, it has to be retransmitted. Figure 3 shows the probability state transitions of the router system with a buffer and a sequential decoder. $P_{n,t,w}$ is the probability that the decoder's buffer contains n packets including the one being decoded, the decoder is in the tth slot of decoding, and there are w packets that need to be retransmitted. The summation of all the outgoing links (probabilities) from each state must be equal to one.

We use the notations that are mentioned in prior researches [29, 30, 32, 34]. c_k denotes the probability of decoding being completed in exactly k slots, and μ_k denotes the conditional probability that decoding is completed in k slots given that the decoding is longer than $k-1$ slots. Then, the conditional probability μ_k is given by

$$\mu_j = \frac{c_j}{1 - F_{j-1}}, \tag{1}$$

where $F_j = \sum_{i=1}^{j} c_i$ is the cumulative distribution function (CDF) of the decoding time. It can be shown that

$$\prod_{i=1}^{j} (1 - \mu_i) = 1 - F_j. \tag{2}$$

The decoding time of sequential decoders has the Pareto distribution

$$P_F(\tau) = \Pr\{t > \tau\} = \left(\frac{\tau}{\tau_0}\right)^{-\beta}, \tag{3}$$

where τ_0 is the decoding time for which the probability is 1, that is, the minimum time the decoder takes to decode a packet, and β is called the Pareto parameter and it is a function of the SNR of the channel.

4. SIMULATION OF A ROUTER SYSTEM WITH A SEQUENTIAL DECODER

This section illustrates a lot of important requirements for the simulation. It contains two subsections. Section 4.1 includes the simulation setup. Section 4.2 includes and explains the simulation results.

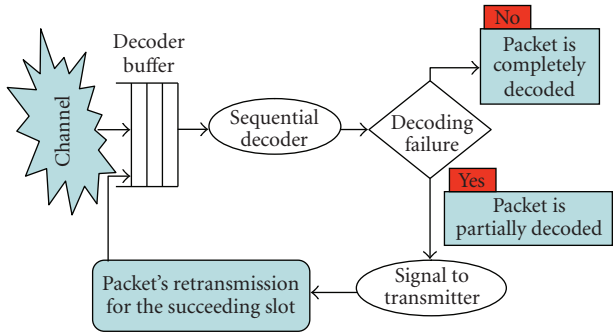

Figure 4: A router system with a sequential decoder using stop-and-wait ARQ.

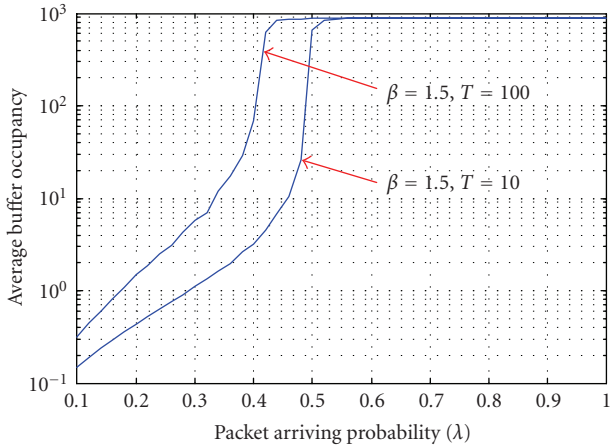

Figure 5: Average buffer size versus packet arriving probability ($\beta = 1.5$).

4.1. Simulation setup

The simulation of the sequential decoding system is done in MATLAB. The goal of this simulation is to measure the average buffer size, channel throughput, system throughput, blocking probability, and decoding failure probability. The sequential decoding system is simulated using stop-and-wait ARQ model. Therefore, the time axis is portioned into slots of equal length where each slot corresponds to exactly the time to transmit a packet over the channel (i.e., propagation time plus transmission time). Figure 4 shows the typical structure of a router system that works on the data link layer (specifically in the logical link control sublayer) since we are working hop-to-hop not end-to-end. We assume that all the incoming packets have equal lengths (e.g., ATM networks or wireless links). Accordingly, the decoder can receive at most one new packet during a slot.

The primary steps in our simulation are as follows. A random number generator for Bernoulli distribution is invoked at the beginning of every time slot to demonstrate the arrival of packets. A random number generator for Pareto distribution is invoked at the beginning of any time slot as long as there are packets in the queue waiting for service to demonstrate the heavy tailed service times. The minimum service

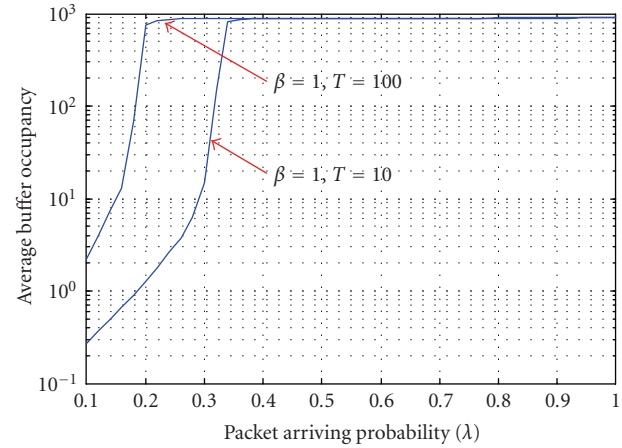

Figure 6: Average buffer size versus packet arriving probability ($\beta = 1.0$).

time is assumed to be one. The decoding time-out slots (T) and system capacity are taken as inputs of the simulation.

4.2. Simulation results

Figure 5 shows the average buffer size versus packet arriving probability (λ) for fixed channel condition ($\beta = 1.5$), system capacity of 900, and different decoding time-out slots (10 and 100). The simulation time is 4×10^5 slots. For fixed decoding time-out slots T and β, the average buffer size increases as packet arriving probability increases and it reaches the system capacity accordingly. This is expected since increasing λ means increasing the probability of arriving packets to the router system. For fixed λ and β, it is also seen that the average buffer size increases as the decoding time-out slots (T) increase. This is expected since increasing the decoding time-out limit means getting low probability to serve more packets and high probability for the buffer to be filled up early accordingly.

Figure 6 represents the average buffer size versus λ for $\beta = 1.0$. The simulation time is 4×10^5 slots, system capacity is 900, and decoding time-out slots are 10 and 100. From Figures 5 and 6, it is noticed that the average buffer size increases as channel condition β decreases of course for fixed T and λ. This is expected since decreasing β means that the channel gets worse (i.e., noisy). Thus, high decoding slots are generated from Pareto random number generator. Consequently, the buffer in Figure 6 is filled up earlier than that in Figure 5. It is also interesting to see that the buffer is filled up too early in terms of packet arriving probability. For example, for $\beta = 1.0$, the system reaches its capacity around packet arriving probabilities $\lambda = 0.35$ and $\lambda = 0.25$ for $T = 10$ and $T = 100$, respectively. While in Figure 5, for $\beta = 1.5$, the system reaches its capacity around packet arriving probabilities $\lambda = 0.52$ and $\lambda = 0.44$ for $T = 10$ and $T = 100$, respectively.

