Carnegie Mellon University
Information Networking Institute

A Hybrid Approach to Effort-Limited Fair (ELF) Scheduling for 802.11

A Thesis Submitted to the
Information Networking Institute
in Partial Fulfillment of the Requirements
for the Degree

MASTER OF SCIENCE
in
INFORMATION NETWORKING

by

David Matsumoto

Thesis Advisor: Peter Steenkiste
Thesis Reader: David Eckhardt

Pittsburgh, Pennsylvania
August 4, 2003

Table of Contents

List of Figures and Tables

Abstract

Providing quality of service (QoS) in wireless networks is a research topic that is still maturing. As more people use wireless technology for various applications, there is demand to provide some level of QoS for these applications. However, there are several challenges to providing QoS in a wireless network that would not normally exist in a wireline network. A variety of dynamic and location-dependent error environments can make it difficult to employ “fair” scheduling on a wireless link. Although techniques have been introduced to reduce the amount of lost link capacity, some errors still result in variable link capacity.

Effort-Limited Fair (ELF) scheduling was proposed by Eckhardt & Steenkiste [1] to extend Weighted Fair Queuing (WFQ) for wireless networks. ELF guarantees that all flows experiencing an error rate below a per-flow threshold receive their expected service, defined as a specified rate for reserved flows, or a specified share of best-effort capacity for best-effort flows.

This thesis builds upon the ELF scheduling model, and shows that it can be integrated into a simulated 802.11 implementation. This could be a substantial contribution towards providing QoS to the average wireless link user, since most commercial networks currently employ the 802.11 protocol. We also seek to take advantage of the distributed nature of the 802.11 MAC protocol, and extend ELF scheduling to operate in both a distributed and centralized environment. We propose a practical algorithm intended to remain faithful to ELF’s original design principles, while employing a hybrid scheduling model. Finally, we evaluate this proposed algorithm through both trace-driven simulation and computer generated error models.

1 Introduction

Wireless networks are inherently different from wireline networks in several ways. For example, wireless signals are unprotected from outside signals and must travel over a medium significantly less reliable than wireline networks. They have dynamic topologies and lack full connectivity (which means that the assumption that every host can hear every other host is invalid), and have time-varying and asymmetric propagation properties. Because of these differences, many assumptions that are normally made about wireline networks do not hold. Thus, providing quality of service (QoS) over wireless networks is a challenge.

Wireless media can exhibit high, variable error rates that affect network users in a number of ways. Most applications and end-to-end transport protocols perform poorly when many packets are lost due to link errors. Furthermore, link errors reduce the useful throughput of the link. Even though techniques have been introduced to reduce the amount of lost link capacity [2] [3], some errors still result in variable link capacity. This high variability in link error rates makes it extremely difficult to make guarantees for individual flows, and for a scheduler to meet its commitments in all circumstances. Unlike a wireline network (which can assume almost no link errors, and virtually constant bandwidth), a wireless network can have huge fluctuations in bandwidth availability, which forces variable transmission rates. In the presence of this variability, one must decide which flows should be short-changed. Which flows should get preferential treatment when bandwidth becomes scarce? Also, in a wireless network with location-dependent errors, we must decide how much extra airtime to assign to high-error stations at the expense of others. These problems center on the question of how a scheduler should respond to capacity loss.

In this thesis, we first introduce the Effort-Limited Fair (ELF) scheduling model presented in [1]. We describe its design principles, and go into some detail on the ELF algorithm and implementation. We then discuss the IEEE 802.11

MAC protocol [4], specifically focusing on the Distributed Control Function (DCF) and Point Control Function (PCF). Next, we show that it is possible to integrate an ELF scheduler into a standard 802.11 implementation, and thus ensure sensible outcomes for flows in response to unrecoverable capacity loss. After showing that ELF scheduling can be an effective policy for an 802.11 PCF, we propose a novel notion for improving the efficiency of ELF scheduler within an 802.11 MAC by using a hybrid approach to regulate stations within both a DCF & PCF. Finally, after motivating and defining this hybrid approach, we present a practical algorithm which we evaluate both through trace-driven simulation and computer generated error models. Considerable time was spent laying the groundwork to run these simulations. Specifically, we added much more functionality to the 802.11 MAC implementation in *ns-2* [5]. This included ironing out several bugs along with extending the functionality of the PCF. After this groundwork was laid, we integrated both the ELF algorithm and our extension into the *ns-2* code base.

2 Earlier Work

There has already been a fair amount of work done on ways to provide QoS in Wireless Local Area Networks (WLANs). Currently, the IEEE 802.11 standard [4] for WLANs is the most widely used WLAN standard. There is a mode of operation in IEEE 802.11 that can be used to provide service differentiation (PCF – Point Control Function), but it has been shown to perform badly and give poor link utilization [6]. Other schemes include Distributed Fair Scheduling [7], Blackburst [8] and a scheme proposed by Deng *et al.* [9]. Comparisons of these schemes have been done in various simulations, and it has been shown that the PCF mode of the IEEE 802.11 standard performs poorly compared to the other schemes evaluated [10].

One scheduler model that has been proposed is the Effort-Limited Fair (ELF) Scheduler for wireless networks. This model is based on the observation that while “effort” (air time spent on a flow) and “outcome” (actual useful

throughput achieved by the flow) are equal in a wireline environment, they can be substantially different in a wireless environment. The ELF scheduler strives to achieve the outcome that is envisioned by users subject to limits on the effort spent on each flow by using a per-flow *power factor* setting. The power factor is a control knob that can be used to administratively implement a variety of fairness and efficiency policies. ELF essentially extends the Weighted Fair Queuing (WFQ) model to allow for dynamic weight adjustments. The ELF scheduling model itself can be compared to other scheduling models [1]. This includes the wireless packet scheduler (WPS) [2], the Channel-condition Independent packet Fair Queuing (CIF-Q) algorithm [11], the Server-Based Fairness approach [12], and the utility-fair bandwidth allocation approach [13]. Each of these gives an explicit model of the desired outcome of a scheduling algorithm in the face of errors. However, the ELF scheduling approach argues that in certain situations, when flows encounter errors, it is important to give special treatment to these error prone flows [14]. This can allow best-effort flows to obtain some amount of outcome fairness, even if this entails slightly reducing overall link efficiency. However, it must also be possible to protect the link against flows with very high error rates that can take up all the available bandwidth.

3 The Thesis

The primary thrust behind this thesis comes from previous work done exploring ways to provide an acceptable level of quality of service to individual flows, in the face of error-prone wireless links.

The thesis: **It is possible to integrate an Effort-Limited Fair scheduler (ELF) into a standard 802.11 implementation, and thus ensure sensible outcomes for flows in response to unrecoverable capacity loss. Moreover, it is possible to improve on the efficiency of a PCF-based ELF scheduler within an 802.11 MAC by using a hybrid approach to regulate stations within both DCF & PCF.**

4 IEEE 802.11 Protocol Overview and Modifications

The 802.11 MAC protocol has many layers, and its specifications are voluminous. We focused primarily on the MAC sub-layer, including the distributed coordination function (DCF), the point coordination function (PCF), and their coexistence in an IEEE 802.11 LAN. Special attention was given to the integration of the ELF scheduler into the MAC sub-layer.

The fundamental access method of the 802.11 MAC is DCF, which is implemented by all the stations within a basic service set. For a station to transmit, it senses if another station is transmitting (using *carrier sense multiple access with collision avoidance* (CSMA/CA)). If the medium is not busy, the transmission may proceed. For frames above a certain size threshold, it may be beneficial to use RTS (Request to Send) and CTS (Clear to Send) frames. This method helps to further minimize collisions among stations during a DCF. A station that is using RTS/CTS first sends an RTS frame to the destination station. It would receive a CTS frame from the destination station, and then be cleared to send its Data frame. The sending station would then wait for the destination station to send an acknowledgement frame (ACK) back to itself. If the station senses that the channel is busy, it will defer until the end of the current transmission, and then back off (at some random interval) while it senses that the medium is idle. Following this interval, the station may send a packet (providing it detects an idle medium). In the event that two frames collide, both stations perform a random backoff procedure, and then try to send their frames again.

The 802.11 MAC also has another optional access method called PCF, which is only usable on network configurations that utilize an access point (AP). This access method uses a point coordinator (PC), which operates at the AP, to determine which station has the right to transmit. Basically, the PC performs the role of polling master. The access priority provided by a PCF can be utilized to

create a contention-free (CF) access method. Thus, the PC controls the frame transmissions of the stations so as to eliminate contention for a limited period of time. When polled by a PC, a pollable station may transmit only one frame, which can be to any destination (not just to the PC), although the frame must be forwarded on from the PC. If the data frame is not in turn acknowledged, the station will not retransmit the frame unless the PC polls it again, or it decides to retransmit during the contention period.

The DCF and the PCF are able to coexist within the same network. The two access methods alternate, with a contention-free period (CFP) followed by a contention period (CP) (see Figure 1 below). The protocol specifications state that at the beginning of a CFP, a beacon frame must be sent out to all stations, informing them that the PC now has control over the medium. (The PC gains control of the medium through the use of a shorter inter-frame spacing (PIFS), see Section 9.2.3.1 of [4].) At the end of a CFP, the PC will send out another beacon frame signifying that a CP is now in progress under the DCF. This alternation continues indefinitely.

In general, the minimum amount of time allotted for a CFP is equivalent to the amount of time needed to send one data frame to a station, while polling that station, and for the polled station to respond with one data frame. Moreover, the minimum amount of time allotted for a CP is equivalent to the amount of time needed to send at least one data frame during a CP.

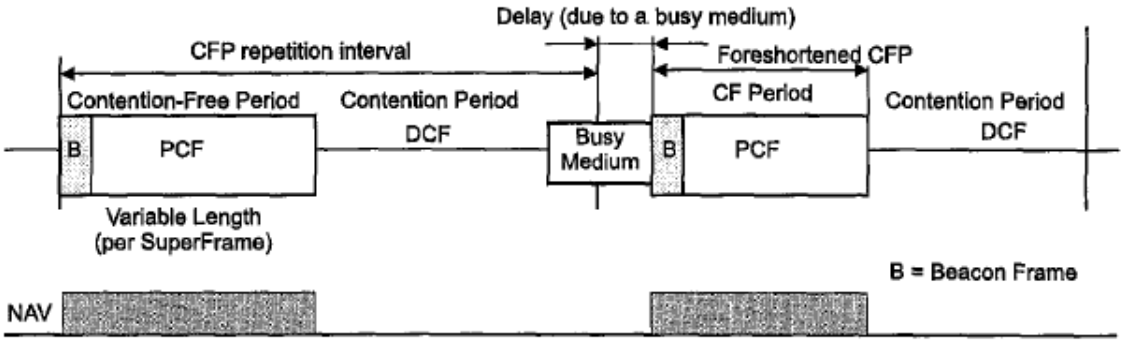

Figure 1: CFP/CP Alternation (excerpted from [4])

IEEE's 802.11 is now a networking standard, and used by academia and industry worldwide. Its attractiveness comes from its design to be efficient and robust. However, although there has been work done in recent past (i.e., 802.11e drafts [17]), there is no current standard built into 802.11 to provide quality of service (QoS) to mobile hosts. The closest thing that fits that criterion is the (optional) PCF, which most commercial implementations have chosen to omit. There are probably at least two main reasons why PCF has not been used.

First, DCF is more efficient to use when stations do not have data queued at all times (which is a majority of the time). It will allow a station to contend for airtime to send a frame, and provides a method to ensure reliable transmission of the packet (i.e., acknowledgements and re-tries), up to a certain threshold of retransmissions. Provided that the collision rate is not high, DCF is efficient, and can deal with hidden stations by using the RTS/CTS protocol. On the other hand, PCF is effective primarily because it does not have to contend for airtime, and thus, there should be no collisions during a CFP. Note that the PCF is quite efficient if there is extremely high contention on the channel (e.g., stations have data queued at all times), since the PC is likely to receive a data packet back from the polled station, and bypass the RTS/CTS protocol and collision recovery. For example, an average RTS/CTS transaction consumes roughly the same amount of air-time that a 100-byte data packet would consume. If there are many RTS/CTS transactions required (e.g., for an average packet size of 512 bytes), this can consume a significant amount of air-time and result in approximately a 16% inefficiency figure. However, if contention is low, then the PC is likely to receive many NULL frames back from stations that have no data queued. Thus air time will be wasted while preventing stations that *do* have data to send from sending. For example, suppose that there are four stations sharing a wireless channel, but only one station has data to send. If the PC polls each of those stations per CFP, this can unnecessarily limit the throughput for that one station, while wasting the significant airtime on the polling the other three stations.

Second, the PC is allowed to maintain a polling list and decide which station is allowed to transmit next. According to the specifications, “During each CFP, the PC shall issue polls to a subset of the stations on the polling list in order by ascending (association ID) value. While time remains in the CFP, all CF frames have been delivered, and all stations on the polling list have been polled, the PC may generate one or more CF-Polls to *any* stations on the polling list…or send data or management frames to *any* stations.” While this specification does allow for some flexibility, it is rigid in determining the order in which polls are initially sent out to stations. This rigidity makes it difficult to provide reasonable levels of QoS to different flows.

Our goal is to take the existing standard, and modify it so that it can provide a reasonable quality of service to mobile hosts, even in the face of high error rates. We believe that the Effort-Limited Fair scheduling model designed by Eckhardt & Steenkiste provides a plausible scheduling algorithm for providing QoS to flows in the presence of capacity loss. Moreover, when ELF is integrated as the policy for the PCF, the PCF can be useful for dealing fairly with flows experiencing high error rates that still demand some level of throughput.

5 Effort-Limited Fair (ELF) Wireless Link Scheduling

The design philosophy of the ELF scheduler is based on a few principles, taken from [1]:

1) In an error-free environment, the outcome achieved by a wireless scheduler should be identical to that of an equivalent wireline scheduler.
2) The amount of capacity loss suffered by a flow should not be proportional to its bandwidth or its error rate, but should be configurable through administrative controls.

3) It must be possible to administratively bound the amount of capacity that is lost due to location-dependent errors.
4) In the absence of information to the contrary, flows experiencing equal error rates should experience the same capacity loss.
5) Capacity unused by one flow should be distributed "fairly" among other flows.

In light of these considerations, ELF scheduling wants to provide QoS such that it balances effort fairness with outcome fairness. When error rates are low, the scheduler should give outcome fairness to best effort flows (the throughput that they would expect with a low error rate). With moderate error rates, the scheduler will use priority to support reserved flows. When error rates are high, the scheduler will fall back to effort fairness, so that the link still provides some useful throughput.

