CHAPTER 3

ERROR CONTROL STRATEGIES

This chapter focuses on the proposed error control strategies and algorithms. Because high error rate cannot be prevented in the wireless environment, power efficiency is an important issue for mobile computing systems. Power is consumed by the physical radio transmission process, computation, signal processing, error control at the transmitter and the receiver, etc.

As explained earlier, two error control strategies have been popular in practice. They are the FEC strategy, which uses error correction alone, and the ARQ strategy which uses error detection combined with retransmission of corrupted data. The ARQ strategy is generally preferred for several reasons. The main reason is that the number of overhead bits needed to implement an error detection scheme is much less then the number of bits needed to correct the same error while the FEC strategy is mainly used in links where retransmission is impossible or impractical. When the FEC strategy is used, the transmitter sends redundant information along with the original bits and the receiver tries its best to find and correct errors.

The rest of the chapter is organized as follows. In Section 3.1, we describe the channel model used in simulation and in the following section, we discuss the error control alternatives and their strategies in order to investigate the efficiency of the error control algorithm. Finally, in Section 3.3, we describe the design of adaptive error control algorithm and their specification.

3.1 The Error Model

In any communication systems, there have always been errors and the need to deal with them. Wireless networks have a much higher error rate than wired networks. The errors that occur on the physical channel are caused by phenomena such as signal fading, transmission interference, and user mobility. Here we describe a model for the wireless channel.

Wang and Moayeri (1995) shows how the statistical time varying nature of the channel can reliably be represented by a Discrete Time Markov Chain (DTMC). The channel in this model is considered to have a number of different states, each representing a different level of fading. The state of the channel is only allowed to change on bit boundaries, such that the model includes the underlying assumption that the channel condition will not change during the reception time of a bit. Thus, the channel model parameters are independent of packet length because this bit level channel model allows channel conditions to vary within a packet.

For each state of the DTMC, it is associated with a binary symmetric channel. The BER associated with each state is a function f of the Signal to Noise Ratio (SNR), where f is imposed by the modulation scheme used. In this experiment, we represent the channel as a two state DTMC with state space *{Good, Bad}* following the Rayleigh model. The transition probabilities between the two states are depicted in Figure 3.1.

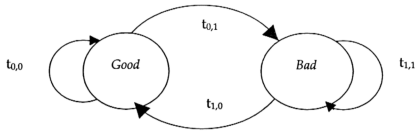

Figure 3.1: Two State Discrete Time Markov Chain

Many researchers have used the Rayleigh fading channel to evaluate the performance of coding schemes. The Rayleigh distribution describes the fluctuation of the received envelope of a flat fading signal with no direct line of sight component as is typical of urban environments.

As the mobile moves through space, it passes through regions of good and bad reception, where the signal strength is either acceptable or not acceptable. When the mobile speed changes, the rate at which these transitions occur and their duration follow proportionally. In the extreme, as the speed of the mobile approaches infinity, bursts become infinitely short but the time between bursts shrinks to nothing as well to keep the overall mean BER constant, as mean BER is independent of speed.

Therefore, in characterizing the channel, there are two variables of importance. First is the *Good* state BER, a function of SNR at the receiver (degraded by such things as path loss etc.). In the simulation, we fixed the *Bad* state BER to 0.5, while allowing up the *Good* state BER to vary over a wide range, between of 10^{-2} to 10^{-8}. The second variable of interest is the speed of the mobile, which dictates the *burstiness* of the channel. In chapter 5, the simulation results are presented in terms of what happens at varying speeds, or correspondingly, levels of burstiness.

As mentioned previously, there are two important variables that need to be considered when characterizing the wireless channel. First, the BER - a function of SNR at the receiver, and second the burstiness of the errors on the channel. This leads to two basic classes of errors: packet erasures and bit corruption errors (Eckhardt, 1996). Error control is applied to handle these errors. Note that when the bit errors are independent, the Packet Error Rate (PER) is related to the size of the packet (L) and the Bit Error Rate (BER) as

$$PER = 1 - (1 - BER)^L$$

While this does not take into account the bursty nature of a wireless link, it gives an idea of the influence of the packet length on the error rate of a packet. Even one uncorrected bit error inside a packet will result in the loss of the packet. Each lost packet directly results in wasted energy consumption, wasted bandwidth, and in time spent. This loss might also result in the additional signaling overhead of an ARQ protocol (Lettieri, 1998a). Because of this, it is important to simultaneously adapt the error control mechanism as packet size increases to minimize the number of transitions between two states.