Figure 7 shows the blocking probability of incoming packets versus incoming packet probability. The results are shown for different decoding time-out limits (T) and channel conditions (β). The blocking probability increases as

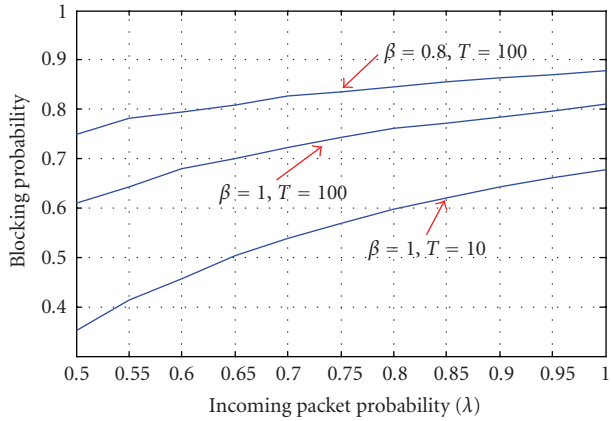

Figure 7: Blocking probability versus incoming packet probability ($\beta = 0.8, 1.0$).

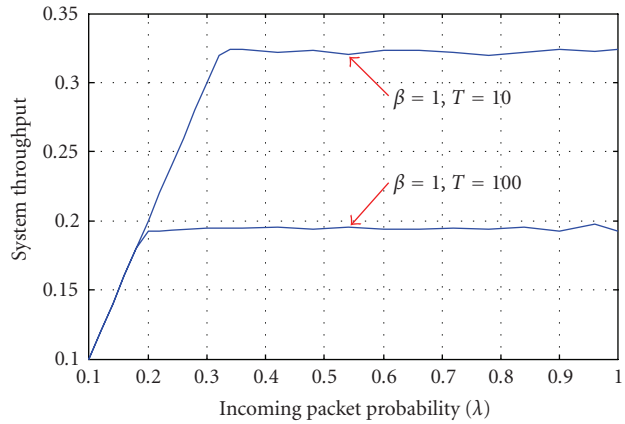

Figure 8: System throughput versus incoming packet probability ($\beta = 1.0$).

incoming packet probability increases. This is expected since there is higher flow rate. For fixed channel condition and incoming packet probability, the blocking probability increases as decoding time-out limit increases. This is also expected since increasing T means getting low probability for the buffer to have available space. For fixed incoming packet probability and decoding time-out limit, the blocking probability increases as channel condition decreases. When the channel condition decreases, the SNR decreases leading to a high noise power (i.e., there is high distortion). Consequently, large T is generated from Pareto random number generators trying to detect and correct the errors in currently noisy served packet.

Figure 8 illustrates the system throughput versus packet arriving probability given the channel condition $\beta = 1.0$. The system throughput can be explained as the average number of packets that get served (decoded) per time slot. One important observation we can notice from this figure is that the system throughput goes firstly linear and then the system cannot respond to increasing incoming packet probability leading to a nonincreasing system throughput. Thus, we have two trends of system throughput. It is so interesting to see that when system throughput is linear, the slope becomes equal to the incoming packet probability (λ). In fact, this indicates that all the incoming packets are being served without any packet loss. The other trend is when the system throughput does not respond to the increase in the incoming packet probability. Actually, there are two interesting explanations for this drastic change. The first one is that change is due to starting discarding (dropping) packets. Therefore, the system throughput is getting lower than the incoming packet probability. Why does the system throughput almost get constant although there is noticed increasing in the incoming packet probability? It is because the blocking probability is not constant when the incoming packet probability is increasing, but instead it is increasing. Figure 7 verifies this explanation. Therefore, it is true that as the packet arrival rate increases, the total number of discarded packets also increases. Thus, the system throughput almost reacts in the same way and does not change significantly. Actually, this is a very good indication that the congestion over the network is obvious since there is not that much gain in the system throughput while increasing the incoming packet probability. The effect of increasing the decoding time-out limit for fixed channel condition and packet arriving probability is shown in Figure 8. In fact, increasing the decoding time-out limit leads to increasing the blocking probability and decreasing the system throughput.

Figure 9 illustrates the system throughput versus packet arriving probability for a different channel condition ($\beta = 1.5$). Figures 8 and 9 show the effects of employing different values of channel condition. Therefore, for a fixed value of packet arriving probability and decoding time-out limit, the system throughput increases as the channel condition increases. This is expected since increasing the channel condition means that the channel gets better (i.e., flipping of the transmitted bits of packets is being reduced).

5. WIRELESS ROUTER SYSTEM WITH PARALLEL SEQUENTIAL DECODERS

This section provides a study over a wireless router that manages all the traffic coming from wireless networks. Our study is applicable for those applications that cannot tolerate any damage or loss packets and need quickness in delivery as much as possible. We reduce the traffic intelligently by mitigating the number of retransmissions since it has significant impact on the QoD in terms of delivery time. In fact, these retransmissions can be a result of lost or damaged packets. The packets can be lost if they arrive to a full buffer. This study includes proposing a wireless router system based on the implementation of hybrid ARQ with parallel sequential decoding. The organization of this section is as follows. It contains five subsections. Section 5.1 explains our stochastic simulation details and flowcharts. Section 5.2 explains the structures, constants, declarations, and initializations that are used in our simulator. Section 5.3 illustrates the system behavior of our simulator, and Section 5.4 includes our parallel simulation results.

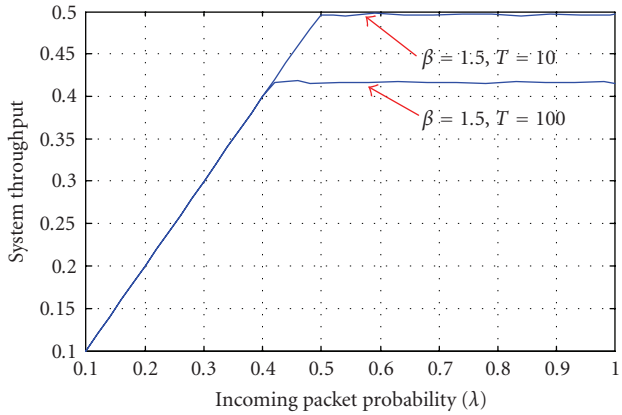

Figure 9: System throughput versus packet arriving probability ($\beta = 1.5$).

5.1. Stochastic simulation details and flowcharts

This program (simulator) is designed to simulate a router system with more than one sequential decoder with a shared buffer. Figure 10 shows this system. It is seen that all the sequential decoders share the same finite buffer. In fact, we manage the traffic over the router system by using sequential decoding to reduce the packet error rate (PER) and this clearly refers to a type of error control. Moreover, we add parallel sequential decoders to mitigate the congestion over the router system due to having finite available buffer space. We see in Section 4 that the average buffer occupancy reaches the system capacity too early leading to an increase in the blocking probability as the incoming packet probability increases when using a sequential decoder with a system capacity of 900 (which is practically very large). In fact, this may be the major drawback of using the sequential decoding. However, we can overcome this drawback and furthermore reduce a clearly possible congestion over the network by implementing parallel sequential decoding environments. There is also one more interesting improvement with this simulator. In fact, it is the ability to extend (increase) the decoding time-out limit in case of noisy channels. We are so worried in a router system model with just a sequential decoder about this limit since it is affecting the buffer badly (as seen in Section 4). This simulator has been performed using Pthreads API that is defined in the ANSI/IEEE POSIX 1003.1, which is a standard defined only for the C language. This program has been executed on a Sun E4500 SMPs (symmetric multiprocessors) system with eight processors. In this system, all the processors have access to a pool of shared memory. Actually, this system is known as uniform memory access (UMA) since each processor has uniform access to memory (i.e., single address space).