ELF introduces the notion of a "power factor", which basically provides the administrative bound that balances effort fairness with outcome fairness.

ELF is an addition to a Weighted Fair Queuing (WFQ) model, as shown by the following equation:

$$A_i = \min(\frac{W_i}{1 - E_i}, P_i \bullet W_i)$$

A_i = adjusted weight of flow i, W_i = weight, E_i = error rate, P_i = power factor

This equation states that we adjust the weights of different flows, depending on what error rate each flow is experiencing. However, the adjusted weight of any given flow cannot exceed that which is defined by its power factor. A flow that experiences 100% error rate for example can be prevented from starving all other flows in the channel. However, with an appropriate power

factor, a flow with a moderate error rate *can* be given enough airtime to achieve its outcome. In general, setting a flow's power factor to 100% will cause it to be scheduled in an effort-fair fashion, while raising its power factor will cause it to experience outcome fairness over a wider range of error rates.

There is also a virtual clock that ticks each loop through the scheduler. Each time a flow's transmission interval passes (the interval is calculated according to the flow's weight), the scheduler allocates the flow both deserve and effort. The flow is then eligible for transmission until it either achieves that throughput or exhausts that effort. Note that the amount of effort a flow is allocated per tick is a function of the flow's power factor. Thus, flows with higher power factors can use up more effort when they are experiencing error conditions in order to meet throughput requirements.

Guaranteed flows are separated from best-effort flows in that the scheduler will first try to select an eligible guaranteed flow before moving to best-effort flows. Thus, guaranteed flows are served until each is either satisfied with its outcome or limited by expended effort. The best-effort flows share the leftover bandwidth. This is achieved by having the scheduler only tick the best-effort flows' clock when there is no guaranteed flow ready to transmit. This is a consequence of the fact that the best-effort flow is almost always over committed. If no guaranteed flow is eligible and there are no best-effort flows, the scheduler will slightly advance the schedule of the next eligible guaranteed flow.

For example, suppose there are two best-effort flows and one video flow. The link capacity is 800 kbit/s, and the video flow requires 200 kbit/s of throughput (one-quarter the total air time available). Thus, the two best-effort flows split the remaining bandwidth equally (i.e., each flow receives 300 kbit/s). Suppose the video flow has a 50% error rate, and a power factor of 200. According the equation above, the video flow will still receive 200 kbit/s of throughput (since it will now consume half the air time), while the best-effort flows

will split the remaining bandwidth equally (i.e., each flow receives 200 kbit/s). Thus, the video flow will meet its reservation requirements and the best-effort flows will avoid starvation even if the error rate rises above 50%.

Eckhardt shows that ELF performs as expected on various simulations and traces capturing “real-world” environments. The simulator assumed all flows are continuously busy, and records throughput allocation decisions made by the scheduler (ignoring how real transport protocols might react to allocation variations). Each flow on the simulator is assigned “deserve” and “effort” banks. The deserve bank keeps track of how much outcome the flow deserves (i.e., throughput). The effort bank keeps track of how much effort (i.e., air-time) the flow is allowed to spend.

The link layer used in his simulations was based on earlier work [15]. It uses link-level re-transmit, adaptive packet sizing, and adaptive error coding to reclaim substantial capacity even in the face of challenging error patterns. In addition to the simulator-based evaluation, Eckhardt tested an ELF implementation. He inserted the ELF scheduler into his existing prototype wireless LAN, built from Intel 80486 and Pentium laptops running NetBSD 1.2 and using 915 MHz PCMCIA card WaveLAN units. The kernel device driver included a simplified poll/response MAC protocol, which was similar in spirit to the IEEE 802.11 PCF. However, he suggests that possible extensions would include using ELF as the policy for an 802.11 PCF. He writes, “If we assume that one station is in a position to discover the outcomes of most packet transmissions, the LAN could operate according to the more efficient DCF most of the time, and the controlling station could then use the PCF period to allocate extra effort to stations according to their power factors.” This thesis is an attempt to take up that challenge.

6 ELF Implementation within IEEE 802.11 on ns-2 (Network Simulator)

Network Simulator-2 (ns-2) [5] provides a convenient platform on which to conduct experiments. On *ns-2*, we are able to plug in data from trace files, as well as use random error models to simulate traffic across wireless channels. This allows us to simulate location-dependent errors, and test boundary cases. *Ns-2* also provides well-established plumbing for each node: We are able to attach a variety of applications (i.e., FTP, Constant bit rate, web flows, etc…) to a TCP SACK or UDP implementation that connects down to the physical layer, which attempts to model the real world.

We extended the mobile node for an 802.11 implementation, so that ELF could be integrated as the scheduler for the AP. Most of the implementation work was done at the MAC layer in ns-2, building upon earlier work done by the Monarch group on mobile ad-hoc networking [19]. We also extended Anders Lindgren's PCF implementation [10] to make it more conformant to the 802.11 protocol specifications. Our ELF implementation can be divided into two main components:

1) Extending 802.11 PCF functionality.
2) Implementing ELF as the scheduler for the PCF.

Figure 2 below shows a simplified schematic layout of a typical mobile node in *ns-2*.

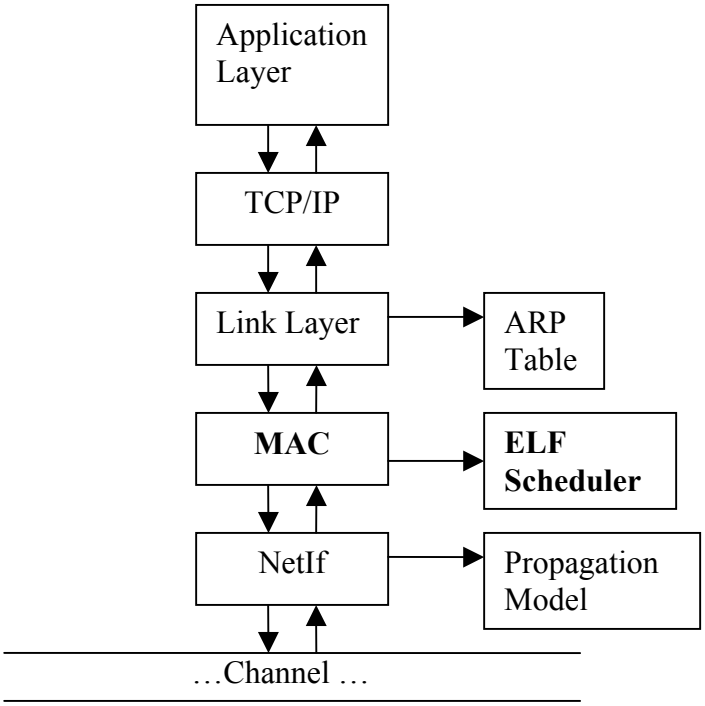

Figure 2: Schematic of a mobile node in *ns*-2

We made several assumptions upon implementing these components, which allowed us to reduce complexity and still produce useful results.

First, we assume that the scheduler views traffic as a set of flows. Flows can be individual (e.g., a single TCP connection) or aggregates (e.g., all traffic to a specific host). The 802.11 PCF specifications handle polling on a per station granularity. Thus, the ability to poll an individual flow on a station is not part of the PCF protocol. Thus, for our purposes, we deal with flows as aggregates, knowing that this can be extended to deal with individual flows if we modify the protocol. Second, the ELF algorithm is meant to run over a traditional WFQ scheduler. However, for implementation purposes, we use packet weighted round robin (WRR) instead of WFQ, measuring throughput in packet slots. This is for

simplicity, and also because wireless networks are often slot-based for synchronization and power-management purposes. Third, in order to guarantee fixed-rate reservations, we assume that there is an admission control module that can set appropriate per-flow power factors. The manner in which this module would choose power factors is outside the scope of this paper, but obviously it would need to avoid over committing the link, given an expected error rate. Eckhardt does provide some guidance [1, 14] on how to select appropriate power factors. Fourth, our implementation is concerned with throughput for flows. It may be possible to implement into ELF a scheduler which offers more complex service characterizations (i.e., deadline-based, delay or jitter guarantees, etc…), but that requires tracking individual packet outcomes and is beyond the scope of this work. Fifth, both the CBR and TCP flows always have data to send. This means that a drop in throughput among any of the flows is not due to idle time, but rather either congestion control or high error rates. Finally, we do not handle data fragments as specified by 802.11, and in fact all CBR flows send fixed sized packets. Each packet transmission fully succeeds or is dropped.

6.1 Extending 802.11 PCF Functionality

The *ns-2* implementation that we began with included support for DCF & partial PCF. The DCF included CSMA/CA along with support for using RTS/CTS. The PCF allowed the PC to poll stations for data, and forward that data on (e.g., to a wired node). If a station did not have any data to send upon receiving a poll, the station would not send anything and there would be a timeout. After the timeout, the PC would continue polling until it reaches the end of its polling list or the CFP ended.

We added to that implementation by adding bi-directional data flow during a CFP. This means the PC had to be able to poll stations for data, as well as maintain a set of queues for each flow, to pass data to a station that is next to be polled. This was necessary to allow us to incorporate TCP flows (into the

experiments which are bi-directional, requiring both Data and Ack packets). Significant amount of time was spent reworking the functionality of the timers to allow for bi-directional data flow, and to account for decisions made by the ELF scheduler. For simplicity (and in conformance with the protocol specifications), the decision on whether or not to send a station data or a poll was decided by whether or not there was data in the queue to send to that station. If there remained data to be sent, it would be sent to that station in place of a poll. Otherwise, that station would be polled for data.

Our implementation is does not fully implement the PCF, in that we do not allow for all possible frame types to be sent. For example, we do not currently allow for Acks to be piggybacked onto Data frames. We also do not do fragmenting as specified by the 802.11 protocol. Finally, station association and authentication are not considered in our implementation.

6.2 Implementing ELF as the scheduler for PCF

The core of the ELF algorithm remains the same as Eckhardt's model described above.

If the PC polls a station, and does not receive any data (or the data is corrupted due to errors), it charges the station the amount of effort equivalent to one data packet. However, it does not reduce the amount of throughput the station deserves. If the station did receive the poll, but has no data to return, it can send a NULL frame. The PC will then reduce both the effort and deserve "accounts" (variables) for the station. Since acknowledgements (ACK) are built into the protocol, we take advantage of this when updating the "effort" and "deserve" variables for each flow. Specifically, when the PC sends data to a station, if the station returns a MAC level ACK, the PC will reduce both deserve and effort for that station. However, if the ACK is lost (or the data packet was

never received by the station), the PC will reduce effort, but leave the deserve quantity unchanged.

When we add a CFP, our implementation deviates from the 802.11 standard in one aspect. Instead of polling stations in the order of their station ID value, we use the ELF algorithm to decide which station should be polled next.

We note that although ELF controls the polling of stations during a CFP, it has no control over how stations operate during a CP. The CP remains completely distributed, with stations able to contend for as much bandwidth as they desire. Obviously, the longer the CFP as a percentage of the superframe, the more ELF will be able to meet its guarantees, and provide QoS to flows.

7 ELF-DCF: Improving upon ELF for DCF/PCF operation

We stated above that the DCF is more efficient than the PCF in general. Thus, our motivation was to design an extension to ELF that was as consistent as possible with the original design of ELF, and allowed more flexibility with respect to the distribution of time allotted for a CP and a CFP. Specifically, we wanted to have more time with the DCF operating, so that we would improve overall link efficiency, but still remain able to meet the same throughput goals per flow that ELF provides.

We first discuss two design principles that we used to develop the algorithm for ELF-DCF[1]. Then we explain our implementation, and how it is

[1] From this point forward, we use "ELF-PCF" to refer to the original ELF algorithm designed by Eckhardt, operating under a CFP. "ELF-DCF" refers to our new algorithm, which operates under a CP. They can operate independently of each other (i.e., one can be enabled, while the other is disabled), but are designed to work together to achieve ELF's design goals.

consistent with our design principles. Finally, we discuss a few low-level details that we needed to consider in deciding how to charge flows.

7.1 Design Principles

1) **DCF must continue to remain completely distributed. Stations must continue to contend for air-time in exactly the same fashion.** We want to minimize any sacrifice in efficiency during a DCF, and thus minimize any interference the PC has with the mobile stations. This includes any type of polling, or other centralized control that would limit stations from sending data.

2) **The integration of ELF (PCF) & ELF-DCF should stick as closely as possible to the standard.** Since PCF and DCF alternate, we need to make sure that ELF continues running smoothly during each transition. This helps to keep the implementation simple, and also minimize the risk of unnecessary spikes or dips in throughput.

7.2 Design

The basic idea is to allow the ELF scheduler to work as before (during the CFP), but to give it an additional role.

- The scheduler will use the data in the effort and deserve banks for each flow to decide how much effort and deserve each flow should be allowed during the next CP.
- At the end of a CFP, the AP will send out a beacon that alerts each of the stations that a CP has begun. Inside the beacon will be information inserted by the AP's ELF scheduler that is pertinent to individual flows.
- When receiving the beacon, each flow will know exactly how much effort and outcome it is allowed during that specific CP, and will regulate itself so

that it does not exceed its limits. Obviously, since the scheduler can assign different effort and deserve values per flow during a CP, it can help flows that might be experiencing high error rates to meet their throughput expectations during a CP. This would be consistent with the original design of ELF.

- At the end of a CP, the scheduler would be in a position to discover how each station has performed during the CP, and make the appropriate scheduling adjustments during the CFP. For example, a flow that achieved low throughput during the CP might be given extra airtime during the CFP to help meet throughput expectations. (see Figure 3 below)

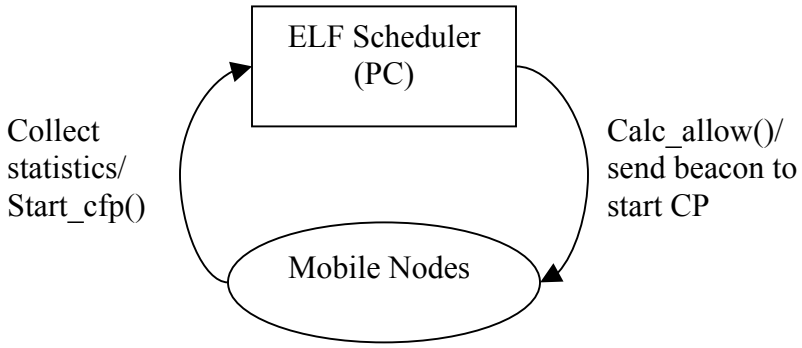

Figure 3: ELF Design (High Level)

There must be a sensible way of deciding how to regulate stations during a CP. Our objective is to allow stations to achieve their desired throughput during a DCF, while aiding error-prone stations in a way that is consistent with the goals of ELF. ELF uses a few variables to regulate flows: *interval* (which establishes the weight of the flow), *power* (power factor), *effort*, and *deserve*. We described above how each of these variables are used during normal ELF operation.

