Chapter 13

QoS Support in Multimedia Wireless Environments

Klara Nahrstedt, Wanghong Yuan, Samarth Shah, Yuan Xue, Kai Chen

13.1 Introduction

Over the couple of years, there has been a strong proliferation in the use of multimedia wireless technology all over the world, creating new research and business opportunities for producers and consumers of these technologies. There are several reasons for this fast proliferation. First, wireless networking technologies such as cellular networks, wireless local area networking (WLAN) and Bluetooth are becoming an integral part of our communication environment. Second, new wireless devices such as cellular phones, PDAs, laptops are emerging to assist people in their lives. Third, multimedia applications became known, first in Internet environment, and now are embedded in wireless environments, due to (a) standardization of digital multimedia content such as MPEG-2, MPEG-4, H.263 and others, and (b) understanding of multimedia applications and users behaviors under different networking conditions. This allows service providers to build larger scale known (from analog world) multimedia services such as video conferencing, video-on-demand, and offer them to larger population. Fourth, new hardware opportunities are appearing such as multi-frequency energy-efficient processors allowing for more efficient use of mobile devices.

However, these new opportunities bring with them also various challenges. We will concentrate on addressing two major challenges. First, mobile devices running distributed multimedia applications and communicating over wireless networks must deal with scarce and variable resource allocations such as battery
power, processor speed, memory and wireless bandwidth, hence the resource management problem for support of Quality of Service (QoS) must be solved. Second, multimedia applications running over wireless networks must achieve some level of performance QoS guarantees, hence modeling of application QoS, QoS management, and its connectivity to underlying resource management must be addressed.

In this chapter, we will aim to answer these application QoS and resource management challenges, and describe some of the solutions that would contribute to solving these challenges. Since these two challenges are still very broad, we will narrow them down to address the following problem scope:

- The topic of multimedia applications and QoS is very broad and there is an extensive pool of solutions in the literature. We will concentrate on modeling of conversational applications with strict delay requirements such as Voice over IP and retrieval applications with sensitive throughput requirements such as multimedia on demand using mobile multimedia devices. We will consider three QoS metrics for multimedia distributed services and those are throughput guarantee, end-to-end delay guarantee, and application lifetime guarantee implying QoS-aware and energy-efficient solutions.

- The topic of resource management in wireless networks is also very broad and there are multiple techniques which optimize different resource usage. Furthermore, resource management is required for all types of wireless networks such as cellular networks, wireless LANs, mobile ad hoc networks, and sensor networks. In this chapter we will concentrate on resource management schemes meant only for networks based on, or compatible with the widely used network standard IEEE 802.11. Also, we will consider four major resources to deliver application QoS and those are wireless network bandwidth, CPU bandwidth, memory and energy, and provide algorithms, services and protocols at the operating system and middleware layer with cross-layered access to selected information in lower-level network solutions. We will consider resource management in single mobile devices, in mobile devices connected via single hop ad hoc networks, and in mobile devices connected via access-point-based networks.

To design solutions that address the end-to-end QoS issues and corresponding resource management in 802.11 wireless single hop environments, we take the top-down approach in this chapter. First, we decide on multimedia applications and their models that will run in these environments. Chapter 13.2 discusses the modeling of these applications and their QoS requirements, especially the application task, connection and QoS (quality) models, and the cross-layer application-OS-network models that drive correct resource allocations in mobile nodes. Second, once it is clear what applications are primarily running in the 802.11 wireless environments, resource management techniques need to be chosen that execute according to QoS requirements. These resource management techniques must span within individual mobile nodes via their
13.2 Application Modeling

Wireless multimedia applications on various mobile devices are becoming an integral part of our life. Examples are music on demand, using the Apple iPOD devices, short video clips on demand using cell-phones, DVD players on laptops, voice over IP using laptops and PDAs. We will first specify the common model of these applications so that we can then address them easier in the resource management and design solutions that will serve these applications. We consider computational and communication requirements of multimedia applications to have comprehensive and expressive model for OS and network resource management and their support for QoS guarantees.

During their lifetime, distributed multimedia applications use computational and communication resources on their mobile nodes. Hence when modeling multimedia applications, we need to consider requirements that these applications have on both resources, and include them into the overall application model. Furthermore, we need to consider the overall quality goal of the end-to-end application. Therefore, the application model will consist of two parts: (a) the operating systems, and across mobile nodes via the distributed network management. Therefore, the rest of the chapter addresses (a) operating-system-internal resource management techniques that help delivering application QoS requirements inside an individual mobile node, and (b) network-specific resource management techniques to deliver QoS in end-to-end fashion from the sender(s) to the receiver(s).

The operating-system-internal techniques can be found in Chapter 13.3 and the network-specific techniques will be discussed in Chapter 13.4 as follows: Chapter 13.3 concentrates on the energy-efficient operating system (EOS) at mobile end-points. The Linux-based EOS includes an integrated and cross-layer-optimized CPU/energy resource management to guarantee node delay and application lifetime QoS requirements. This end-point resource management must be addressed to achieve true end-to-end quality guarantees for any multimedia application [42]. Chapter 13.4 addresses the cross-layer-optimized network resource management to guarantee end-to-end delay and bandwidth QoS requirements in single hop wireless networks. The reason for concentrating on single hop wireless networks is that in commercial applications such as music on demand or phone conversations we believe there will be only few hops before the multimedia stream reaches the wired infrastructure through which the information will be transported. Hence, what we need to ensure in wireless networks for these types of applications is that the multimedia data be transmitted over the first/last wireless mile in a quality-aware manner.