3.2 Error Control Alternatives

There is a large variety of error control strategies, each with their own strengths and weaknesses in terms of latency, throughput, and energy efficiency. As mentioned in the previous chapter, there are basically two methods for dealing with errors: ARQ and FEC schemes. Hybrids of these two schemes also exist. Within each category,

there are numerous options. Computer communication generally implements a reliable data transfer using either methods or a combination of them at different levels in the communication protocol stack. FEC is normally used at the data link layer to reduce the impact of errors in the wireless connection. In many cases, these codes provide less than perfect protection and some amount of residual errors pass through.

When using FEC, redundancy bits attached to a packet allow the receiver to correct errors which may occur. In principle, FEC incurs a fixed overhead for every packet, irrespective of the channel conditions. This implies a reduction of the achievable data rate and causes additional delay. When the channel is good, this overhead is still incurred. Areas of application that can benefit from error correction mechanisms are *multicast applications* (Stemm, 1996). Even if the QoS requirement is not that demanding, ensuring the QoS for all receiving applications is difficult with retransmission techniques since multiple receivers can experience losses on different packets. Individual repairs are not only extremely expensive, they also do not scale well to the number of receivers. Reducing the amount of feedback by the use of FEC leads to a simple, scalable and energy efficient protocol.

Using ARQ, feedback is propagated in the reverse direction to inform the sender of the status of packets sent. The use of ARQ results in an even more significant increase in delay and delay variations than FEC (Schuler, 1998). The retransmission requires additional buffering at the transmitter and receiver. A large penalty is paid in waiting for, and carrying out the retransmission of the packet. This can be unacceptable for systems where QoS provisioning is a major concern, for example in wireless ATM systems.

ARQ schemes perform well when the channel is good, since retransmissions are rare, but performs poorly when channel conditions degrade since much effort is spent in retransmitting packets. Another often ignored side effect in ARQ schemes is that the round-trip-delay of a request-acknowledge can also cause the receiver to be waiting for the acknowledge with the receiver turned 'on', thus, wasting energy.

The next is a hybrid of these two schemes. Hybrids do not have to transmit with maximum FEC redundancy to deal with the worst possible channel. Under poor channel condition, FEC should be used o avoid many retransmissions. Although more efficient than the pure categories, a hybrid system is still a rigid one since certain channel conditions are assumed.

An adaptive error control allows the error control strategy to vary as the channel conditions vary. The error control can be FEC, ARQ, or a hybrid. The wireless channel quality is a function of the distance of a user from the base station, local and average fading conditions, interference variations, and other factors. Furthermore, in packet data systems the bursty nature of data traffic also causes rapid changes in interference characteristics. In a wireless channel, link adaptations should occur frequently because of the rapid changes in signal and interference environment. In such a dynamic environment it is likely that any of the previous schemes is not optimal in terms of energy efficiency all the time. Adaptive error control seems likely a source of efficiency gain.

An adaptive error control can be added fairly easily to a MAC protocol and link layer protocols. First of all, the adaptive error control techniques have to be present in the sender and the receiver.

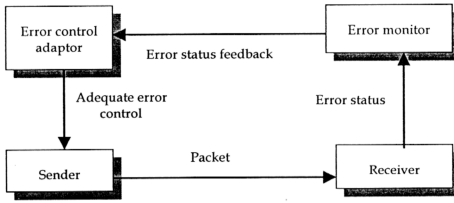

Figure 3.2: Feedback loop for adaptive error control

Secondly, a feedback loop as shown in Figure 3.2 is required to allow the transmitter to adapt the error coding according to the error rate observed at the receiver. Normally, such information consists of parameters such as mean Carrier-to-Interference ratio (C/I) or Signal-to-Noise Ratio (SNR), standard deviation of SNR channel impulse response characterization, bit error statistics (mean and standard deviation), and packet error rate. The required feedback loop limits the responsiveness to the wireless link conditions. Additional information can be gathered with a technique that performs link adaptation in an implicit manner by purely relying on acknowledgement information from the radio link layer.

The design goal of an error control system is to find optimum output parameters for a given set of input parameters, e.g. channel BER or maximum delay. Examples of output parameters are FEC code rate and retransmission limit. The optimum output might be defined as maximum throughput, minimum delay, or minimum energy consumption, depending on the service class (or QoS) of a connection. Real-time traffic would prefer minimum delay, while most traditional data services would prefer a

maximum throughput solution. All solutions in a mobile environment should strive for minimal energy consumption.

3.3 Algorithms Specification

In this section, a description of algorithms for the simulator development is given. We have designed the algorithms for the purpose of examining which error control strategies should be used in a wireless environment with stringent QoS, and which parameters should be applied to obtain the maximum throughput solution for a variety of error conditions existing in a wireless environment.