Normally, in a DCF, stations have no constraints to work with except CSMA/CA. Therefore, they will continue to contend for airtime as long as they have data to send. However, it *is* possible to have stations regulate themselves beyond CSMA/CA (i.e., a station defers from sending a packet due to some

constraint on effort spent or throughput limits), by taking advantage of the use of beacons. We used these beacons in our implementation to send out specific information unique to each station that told them “how to behave” during a DCF. Our goal was to leave the DCF being “distributed” and de-centralized, but still regulate the nature of its distribution.

7.2.1 Allocating Effort & Deserve

The algorithm for deciding what regulatory values to send each station is critical. It makes sense to send each station a “deserve” and “effort” value (with equivalent meaning to those used by the ELF scheduler operating on the PC). The deserve value would tell a station how much throughput it is allocated during this CP. The effort value would tell a station how much airtime it is allowed to use during a CP. Sending these values also helps us to maintain a seamless continuation from CP to CFP. In particular, we would like to use the values for “effort” and “deserve” coming out of a CFP, and partially use them to compute the new CP values (without altering either account unfairly).

It is critical to send each station *both* of these values. If we only told a station how much throughput it deserves, then actually we would be encouraging error-prone stations to do the very thing ELF seeks to prevent. Specifically, an error prone station would continue contending for airtime indefinitely, until it met its “deserve” quota. This could have the effect of starving out other stations and monopolizing the bandwidth.

In the case of “deserve,” it makes sense to allocate a station all of the remaining amount of deserve (coming out of a CFP), plus whatever throughput the station would expect during a CP. Even if the amount of deserve is much higher than can actually be expended during a CP, this would be constrained by a more rigid “effort” value.

Allocating effort in the same way as deserve (see above) would be undesirable for a few reasons. Ideally, each station should use up all the effort we allocate for it. However, experiments showed that often when a station experienced a high error rate, the amount of effort it expended would drop significantly. This may be due to large intervals of time in back-off, or waiting for a timeout so it can retransmit. Therefore, assigning the error prone station a large allocation of effort would do little good during the CP, since it wouldn't use it up. On the other hand, one station may have a substantial amount of "left over" effort from the CFP. Therefore, combining that with the effort that would normally be expended (assuming no errors) during a CP could over allocate effort, so as to allow an "effort inclined" station to monopolize the bandwidth during a CP. In general, we would like to have a good statistic (estimation) of a station's willingness to expend effort. This helps us to determine how much effort to allot a station during a CP.

We would also like to assist error-prone stations in a similar way as ELF does during a CFP. Specifically, we'd like to make use of the power factor, giving a needy station enough effort (up to a bound) to receive its desired throughput. However, we cannot allocate effort in the same way during a CP. During a CFP, the PC credits a station with enough effort for one packet, multiplied by its power factor. However, the PC decides when each station gets to send data by polling them. Under a DCF, there is no such regulation. Thus, suppose there is a station with a higher power factor, but is not experiencing any errors. If we allocate that station more effort during a CP (proportional to its power factor), then it may start to use more bandwidth than is desirable from the scheduler's standpoint. This is because the station may have a large amount of deserve accumulated, and would now have unrestricted use of the air time as long as it has effort to spend. We needed a way to allocate additional effort *as needed* during a CP.

7.2.2 The ELF-DCF Algorithm

Based on these factors, we designed the algorithm shown below (see calc_allow in Figure 4). We believe it meets the design criteria, and takes into account each of the implementation concerns mentioned above. First, we discuss how the algorithm works, and deals effectively with error conditions. We discuss exactly how we take into account the fact that stations have different willingness to expend effort, and also how we manage the “effort” and “deserve” banks for each flow. Then we give a brief description of our pseudo-code below, and discuss the main variables involved in the implementation.

For each flow, we keep a history of past effort utilization (as a percentage of effort allocated) during a CP. This history spans five previous CP’s, though this number can be experimented with further. We also keep track of the immediately preceding CP’s effort utilization. The logic is that recent history is a more accurate predictor than older history. We weigh the average of the history table’s utilization with the value for the last CP’s effort utilization, and compute a moving average. We use this to predict a station’s willingness to expend effort in the next CP.

Then we allocate a station’s “deserve” by combining its current “deserve” value with what is expected during this CP. The “effort” is allocated based on the moving average value. If it is less than 50%, then we allocate that station’s CP effort as a percentage (i.e., the moving average percentage) of what it would normally expect in a CP. The remaining effort that is unused is factored back into the “effort” value that will be used during a CFP. This way, the station is not penalized unfairly for unexpended effort during a CP. If the moving average value is greater than 50%, then we take that to mean the station has a higher willingness to spend effort. In this case, effort allocation is a sum of two components: 1) we give a station as much effort as would be needed to meet its

throughput requests during an error-free CP. 2) we recognize that this station may have use for more effort (to make up for unmet throughput requests due to high error rates). Therefore, we allocate extra effort according to its power factor, as a percentage of one minus the moving average value. The purpose is to give a *willing* station the opportunity to expend extra effort during times of high error rates, with the amount of extra effort being inversely proportional to how much effort was utilized in previous periods with low utilization. Note that in both cases, we assign a value for "effort" that is independent of the remaining "effort" in the CFP (to avoid "bloating" the effort allocation in the CP). However, we do conserve that remaining effort for future use, so we do not penalize that station unfairly. One might argue that extra effort should be assigned as a function of percentage "deserve" underutilization in previous CP's. However, since the "deserve" values can become increasingly large, the moving average of "deserve" underutilization could grow too small, and thus not assign extra effort fairly. We choose not to use two different "deserve" values for a CP and CFP (like we did for "effort") to stay as close as possible to the goal for seamless integration as described above. In Table 1 below, we give a short description of the key variables used.

Variable (per flow state)	Brief Description
Flow->deserve	Amount of outcome a flow deserves
Flow->effort	Amount of effort a flow is entitled to
Flow->rts_reserve	Internal administrative information; informs the PC how much effort was allowed for RTS frames during a CP
Flow->reserve	Amount of outcome a flow deserves per CP (constant value)
Mix_avg_effort[i]	Weighted average value used to determine if a flow is effort averse/prone
Cfend_frame->deserve[i]	Deserve value incorporated into beacon to let station know how much outcome it deserves for that CP.
Cfend_frame->effort[i]	Effort value incorporated into beacon to let station know how much effort it is entitled to for that CP.

Table 1: List of state variables

/* Setup beacon frame to regulate stations' throughput distribution during a CP. */

```
void Mac802_11::calc_allow(Beacon CF-end frame){
      int rts_reserve[MAX_STA] = 0;
      int unused_deserved_effort[MAX_STA] = 0;

foreach flow (Protected & Unprotected){

      /* Calculate weighted average of effort spent during a CP.
      50% of weight comes from last CP's effort statistic; 50% comes
      from the previous 5 history's of effort averaged together; */

      mix_avg_effort[flow->macId] = ((50% * recent_avg_effort[flow-
      >macId]) + (50% * hist_avg_effort[flow->macId]));

      cfend_frame->deserve[j] = flow->deserve + flow->reserve;

      if(mix_avg_effort[j] <= 50%){

            /* The station hasn't demonstrated that it has much
            inclination to spend effort, so give it less allocation. */

            cfend_frame->effort[j] = (mix_avg_effort[j] *
                  flow->reserve * 150%);
            flow->rts_reserve = (mix_avg_effort[j] * flow->reserve *
      50%);
            unused_deserved_effort[j] = (1 - mix_avg_effort[j]) *
                  flow->reserve * flow->power;
      }
      else{

            /* Give it extra effort (with the power factor) to try and
            meet its requirements */

            cfend_frame->effort[j] = (flow->reserve * 150%) + (1 -
                  mix_avg_effort[j]) * (flow->reserve * 150% * flow-
                  >power));
            flow->rts_reserve = (1 * flow->reserve * 50%) + ((1 -
                  mix_avg_effort[j]) * flow->reserve * 50% * flow-
                  >power);
      }

      /* Update figures to be used after CP is finished */
      flow->deserve_allowed = cfend_frame->deserve[flow->macId];
      flow->effort_allowed = cfend_frame->effort[flow->macId];
      flow->deserve = 0;
      flow->effort += unused_deserved_effort[flow->macId];

      flow = flow->next;
      }
return;
}
```

```
/* Before beginning polling stations, plug statistics from the CP into ELF. */

void Mac802_11::start_cfp(){
int effort_gap = 0, deserve_gap = 0;
foreach flow (Protected & Unprotected){

      /* History for effort outcome during CP */
      flow->hist_effort[(flow->mod)%5] = flow->recent_effort;
      flow->allow_hist_effort[(flow->mod++)%5] =
            flow->recent_effort_allowed ;

      /* Most recent history (deserve & effort) for CP */
      flow->recent_deserve = flow->cp_deserve_outcome;
      flow->recent_deserve_allowed = flow->deserve_allowed;
      flow->recent_effort = flow->cp_effort_outcome;
      flow->recent_effort_allowed = flow->effort_allowed;

      effort_gap = flow->recent_effort_allowed –
            flow->recent_effort;
      deserve_gap = flow->recent_deserve_allowed –
            flow->recent_deserve;

      /* reduce gap by accounting for RTS inflation */
      if(effort_gap > flow->rts_reserve)
            effort_gap -= flow->rts_reserve;
      else
            effort_gap = 0;

      /* Reset temporary storage variables */

      /* Update main ELF statistics */
      flow->deserve += deserve_gap;
      flow->effort += effort_gap;
      effort_gap = 0;
      flow = flow->next;
      }
sendPoll();
}
```

Figure 4: Pseudo-code for the ELF-DCF algorithm

7.2.3 Handling statistical data received during a CP

We also need to plug statistics received during a CP into ELF's operation during a CFP (see start_cfp function above in Figure 4). Since we set up the algorithm for assigning effort/deserve as we did above, this part of the implementation is more straightforward. After a CP completes, we update the history table, and recent effort and outcome statistics. Then we compute the "lag" (gap) between the amount of "deserve" allocated, and how much outcome (throughput) was actually achieved. We also compute the gap for "effort."

Finally, we update the “effort” and “deserve” values for the CFP simply by adding the respective lag values from the previous CP period. This conserves both values, and allows for more seamless integration between CP and CFP ELF values. Most importantly, the ELF scheduler is in a good position to “fix” any imbalances that might have occurred during the CP now, under the PCF.

We note that although ELF is seamlessly integrated into both CP and CFP with respect to effort & deserve values, there is a departure from the original classification of flows under ELF-PCF. Specifically, under ELF-PCF, we make a distinction between protected (guaranteed) and best-effort flows. However, we make no such distinction in ELF-DCF. This is mainly to reduce the complexity of implementation and processing on the mobile station. Therefore, throughput for best-effort flows will be higher on average during a CP than a CFP.

Obviously, there are several areas that can be explored further. For example, the values for different weights (calculating the moving average), or the determination of how much extra effort to assign, are open to change. However, based on empirical results and analysis of the boundary cases, we believe that this algorithm takes into consideration each of the design criteria given above.

7.3 Policy decisions for charging flows

The PC needs to be able to monitor how stations perform during a CP. As we stated above, stations are able to send data directly, or use an RTS/CTS first (depending on whether the size of the data crosses a certain threshold which can be decided administratively). Thus, it is important for us to account for the fact that RTS/CTS transactions involve considerable overhead. On a 2 Mbps channel, an RTS/CTS transaction costs about 350 microseconds, while a DATA/ACK transaction (where the size of the data is 100 bytes) costs about 400 microseconds. (A data packet of size 512 bytes costs about 2000 microseconds). It is clear that the overhead involved can still be substantial when viewed in

aggregate, especially if one highly error prone station continues to send an RTS unsuccessfully. This leads us to believe that it is reasonable to charge stations for initiating an RTS/CTS transaction during a CP. Our goal is for stations to be charged a “reasonable” amount of effort so as to prevent one station from monopolizing the bandwidth with bogus RTS packets. On the other hand, we recognize that the overhead taken by the RTS/CTS transaction is much smaller than the time for data transmission, and thus should not be charged as much for a regular data packet. We settled on charging 50% of a data packet’s effort as a plausible billing system.

Since we charge stations during a CP for using RTS/CTS, we designed ELF-DCF to give each station enough “effort allowance” so that it can theoretically use up all the “deserve” that it is allotted. For example, if we allocate enough deserve for five packets, then we also allocate (150% * five packets) worth of effort (to account for the possibility of sending an RTS/CTS). Thus, for every data frame that is allocated deserve, we allocate an additional 50% of one data frame’s worth of effort to allow for the RTS/CTS. It is possible for a station to not send RTS/CTS packets if the packet sizes do not cross the threshold. Also, a station may not expend much effort during the CP. In this case, there would be a substantial amount of effort left over at the end of the CP. We take into account that we made an adjustment how much effort was allocated to stations, because of the RTS/CTS overhead required for one data frame transaction. Therefore, since the CFP does not have that overhead, we reduce the gap in effort by whatever incremental effort was allocated to account for RTS/CTS overhead (see start_cfp function above in Figure 4). The effort spent for data frames and RTS both draw from the same effort bank. The logic here is that we want to make sure we deduct all the effort that was allocated for RTS/CTS for that CP. If we were to have two effort banks (i.e., one for RTS/CTS and one for data frames), then at the end of the CP we’d only be able to deduct from the RTS/CTS bank (which may be close to empty). This way, we ensure that the station is fully deducted for all of the RTS/CTS effort that was allocated, or that the station used up all its

effort during the CP. The key point is that a station with a high error rate will be prevented from continuously sending faulty RTS, since they will use up effort with each RTS sent.