We have built multiple cross-layer QoS-aware systems that utilize techniques in Chapters 13.3 and 13.4. The design principles and overall lessons learned from their design and development are summarized in Chapter 13.5. The chapter concludes with possible future directions with respect to wireless multimedia and the corresponding QoS support in Chapter 13.6.
CHAPTER 13. QOS SUPPORT IN WIRELESS ENVIRONMENTS

Application Task, Connection, and Quality Models: We consider multimedia distributed applications (either video, voice or music) as periodic tasks, running distributed application functions over single or multiple network connections between sender(s) and receiver(s). Each task consumes CPU time, energy, network bandwidth resources and provides an output quality. Multimedia applications are adaptive tasks, which means that from the computational point of view they are soft-real-time tasks that can operate at multiple application QoS (quality) levels. For example, a QoS level may correspond to a video frame rate in a video task. Each task $i$ supports a discrete set of QoS levels, $q_{i1}, ..., q_{im}$\[43, 7\]. Each task can provide different quality levels, trading off quality with resource consumption or trading off consumption between different resources \[37\]. We aggregate all best effort (non-multimedia) applications into one logical adaptive task. This logical task delivers either average (in lightly loaded environment) or no (in a heavily loaded environment) quality guarantees to individual best effort tasks.

Each connection connects multiple tasks to form a transmission medium between sender(s) and receiver(s) to exchange multimedia data and control information among mobile nodes. Also, each connection consumes through its distributed tasks CPU time, energy, and bandwidth resources, and based on the shared resource availability, especially the wireless channel, it provides an output quality. It is important to stress that from the 802.11 wireless networking point view, multimedia distributed applications must be adaptive. This means that the end-to-end connections can yield only soft end-to-end guarantees, and in many cases only statistical or best effort guarantees.

Cross-Layer Application-OS-Network Model: Each wireless multimedia application must have a strong relation to the underlying computing and communication layers that allocate resources to provide QoS guarantees $q$ and utility $u(q)$. We will consider two layers where multimedia applications will interface to: (1) process management (representing the operating system and its access to processor hardware) with its soft-real-time task scheduling and (2) middleware layer (representing entrance to the network protocol stack) with its connections/packets scheduling and bandwidth management.

Each application QoS level $q$ has a utility $u(q)$, which measures the perceptual quality at a QoS level from the user’s point of view, and consumes $C(q)$ cycles and $B(q)$ network bytes per period $P(q)$. Furthermore, we assume that for each QoS level, the task has probability distribution of its cycle demand; that is, $F'(x) = Pr(X \leq x)$ is the probability that the task demands no more than $x$ cycles for each job. This distribution can be obtained with our previously developed kernel-based profiler \[49, 50\]. Specifically, the operating system uses a profiling window to keep track of the number of CPU cycles each task has consumed for its recent jobs. The operating system then builds a histogram
13.3. QOS SUPPORT IN MOBILE OPERATING SYSTEMS

Based on the result in the profiling window. The histogram estimates the probability distribution of the cycle demand of the task for each job. With respect to network connections, we assume two network models, the integrated service ("IntServ") model and the differentiated service ("DiffServ") model that determine the network bandwidth allocation in 802.11 wireless networks and will be discussed in detail in Chapter 13.4.1.

In the hardware layer, a multimedia application uses two adaptive resources, CPU and wireless network interface card (WNIC). The CPU can operate at multiple speeds (frequencies/voltages), \( \{f_1, ..., f_{\text{max}}\} \), trading off performance for energy. The power consumption of the CPU is \( p(f) \) at speed \( f \). The lower the speed is, the lower the power is. We assume that the overhead for adapting CPU speed is negligible.

The WNIC supports three operation modes: active, idle, and sleep, where the sleep mode has much less power. The power consumption at the above states are \( p_{\text{act}} \), \( p_{\text{id}} \), and \( p_{\text{sl}} \), respectively. The overhead for switching the WNIC into sleep and from sleep is \( t_{\text{sl}} \), which is not negligible (e.g., around 40ms for Lucent WaveLan card).

13.3 QoS Support in Mobile Operating Systems

In mobile wireless environments, QoS-aware operating system support for battery-powered mobile nodes is crucial in order to run multimedia applications. Such multimedia-enabled mobile systems need to save energy while supporting multimedia QoS requirements. There is a conflict in the design goals for QoS provisioning and energy saving. For QoS provisioning, system resources often need to provide high performance, typically resulting in high energy consumption. For energy saving, system resources should consume low energy. As a result, the operating system of mobile devices needs to manage resources in QoS- and energy-aware manner and provides the flexibility to trade off QoS and energy based on the user’s preferences.

Recently, a number of soft real-time operating systems has been proposed to support QoS for multimedia applications. These operating systems typically integrate predictable CPU allocation (such as proportional sharing [9, 31] and reservation [18, 36]) and real-time scheduling algorithms, such as earliest deadline first (EDF) and rate monotonic [21]. Energy management is also an important part of the operating system. For example, ECOSys tem [52] and Nemesis [29] manage energy as a first-class OS resource. Vertigo [16] saves energy by monitoring application CPU usage and adapting the CPU speed correspondingly. More recently, there is some work on QoS and energy aware cross-layer adaptation [27, 33, 34, 51, 50]. Pereira et al. [33] proposed a power-aware application programming interface that exchanges the information on energy and performance among the hardware, OS and applications. Mohapatra et al. [27] proposed an approach that uses a middleware to coordinate the adaptation of hardware and applications at coarse time granularity (e.g., at the time of admission control). EQoS [34] is an energy-aware QoS adaptation framework,
which formulates energy-aware QoS adaptation as a constrained optimization problem. GRACE [49, 51, 50] coordinates the adaptation of the CPU speed in the hardware layer, CPU scheduling in the OS layer, and multimedia quality in the application layer in response to system changes at both fine and coarse time granularity.

We next introduce the design of our operating system, which is a part of the GRACE project [51, 50].