The main problem of SMP is synchronization [35, 36]. Actually, synchronization is very important in SMP and needs to be accomplished since all processors share the same memory structure. It can be achieved by using mutual exclusion, which permits at most one processor to execute the critical section at any point. It is known that the enforcement of mutual exclusion may create deadlock and starvation control problems. Thus, the programmer/developer must be careful when designing and implementing the environment. There are a lot of methods for controlling access to critical regions. For example, there are lock variables, semaphores (binary or general semaphores), and monitors. Pthreads API uses mutexes; mutex is a class of functions that deal with synchronization. Mutex is an abbreviation for "mutual exclusion." Mutex functions provide creation, destruction, locking, and unlocking mutexes. Mutex variables are one of the primary means of implementing thread synchronization and protecting shared data when multiple writes occur.

5.2. Structures, constants, declarations, and initializations

The key structures and entities that are required for the simulation are *buffer status* structures, *threads* structure, *target* structure, *corrupted packets* structure, *Bernoulli random number generator* (BRNG) structure, *constants* entity, and POSIX critical sections entity.

Buffer status is represented with five attributes: *current_slot* (the current slot of the simulation), *Sys_curr_state* (the number of available packets in the system), *arrival_lost_packet* (the accumulative number of the packets being lost due to full buffer), *total_arriving_packets* (the total number of arriving packets to the router system), and *time_history* (history or record of the total number of the packets in the system for every slot time). Threadsstructure refers to sequential decoder (leader or slave decoder). It has two important attributes: *leader_thread* (set when it is in decoding process) and *threads_counter* (the number of jobs waiting for a slave decoder). The threads and mutex are initialized inside main thread initialization. In our simulations, there is one leader sequential decoder, and the rest are considered as slave sequential decoders. Target structure is used for simulator statistics and has two important attributes: *mean_buffer_size* (the average buffer occupancy at stationary conditions) and *blocking_prob* (the probability of packets being dropped (discarded) due to limited system capacity). Corrupted packetsstructure has two attributes used for the management of corrupted packet pool: *packet_failure* (the number of packets facing decoding failure) and *corrupted_pcts_counter* (the number of packets that cannot be decoded even with retransmission). Bernoulli random number generator (BRNG) represents the probability of arrival packets for a certain slot time. Constants entitymaintains the six input attributes: *num_threads* (the maximum number of threads in the simulation), *sim_time* (the simulation time), *system_cap* (the maximum buffer size), *beta* (the channel condition), *min_serv_slots* (the minimum decoding time in terms of time slots), and T (the maximum time-out decoding slots limit). POSIX critical sections entity declares the mutex for three necessary critical sections (shown in Figures 11–14) for synchronization.

5.3. System behavior

Figure 10 explains the system architecture of our approach in a wireless router. This subsection addresses the major duties

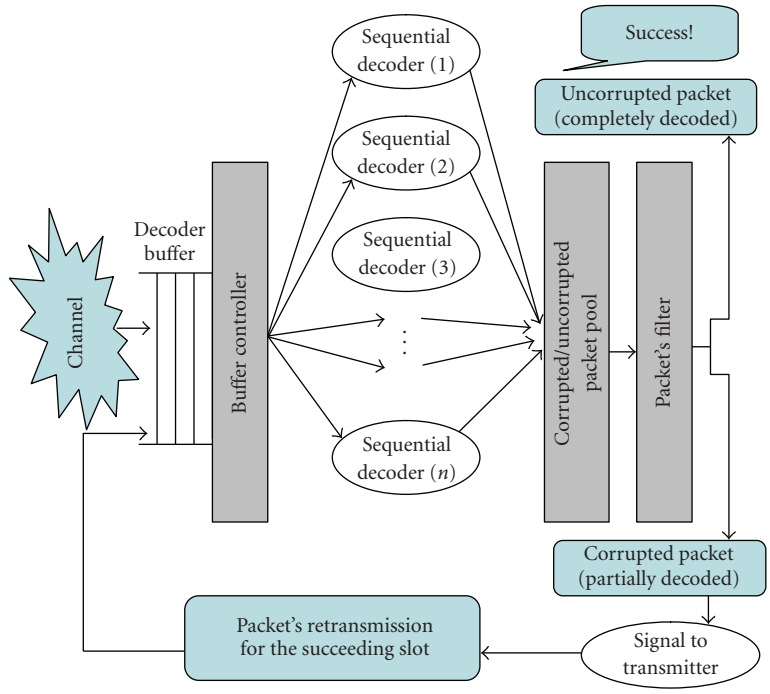

Figure 10: Router system with parallel sequential environment and single address space structure.

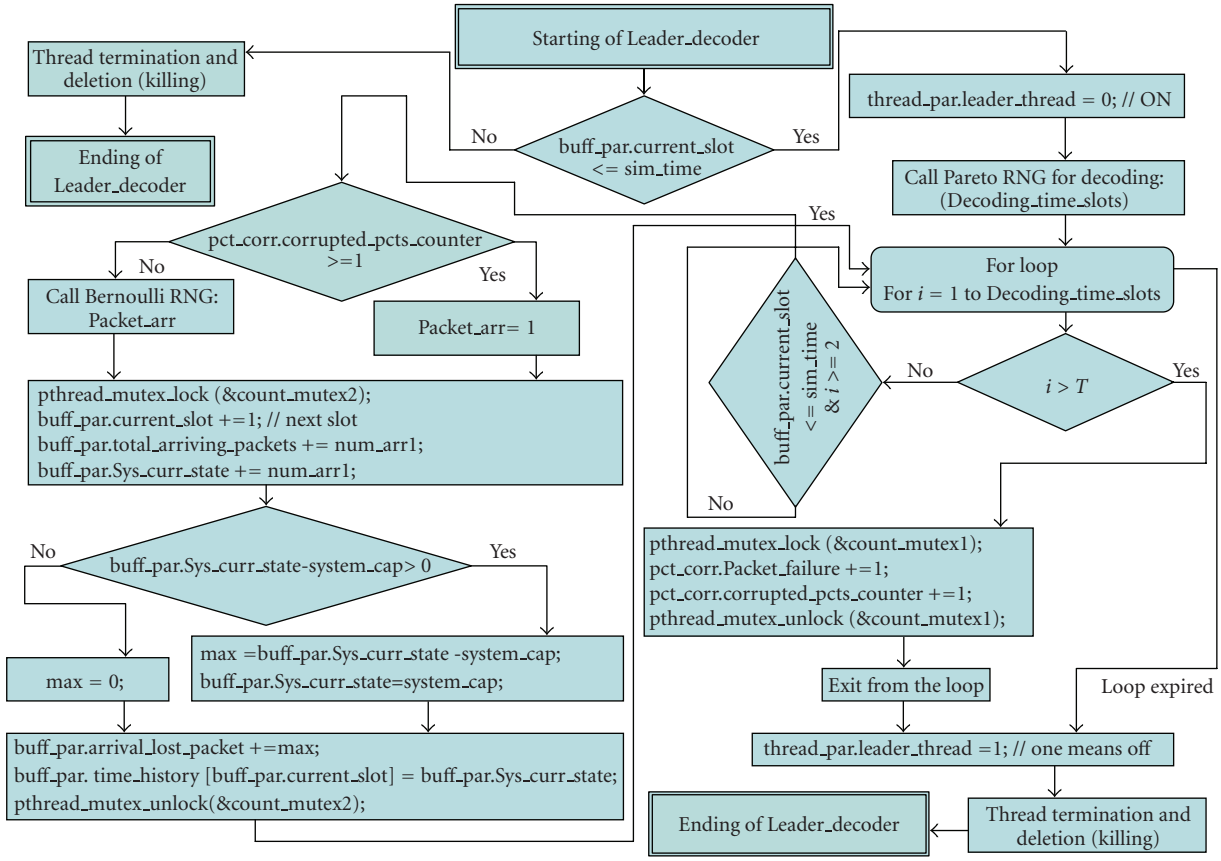

Figure 11: Major duties of the leader thread.