Table 2 below briefly illustrates the operation of the ELF scheduler between a DCF & PCF. In this example, we assign both guaranteed flows a power factor of 200, and they are assigned weights such that they each should deserve 4 frames worth of throughput per CP. For simplicity, we assume that the first CFP in the example ends with all values being 0. We also assume that there have been enough superframes to allow the effort history table to accumulate data values. At this snapshot in time, flow 1 has an average effort history of 75% (i.e., effort prone (see above algorithm)), while flow 2 has an average effort history of 50% (i.e., effort averse (see above algorithm)). In this example, we credit 4 frames worth of deserve to flow 1. We credit 150% of 4 frames worth of effort, along with 25% (1 - .75) of the effort normally assigned to flow 1 during a CP (taking the power factor into consideration) for a total of 900. In this example, flow 1 expends 6 RTS worth of effort, along with 3 data frames worth of effort (all data frames are successful). Thus, 3 frames worth of deserve are deducted from flow 1's account, along with 6 frames worth of effort (accounting for the RTS expenditures).

Flow 2 is effort averse according to its effort history table. Therefore, although we credit the same amount of deserve for the CP, the amount of effort allotted is only 50% of what would normally be allotted. The unused effort that was not allotted for the CP is "stored away" to be credited back into the flow for the CFP. In this case, we assume that the flow expends 2 RTS worth of effort, and 2 data frames worth of effort (all data frames are successful). Thus, at the start of the next CFP, flow 2 has 2 frames worth of deserve deducted. However, it also has the remaining effort that was unused from the CP credited back into its effort account (taking the power factor into consideration).

	End of CFP	CP summary	Start of next CFP
Flow 1		75% -effort prone	
Deserve	0	400	100
Effort	0	900	300
RTS Effort spent	0	300	0
Outcome	0	300	0
Flow 2		50% -effort averse	
Deserve	0	400	200
Effort	0	300	400
RTS Effort spent	0	100	0
Outcome	0	200	0

Table 2: Blow-by-blow trace of ELF-DCF & ELF-PCF Scheduling

According to the specifications (for a non ad-hoc network), all stations should send data through the access point, which often acts as the PC. We take advantage of this by having the PC keep track of packets during a CP. In our simulation, we are able to determine the type of an incoming packet (i.e., RTS, CTS, DATA, ACK, etc…), and therefore we know how much to charge a station. Even if the packet is corrupted, we still charge the station for effort spent. However, like the CFP, we only bill a station for "deserve" (throughput) if the data packet's delivery is confirmed. Note that effort and deserve are billed to a given station both for traffic coming *from* the station, and going *to* the station from the PC. This remains consistent for both CP's and CFP's. This is especially important for TCP flows, since both TCP level data and ACK packets are billed on the same flow's account. It seems logical to consider a TCP flow as being *one* flow from ELF's perspective (rather than to break it up into two separate flows) since ELF is concerned about the link-layer's throughput for a particular flow, and doesn't make higher level classifications.

8 Experimental results

8.1 Simulation Setup

We evaluated various possible error configurations, using both computer-generated and real world trace files. We use the computer-generated models to show that the algorithms work as expected for simple scenarios. We also cover the extreme cases, to see where the algorithm (ELF-PCF & ELF-DCF) works, and where it can be improved.

Our experimental setup has four simulated wireless stations, and a base station that serves as the access point (AP). All stations are equipped to run 802.11 DCF/PCF, and we can administratively enable ELF-PCF and/or ELF-DCF. Each station sends one flow to a wired node through the access point. Two stations each manage a CBR flow, while another two stations each manage an FTP flow. The CBR flows are classified as guaranteed flows which are more sensitive to throughput fluctuations, while the FTP flows are classified as best-effort (i.e., non-guaranteed) flows. There are many possible topologies over which the experiments can be run, but we chose this because it is typical of how many flows realistically operate (i.e., wireless node to wired node). We also ran our experiments with data flows going from a wired to a wireless node and found that the results were equivalent, as expected (assuming all other variables are held the constant, e.g., each flow has its own queue on the AP, to prevent one flow's data frames from monopolizing a global queue).

We used four main types of error patterns: 1) Uniform error rate, 2) Multiple state Markov model, 3) "Walls" (a real-world trace file), with a low to moderate error rate, and 4) "Adjacent" (a real-world trace file), with a high error rate. The real-world trace files are derived from Eckhardt's previous experiments [1]. For the computer-generated error rates (i.e., non-trace error rates), there are three possible error states shown in the graphs below: 0%, 50%, 100%. Flows

with a Markov state model error rate fluctuated between these three states. Each station (flow) can have its own error rate.

In each experiment, we modify six main variables. First, we decide what the **duration of each superframe** should be. Next, we decide what **percentage of each superframe** should be devoted to a CFP (the remaining time would be used for a CP). This is important because we want to be able to track the effectiveness of both ELF-PCF (CFP) and ELF-DCF (CP). We would expect that the larger percentage CFP, the closer we will be to meeting ELF's goals. Third, we are able to **enable/disable ELF-DCF**, so we can see how much of a difference it makes from just using ELF-PCF. Specifically, we want to know how much additional throughput an error-prone flow might receive, and how much of a throughput drop the other flows might suffer. It is also important to know how much overall channel throughput changes as a result of ELF-DCF. Fourth, we effectively control whether RTS/CTS frames are sent by each flow, by controlling the frame sizes sent by each flow, and the corresponding **RTS/CTS threshold**. In the results below, we mainly discuss flows that did not use RTS/CTS, since most real-world traffic frames tend to be small and not require RTS/CTS (i.e., average 402 bytes [16]). However, we did run experiments using RTS/CTS, and found that ELF-DCF/PCF performs similarly to those experiments that did not use RTS/CTS. The main difference is that the throughput fluctuations tend to be smaller across all the flows, with a small drop in overall throughput due to the additional overhead. This drop in throughput jitter is probably because we reduce the probability of collisions involving large frames. Finally, we set both the **power factor** and **error rate** for each flow. A flow with a power factor of 100% should be scheduled in an effort-fair fashion, and raising its power factor will cause it to experience outcome fairness over a wider range of error rates.

8.2 Uniform error rate

We ran a series of experiments having one flow with a uniform error rate, and the rest of the flows having a 0% error rate (besides collisions, etc…). We want to show first that ELF-PCF (the original ELF algorithm) works well under 802.11 by running 100% CFP with ELF-DCF disabled. Then we show a simulation at 50% CFP with ELF-DCF disabled to show that ELF-PCF gives some assistance, but not enough to meet throughput expectations. Finally, we show that ELF-PCF (50% CFP) used together with ELF-DCF can effectively boost throughput to acceptable levels for the error prone flow.

Each of the CBR flows sends constant sized packets of 512 bytes each, with the transmission rate set to 125 packets/second (a theoretical throughput of 512 Kbit/s). All flows have a weight of 25%. The link has a maximum practical throughput level of approximately 1.5 Mbit/s. When we run all flows with 0% error rate, with each superframe being 100% CFP, we find that each of the CBR flows receive about 500 Kbit/s, while the best-effort flows receive about 250 Kbit/s (see Table 3 below). This is expected, since we are purposely trying to fill the link with traffic to capacity, so we can see how flows are treated with a loss in total capacity. Also, note that as mentioned in section 7.3, the ELF scheduler bills ACKs from a TCP flow on the same account (and for the same amount) as the Data packets. Thus, it makes sense that the best-effort flows receive half as much throughput as the CBR flows (which run over UDP).

Alternatively, when we retained the 0% error rate but changed from 100% CFP to 100% CP, we find that each of the CBR flows receives about 450 Kbit/s, while the best-effort flows receive about 200 Kbit/s a piece (see Table 3 below). Thus, the total throughput possible on the link is about 1.3 Mbit/s. This decrease in overall throughput (200 Kbit/s decrease) is because there are virtually no collisions when running only PCF. Thus, if PCF receives data each time it polls a

station, it will give higher throughput than DCF. The fact is that this scenario is extremely unlikely, and there are indeed significant idle times where stations do not return data upon receiving a poll. Thus, from the standpoint of increasing overall throughput, it is normally more desirable to run a DCF.

% CFP	CBR flow throughput (Kbit/s)	TCP flow throughput (Kbit/s)	Total throughput (Mbit/s)
0	500	250	1.5
100	450	200	1.3

Table 3: Throughput comparison with boundary superframe cases (0% error rate)

When we increase the error rate for one of the CBR flows (CBR(1)) to 50%, we can start to observe ELF in action (see Figures 5-8). First, observe that if we run only DCF (i.e., 100% CP), the error prone flow receives only 113 Kbit/s (see Table 4 below), while the best-effort flows have substantially more throughput (see Figure 5 below). However, if we use ELF-PCF (without ELF-DCF), we see that it does a remarkable job of regaining throughput for the error-prone flow. In fact, there is a gain of almost 300 Kbit/s (see Figure 6 below). The jitter for each of the flows significantly decreases as well. Note that the other three flows do suffer a capacity loss, but the best-effort flows receive the greatest drop (which is expected). Overall, running strictly ELF-PCF under 802.11 works well, and achieves its objectives.

If we reduce ELF-PCF to 50% CFP, we can see that throughput for the error-prone flow falls to only about 275 Kbit/s (see Figure 7 below). However, if we turn ELF-DCF on, the throughput rises to about 355 Kbit/s. The other CBR flow's throughput falls only marginally, while the best-effort flows suffer a drop to about 143 Kbit/s (see Figure 8 below). Clearly, while the error-prone CBR flow suffers a drop in throughput, ELF-DCF manages to adjust both CBR flows to have comparable throughput (which would be expected with a 200% power factor and a 50% error rate). Also, the total loss in best-effort throughput is about 120 Kbit/s. However, this is consistent with ELF's goals since the error-prone

flow gains about 80 Kbit/s. Note that the overall channel throughput does decrease when we enable ELF-DCF, but we found that as we increase the percentage CFP and/or enable ELF-DCF, the throughput jitter decreases. Comparing Figures 5 & 8, we can see that by doubling the amount of air time spent on CBR(1), we gained almost 250 Kbit/s of throughput, while the other flows suffer a cumulative loss of about 370 Kbit/s in throughput. This successfully demonstrates that ELF-DCF can achieve ELF's design goals.

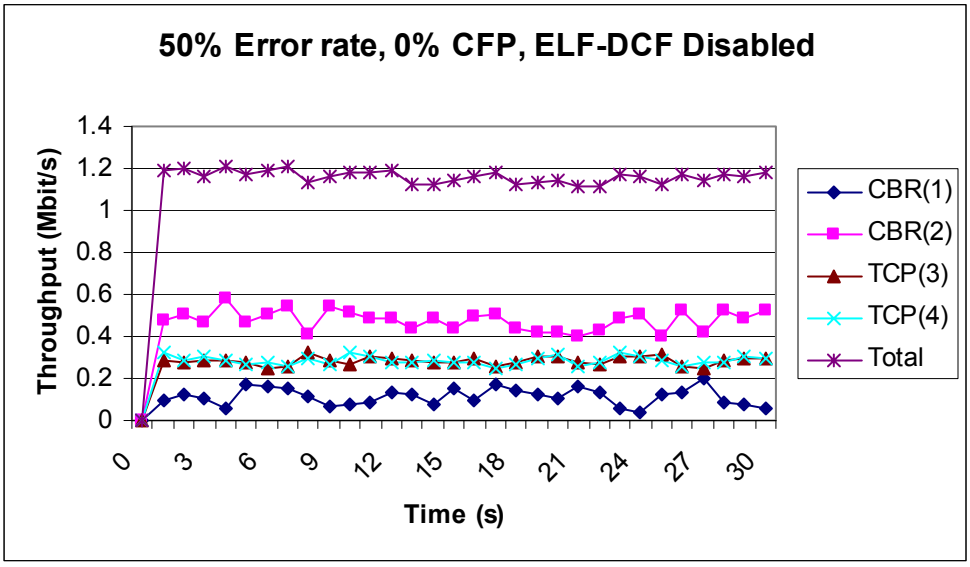

Figure 5: 50% Error rate, 0% CFP, ELF-DCF Disabled

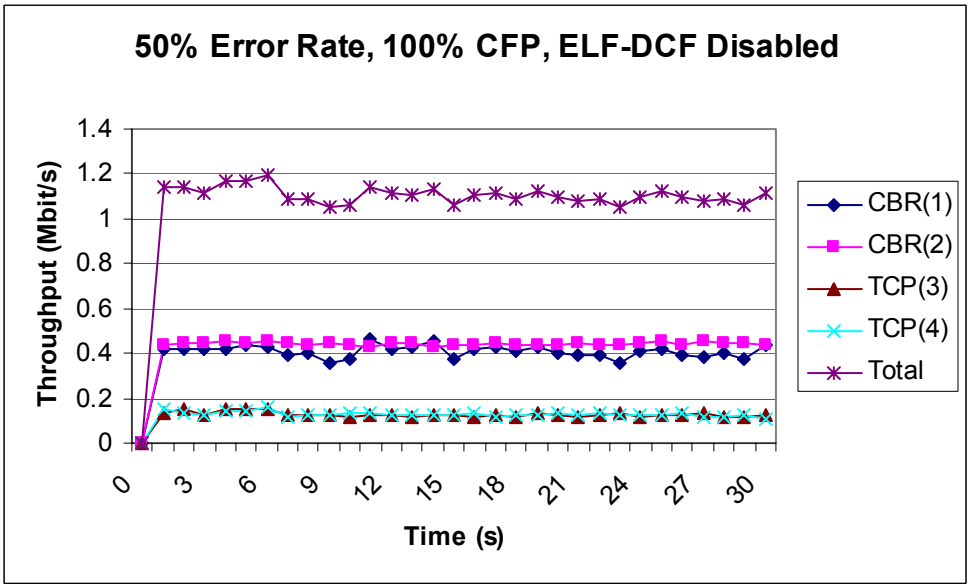

Figure 6: 50% Error rate, 100% CFP, ELF-DCF Disabled

Flow ID	ELF-DCF (on/off)	Percentage CFP (%)	Power factor (%)	Uniform Error Rate (%)	Average Throughput (Mbit/s)
CBR(1)	Off	0	200	50	.113
CBR(2)	Off	0	100	0	.477
TCP(1)	Off	0	100	0	.284
TCP(2)	Off	0	100	0	.285
CBR(1)	Off	100	200	50	.408
CBR(2)	Off	100	100	0	.441
TCP(1)	Off	100	100	0	.127
TCP(2)	Off	100	100	0	.128
CBR(1)	Off	50	200	50	.275
CBR(2)	Off	50	100	0	.391
TCP(1)	Off	50	100	0	.208
TCP(2)	Off	50	100	0	.208
CBR(1)	On	50	200	50	.355
CBR(2)	On	50	100	0	.388
TCP(1)	On	50	100	0	.143
TCP(2)	On	50	100	0	.143