### 13.3.1 Design and Algorithm

The goal of the operating system is to maximize multimedia quality $q$ of all concurrent tasks in the mobile device under the constraints of CPU, network bandwidth, and battery energy. Figure 13.1 shows the architecture of the operating system, which includes four major components: a coordinator, a soft real-time CPU scheduler, a CPU adapter, and a WNIC adapter. The coordinator coordinates tasks and the CPU and WNIC resources to determine the quality level and CPU allocation for each task and the average power consumption for the CPU and WNIC. The CPU scheduler enforces the coordinated allocation to support the coordinated QoS levels of individual tasks. Finally, the CPU and WNIC adapters dynamically adapt the CPU and network card to minimize their power consumption. We next describe each component in turn.
13.3. QOS SUPPORT IN MOBILE OPERATING SYSTEMS

Coordination: The goal of the coordination is to maximize the aggregate utility of all concurrent tasks in the device subject to the constraints of CPU, network, and energy in the device. More formally, let’s assume that (1) there are \( n \) tasks concurrently running in the device. Each task has multiple QoS levels, \( \{q_{i1}, \ldots, q_{im}\} \). Each QoS level has a utility \( u(q_{ij}) \), consumes \( C(q_{ij}) \) cycles and \( B(q_{ij}) \) network bytes per period \( P(q_{ij}) \); (2) the remaining battery energy in the device is \( E \); (3) the estimated operating time of the device is \( T \); and (4) the available network bandwidth is \( BW \). The coordination needs to determine a QoS level for each task, the CPU speed \( f \) and power \( p(f) \), and the network power \( p_{net} \). Intuitively, when tasks operate at a higher QoS level, they demand more CPU and network resources; consequently, the CPU and WNIC perform at higher performance and hence consume more energy.

The coordination problem can be formulated as follows:

\[
\text{maximize} \quad \sum_{i=1}^{n} u(q_{ij}) \quad \text{(total utility)} \quad (13.1)
\]

subject to
\[
\sum_{i=1}^{n} \frac{C(q_{ij})}{P(q_{ij})} \leq 1 \quad \text{(CPU constraint)} \quad (13.2)
\]
\[
\sum_{i=1}^{n} \frac{B(q_{ij})}{P(q_{ij})} \leq BW \quad \text{(network constraint)} \quad (13.3)
\]
\[
(p(f) + p_{net}) \times T \leq E \quad \text{(energy constraint)} \quad (13.4)
\]
\[
f \in \{f_1, \ldots, f_{\text{max}}\} \quad \text{(CPU speeds)} \quad (13.6)
\]

The CPU power \( p(f) \) is directly determined by the speed \( f \). We determine the network power as follows: If the transmission speed for the WNIC is \( S \), the WNIC needs to be in the active state for \( \frac{B}{S} \) and in idle state for \( 1 - \frac{B}{S} \) for every second, where \( B \) is the aggregate bandwidth requirement of all tasks, i.e. \( \sum_{i=1}^{n} \frac{B(q_{ij})}{P(q_{ij})} \). Then the network power is

\[
p_{net} = p_{act} \times \frac{B}{S} + p_{slp} \times \left(1 - \frac{B}{S}\right) \quad (13.7)
\]

Note that in the above equation, we switch the WNIC into sleep when it is idle.

The above constraint optimization happens at coarse time granularity, for example, when a task joins or leaves the system. The coordination problem is NP-hard, since we can prove that NP-hard Knapsack problem is an instance of the above constraint optimization problem. We therefore use the dynamically programming algorithm \([28]\) that provides a heuristic solution. As a result of this solution, we determine the QoS level and CPU allocation for each task as well as the average CPU power and network power.

Soft Real-Time CPU Scheduling: Soft real-time scheduling is a common mechanism to support timing requirements of multimedia applications \([9, 30, \ldots]\).
CHAPTER 13. QOS SUPPORT IN WIRELESS ENVIRONMENTS

Here, we focus on the CPU scheduling. Previous soft real-time scheduling algorithms, however, often assume that the CPU runs at a constant speed. This assumption does not hold for our target mobile devices with a variable-speed CPU. As a result, we cannot directly use existing scheduling algorithms in our system. We therefore extend traditional real-time scheduling algorithms by adding another dimension—speed. That is, the scheduler also sets the CPU speed when executing a task and hence enforces the CPU allocation on a variable-speed CPU [48].

The operating system uses an energy-aware EDF scheduling algorithm, which enforces the globally coordinated CPU allocation on a variable-speed CPU [48]. Specifically, in this scheduling algorithm, each task has a deadline and a cycle budget:

- The deadline of the task equals to the end of its current period. That is, when a task begins a new period, its deadline is postponed by the period.

- The budget of a task is recharged periodically. In particular, when a task begins a new period, its budget is recharged to the coordinated number of cycles.

The scheduler schedules all tasks based on their deadline and budget. In particular, the scheduler always dispatches the task that has the earliest deadline and a positive budget. As the task is executed, its budget is decreased by the number of cycles it consumes. When the budget of a task is decreased to 0, the task is preempted to run in best-effort mode until its budget is replenished again at the next period.

This preemption provides temporal and hence performance isolation among tasks; i.e., a task’s performance is not affected by the behavior of other tasks [12, 18, 30].

CPU Energy Saving: As the coordination problem above shows, the coordinated CPU power consumption is $p(f)$, where is $f = \sum_{i=1}^{n} \frac{C(q_{ij})}{P(q_{ij})}$. In other words, we expect the CPU to execute at a uniform speed for all concurrent tasks. If each task uses exactly $C(q_{ij})$ cycles per period $P(q_{ij})$, this uniform speed technique would consume minimum energy due to the convex nature of the CPU speed-power function [17]. However, the instantaneous cycle demand of multimedia tasks often varies greatly. In particular, a task may, and often does, complete a job before using up its allocated cycles. Such early completion often results in CPU idle time, thereby wasting energy. To avoid this energy waste, we dynamically adapt the CPU speed during each task’s execution.

On the other hand, we cannot lower the speed too much; otherwise, the task may miss its deadline or cause other tasks to miss their deadlines. To do this, we allocate the task a time as follows: If there are $n$ concurrent tasks and each task is allocated $C_i$ cycles per period $P_i$, then the scheduler allocates the $i^{th}$ task CPU time $T_i = \frac{C_i}{\sum_{j=1}^{n} \frac{1}{P_j}}$ every period $P_i$. The reason for time allocation
13.3. QOS SUPPORT IN MOBILE OPERATING SYSTEMS

(in addition to cycle allocation) is to guarantee that each task executes for up to its allocated cycles within its allocated time, regardless of speed changes.