In this study, we intend to improve the algorithm developed by Lettieri et al specifically to make it adaptive. In this case, our adaptive error control algorithm might be able to compensate for the changing conditions by changing which type of ARQ or FEC it is using. However, in their work, a comparison of the performance of various error control strategies is given without concentrates on adaptive algorithm.

For this purpose, every time a new packet arrived at *Channel* entity, each bit will be check for the error, count and insert that amount of error to the *PacketMsg* message before sending to the *Receiver* entity. For Good state, if the random number generated by *pc_erand()* is less than the default value of BER good than error will be add to the error counter and the status of that bit was assumed has been filled . Otherwise, for Bad state, if the random number generated by *pc_erand()* is less than the default value of BER bad than error will be add to the error counter and the status of that bit was assumed has been filled. The process is executed until all bit of packet has been parsed.

Since we have tested various error control strategies over the channel conditions, we can select the best performance in terms of the number of packet dropped, BER, user throughput etc, from various hybrid schemes for a given channel condition. In this work, we only test ATM packet type.

The algorithm is split into three sub-algorithms so that it is easier to understand. The sub-algorithms are explained below.

3.3.1 The Non-Adaptive Error Control Algorithm

The first sub-algorithm is designed to be executed by the transmitter. The second and third are for the channel and receiver, respectively. For each phase, a brief description is elaborated with regards to entire simulation algorithm.

Phase One (Transmitter):

1. Set up parameter and create buffer space.
2. While *data-sent* less than *total-data*, buffer still empty
   - 2.1 Determine if data source type is *data* or *speech*
     - 2.1.1 If *data*, have request data from source first.
     - 2.1.2 If *speech*, directly receive data from source.
   - 2.2 Form the packet and perform coding process if required.
   - 2.3 Put the data into actual packet.
3. Request the channel, if granted then send the oldest located packet.
4. If an acknowledgement is activated
   - 4.1 Waits for an acknowledgement to arrive
     - 4.1.1 If an acknowledgement arrives, go to step 3.
       - 4.1.2 If after timeout, no acknowledgement has arrived, perform retransmission.
5. Go to step 3.
6. Algorithm ends for this phase.

A Brief Description:

In this phase, a parameter for ATM packet type is set up, followed by the creating of buffer space. Here, parameter used for packet is head and body. This is the second entity (first is *Source* entity) in the simulation that transmits data to the *Receiver* entity. While the buffer is still empty and amount of current data is less than total data, then it requests the data source type either a speech or data type sources. For speech source type, a new data is produced at a fixed rate and directly feed to the *Sender* entity at fixed intervals. For data source type, the *Sender* entity requests the data directly from the *Source* entity. After that, the *Sender* entity segments the data and putting the data into an actual packet.

After that, the entity sends a request message to the *Channel* entity. If the request is granted, in another word it is ready to receive a packet, it sends a response message to this entity. After this notification, the *Sender* entity sends a packet to the *Channel* entity. If acknowledgements are activated, it waits for an acknowledgement before proceeding. If after a timeout, no acknowledgement arrives, a retransmission is performed. Error control coding such as checksum is applied here as required.

Phase Two (Channel):

1. For each new transfer request from a sender, determine if it is free
    - 1.1 If it is free, go to step 2.
    - 1.2 If it is in use, go to step 3.
2. Granted, send a *clear-to-send* message to the sender
    - 2.1 Wait for packet to arrive from the sender.
    - 2.2 Errors are added to the data, go through the data one bit at a time
    - 2.3 Send the packet to the receiver's queue after the channel delay.
3. Algorithm ends for this phase.

A Brief Description:

The *Channel* entity waits for request to send a message from other entities and grants it if the channel is not in use. If granted, no further request is serviced until a packet has been taken in. Errors are injected to the data and passed to the receiver's queue with a delay factor specified by the channel delay. For a given packet size, number of errors applied in the *Channel* entity based on the channel model used are returned and inserts those errors into the data buffer. This mechanism has been explained in Section 3.3. Any time a *Sender* and *Receiver* entity wants to communicate with the *Channel* entity, it must assert a *RequestToSendMsg* message, which the channel responds to with a *ClearToSendMsg* message if it is free, and then waits for a packet only from the *Sender* entity who was granted access.

Phase Three (Receiver/Error Control module):

1. Wait for incoming packets from the channel
    - 1.1 If no more packet arrives, go to step 5.
    - 1.2 If packet arrives, go to step 2.
2. Read the packet
    - 2.1 Clear out the buffer used before.
    - 2.2 Performs decoding process, if no FEC assume it was good channel.
    - 2.3 Do a spot check on the header, need to consider ARQ type.
3. If an acknowledgement is enabled and the packet is error-free
    - 3.1 Send an acknowledgement back to the channel.
4. Go to step 1.
5. Algorithm terminates.