Table 4: Flow statistics for 50% uniform error rate

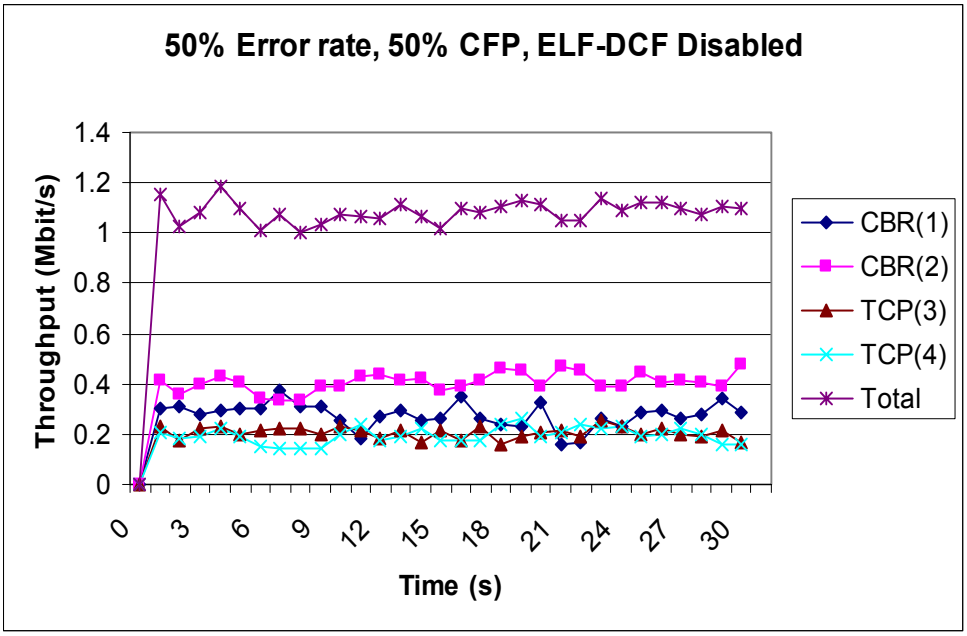

Figure 7: 50% Error rate, 50% CFP, ELF-DCF Disabled

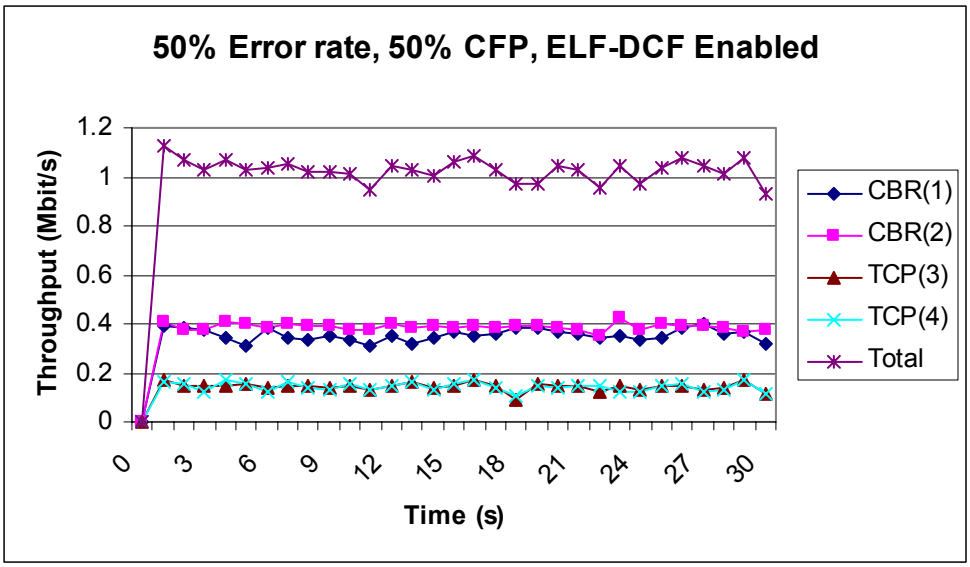

Figure 8: 50% Error rate, 50% CFP, ELF-DCF Enabled

When we increase the error rate to 100%, CBR(1) will obviously not get any throughput. If we run all four flows completely with DCF (i.e., 0% CFP), and no ELF-DCF, total throughput is about 1.2 Mbit/s. Running 50% CFP *with* ELF-DCF gives a total throughput of about 700 Kbit/s, which is a marked decrease from 1.2 Mbit/s (see Figure 9 below). However, this is consistent with ELF. Normally the error-prone flow would take 1/4 the total air time, whereas now it has taken roughly 40% the total air time (since we have a power factor of 200). However, we have succeeded in avoiding link starvation in the worst-case scenario. Thus, we show that ELF performs acceptably even in a pathological case that would be problematic for a priority scheduler.

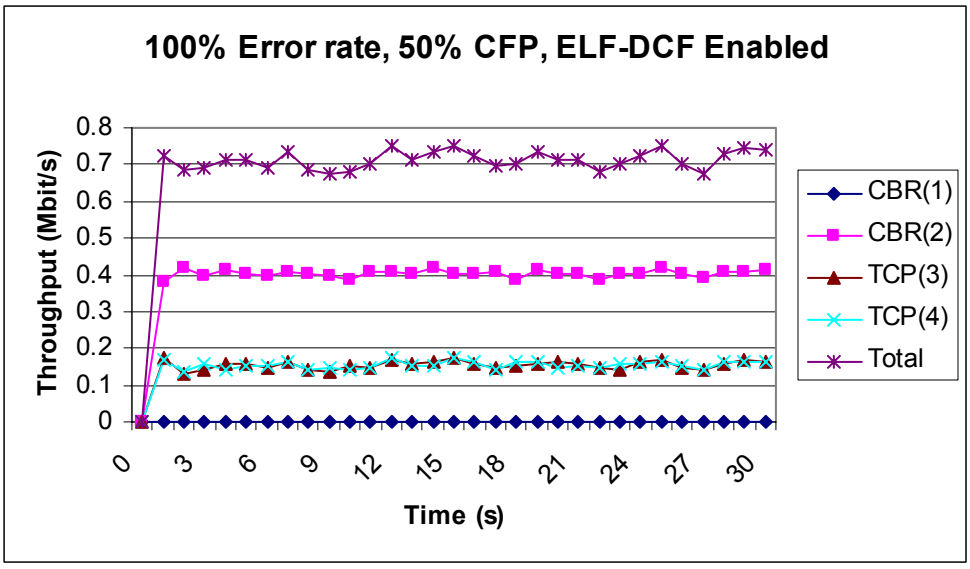

Figure 9: 100% Error rate, 50% CFP, ELF-DCF Enabled

8.3 Three-state Markov model

The Markov error model allows us to test how ELF-PCF/DCF performs under fluctuating error conditions. We set up three error states (0%, 50%, 100%), with a 50% probability of transitioning to any other state (see Figure 10). The error rates in each state are uniform in distribution. Table 5 below shows the time duration the error prone flow remains in a particular error state.

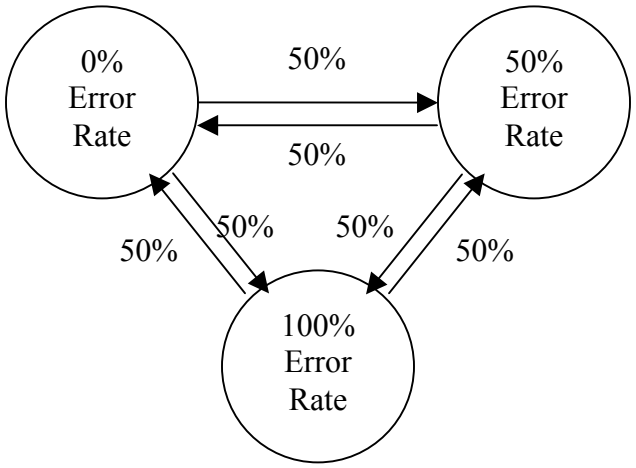

Figure 10: State machine for Markov error model

We observe that ELF-PCF/DCF performs as expected, though it leads to questions about how much of a gain in throughput (and for what time period)

should result after a time of substantial errors. Specifically, as we increase the percentage CFP, the average throughput for the error prone flow gradually increases, which is consistent with ELF's design. However, the throughput jitter also increases. The reason for this increase in jitter is because of a substantial amount of "deserve" accumulation. During periods where the error rate is 100%, the flow can accumulate a large amount of deserve, which causes throughput to be bursty during periods where the error rate is 0%. If the error durations are short, then the throughput jitter tends to be smaller because the amount of deserve accumulation is smaller (see Figure 11 below). However, if the error period is lengthy, it can cause the deserve accumulation to build up to large levels. Thus, after the error duration has passed, the flow's throughput will make a *huge* jump (whereas, with smaller deserve levels, the throughput would return to slightly above its normal level). For example, when the Markov state duration is increased to two seconds (last row of Table 5), the throughput jitter goes up to about 245 Kbit/s because of both the throughput outage and the ensuing throughput spike (see Figure 12 below). It might be worthwhile to find a plausible heuristic to cap or retire deserve (especially deserve unused coming out of a CP, or after a period of sustained errors). This could allow ELF to remain faithful in providing an average throughput to a flow, while not excessively penalizing other flows immediately following a time of high error rate. (We chose to use the .25s time duration because it allows for reasonably small error bursts. The 2s time duration allows us to demonstrate a relatively lengthy error burst that illustrates the need for a capping heuristic.)

State duration	Percentage CFP (%)	Power factor (%)	Average Throughput (Mbit/s)	Throughput jitter (Mbit/s)
.25s	0	200	.223	.062
.25s	50	200	.336	.081
.25s	100	200	.394	.085
2s	50	200	.335	.245

Table 5: Flow statistics for CBR(1) under Markov Model (ELF-DCF/PCF Enabled)

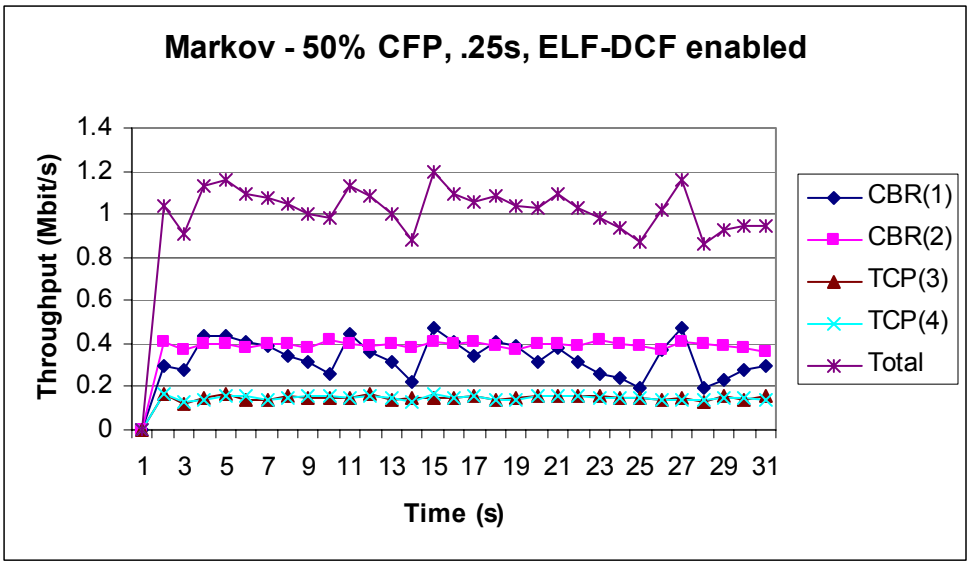

Figure 11: Markov error model, 50% CFP, .25s, ELF-DCF enabled

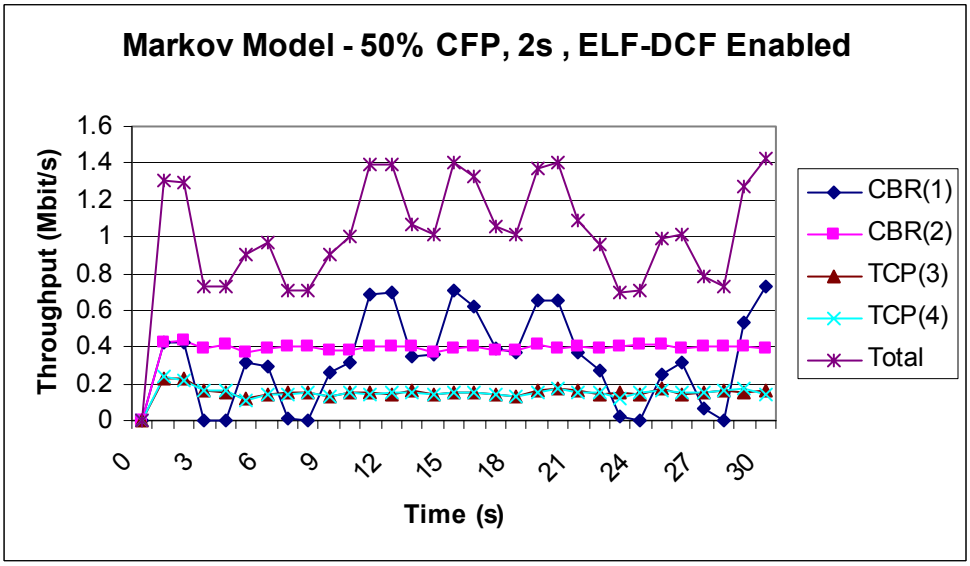

Figure 12: Markov error model, 50% CFP, 2s, ELF-DCF Enabled

8.4 Trace errors: Walls

We will now use the Walls trace from [3] to investigate whether ELF works well in a real-world setting with moderate error rates.

"Two units were separated by placing them in two rooms across a hallway. The direct path was approximately 17 feet long and passed through two

thick concrete block walls, a metal display case, and some classroom furniture. As compared to the interference environments, attenuation has an almost insignificant truncation rate, but has significant packet corruption."[2]

We will examine this by presenting two test cases. In the first, we will disable both ELF-PCF & ELF-DCF, and show to what degree CBR(1) performs poorly. Then we will test at 50% CFP with ELF-DCF enabled, and show that ELF can work remarkably well under these kinds of error conditions.

In Figure 13 below, CBR(1) is seen to experience a substantial increase in errors, and then that tapers off. The error rate increases substantially again towards the end of the trace.