Without loss of generality, we focus on the speed adaptation for an individual task, which is allocated $C$ cycles and $T$ time per period and has a probability distribution of its cycle demand $F(x) = \Pr(X \leq x), 1 \leq X \leq C$. Our goal is to minimize the expected energy consumption of each job of the task. To do this, we find a speed for each of the allocated cycles of this task, such that the total energy consumption of these allocated cycles is minimized while their total execution time is no more than the allocated time. More formally, if a cycle $x$ executes at speed $f_x$, its execution time is $\frac{1}{f_x}$ and its expected energy consumption is $(1 - F(x)) \times \frac{1}{f_x} \times p(f_x)$ [10]. We can then formulate the speed adaption schedule problem as follows:

$$\min: \sum_{x=1}^{C} \left( (1 - F(x)) \times \frac{1}{f_x} \times p(f_x) \right) + \left( T - \sum_{x=1}^{C} \frac{1}{f_x} \times (1 - F(x)) \right) \times p_{idle}$$

subject to:

$$\sum_{x=1}^{C} \frac{1}{f_x} \leq T$$

$$f_x \in \{f_1, \cdots, f_{max}\}$$

where $p_{idle}$ is the CPU idle power at the lowest speed. Note that the energy consists of two parts: The first part is the energy consumed when executing all allocated cycles. The second part is the energy consumed during the residual time (i.e., the time budget minus the expected execution time of all allocated cycles). During this residual time, the CPU is often idle since the process needs to wait until the next job is available. During this idle time, we set the CPU to the lowest speed during the idle slack.

We refer to the above optimization as a statistical DVS (Dynamic Voltage Scaling) approach. This optimization happens at fine time granularity, for example, within a multimedia frame execution. The optimization problem is NP hard. To provide an approximate solution, we develop a dynamic programming algorithm, based on the algorithm proposed by Pisinger [35]. Specifically, we first divide the allocated cycles into groups and find a speed for each group, rather than for each cycle. We then consider the combinations of all speed options for all cycle groups and sort them in the non-decreasing order of a slope that is defined as the ratio of the increased energy to the decreased time by increasing a group’s speed to the next higher speed. We initially set all cycle groups to the lowest speed and then visit the sorted slope list. For the currently visited slope, we try to increase the speed of its associated cycle group to the next higher speed. We finish the visit when the total execution time of all cycle groups is no more than its allocated time.

Each task has its own speed schedule and its speed schedule applies to all its jobs. In other words, the OS changes the CPU speed in three cases (Figure 13.2):

- **Context switch.** After a context switch, the OS sets the CPU speed
based on the speed schedule of the switched-in task. This provides isolation of speed scaling among different tasks.

- **New job.** When the current task releases a new job, its execution speed is reset to the speed of its first cycle group.

- **Job progress.** The OS also monitors the progress of each job execution and changes the CPU speed when the job reaches its next cycle group.

**Network Energy Saving:** Dynamic power management (DPM) is a common technique to save network energy by switching the WNIC into sleep when it is idle. DPM, however, cannot be directly applied in our target multimedia systems for the following reason. Multimedia applications are periodic and need to transmit or receive data in each period. Consequently, the idle interval of the WNIC is often shorter than the period. Since the period is often shorter than the DPM overhead, the WNIC cannot enter the lower-power sleep mode. To save network energy, we use a buffering approach. In this approach, each task still performs computation every period in a timely fashion, but delays the transmission by buffering frames and sending them in bursts at longer intervals (Figure 13.3).

Specifically, let’s assume that the buffer size is \( k \) frames and each frame needs to transmit for \( t_{act} \) time. In each period \( P \), the task processes a frame and stores it in the buffer. When the buffer has \( k \) frames (i.e., every \( k \) periods), the OS sends all buffered frames in batch. The buffering approach combines short WNIC idle intervals with length \( (P - t_{act}) \) into longer ones with length \( k(P - t_{act}) \). Such aggregate idle intervals are larger than the DPM overhead;
so, the WNIC can enter sleep. The buffering approach saves more energy. That is, the network power in the $k$ period is

$$p_{net} = \frac{p_{act} \times k t_{act} + p_{slp} \times k(P - t_{act})}{kP} = p_{act} \times \frac{t_{act}}{P} + p_{slp} \times \left(1 - \frac{t_{act}}{P}\right)$$

which is equivalent to Equation 13.7. That is, we enable the WNIC to consume the coordinated power.

### 13.3.2 Experimental Results

We have implemented a prototype of the OS. The hardware platform for our implementation is the HP Pavilion N5470 laptop with a single AMD Athlon 4 processor, which supports six different frequencies, 300, 500, 600, 700, 800, and 1000 MHz. The laptop has a Cisco Aironet 350 wireless card. The coordinator, scheduler and CPU adapter are implemented as a set of patches and modules that hook into the Linux kernel 2.6.5. The WNIC adapter is implemented as a user-level process that switches the WNIC into the power saving mode (PSM) when it is idle and into the continuous access mode (CAM) when it is active.

We next evaluate the OS prototype. Since the coordination requires the utility function for each task, which is application specific, we focus on our evaluation on energy saving. To measure energy, we remove the battery from the laptop and let it use the power from the AC adapter. The power consumption is the product of the input voltage and input current from the AC adapter. We use the Agilent 54621A oscilloscope to record the measurement. The sampling rate of the oscilloscope is 5 kHz, i.e., making a sample every 200 microseconds. Figure 13.4 shows the setup for power measurement.

First, we analyze the impact of the CPU adaptation and WNIC adaptation together. To do this, we use a H263 encoder that encodes local raw images into frames in real-time and sends the encoded frames to a receiver through a wireless network. The input pictures are `paris.cif`. The H263 encoder can
process two or three frames per second in real-time. We measure average energy consumption characteristics for three different system scenarios:

- **no-adapt**: the CPU always runs at the highest speed and the WNIC always runs at the CAM mode.
- **CPU-only**: the CPU runs at a uniform speed that meets the total average demand of applications.
- **CPU+NW**: The CPU and WNIC both adapt.