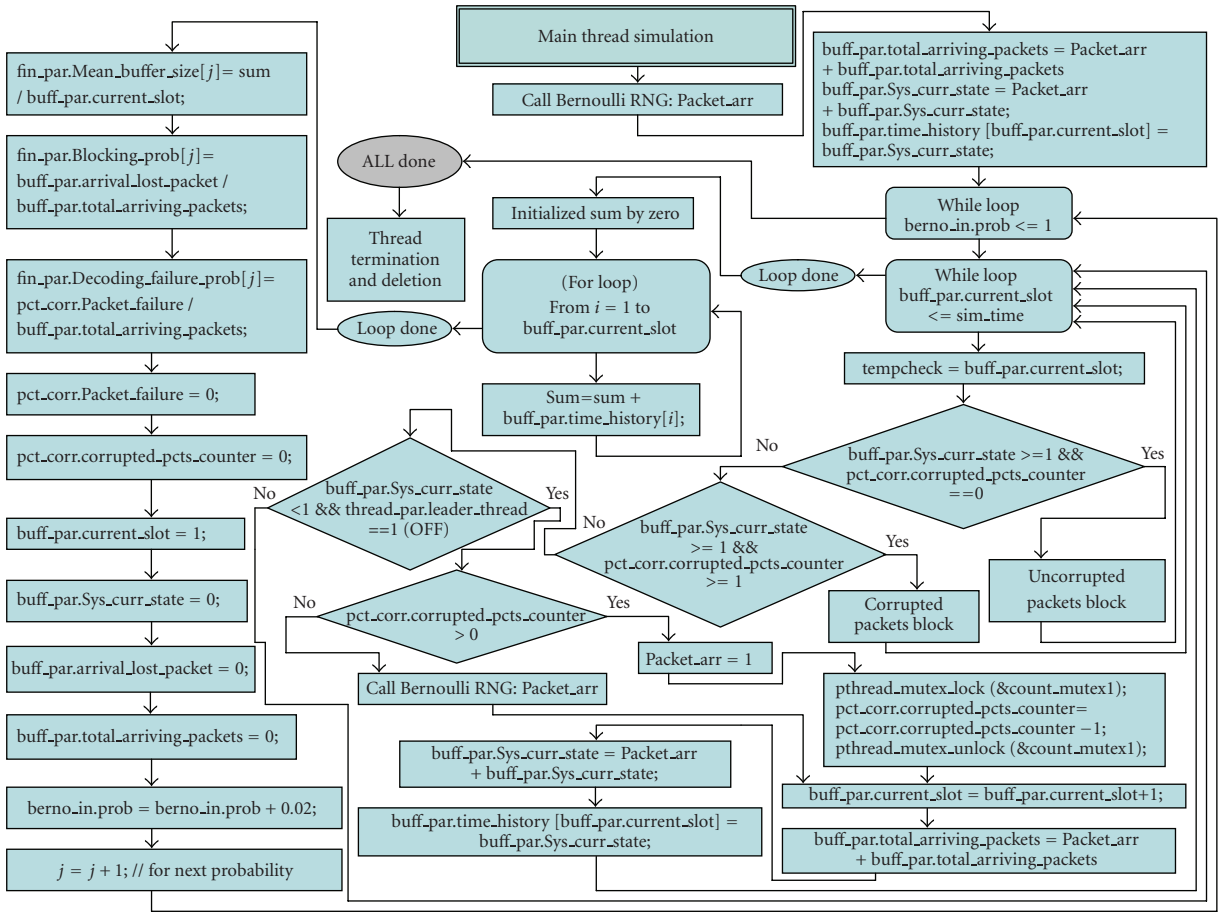

Figure 12: Major decoding steps for main thread.

and responsibilities of these components in our simulator. In this section, we use the terms thread, processor, and decoder interchangeably.

In our simulation environment, each thread represents a *sequential decoder* except the main thread (processor). We assume that there is just one packet that may arrive for any arbitrary slot time to be fully compatible with stop-and-wait handshaking mechanism. All the attributes of the *buffer status* structure are required to be updated accordingly. During the decoding slots, there may be arrivals to the system. We need to use sequential decoders to demonstrate the arriving process during decoding slots. Unfortunately, we cannot attach the arriving process to every decoder since one packet may arrive during any decoding slot. Therefore, we have defined (classified) two types of decoders: leader and slave. There is only one leader decoder but there might be many slave decoders. In our model, the main thread gives the highest priority for decoding for the leader decoder. But, in other cases, we cannot attach the arriving process to those slaves (since there are many) when the leader processor is not busy (decoding). Thus, in our model, this arriving process at such cases is handled by the main thread especially the first slot of our simulation. Furthermore, the leader and slave decoders have common responsibilities that are packet decoding and management of corrupted packet pool.

Before the leader processor starts decoding, it modifies the *leader_thread* attribute of *threads structure* to 0 indicating that it is currently busy and then it starts decoding. After finishing its decoding, it increments this attribute to 1 indicating that it is currently free waiting to serve another packet. On the other hand, slave processors start decoding after decrementing the *threads_counter* attribute of the *threads structure*. In fact, this attribute represents the level of utilizing the slave decoders. Whenever they finish decoding, they increment this attribute. Since all slave processors may access this attribute at the same time, it is synchronized by the third critical section inside the *POSIX critical sections* entity. Each decoder before decoding calls Pareto random number generator (PRNG) to get the number of decoding slots needed for that packet. The inputs for that PRNG are *min_serv_slots* and *beta*. Figure 11 shows the flowchart that shows the duties of the leader decoder. The leader and slave processors are responsible for *corrupted packet pool*. Whenever a packet decoding exceeds the given decoding time-out limit, a partial decoding occurs and the corrupted packet pool is updated. In our simulation, this process can be managed through the *corrupted packets* structure. These attributes are shared (i.e., all parallel decoders may need to use these simultaneously).

The *arriving process* is handled by calling BRNG. If the probability of arriving packets is one, this means that every