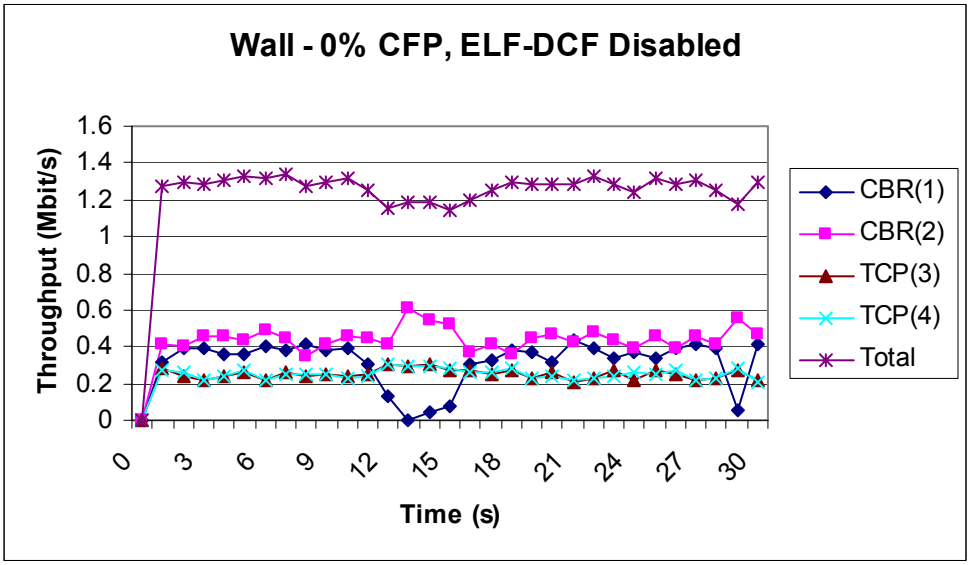

Figure 13: Walls trace, 0% CFP, ELF-DCF Disabled

[2] This error trace description and the rest of them are excerpted from Eckhardt, et al. [7]. Eckhardt's goal was to "characterize error environments by monitoring data transfers between two identical DECpc 425SL laptops (25 MHz 80486) running NetBSD 1.1. For different tests, we placed the PCs in different environments or added competing radiation sources. Our laptops used PCMCIA WaveLAN interfaces operating in the 902-928 MHz frequency band….To explore the range of error severity due to interference, we investigated 30-second traces in four different situations." For our experiments, we always discard packets with truncations and bit corruptions for simplicity purposes.

When running at 0% CFP (ELF-DCF disabled), although CBR(1) suffers a huge drop in throughput (to about 0 Kbit/s), the overall throughput barely falls (by about 100 Kbit/s). The other three flows increase in throughput while CBR(1) suffers. (Note that CBR(2) and the TCP flows gain an equally proportional amount throughput as a consequence of the fact that the link is slightly overcommitted even with each flow having a 0% error rate. The gain in TCP throughput is not as evident because we bill a station for both TCP DATA and ACK packets equally.) In contrast, when ELF-DCF is enabled at 50% PCF, we note three things (see Figure 14 below). First, the amount of throughput lost by CBR(1) is lessened during the error period. Second, because CBR(1) has a lot of unused "deserve," it tends to have a higher throughput than the other flows for a short duration after its last throughput loss (this further highlights the need for a capping mechanism). Third, with ELF-DCF disabled, the total throughput suffers only a small amount. With ELF-DCF enabled, we are now allocating more effort to CBR(1) (and much of that effort is wasted during error periods). Thus, the total throughput loss is substantial during an error period. Overall, this demonstrates that ELF-DCF can work successfully in a real-world error environment.

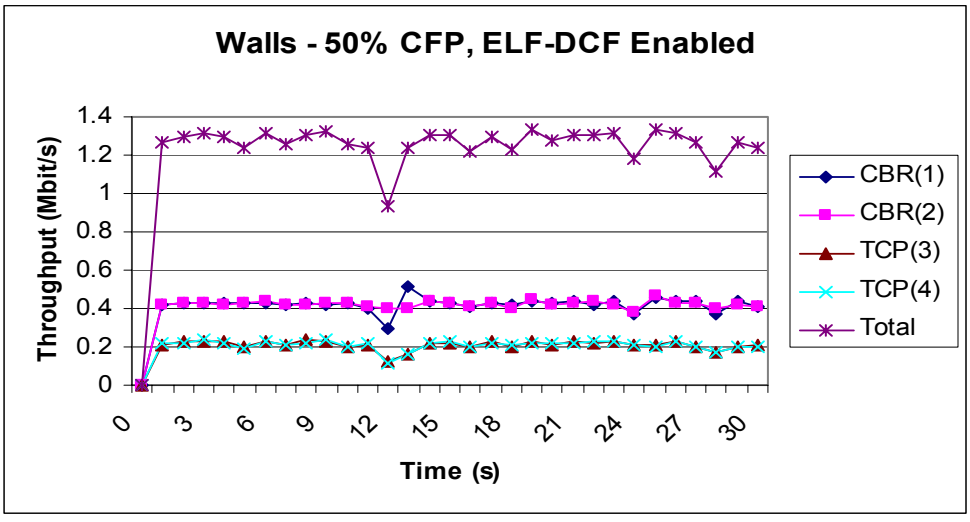

Figure 14: Walls trace, 50% CFP, ELF-DCF Enabled

Flow ID	ELF-DCF (on/off)	Percentage CFP (%)	Power factor (%)	Average Throughput (Mbit/s)
CBR(1)	Off	0	200	.319
CBR(2)	Off	0	100	.446
TCP(3)	Off	0	100	.250
TCP(4)	Off	0	100	.251
CBR(1)	On	50	200	.421
CBR(2)	On	50	100	.421
TCP(3)	On	50	100	.209
TCP(4)	On	50	100	.209

Table 6: Flow statistics for Walls trace

You can see clearly in Table 6 above that ELF does a remarkable job in assisting CBR(1) to meet its desired throughput. The best-effort flows suffer a 40 Kbit/s loss in throughput, while CBR(1) gains about 100 Kbit/s of throughput. In fact, CBR(1) and CBR(2) are almost equal in throughput. Also, the jitter in CBR(1) is substantially reduced now, as well as all the other flows. Thus, we show that ELF-DCF can do a good job of meeting ELF's original design principles in the real world.

8.5 Trace errors: Adjacent

We will now use the Adjacent trace from [3] to investigate whether ELF works well in a real-world setting with high error rates.

> "In the adjacent scenario, the communicating machines were separated by roughly 3 feet. A cordless telephone base station was adjacent to the receiver's modem unit, and the telephone handset moved repeatedly at walking speed back and forth from roughly a foot away from the base station to a point approximately 30 feet away (where the handset complained about signal loss)."

First, we show that the error-prone flow performs much worse than in the Walls trace without any intervention from ELF. Then we show how a larger power factor is needed if we want to give the error-prone flow outcome fairness over a wider range of error rates. Finally, we show that ELF-DCF can still be effective in boosting throughput for the error-prone flow, while decreasing the percentage CFP.

The packet loss for CBR(1) in this trace is much higher (see Figure 15 below), and thus the throughput for the other flows substantially increases (e.g., CBR(2) average throughput is 560 Kbit/s, best-effort flows average 300 Kbit/s).

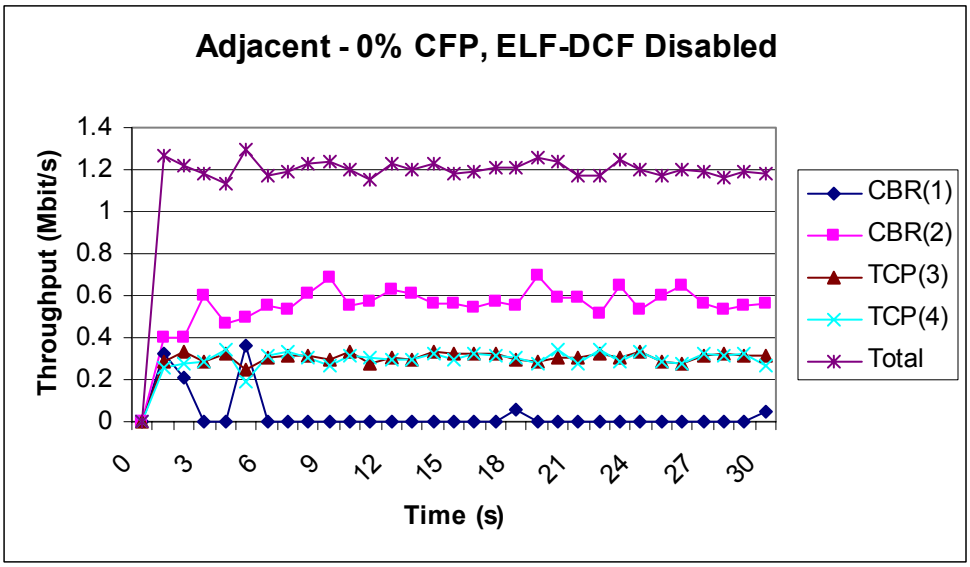

Figure 15: Adjacent trace, 0% CFP, ELF-DCF Disabled

Because of the high packet loss rate for CBR(1), we found that a high power factor was needed in order to boost CBR(1)'s throughput (see Figures 16, 17, & 18). With power factors of 100 or 200, CBR(1) gained more throughput. However, only at 300 was it able to meet its throughput expectations. At 100% CFP we note that the best-effort flows suffer the brunt of the throughput loss, as expected. Also, note that with increasing power factors assigned to CBR(1), the best-effort flows suffer more throughput loss. Moreover, ELF-DCF smoothes out the jitter in throughput, except during and following a period of high packet loss. In that case, CBR(1) is administratively bounded from receiving any additional

effort. However, it continues to accumulate deserve. Thus, when the loss rate decreases, CBR(1) gets more throughput because of the unused deserve (see Figure 18 below).

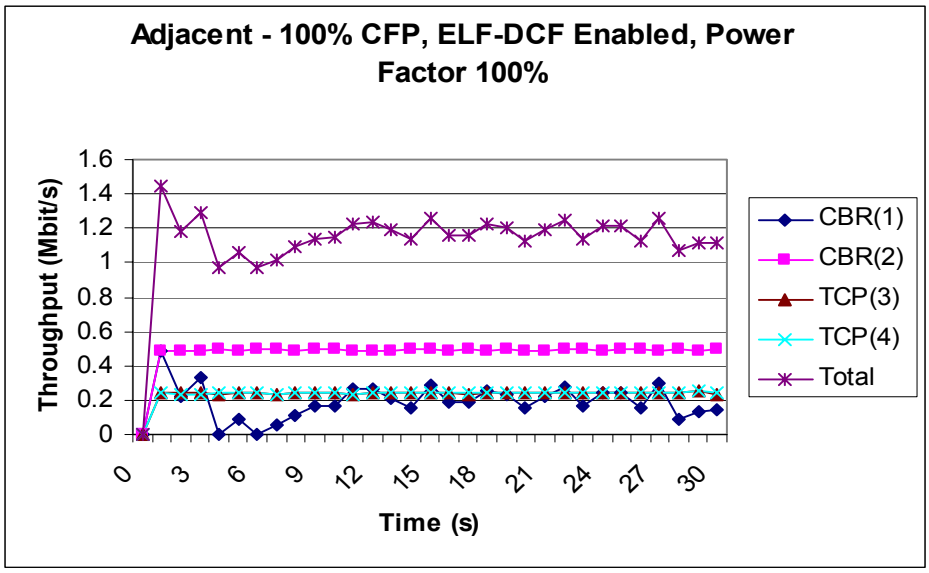

Figure 16: Adjacent trace, 100% CFP, ELF-DCF Enabled, Power factor 100%

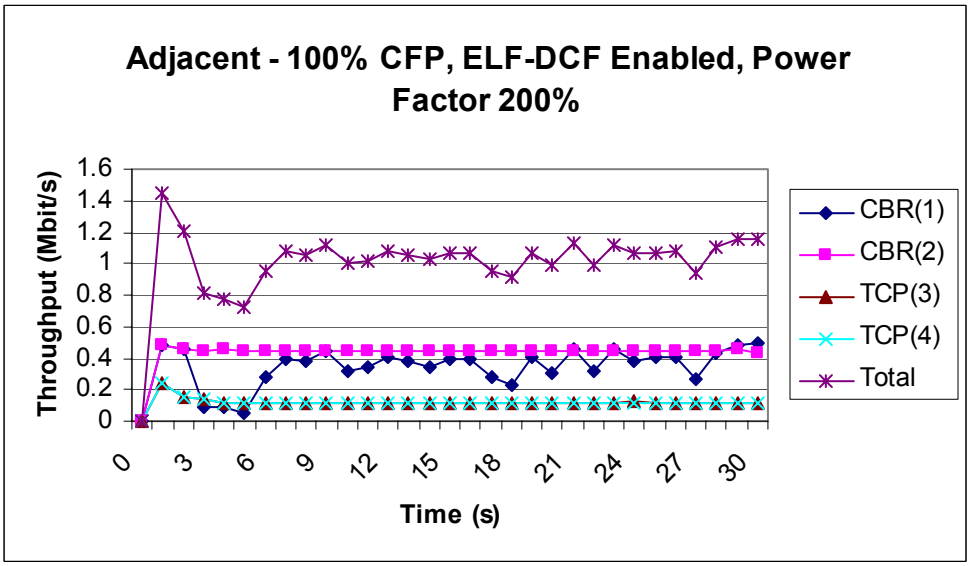

Figure 17: Adjacent trace, 100% CFP, ELF-DCF Enabled, Power factor 200%

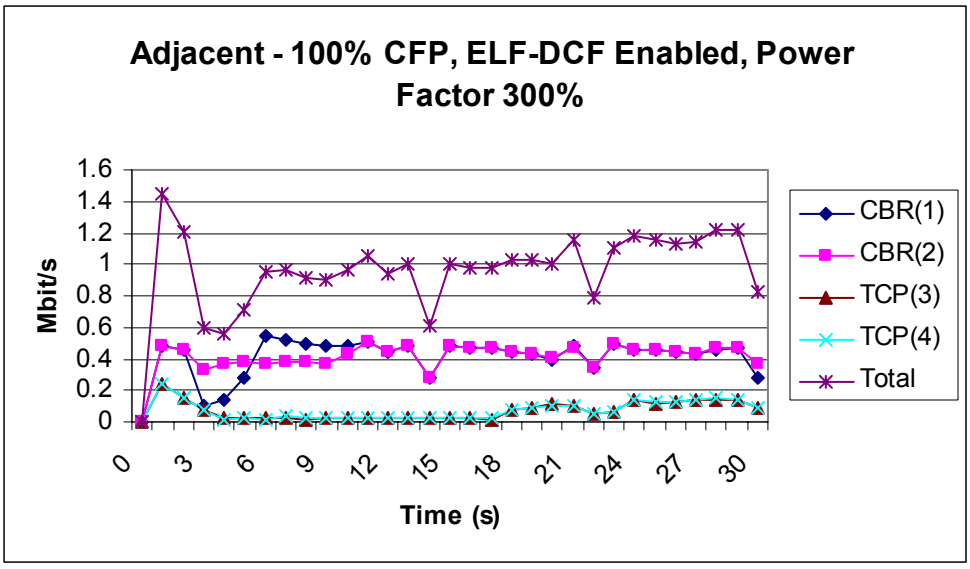

Figure 18: Adjacent trace, 100% CFP, ELF-DCF Enabled, Power factor 300%

We clearly saw in each simulation that higher percentage CFP combined with ELF-DCF, *does* improve CBR(1) throughput performance, but at an expected loss of total channel throughput. At 50% CFP with ELF-DCF enabled, CBR(1) throughput rises to about 300 Kbit/s (see Figure 19 below). However, CBR(2) throughput falls to 340 Kbit/s, and both best effort flows average about 110 Kbit/s. Thus, total channel throughput has fallen from about 1.2 Kbit/s to about 860 Kbit/s. However, this is in line with ELF's expectations.