This gives us an idea of the energy-saving resulting from CPU speed adaptation and wireless card mode changing.

Figure 13.5 shows the energy consumption of the laptop. We note that the CPU speed adaptation reduces base energy consumption of the laptop by about 34% at 3fps, and the network card mode-changing reduces the energy consumption of the network interface by about 42% at the same frame-rate. On the other hand, we find that the total energy saving in the CPU+NW case over the CPU-only case is only 2-3% with the HP Pavilion laptop. The reason is that the WNIC consumes much less energy than the CPU in the HP laptop.

Second, we evaluate the benefits of our proposed CPU energy saving technique. To do this, we disable the wireless connection and let the H263 encode
three frames per second in real-time and store the encoded frames in a local file. We run the stand-alone H263 encoder under the following DVS techniques:

- **No DVS.** This is the baseline technique that always runs the CPU at the highest speed.
- **Uniform DVS.** It sets the CPU speed based on the average CPU demand of the H263 encoder.
- **Reclamation DVS.** It first sets a uniform speed and sets to the lowest speed when the H263 encoder completes a job early.
- **Statistical DVS.** It adjusts the execution speed for each task job based on its demand distribution.

Figure 13.6: Benefits of statistical DVS comparing to other DVS techniques.

Figure 13.6 shows the energy results. Compared to the baseline algorithm without DVS, all DVS techniques save energy significantly. In particular, the statistical DVS reduces energy by 26.4% compared to the no-DVS approach. The reason is that the CPU does not need to always run at the highest speed. This clearly shows the benefits of energy saving by dynamically adapting the CPU speed. Compared to other DVS techniques, the statistical DVS further reduces the total energy by 2% to 10%. This clearly shows the benefits of adapting the CPU speed based on demand distribution of tasks.

### 13.4 QoS Support in Mobile Wireless Networks

To achieve end-to-end QoS guarantees in multimedia wireless networks, a strong QoS-aware cross-layer networking system support for wireless multimedia applications must be present. We will present a complete cross-layer networking
system support for a single-hop ad hoc network based on the IEEE 802.11 MAC layer. In this network, all the nodes are within one-hop transmission range of each other. They are able to talk to each other in a peer-to-peer fashion. They all share the same wireless medium and hence need to cooperate with each other in satisfying their QoS needs.

The nodes in our network use the IEEE 802.11 MAC protocol’s Distributed Co-ordination Function (DCF) mode for communication. The IEEE 802.11 standard specifies two operating modes: Point Co-ordination Function (PCF) and Distributed Co-ordination Function (DCF). The former requires a single coordinator to arbitrate access to the shared wireless channel. The latter mode allows peers to arbitrate channel access without any centralized coordinator, using a CSMA/CA protocol. Wireless nodes using the DCF mode carrier sense the medium. If the channel is busy, transmissions are deferred. When the channel is clear, nodes backoff for collision avoidance. A node that captures the channel for transmission uses a RTS-CTS-DATA-ACK cycle to transmit a MAC frame. The RTS/CTS handshake is used mainly to deal with the hidden terminal effect.

Multimedia applications running over this 802.11 DCF network need to pay attention to the following conditions: 1) interference of wireless communications between different flows within the network; and 2) dynamic network environment where resource usage pattern and wireless signal may vary with time. As a result, multimedia applications need to adapt to these conditions with proper system support.

To date, most of the existing work has been proposed within the context of an individual layer, such as routing and MAC layer. Much less progress has been made in addressing the overall system support for running multimedia applications over wireless networks. One solution for overall system support for wireless multimedia applications is to adopt a cross-layer system architecture between MAC, transport, middleware and application layers. All these layers communicate and coordinate with each other to support QoS for multimedia applications.

### 13.4.1 QoS Models

Before discussing details of our cross-layer networking system architecture, we first re-visit the QoS models proposed in the Internet and review their applicability in the wireless environment. There are two different QoS models: 1) integrated service (“IntServ”) model, and 2) differentiated service (“DiffServ”) model.

The IntServ QoS model defines two types of services: guaranteed and best effort. In guaranteed service, each flow can request certain level of QoS from the network, such as minimum bandwidth and maximum delay. Over the Internet, IntServ is usually implemented by per-flow resource reservation in the routers. To apply the IntServ model to a wireless network, admission control must be designed to work with imprecise and time-varying resources information. Furthermore, the reserved resources of a flow may have to change in response to wireless resource fluctuations. As a result, in wireless networks, multimedia...
applications often specify their QoS requirements over a range, e.g., minimum and maximum bandwidths, and the granted resource can be a QoS level within that range.

In the DiffServ model, flows are aggregated into multiple traffic classes. A router needs to provide certain per-hop forwarding behavior for each class of packets. In particular, we are interested in the relative DiffServ model [13] which assures the relative quality ordering between different classes. No guarantee is provided for any of those classes. This QoS model is appealing especially in wireless networks because it does not need to provide any bandwidth guarantees for any class of packets. Instead, it relies on the end-host’s adaptation behavior to dynamically select an appropriate service class for each of its applications.

These two QoS models address different needs of multimedia applications. IntServ is more stringent in resource provisioning. An application has a better level of QoS guarantee but at the same time there is a higher probability that the QoS request may be rejected in admission, or terminated due to resource fluctuations. The relative DiffServ model has less guarantee for each application, but each application is always allowed to send out packets, although with different levels of QoS.

In the following subsections, we discuss in detail our design of two cross-layer architectures that realize the QoS models mentioned above. There have been several other cross-layer architectures for dynamic bandwidth management and adaptation, such as INSIGNIA [20], SWAN [2], TIMELY [6], dRSVP [26], and most recently MPARC [47] and PBRA [46]. These cross-layer architectures assume different QoS models and network topologies, but the underlying mechanisms (sub-tasks) implemented in order to manage bandwidth are, with some exceptions, similar: available bandwidth monitoring at the MAC, soft state reservation, application adaptation to network variations, and fair bandwidth allocation. QoS research for wireless networks has also addressed fair scheduling at the MAC layer [44, 4, 24, 19, 23, 22, 15] and new transport mechanisms [40, 25, 3, 46] to improve application performance.