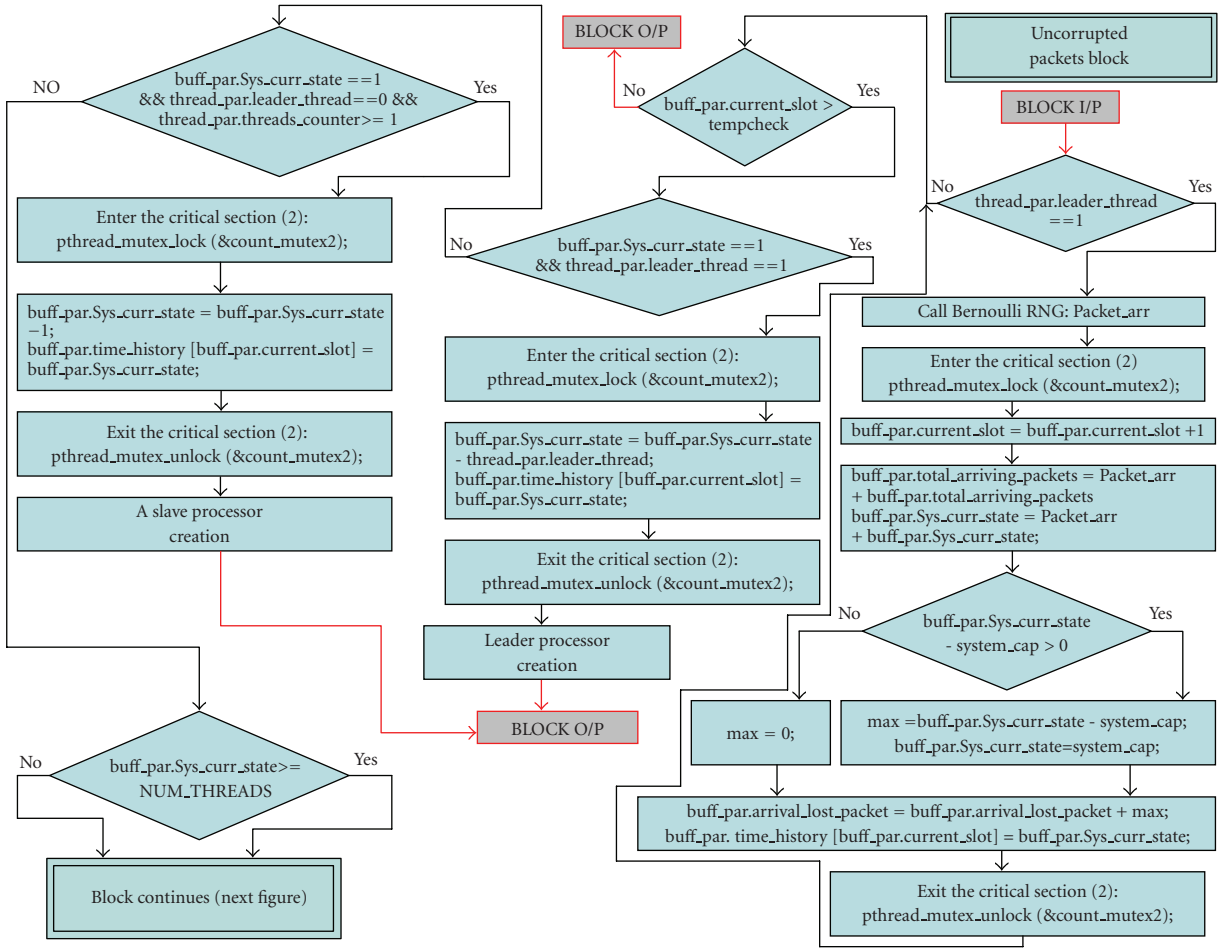

Figure 13: Major simulation steps for uncorrupted packets block shown in Figure 12.

slot has an arriving packet. The *buffer status* structure is required to be updated. Furthermore, the *sys_curr_state* attribute is required to be checked every slot time in order to ensure that it will not exceed the system capacity. Therefore, if a loss occurs, the *arrival_lost_packet* attribute must be immediately updated. The system current state must be maintained to be equal to the system capacity accordingly. Moreover, the *total_arriving_packets* attribute must be incremented if there is an arrival regardless of whether loss happens or not. In our approach, controlling the arriving process is done once either by the buffer controller or by the leader decoder. For every slot time, the system current state becomes less than one and leader decoder is not busy; the buffer controller calls BRNG if *corrupted_pcts_counter* attribute (defined in *corrupted packets* structure) is zero. Otherwise, there is a retransmitted packet. The *corrupted_pcts_counter* attribute is required to be decremented accordingly. It is possible that in those cases the slave decoders are performing decoding. Therefore, they may update this attribute simultaneously with the buffer controller.

The *buffer controller* is responsible for dispatching decoding for a packet if there is any in the buffer waiting for service and there are available (not busy) sequential decoders. It may dispatch decoding for every slot time except the first slot since we started from that slot (i.e., system current state is zero at that slot). Whenever it dispatches decoding, it updates the system current state by decrementing a number referring to the total number of dispatched decoders. It also updates the time history vector of the *buffer status* structure with current number of waiting packets. Actually, the dispatching process depends on *sys_curr_state* attribute, leader processor, and slave processors. For example, if the *sys_curr_state* is zero, the buffer controller cannot dispatch decoding. If the *sys_curr_state* is equal to one, the buffer controller dispatches the decoding for the leader decoder if the leader decoder is not busy. Otherwise, the buffer controller dispatches the decoding for any free slave decoder. Moreover, it has to have a strong connection with the total number of available decoders, the attributes of *sequential decoding threads* structure (i.e., *leader_thread* and *threads_counter)*, and *sys_curr_state* in order to maintain system validity through judging and knowing to whom it will dispatch, how many it should dispatch, what is left and then update the required attributes accordingly. As we mentioned earlier, the buffer controller keeps monitoring for every slot time to see whether a packet comes and/or decoders finish. How are the system current state and