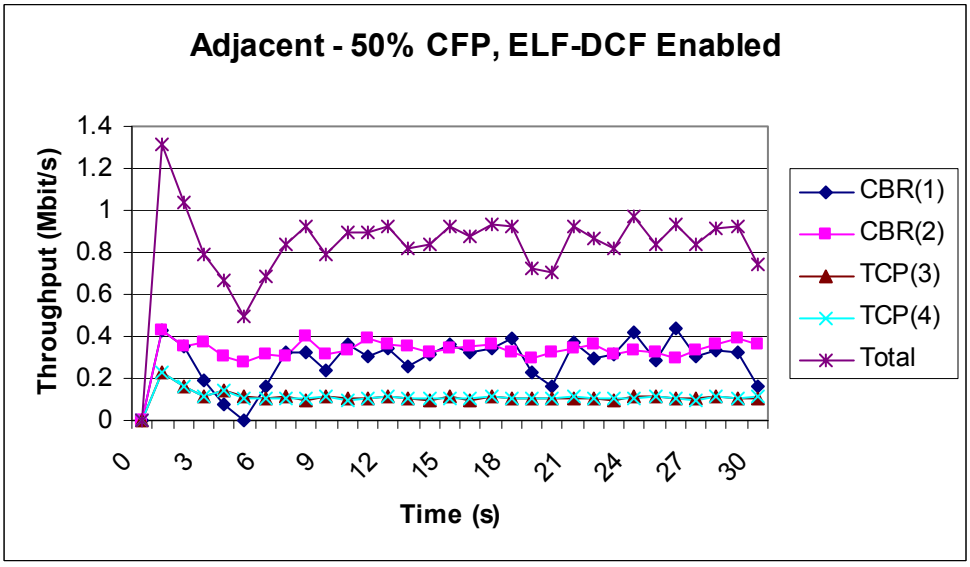

Figure 19: Adjacent trace, 50% CFP, ELF-DCF Enabled

8.6 Multiple flows with errors

We have done several simulations on both guaranteed and best-effort flows, where more than one flow experiences a significant packet loss rate. There are two main factors unique to ELF that administratively govern how two error-prone flows will perform relative to each other. First, flows with a higher power factor will be given more air-time than those with a lower power factor during times of high packet loss. Second, a guaranteed flow is serviced before a best-effort flow. Thus, even if a best effort flow is experiencing a high error rate and has a large power factor, it will be preempted by the guaranteed flow until there are no eligible guaranteed flows for servicing. However, relative to other best-effort flows, the best-effort flow with the higher power factor can receive more air-time to achieve is desired throughput outcome relative to the air-time of the other best-effort flows. Our simulations have verified that these principles hold true for ELF-PCF. The slight variation with ELF-DCF lies in the fact that the PC makes no distinction between a guaranteed or best-effort flow when sending out a beacon. Thus, a mobile station with a guaranteed flow is just as likely to send a frame as is a mobile station with a best-effort flow during a CP. Thus, in general, best-effort flows will receive higher throughput during a DCF (with ELF-DCF enabled) than during a PCF. As we stated in section 7, we desired to keep the overhead in the beacon minimal – hence the lack of distinction.

Figures 20 & 21 are used to show that ELF does work even when you have multiple flows with errors. In Figure 20, we have both ELF-PCF & DCF disabled. CBR(1) & CBR(2) both have error rates of 50%, and it is clear that they get very little throughput. In Figure 21, we assign both error-prone flows a power factor of 200, and run ELF-PCF continuously. Obviously, they are both able to regain much of their expected throughput at the expense of CBR(3) & CBR(4). This makes sense because by assigning both flows a power factor of 200 (each has a weight of 1/4), we allow them to consume virtually the whole link with an

error rate of 50% per flow. Note that although this is in conformance with ELF's design principles, it is probably undesirable in practice to starve the best-effort flows as shown below. We would probably want to assign a lower power factor to the guaranteed flows, so that starvation is avoided even in the worst case scenario.

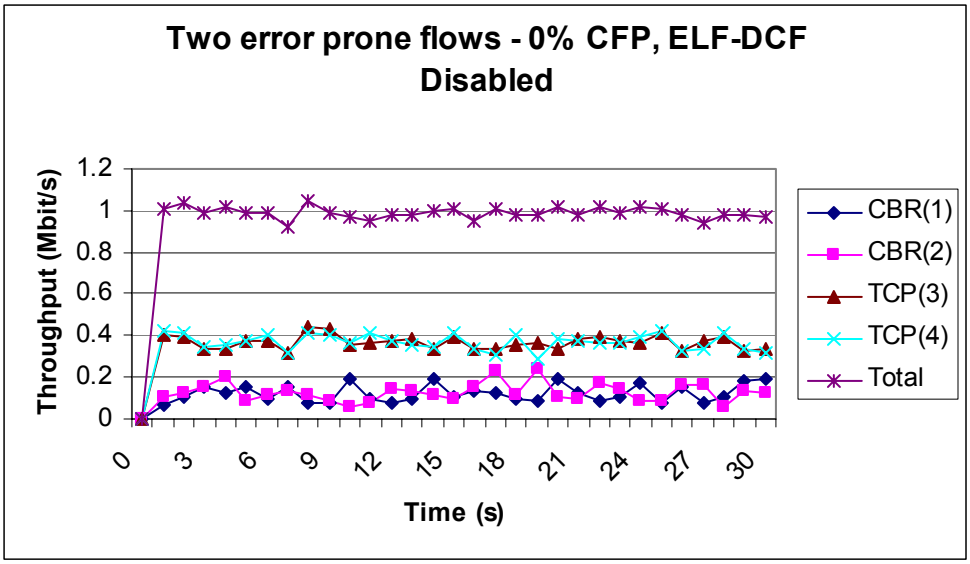

Figure 20: Two error prone flows, 0% CFP, ELF-DCF Disabled

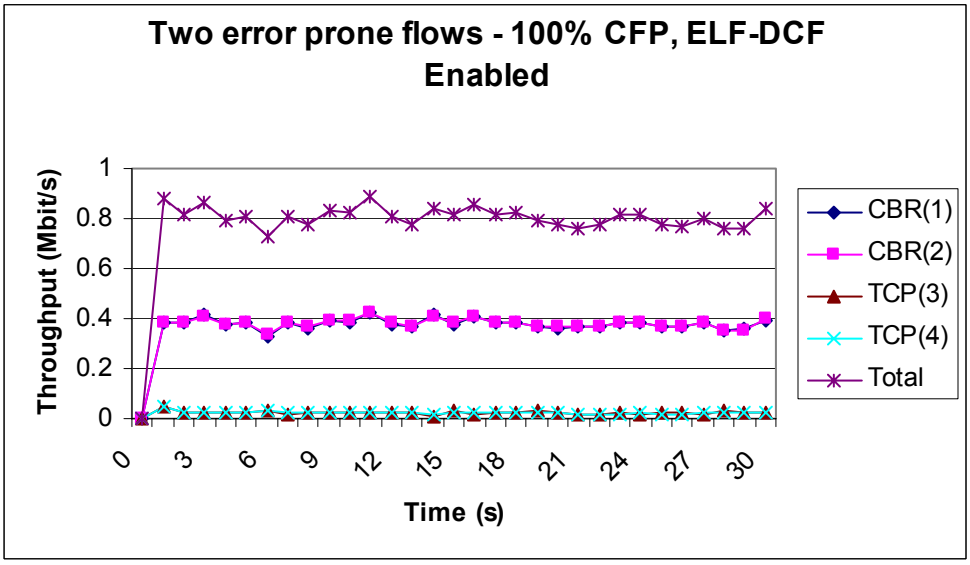

Figure 21: Two error prone flows, 100% CFP, ELF-DCF Enabled

8.7 Best-effort traffic

The most common flows on the network are best-effort flows. Thus, it is useful to show that ELF still works well even if there are no guaranteed flows on the network. In Figures 22 & 23 below, we show four best-effort FTP/TCP flows (equally weighted). Both TCP(1) & TCP(2) suffer with a uniform error rate of 30% , and we can see a slight loss in throughput in Figure 22. In Figure 23, we give all the flows a power factor of 150, while only running a 50% ELF-PCF and enabling ELF-DCF. Our main observation is that TCP(1) & TCP(2) both regain their lost throughput relative to the other flows, and that throughput jitter substantially decreases for each flow. ELF is seen to be able to succeed even among best-effort flows with smaller loss rates, and with a nominal PCF intervention.

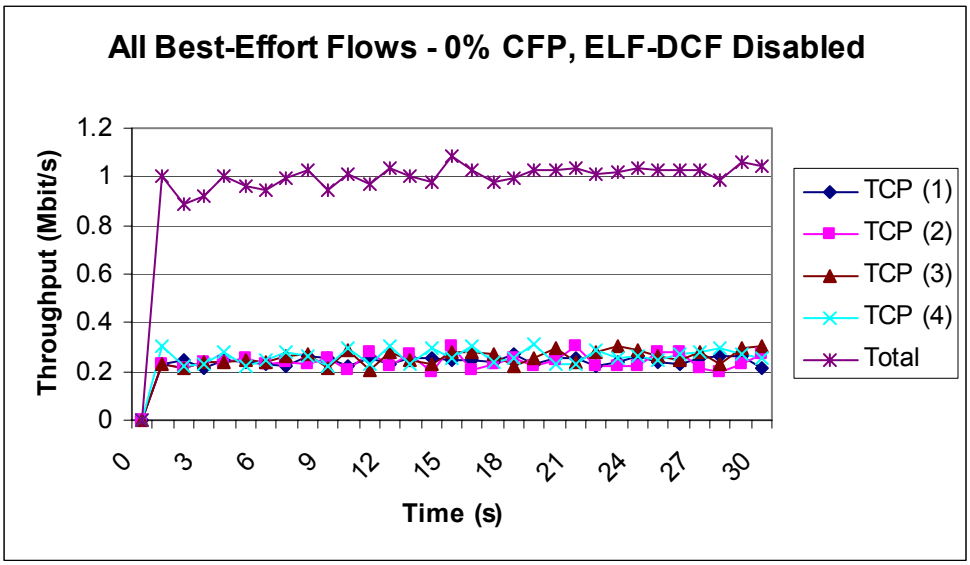

Figure 22: Best-effort flows, 0% CFP, ELF-DCF Disabled

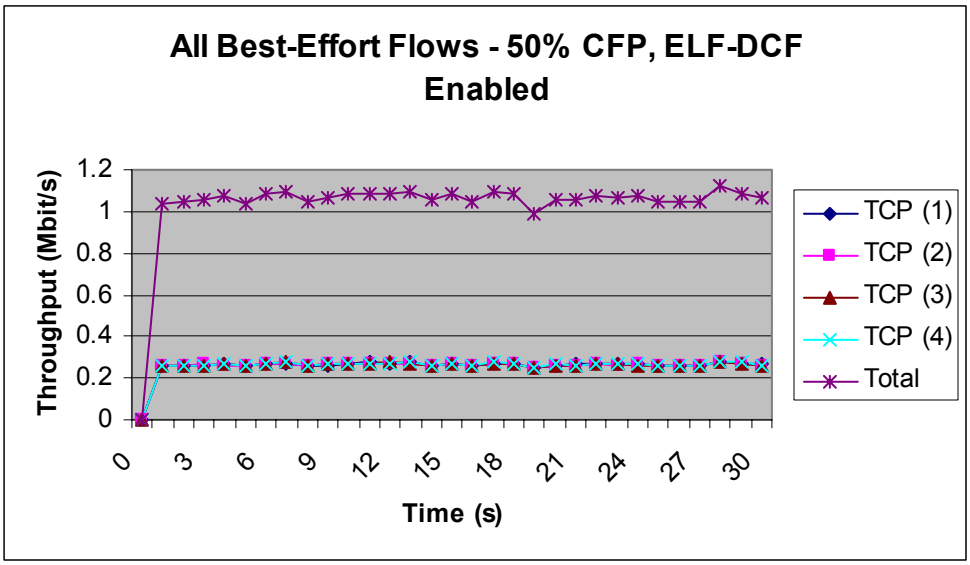

Figure 23: Best-effort flows, 50% CFP, ELF-DCF Enabled

8.8 Varying superframe duration

All of the preceding simulations were done with superframes of about 100 milliseconds in length. However, it certainly is possible that one might want to use smaller superframe duration so that they can alternate between CP & CFP at a quicker pace. This would depend on the size of the packets being sent, the desired round-trip times (e.g., audio applications), how much jitter is tolerable, and other factors. The shorter the superframe duration, the less efficient the system will be overall (because more time will be spent in overhead alternating between CP & CFP).