A Brief Description:

The *Receiver* entity is the final destination for a packet after parsing through the *Channel* entity. In this phase, the *Receiver* entity waits for packets to arrive from the *Channel* entity. After receiving a packet, the decoding process is performed on each packet. If acknowledgements are enabled and the packet is error free, then an acknowledgement is sent back to the *Channel* entity. Otherwise, no further action is taken. Once the entire file has been sent to the *Receiver* entity, the simulation terminates and a some of results are displayed to the user, including energy consumed by both sender and receiver, the number of packets dropped etc. The details about simulation results are discussed in Chapter 5.

3.3.2 The Adaptive Error Control Algorithm

The most complex part of the adaptive algorithm is located on receiver side, where the packet errors are detected, estimate the current channel BER and the efficient error control can be determined. The next data transmission is executed, the channel BER rate from the previous execution is used to estimate which of error control (Reed-Solomon code, SACK) scheme is beneficial.

Basically, this algorithm requires no priori knowledge of the channel BER or channel conditions. The only knowledge necessary is a good estimate of the packet error probability and channel BER. The adaptive algorithm learns and adapts its decision based on previous execution of data transmission. The important task in this algorithm is the ability to develop communication between sender and receiver in order to indicate a new parameter setting involved which includes measurement of

packet error rate, estimation of channel BER and a suitable error control type. For example, if the algorithm has detected a change of the channel condition, then it will react to this change by modifying certain error control parameters and will used this kind of error control for the next data transmission. The flowchart for this algorithm is depicted in Figure 3.3 below.

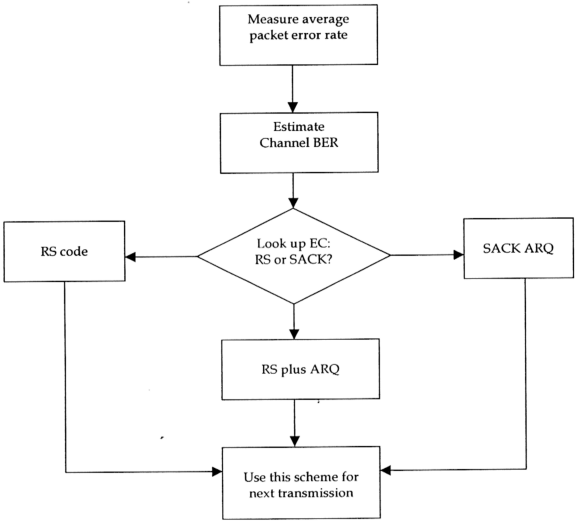

Figure 3.3: Adaptive algorithm at Receiver side

3.3.3 The Basic ARQ Algorithm used in the Simulation

Before explaining the simulation methodology, we describe the basic ARQ algorithms used in our simulation. As mentioned in the previous chapter, there are four options of error control schemes. We explain each of them in the scope of our simulation.

Stop and Wait

This is a simple algorithm where each packet is acknowledged individually. The transmitter waits for each packet to be acknowledged before sending the next packet. A time out ensures that we do not reach a state of deadlock.

Go Back N

Each packet is acknowledged, but the transmitter does not wait for the acknowledgement. If an acknowledgement (ACK) is received out of sequence at the transmitter, this means either a lost packet or a lost ACK since the link is a serial one. In the first case, we need to retransmit the packet, while in the second case, we will retransmit needlessly. Either way the transmitter sets the buffer pointer back to the first lost packet (as per the first ACK not received) and starts retransmitting from there. Anything already in the pipe is assumed lost by the transmitter and abandoned by the receiver (who knows to abandon it by the later reception of an earlier packet).

Alternatively, a time out is available for each packet such that if its ACK is not received within a specified time, the buffer will be reset as above to the first offending packet, and anything already in transit is again abandoned.

Cumulative ACK

Once acknowledgement packet acknowledges all previous packet, i.e. an acknowledgement for packet N acknowledges packet N and all packets before N. If a packet is lost, the receiver stop sending ACK packets. The sender would time out retransmit the outstanding unacknowledged packets.

Selective ACK

This scheme tends to require the fewest retransmissions for a given set of lost packets. Whenever a packet is received, an ACK which describes the last N packets received correctly is sent. N bits of the ACK packet correspond to N packets, where a one in a bit position corresponds to a successfully received packet and a zero corresponds to a failed packet. Another part of the ACK packet gives the offset from which the N bits start counting. Upon reception of the ACK, failed packets are put back into the queue for retransmission. If at any time a packet in the queue is found to have been successfully received, it is dequeued. If a packet is already in the queue and noted in the ACK packet as having not made it, it is simply left in the queue.