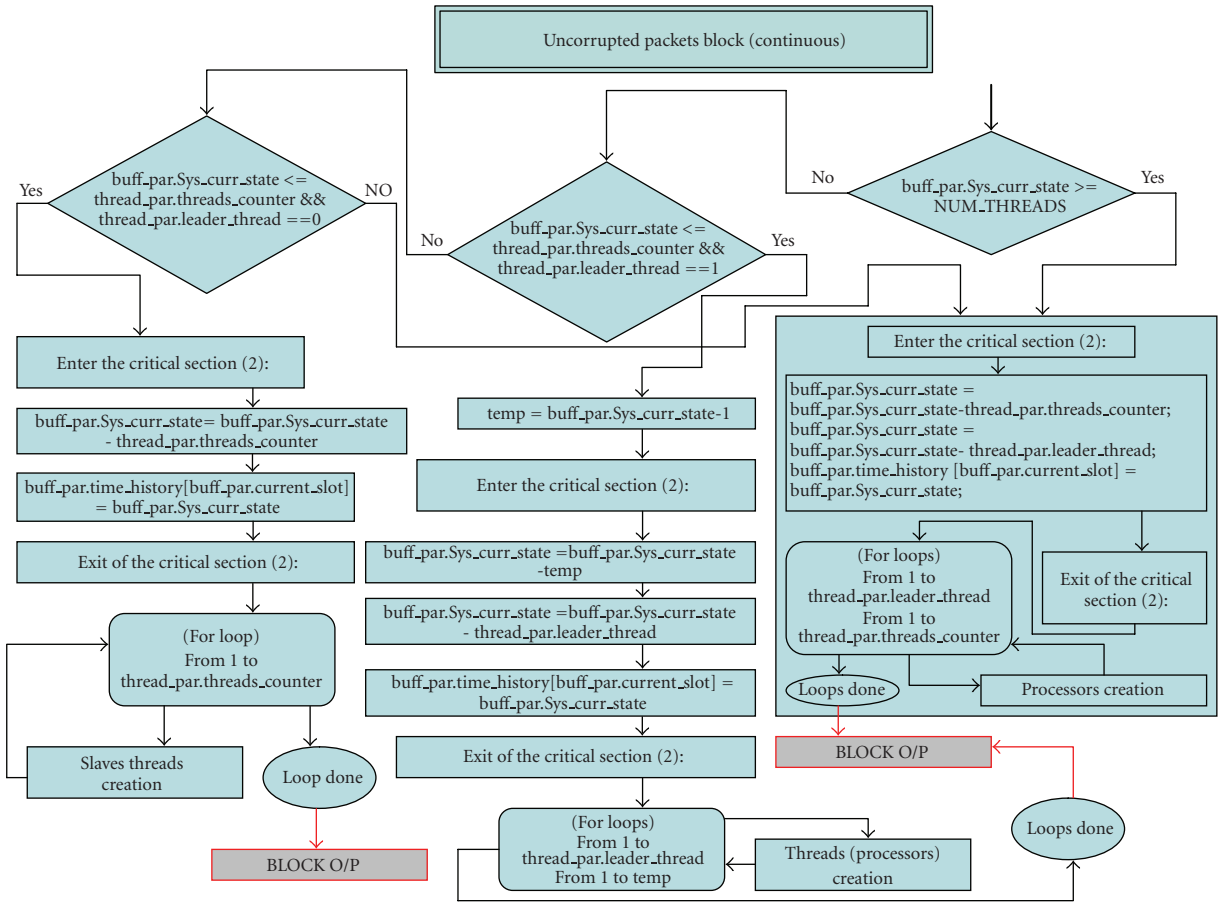

Figure 14: Major steps of uncorrupted packets block simulation (extension for Figure 13).

time history attributes handled at the times when arriving process is handled by the leader decoder? Basically, the buffer controller may access system current state and time history attributes simultaneously with the leader decoder in case of decoding dispatch.

Main thread (processor) manages buffer controller, packet's filter, and simulator statistics. The buffer controller keeps monitoring (or tracking) on every slot time until the end of the simulation time since every slot time may have an arriving packet and completion of packet decoding. Therefore, it checks for every slot time the status of every sequential decoder so that it dispatches decoding for any sequential decoder to become free and available for decoding. The main processor isolates the management of corrupted and uncorrupted packets. Figure 12 illustrates the steps implemented in the main thread through the flowchart. The simulation goes for *corrupted packets* block when the *corrupted_pcts_counter* is equal to or greater than one. In fact, this block is very similar to *uncorrupted packets* block explained in Figures 13 and 14 that show the major simulation steps. There are two major differences. The first one is that there is no need to call BRNG since the probability at this slot is one. The other one is that *corrupted_pcts_counter* attribute is decremented once entering the execution of that block. Definitely, a lock is required to decrement this attribute since the decoders may update it simultaneously. In our simulation, the *mean_buffer_size* performance metric in target structure is estimated by adding up all the numbers at the time history array attribute (in the *buffer status* structure) and dividing them by the simulation time. The *blocking_prob* is measured by dividing the *arrival_lost_packet* attribute by *total_arriving_packets* attribute.

5.4. Parallel simulation results

Our major goal when using parallel sequential decoding is to reduce the average buffer occupancy and blocking probability and to increase the system throughput. We see in our simulator in Section 4 (from Figures 5 and 6) that at those values of incoming probability like from 0.5 to 1, the average buffer occupancy gets constant by reaching the maximum system capacity of 900. We see in Section 4 (from Figure 7) that the blocking probability is very high at these values of incoming packet probability. Moreover, we see (from Figures 8 and 9) that the system throughput gets constant at these probability values. Hence, we have simulated our wireless router system with parallel sequential decoding using the same values of incoming packet probability to see the reaction of these parallel sequential decoders over just one sequential decoder. Furthermore, we employ the same values of channel conditions

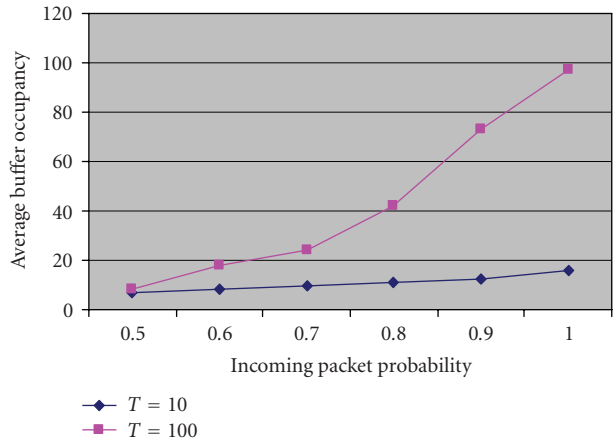

FIGURE 15: Average buffer occupancy versus incoming packet probability ($T = 10, 100$).

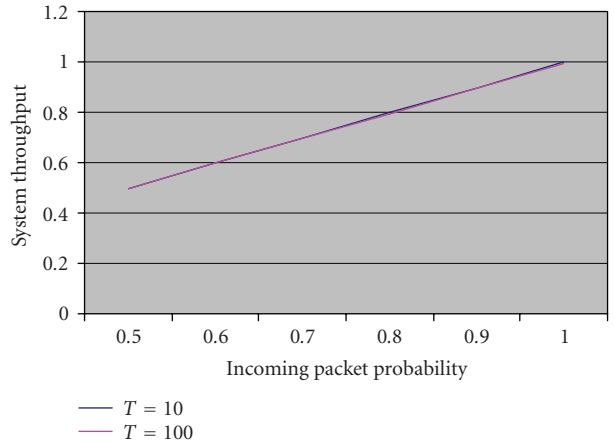

FIGURE 17: System throughput versus incoming packet probability ($T = 10, 100$).

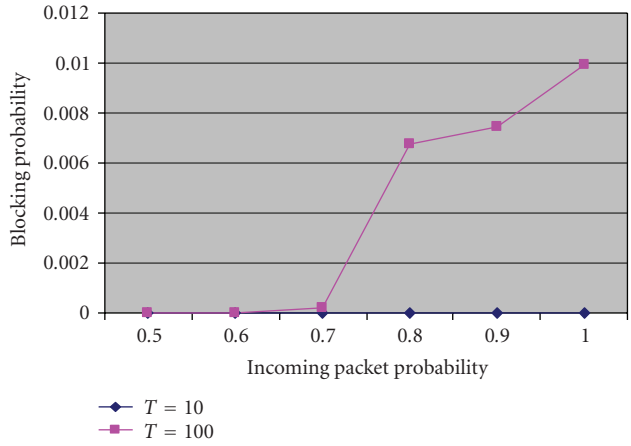

FIGURE 16: Blocking probability versus incoming packet probability ($T = 10, 100$).

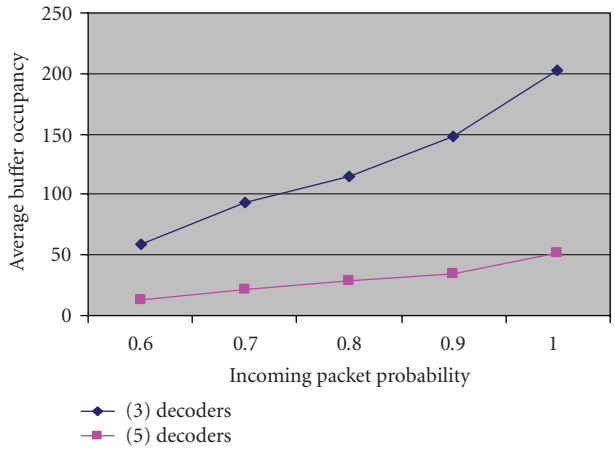

FIGURE 18: Average buffer occupancy versus incoming packet probability for different number of sequential decoders.

and even values lower than those which are employed with just one decoder to see the behavior of our wireless router system using parallel sequential decoding even under noisier channel conditions.