In Figures 24 & 25 below, we show that ELF is still successful even with shorter superframe duration (30 ms). Again we have two guaranteed flows and two best-effort flows, and CBR(1) has an error rate of 50%. We can see that in Figure 24, CBR(1) suffers a significant throughput loss, while the other flows operate under a normal DCF and receive similar throughput. However, in Figure 25 we see that if we give CBR(1) a power factor of 200 (at 60% PCF with ELF-DCF enabled), CBR(1) is able to recover much of its throughput. Also,

throughput jitter for each flow has been significantly reduced, while total throughput falls by a minimal amount. Note that with smaller superframe duration, the overhead of alternating between CP & CFP becomes increasingly large as a percentage of the superframe (i.e., around 5 ms). Thus, it becomes impossible to truly have a 100% CFP. However, we can see that even with a 30 ms superframe length, ELF still performs well.

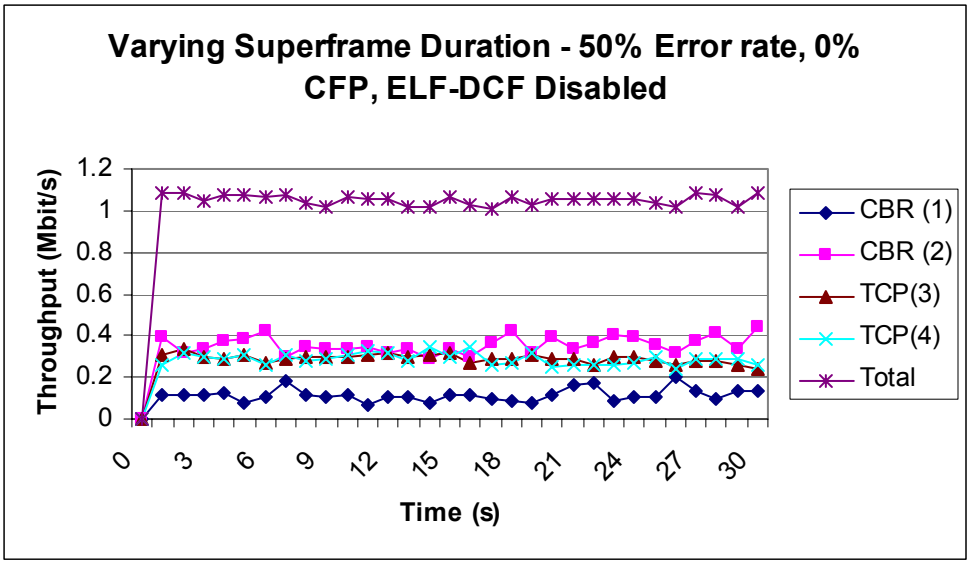

Figure 24: Varying Superframe Duration, 50% Error rate, 0% CFP

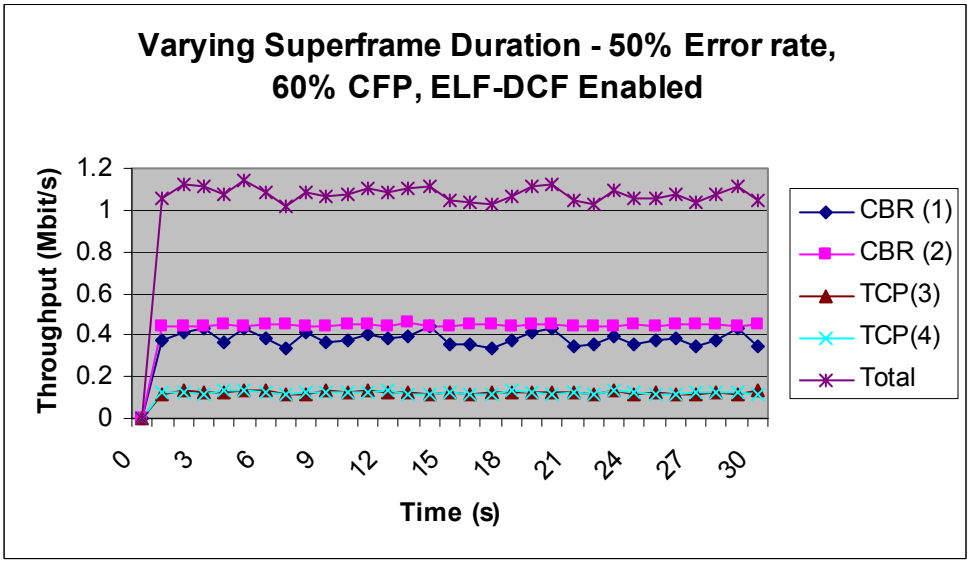

Figure 25: Varying superframe duration, 50% Error rate, 60% CFP

9 Future Work

We have pointed out several areas of possible further exploration in this paper. We stated at the beginning that we treated flows as being aggregates, so that one station's flows can be considered to be one flow. This was partially for simplicity, and also because the current protocol specifications call for polling on a station-by-station basis (instead of individual flows). It may be worthwhile to alter the protocol and allow for individual flows to be polled and beaconed. This might provide for better QoS delivered to individual flows, since the PC can monitor each flow at a better granularity. Of course, admission control would still be critical, so as to not over commit the link, nor increase the complexity on the server beyond a reasonable threshold.

Another question (which was also posed by Eckhardt) which becomes more important in light of ELF-DCF, is whether a flow that experiences errors for a period and loses throughput as a result, is entitled to regain that lost throughput later. Currently, we have no cap on our "deserve" value, and so we have shown (i.e., multiple state Markov model simulation) that ELF will allow this flow to reclaim the lost throughput at the expense of other flows. Since unused deserve and effort from ELF-DCF is currently plugged right back into the CFP, one can see how this problem can be exacerbated (especially when a station has low effort utilization and/or high error rate). Another area of concern is with respect to "deserve" accumulation during an idle time period. In our simulations, the flows constantly had data to send. However, it is entirely possible for flows to experience significant idle time. This raises the question of how to bill flows when they return a NULL frame in response to a poll, because they are idle. Currently, our implementation charges a flow for effort and deserve even if the station returns a NULL frame because the flow is idle. However, it may be fairer to not charge a flow if it is idle. In that case, once again the question is raised about how to deal with flows that are idle for extended periods that accumulate large

"deserve" values. One possibility is that deserve might cease to increase, so that they would accumulate up to some threshold. Another possibility is to age these values.

Although, the ELF-DCF extension does assist error-prone flows significantly, one might question if we can achieve the same goal, but not suffer as great a loss in overall throughput. Is it possible for stations to be "encouraged" to spend effort during a DCF, and yet done so in a way that accurately predicts the nature of a flow on an individual station? Currently, the PC only serves to *limit* effort and deserve during a CP. But if a station could be allowed to spend *more* effort when it needs to, that might result in being more faithful to ELF's design. One possibility of getting stations to spend more effort might be to allow them to acquire control of the medium in a similar way that the PC does. We could then specify how long they are allowed to hold on to the medium (i.e., up to some administrative bound), before giving it back to the rest of the stations under a CP. One might look into the possibility of having each individual mobile node run its own ELF scheduler, and link to a global ELF scheduler every beacon period. This could reduce processing on the PC, while giving each station more flexibility in how to handle individual flows on the machine.

Another interesting area of exploration involves the variable length PCF & DCF periods. For example, we note that under PCF, the ELF scheduler can guarantee that flows receive their desired outcome up to some administrative bound. Under DCF, flows will receive a "boost" in achieving their desired outcome. However, in times of severe errors, it would be more useful to allow the PC to take control of the medium and guarantee that the error-prone station is allowed to spend the appropriate amount of effort. If we could vary the percentage of time that is devoted to PCF versus DCF per superframe, we could devote more time to PCF during times where flows are experiencing high errors. When flows are doing relatively well, and do not need as much direct intervention from the ELF scheduler, we could devote most of the superframe to running

DCF. Currently, the low-level MAC protocol design technically allows the PC to seize the medium at any time, even though the specifications specify that it should be done only on a “per superframe” basis.

Finally, most of our research was done to investigate how the ELF algorithm and our modifications affect the overall throughput of the channel, along with individual flows’ throughput. However, we did not look in depth at how ELF scheduling within 802.11 affects TCP congestion control. One could imagine asking if ELF should interact explicitly with TCP, so as to increase or decrease an individual flow’s throughput and reduce jitter.

10 ns-2 Extensions

We have posted our source code and documentation of the changes we have made on the following web site: http://www.cs.cmu.edu/~davidm. Please feel free to modify or extend our contribution to the *ns-2* code base.

11 Recommendations for IEEE Committee

There is already being work done to enhance 802.11 to provide QoS in wireless networks (e.g., 802.11e [17]). However, to the best of our knowledge, there has been no work within 802.11 that addresses the issues that ELF focuses on. Also, while there has been some work on “hybrid” networks, this has been more related to enabling an ad-hoc network to interoperate with an infrastructured network (or an 802.11 WLAN to interoperate with a high-speed error-free backbone [18]). We are not aware of any work to date that hybridizes DCF & PCF in a way that is consistent with ELF’s design principles. Therefore, we recommend that the IEEE Committee consider making a few modifications to the 802.11 specifications.

One modification would be to allow the PC to incorporate some sort of feedback mechanism into beacon frames (similar in spirit to what we have shown in ELF-DCF). This way, individual hosts can have some way of "policing" themselves during DCF periods. This would make it easier to implement the principles upon which ELF was designed (i.e., administrating the bounds on outcome and effort that each flow is allowed to achieve, thus making it easier to provide quality of service to flows experience high error rates).

Another modification would be to change the specifications of how the PCF should be implemented. Specifically, instead of just choosing stations to poll based on their station number, we should be able to insert any algorithm we choose (e.g., an ELF scheduling algorithm).

12 Conclusion

We have sought to build upon previous work done, both by David Eckhardt and by the IEEE's 802.11. Eckhardt's work provided us with a plausible way to address severe wireless errors, by proposing an "effort –limited fair" (ELF) scheduling approach. This approach extends a normal WFQ scheduler with an explicit mechanism for controlling behavior in the presence of capacity loss (which may be location dependent).

Our contribution shows that the ELF scheduler can be integrated into 802.11's PCF, while remaining as faithful as possible to both the goals of ELF, and the efficiency benchmarks for 802.11. We also implemented an ELF WRR scheduler (similar to Eckhardt's), showing that ELF does work in 802.11 as the policy for a PCF. We contribute further by extending ELF's influence to a DCF's operation, effectively regulating flows during a CFP, but still keeping the function distributed. Finally, we used simulations to illustrate the operation of ELF-PCF/ELF-DCF under various environments. These experiments were conducted using the *ns-2* simulator, after both extending the 802.11 implementation and

integrating ELF into that code base. We then presented experimental results that show ELF-DCF does achieve its goals of assisting ELF-PCF and providing quality of service to flows with high error rates, up to an administrative bound.

13 Acknowledgements

Much thanks goes to Peter Steenkiste and David Eckhardt for guidance and patience in completion of this thesis. Their input and assistance from design, to implementation, to analysis and feedback, has been invaluable.

14 Bibliography

[1] David A. Eckhardt and Peter Steenkiste, “Effort-limited Fair (ELF) scheduling for Wireless Networks” in *IEEE INFOCOM 2000*

[2] Songwu Lu, Vaduvur Bharghavan, and Rayadurgam Srikant, “Fair scheduling in wireless packet networks,” in *Proceedings of ACM SIGCOMM ’97.* September 1997, IEEE Computer Society.

[3] David Eckhardt and Peter Steenkiste, “A Trace-based Evaluation of Adpative Error Correction for a Wireless Local Area Network,” *Mobile Networks and Applications (MONET),* 1999, Special Issue on Adaptive Mobile Networking and Computing.

[4] The Institute of Electrical and Electronics Engineers, Inc., *IEEE Std 802.11 – Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications,* 1999 edition.

[5] Kevin Fall and Kanna Varadhan, “The *ns* Manual”, April 2002

[6] M.A. Visser and M. El Zarki, “Voice and data transmission over an 802.11 wireless network,” in *Proceedings of PIMRC’95*, *Toronto, Canada,* September 1995, pp. 648-652.

[7] N. H. Vaidya, P. Bahl, and S. Gupta, “Distributed fair scheduling in a wireless LAN,” in *Sixth Annual International Conference on Mobile Computing and Networking,* Boston, August 2000.

[8] J. L. Sobrinho and A.S. Krishnakumar, “Real-time traffic over the IEEE 802.11 medium access control layer,” *Bell Labs Technical Journal*, pp. 172-187, Autumn 1996.

[9] D-J. Deng and R-S Chang, “A priority scheme for IEEE 802.11 DCF access method,” *IEICE Transactions on Communications,* vol. E82-B, no. 1, January 1999.

[10] Lindgren, Almquist, and Schelen, “Quality of Service Schemes for IEEE 802.11 – A simulation study”

[11] T.S. Eugene Ng, Ion Stoica, and Hui Zhang, “Packet fair queuing algorithms for wireless networks with location-dependent errors,” in *Proceedings of INFOCOMM ’98.* IEEE Communication Society.

[12] Parameswaran Ramanathan and Prathima Agrawal, “Adapting Packet Fair Queuing Algorithms to Wireless Networks,” in *Proceedings of the Fourth Annul ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM ’98),* Dallas, TX, October 1998, ACM SIBMOBILE.

[13] Guiseppe Bianchi, Andrew T. Campbell, and Raymond R. –F. Liao, “On Utility-Fair Adaptive Services in Wireless Networks,” in *Proceedings of the Sixth*

International Workshop on Quality of Services (IWQOS '98), Napa Valley, CA, May 1998, IEEE Communications Society.

[14] David Eckhardt, "An Internet-style Approach to Managing Wireless Link Errors," Ph.D Thesis, May 2002.

[15] David A. Eckhardt and Peter Steenkiste, "Improving Wireless LAN Performance via Adaptive Local Error Control," in *Sixth International Conference on Network Protocols,* Austin, TX, October 1998, IEEE Computer Society.

[16] Agilent Technologies, "Mixed Packet Size Throughput", White Paper, February 2001

[17] The Institute of Electrical and Electronics Engineers, Inc., *IEEE Std 802.11e* –Task group established for enhancing QoS in 802.11 (ongoing)

[18] Qiang Ni, Thierry Turletti and Wei Fu, "Simulation-based Analysis of TCP Behavior over Hybrid Wireless & Wired Networks." The 1st International Workshop on Wired/Wireless Internet Communications (WWIC 2002), Las Vegas, Nevada, U.S.A., June 24-27, 2002

[19] David B. Johnson, David A. Maltz, and Josh Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. in Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pp. 139-172, Addison-Wesley, 2001.