### 13.4.2 IntServ: Bandwidth Management

**Bandwidth Management Architecture:** Bandwidth Management architecture [38] arbitrates the bandwidth requests of all the flows in a single-hop wireless network. In this architecture, every host in the network monitors its MAC layer transmissions to observe wireless channel fading and interference effects. These observations are fed into a central network arbiter which takes the bandwidth requirements and channel effects pertaining to each multimedia stream in the network into account, to decide how much channel time each stream gets to access the network to ensure its requirements are met.

Overview of the architecture is shown in Figure 13.7. It consists of a middleware agent for each host which obtains channel quality updates from the MAC layer monitor and application throughput requirements from each application, including the media application. It translates the throughput requirement into channel time requirement, using the channel quality. The channel time require-
CHAPTER 13. QOS SUPPORT IN WIRELESS ENVIRONMENTS

Figure 13.7: Bandwidth management architecture: overview

ment represents the fraction of unit time that the media stream must have access to the wireless channel of the observed quality, in order to satisfy its throughput requirement. The channel time required thus depends on both the throughput required as well as the channel quality observed. The middleware agent feeds these channel time fractions required by a particular stream to a central network arbiter, called the Bandwidth Manager (BM), which resides on one of the hosts in the network. The BM allocates the unit channel time resource among the various media streams in the network. It can be configured with any logical policy to distribute the resource among the streams, taking into account their requirements, for example by using a fair, utility-based or price-based policy.

The BM returns to the middleware agent at each wireless host the channel fraction allocated to each application running on the host. When there is some change in application throughput requirement or channel quality observed, the BM must re-allocate resources. This may involve revoking partially the resources previously allocated to an application and re-allocating them based on the new network conditions.

Each host also has a rate-control system (i.e., traffic shaper) comprising leaky-bucket queues. It is configured to ensure that each application injects no more traffic than can occupy the channel for the fraction of time the application was allotted. The sequence of events within each wireless host is shown in Figure 13.8.

The centralized architecture shown in Figure 13.7 is flexible enough to work with any single-hop wireless topology. It can be used for single-hop peer-to-peer ad hoc networks, and for access-point (AP) based networks. Furthermore, it can be extended to a network consisting of multiple APs that cover a large area with overlapping frequency bands. In such networks, the BM must also keep track of spatial reuse of the channel resource, apart from tracking the channel time requirements [39]. The channel quality monitor that we describe below is powerful enough to help a host using one AP to detect the presence of interference from a host using a different one.
Channel Quality Monitoring: In the BM architecture, a key component is the monitoring of the channel quality. We monitor the channel quality at the MAC layer, i.e., we observe how fading and interference phenomena affect MAC frame transmissions. We observe the delay in MAC frame transmission, and observe the loss-rate of MAC frames. We explain in this section how fading and interference phenomena are manifested in the MAC frame delay and loss-rate. We do not change the IEEE 802.11 protocol in any way in constructing the channel quality monitor. We merely observe the MAC layer transmissions of data packets in drawing our inferences.

Interference on the network can be estimated by the amount of time a transmitting host senses the channel busy, and must hence back-off and wait before being able to transmit its RTS or DATA frame. Thus the delay $t_r - t_s$ in Figure 13.9(a) reflects the interference levels in the network. Signal fading effects cause bit-errors in individual frame transmissions, thus requiring the frame to be re-transmitted. If ultimately the RTS-CTS-DATA-ACK cycle is successfully completed, then the interval $t_r - t_s$ also measures signal fading effects, since delays due to re-transmissions are also accounted for in the interval. In case the RTS or DATA re-transmission limit is exceeded, then the frame is dropped at the MAC layer, despite a time $T_w$ wasted in trying to send it. Thus measuring this time wasted $T_w$ in Figure 13.9(b), and the frame loss rate is also crucial in estimating signal fading effects.

Our channel quality monitoring scheme also accounts for hidden-terminal effects that might occur in networks spread out over a larger area. Hidden terminal effects cause the CTS frame to be suppressed. This is because the transmitter of the RTS does not know about the transmissions in the receiver’s neighborhood. But these transmissions prevent the intended receiver from responding with a CTS. This may result in multiple RTS re-transmissions, as would be the case if there were bit-errors in the individual frames, and even result in RTS re-transmission limit being exceeded. Both of these scenarios are accounted for when we measure the delay in Figure 13.9(a), and the time wasted and frame loss rate in Figure 13.9(b).
The channel quality monitoring mechanism measures, over a time interval $T$, the number of frames successfully transmitted, the delay $t_r - t_s$ incurred in transmitting them, the number of frames lost, and the time wasted in attempting to transmit them $T_w$. These four measures comprise our channel quality metric. They give us an indication of how many higher-layer packets can be transmitted in unit time, and how many will be lost in the process. The channel fraction required for a media stream to obtain its required throughput, depends on this information.

Note that we have described above only the principle behind our channel quality monitoring mechanism. We have omitted the details pertaining to how different higher-layer packet sizes affect the monitor and also considerations pertaining to packet-header overhead consuming some channel fraction. Details on dealing with these issues can be found in [38].

**Illustrative Example:** We now demonstrate using the network simulator ns-2 the performance of our channel quality monitoring and rate-control schemes that together constitute our bandwidth management solution. We assume a network topology shown in Figure 13.10 with two APs, node mobility, handoff, and hidden node effects. The transmission range of a wireless node is 250 meters and the carrier-sense range is 550 meters.