Table 1 shows the results of ABO, BP, and ST for different incoming packet probabilities. It also contains the simulation inputs. The increase in the average buffer occupancy as the incoming packet probability increases is noticed.

It is interesting to see the drastic difference between the average buffer occupancies when considering three processors and just one processor as in Figure 6. Also notice the big difference between blocking probabilities when employing three processors and just one processor as in Figure 7 for the same inputs. Moreover, notice the impressive linear trend of system throughput with λ using parallel sequential decoding over a constant trend in Figure 8. In fact, when employing a decoder (processor), the system may not be stable since it can reach the system capacity too early at a relative low packet arrival rate. Thus, high traffic, low system throughput, and worse QoD are obtained. On the other hand, employing parallel decoding environment leads to stable system, low traffic, and good QoD. Briefly, through parallel sequential decoding, we have adaptive decoding of the channel condition instead of fixed one that is used typically nowadays. We reduce the system flow due to low blocking probability. Consequently, we reduce the congestion over the network. Therefore, we have high system throughput. Thus, better QoD is attainable.

Table 2 shows the average buffer occupancy and blocking probability for different decoding time-out limit ($T = 100$). The blocking probability increases as the packet arriving probability increases. It is seen that the average buffer occupancy increases as packet arriving probability increases. From the comparison between Table 2 and Table 1, it is noticed that the average buffer occupancy and blocking probability increase as the decoding time-out limit increases.

Furthermore, we make plots for the average buffer occupancy, blocking probability, and system throughput in Figures 15, 16, and 17, respectively, when employing three

Table 1: ABO, blocking probability, and system throughput ($\beta = 1.0, T = 10$).

Inputs			
Channel condition $\beta = 1$		Number of threads = 3	
System capacity $N = 900$		Decoding time-out limit = 10	
Outputs			
Incoming packet probability (λ)	Average buffer occupancy (ABO)	Blocking probability (BP)	System throughput (ST)
0.5	6.984940	0	0.50000
0.6	8.000000	0	0.60000
0.7	9.481100	0	0.70000
0.8	10.96340	0	0.80000
0.9	12.71200	0	0.90000
1	15.58585	0	1

Table 2: Average buffer occupancy, blocking probability, and system throughput ($\beta = 1.0, T = 100$).

Inputs			
Channel condition $\beta = 1$		Number of threads = 3	
System capacity $N = 900$		Decoding time-out limit = 100	
Outputs			
Incoming packet probability (λ)	Average buffer occupancy (ABO)	Blocking probability (BP)	System throughput (ST)
0.5	8.50511000	0	0.5000000
0.6	17.9068175	0	0.6000000
0.7	24.3839000	0.00022021	0.6998458
0.8	41.9367775	0.00676657	0.7945867
0.9	73.2118150	0.00742776	0.8933150
1	97.3598000	0.00990248	0.9900975

sequential decoders to be compared with ABO, BP, and ST in Figures 6, 7, and 8, respectively, when operating just one sequential decoder.

One important observation which can be noticed from Figure 17 is that the system throughput is almost the same when using 10 and 100 for different values of T. Furthermore, the system throughput value is equal to the slope that represents the incoming packet probability. In fact, this indicates that every packet arrival is served. On the other side, consider the behavior of system throughput in Figure 8. Firstly, the system throughput for $T = 10$ is much larger than in $T = 100$ for single sequential decoder. Secondly, the values of the system throughout are almost constant regardless of significant increase in the incoming packet probability for single sequential decoder. This fixed value is about 0.325 for the case of $T = 10$ and about 0.185 for the case of $T = 100$.

One more interesting observation which can be extracted from both Figures 16 and 17 is that choosing larger value of T may not accordingly affect the blocking probability and system that much as in the case of one sequential decoder. Basically, this is a wonderful gain that leads to better QoD. Actually, in the case of a single decoder, we are so worried about employing large value of T (although it is important to do that in case of noisy channels) since it affects the blocking probability (see Figure 7) and system throughput (see Figure 8) negatively. Consequently, we get a worse QoD.

Table 3 shows the average buffer occupancy, blocking probability, and system throughput for a different channel condition (0.4). It is shown that the average buffer occupancy and blocking probability increase as the packet arriving probability increases. From the comparison between Table 3 and Table 2, it is seen that the average occupancy and blocking probability increase as the channel condition decreases. It is also interesting to see that even for the worst conditions (like $\beta = 0.4$), the router system with parallel sequential decoder is stable and reacts much better than that with a sequential decoder having higher values of channel conditions (better conditions); for comparison, see Figures 5, 6, 7, and 8.

Figure 18 illustrates the average buffer occupancy versus incoming packet probability when employing different number of sequential decoders (3 and 5). The inputs for this figure are the system capacity of 900, decoding time-out limit of 100, and channel condition $\beta = 0.6$. It is obvious that the average buffer occupancy decreases as the number of sequential decoders increases. Also the average buffer occupancy increases as incoming packet probability increases.

Figure 19 presents the blocking probability versus incoming packet probability. The simulation inputs are the same as in Figure 18. In fact, as shown, the blocking probability increases as the number of sequential decoders decreases. It is seen also that the blocking probability increases as the incoming packet probability increases.

Table 3: ABO, blocking probability, and system throughput ($\beta = 0.4$, $T = 100$).

Inputs			
Channel condition $\beta = 0.4$		Number of threads = 3	
System capacity $N = 900$		Decoding time-out limit = 100	
Outputs			
Incoming packet probability (λ)	Average buffer occupancy (ABO)	Blocking probability (BP)	System throughput (ST)
0.5	72.8308000	0.0030000	0.4985000
0.6	127.053160	0.0142560	0.5914464
0.7	177.936585	0.0518800	0.6636840
0.8	245.082800	0.0530000	0.7576000
0.9	357.098457	0.0848193	0.8236086
1	512.467600	0.1065770	0.8934230

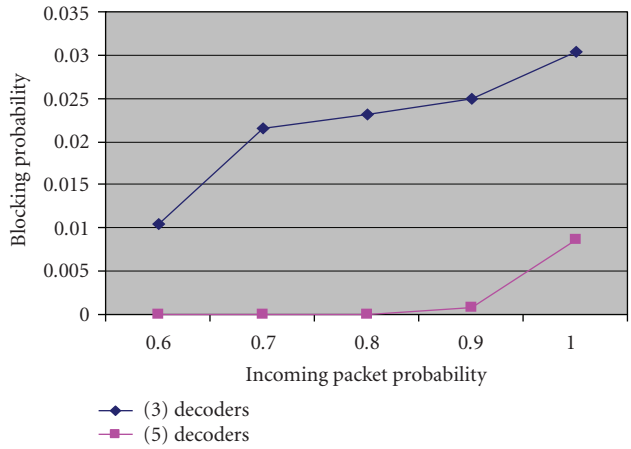

Figure 19: Blocking probability versus incoming packet probability for different number of sequential decoders.

We use different value of channel condition ($\beta = 0.6$) for further parallel system validation. Therefore, it is seen from Figures 18 and 19 and Table 3 that the average buffer occupancy and blocking probability decrease as channel condition increases for fixed number of sequential decoders and decoding time-out limit. On the other hand, from Figures 15, 16, 18, and 19, the average buffer occupancy and blocking probability increase for fixed decoding time-out limit and number of sequential decoders when the channel condition decreases.

6. CONCLUSION

In this paper, we use different discrete time Markov simulation environments to study the QoD of a router system with a finite buffer for wireless network systems that employ stop-and-wait ARQ. The packet errors depend on the channel condition. In other words, if packet error rate is high, this means that the channel is noisy. To make a packet decoding adaptive with a channel condition instead of affording fixed decoding, we use sequential decoding. Since we have finite buffer size, it is clearly noticed that choosing it too small or too large leads to high system flow and congestion over the network. Thus, worse QoD is inevitable. We make our first simulation with just one sequential decoder with very large buffer. Our results show the instability in the router system with bad QoD. Consequently, we have designed a parallel sequential decoding system to mitigate the traffic, and thus getting better QoD. The parallel sequential decoding system has significant improvements on blocking probability, average buffer occupancy, decoding failure probability, system throughput, and channel throughput. We explain extensively through detailed flowchart both simulation environments. We can conclude that increasing buffer size, when the buffer gets full, is just a temporary solution and it is likely to yield severe problems on the network congestion. Rather than investing money on buffers to get worse QoD, the use of parallel sequential decoding systems provides more stable and reliable environment for network congestion.

REFERENCES

[1] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environments: problems and solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27–S32, 2005.

[2] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[3] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, O'Reilly Media, Sebastopol, Calif, USA, 2nd edition, 2005.

[4] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, Morgan Kaufmann, San Francisco, Calif, USA, 3rd edition, 2003.

[5] W. Stallings, *High Speed Networks and Internets: Performance and Quality of Service*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 2001.

[6] N. Olifer and V. Olifer, *Computer Networks: Principles, Technologies and Protocols for Network Design*, John Wiley & Sons, New York, NY, USA, 2006.

[7] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2nd edition, 2004.

[8] R. Sundaresan and S. Verdú, "Sequential decoding for the exponential server timing channel," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 705–709, 2000.

[9] R. O. Ozdag and P. A. Beerel, "A channel based asynchronous low power high performance standard-cell based sequential

decoder implemented with QDI templates," in *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems (ASYNC '04)*, pp. 187–197, Crete, Greece, April 2004.

[10] D. G. Sachs, I. Kozintsev, M. Yeung, and D. L. Jones, "Hybrid ARQ for robust video streaming over wireless LANs," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '01)*, pp. 317–321, Las Vegas, Nev, USA, April 2001.

[11] S. Y. Chang, A. Anastasopoulos, and W. E. Stark, "Energy and delay analysis of wireless networks with ARQ," in *Proceedings of the 61st IEEE Vehicular Technology Conference (VTC '05)*, vol. 4, pp. 2601–2605, Stockholm, Sweden, May-June 2005.

[12] L.-J. Chen, T. Sun, and Y.-C. Chen, "Improving bluetooth EDR data throughput using FEC and interleaving," in *Proceedings of the 2nd International Conference on Mobile Ad-hoc and Sensor Networks (MSN '06)*, vol. 4325 of *Lecture Notes in Computer Science*, pp. 724–735, Hong Kong, December 2006.

[13] E. Ferro and F. Potorti, "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison," *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12–26, 2005.

[14] G. Kabatiansky, E. Krouk, and S. Semenov, *Error Correcting Coding and Security for Data Networks: Analysis of the Superchannel Concept*, John Wiley & Sons, New York, NY, USA, 2005.

[15] R. Togneri and C. J. S. deSilva, *Fundamentals of Information Theory and Coding Design*, Discrete Mathematics and Its Applications, Chapman & Hall/CRC, Boca Raton, Fla, USA, 2003.

[16] R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*, Wiley-IEEE, Piscataway, NJ, USA, 1999.

[17] Y. S. Han, P.-N. Chen, and H.-B. Wu, "A maximum-likelihood soft-decision sequential decoding algorithm for binary convolutional codes," *IEEE Transactions on Communications*, vol. 50, no. 2, pp. 173–178, 2002.

[18] J. B. Anderson and S. Mohan, "Sequential coding algorithms: a survey and cost analysis," *IEEE Transactions on Communications*, vol. 32, no. 2, pp. 169–176, 1984.

[19] S. Kallel and D. Haccoun, "Sequential decoding with an efficient partial retransmission ARQ strategy," *IEEE Transactions on Communications*, vol. 39, no. 2, pp. 208–213, 1991.

[20] S. Kallel and D. Haccoun, "Sequential decoding with ARQ and code combining: a robust hybrid FEC/ARQ system," *IEEE Transactions on Communications*, vol. 36, no. 7, pp. 773–780, 1988.

[21] P. Orten and A. Svensson, "Sequential decoding in future mobile communications," in *Proceedings of the 8th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '97)*, vol. 3, pp. 1186–1190, Helsinki, Finland, September 1997.

[22] W. D. Pan and A. Ortega, "Buffer control for variable complexity Fano decoders," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '01)*, vol. 1, pp. 176–180, San Antonio, Tex, USA, November 2001.

[23] Y. S. Han and P.-N. Chen, "Sequential decoding of convolutional codes," in *Encyclopedia of Telecommunications*, pp. 2140–2146, John Wiley & Sons, New York, NY, USA, 2002, Book chapter.

[24] P. Y. Pau and D. Haccoun, "An analysis of sequential decoding with retransmission procedures," Tech. Rep. EMP/RT-85-19, Ecole Polytechnique of Montreal, PQ, Canada, 1985.

[25] A. Drukarev and D. J. Costello Jr., "Hybrid ARQ error control using sequential decoding," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 521–535, 1983.

[26] J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*, John Wiley & Sons, New York, NY, USA, 2006.

[27] K. Darabkh and R. Aygün, "Quality of delivery evaluation of error control for TCP/IP-based systems in packet switching ATM networks," in *Proceedings of the International Conference on Internet Computing (ICOMP '06)*, Las Vegas, Nev, USA, June 2006.

[28] T. Hashimoto, "Bounds on a probability for the heavy tailed distribution and the probability of deficient decoding in sequential decoding," *IEEE Transactions on Information Theory*, vol. 51, no. 3, pp. 990–1002, 2005.

[29] W. D. Pan, "Queuing analysis of sequential decoders with buffers," in *Proceedings of the Huntsville Simulation Conference (HSC '04)*, Huntsville, Ala, USA, November 2004.

[30] N. Shacham, "ARQ with sequential decoding of packetized data: queueing analysis," *IEEE Transactions on Communications*, vol. 32, no. 10, pp. 1118–1127, 1984.

[31] K. Darabkh and R. Aygün, "Performance evaluation of sequential decoding system for UDP-based systems for wireless multimedia networks," in *Proceedings of the International Conference on Wireless Networks (ICWN '06)*, Las Vegas, Nev, USA, June 2006.

[32] K. Darabkh and W. D. Pan, "Stationary queue-size distribution for variable complexity sequential decoders with large timeout," in *Proceedings of the 44th Annual Southeast Regional Conference (ACMSE '06)*, pp. 331–336, Melbourne, Fla, USA, March 2006.

[33] K. Darabkh and R. Aygün, "Simulation of performance evaluation of error control for packet-to-packet acknowledgment based systems in ATM networks," in *Proceedings of SCS International Conference on Modeling and Simulation—Methodology, Tools, Software Applications (M&S-MTSA '06)*, Calgary, Canada, July-August 2006.

[34] K. Darabkh and W. D. Pan, "Queue-size distribution for Fano decoders," in *Proceedings of the Huntsville Simulation Conference (HSC '05)*, Huntsville, Ala, USA, November 2005.

[35] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*, Addison-Wesley, Reading, Mass, USA, 2003.

[36] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2004.