![Figure 13.9: Successful and unsuccessful transmissions in 802.11](image)
Note that there is no spatial reuse of the channel, i.e., two transmissions are not simultaneously possible on the wireless channel. In order to create the hidden node effect, and illustrate how our scheme deals with it, we assume both APs use the same wireless frequency, although in practice adjacent 802.11 APs tend to use non-interfering frequencies in the 2.4 GHz band. The BM is located in the backbone distribution system and not shown in the figure.

Figure 13.11: Observed weighted throughput without bandwidth management

Figure 13.12: Observed weighted throughput with bandwidth management

Figure 13.13: Perceived channel capacity for each flow

Figure 13.14 shows the fraction of unit time each flow is permitted to be active on the wireless channel. Flows with lower perceived channel capacity
(i.e. worse channel quality) are allowed to spend more time on the channel, and vice-versa. The accuracy of our channel quality estimation, even in the presence of hidden node effects, is illustrated by the fact that the allotted channel fractions exactly compensate channel quality variations and the result is a high degree of fairness in throughput among flows.

Of course, throughput-fairness is only one notion of fairness. Another notion of fairness is channel-time fairness, wherein all flows get equal access to the channel, and flows with better channel quality end up transmitting more packets successfully. Since we provide flows with worse channel quality more access to the channel, we use the channel less efficiently as a result. (In the ideal case, for maximum efficiency, only one flow should transmit on a completely clear channel, but obviously this starves all other flows and is hence not a practical solution.) In our scenario above, we observe up to 15% drop in overall channel efficiency as compared to the baseline case without bandwidth management.

In cases with spatial reuse of the channel, the BM must arbitrate multiple resources. It must identify the flows in a particular region that share the wireless channel in that region of the network, and arbitrate the channel among them. In another area of the network, a different set of nodes share the wireless channel, and channel arbitration must be performed separately for that set of nodes. Details on identifying the flow sets that share the channel, and on the bandwidth management in a scenario with spatial reuse, can be found in [39].

13.4.3 DiffServ: Proportional Delay Differentiation

Delay Management Architecture: Our second cross-layer design is a DiffServ QoS architecture that provides different delays for packets in different service classes [45]. The cross-layer architecture in Fig. 13.15 operates from the MAC layer up to the application layer. At the network level, packets from different service classes are processed differently via per-hop forwarding mechanisms (e.g., packet scheduling and queue management). At the middleware level, a monitor component monitors the performances of applications. Based on the monitored results, it performs appropriate service class adaptation so that different applications are able to meet their required QoS specifications.

A detailed diagram of our delay management QoS architecture, which shows the key components of this architecture, is illustrated in Fig. 13.16. In order to provide QoS support in the wireless networking environment, these components
interact in the following way.

1. **At application level in the end hosts:**
   - The application notifies the *Adapter* in the middleware that it wishes to set up a flow between two end hosts. It also provides its QoS specification and adaptation policy to the *Adapter* in the middleware layer.

2. **At middleware level in the end hosts:**
   - Based on the previous performance of the service classes and the QoS specifications of the applications, the *Adapter* decides the appropriate service class for each application and notifies the *Classifier*. Adaptation is an application-specific process. Based on application-specific adaptation policy, actions are taken to adapt the application’s service class.
   - The packets from applications are delivered through the middleware layer, where the *Classifier* marks the packets with their corresponding service class.
   - The *Monitor* monitors the performance of each service class and notifies the *Adapter* of the observed changes and QoS violations.

3. **At network level in routing nodes:**
   - The *Queue Management* component allocates buffer spaces and marks or drops packets. It deals with packet loss rate differentiation.
   - The *Differentiated Scheduler* selects a packet to transmit. It performs packet-level QoS enforcement, allocates bandwidth for different flows and provides delay differentiation.
This architecture balances very well between architectural flexibility and scalability. At the network level, the service differentiation mechanisms work to bring scalability with per-class packet scheduling and queue management. At the middleware level, the individual QoS requirement of each application is met via the application-specific adaptation process.

In the following, we discuss details of the cross-layer proportional delay differentiation scheduler and the adaptation service at the middleware layer.

**Cross-layer Proportional Delay Differentiation Scheduler:** The model of the proportional service differentiation was first introduced as a per-hop behavior (PHB) for DiffServ in wireline networks [13]. It states that certain class performance metrics should be proportional to the differentiation parameters. In particular, if we consider the case of delay differentiation, in a network with \(C\) service classes the proportional delay differentiation model imposes the following constraints for all pairs of classes.

\[
\frac{\bar{d}_i(t, t + \tau)}{\bar{d}_j(t, t + \tau)} = \frac{\delta_j}{\delta_i}, \quad \text{for all } i \neq j \text{ and } i, j \in \{1, 2, ..., C\} \quad (13.12)
\]

where \(\delta_i\) is the service differentiation parameter for class \(i\), and \(\bar{d}_i(t, t + \tau)\) is the average delay for class \(i\), \((i = 1, 2, ..., C)\) in the time interval \((t, t + \tau)\), where \(\tau\) is the monitoring time scale.
The basic idea of proportional differentiation is that, even though the actual quality level of each class may vary with traffic loads, the quality ratio between classes should remain constant in various time-scales. In addition, such a quality ratio can be controlled by setting the service differentiation parameters, which provides flexible class provisioning and management. Under certain conditions (i.e., the network is well provisioned), applications with absolute delay requirements can select appropriate service classes to meet their requirements [8], even though the network offers only relative differentiation.

One of the packet scheduling algorithms that can realize the proportional delay differentiation model in a short time-scale is the waiting time priority (WTP) scheduler [14]. In this algorithm, a packet is assigned with a weight, which increases proportionally to the packet’s waiting time. Service classes with higher differentiation parameters have larger weight-increase factors. The packet with the largest weight is served first in non-preemptive order. Formally, let $wt_{pkt}(t)$ be the waiting time of a packet $pkt$ of class $i$ at time $t$, we define its normalized waiting time $\hat{wt}_{pkt}(t, i)$ at time $t$ as follows.

$$\hat{wt}_{pkt}(t, i) = wt_{pkt}(t) \cdot \delta_i$$

(13.13)

The normalized waiting time is then used as the weight for scheduling. The packet with the largest weight is then selected by the WTP scheduler for transmission. Formally, at time $t$ it will transmit the packet $pkt$ which satisfies

$$pkt = \arg\max_{pkt \in \mathcal{P}} \hat{wt}_{pkt}(t, i)$$

(13.14)

where $\mathcal{P}$ is the set of backlogged packets. It is shown that WTP scheduler is able to approximate the proportional delay differentiation model in wireline networks under heavy traffic condition [14].

Here we introduce the proportional service differentiation model into the domain of wireless LAN. Different from wireline networks, where flows from the same router contend for the same wireline link, in wireless LANs not only do the flows originating from the same node contend with each other, the flows from different nodes also contend for the same wireless channel. To extend the concept of proportional service differentiation to the wireless LAN, the flows among different pairs of nodes are considered. Specifically, our proportional delay differentiation model for wireless LANs states that the relation Eq. (13.12) holds for all flows within the wireless LAN no matter whether they originate from the same node or not.

As a result of the distributed medium sharing, packet scheduling needs the cooperation among all the nodes. This is in contrast to wireline networks where packets that need to be scheduled originate from the same router, and hence the packet scheduling decision can be made by the route itself only considering its own packets. We argue that delay differentiation in wireless LANs can only be achieved through a joint packet scheduling at the network layer and distributed coordination at the MAC layer. Therefore, we present a cross-layer waiting time priority scheduling (CWTP) algorithm which is able to achieve proportional delay differentiation in wireless LANs.
The cross-layer waiting time priority scheduling algorithm (CWTP) divides the scheduling task into two parts which are performed at two layers in the network stack. At the network layer, *intra-node* scheduling at node $n$ selects a packet $\text{pkt}_n^*$ with the longest normalized waiting time, *i.e.*, a packet $\text{pkt}_n^*$ which satisfies

$$\text{pkt}_n^* = \arg \max_{\text{pkt} \in \mathcal{P}_n} \hat{wt}_{\text{pkt}_n}(t, i)$$

(13.15)

where $\mathcal{P}_n$ is the set of all backlogged packets at node $n$. At the MAC layer, *inter-node* scheduling selects packet $\text{pkt}^*$ among $\text{pkt}_n^*$, which satisfies

$$\text{pkt}^* = \arg \max_{\text{pkt}_n^* \in \mathcal{N}} \hat{wt}_{\text{pkt}_n^*}(t, i)$$

(13.16)

where $\mathcal{N}$ is the set of wireless nodes.

Such an intra- and inter-node scheduling algorithm can fit well the environment of wireless LANs. In particular, the intra-node scheduling can be implemented via network layer packet scheduling at each individual node and the inter-node scheduling can be implemented via media access control (MAC) which coordinates packet transmissions among nodes. Fig. 13.17 illustrates such a cross-layer scheduling architecture. In this architecture, the packet scheduler at the network layer and the distributed coordination function at the MAC layer are coordinated using normalized packet waiting time $\hat{wt}$ as a cross-layer signal.

![Cross-layer architecture](image)

Figure 13.17: Cross-layer architecture

At MAC layer, in order to transmit the packet with larger normalized waiting time before the ones with smaller normalized waiting time, we map the normalized waiting time $\hat{wt}$ to the backoff time $b$ via function $b = \Phi(\hat{wt})$. In [45], we present two mapping schemes, namely linear mapping and piece-wise linear mapping schemes to implement the function $\Phi((\hat{wt}))$. 
In linear mapping scheme, the normalized waiting time of a packet is mapped to its MAC layer backoff time via a linear function. Formally, let us consider a linear function $\phi(x) : \mathbb{R}^+ \rightarrow \mathbb{R}$,

$$\phi(x) = \beta - \alpha \cdot x$$

(13.17)

where $\alpha, \beta > 0$ are parameters of this linear function. To ensure it to be a non-negative integer, the backoff time $b$ (in number of time slot) of a packet with normalized waiting time $\hat{wt}$ is chosen as follows,

$$b = \Phi(\hat{wt}) = \lceil \lfloor \phi(\hat{wt}) \rfloor^+ \rceil$$

(13.18)

where $[x]^+ = \max(0, x)$ and $\lceil \cdot \rceil$ is the ceiling operation. These two operations round up the value of $\phi(\hat{wt})$ to a non-negative integer. It is obvious that $\alpha$ and $\beta$ determines the effectiveness of the mapping function, and thus the performance of the cross-layer scheduling algorithm. We present a dynamic tuning algorithm of $\alpha$ and $\beta$. Let $\overline{cw}$ be the expected value of contention window under IEEE 802.11 DCF without differentiation. The backoff time $b$ is uniformly chosen from $[0, \overline{cw})$. Let $\hat{wt}_{max}$ and $\hat{wt}_{min}$ be the maximum and minimum normalized waiting time respectively. Preferably, the maximum normalized waiting time $\hat{wt}_{max}$ can be mapped to the smallest backoff time (0) for efficient channel utilization; and $\hat{wt}_{min}$ can be mapped to $\overline{cw}$ for similar contention behavior as IEEE 802.11 without differentiation.

Linear mapping scheme neglects the fact that the distribution of the normalized waiting time can be non-uniform. If there is a higher density over a certain interval of time, then it will increase the possibility of packets with different normalized waiting time being mapped into the same backoff time. It can also increase the possibility of packet collision at the MAC layer. To address above problems, we present a piecewise linear mapping algorithm which considers the effect of normalized waiting time distribution. In piecewise linear mapping algorithm the normalized waiting times $\hat{wt}$ are divided into $L$ intervals of equal lengths defined by points $\hat{wt}_{min} = \hat{wt}_0$, $\hat{wt}_1$, $\hat{wt}_2$, ..., $\hat{wt}_{L} = \hat{wt}_{max}$. During each interval, function $\Phi_i(\hat{wt}) = \lceil [\beta_i - \alpha_i \cdot \hat{wt}]^+ \rceil$ will be used for the mapping. Fig. 13.18 compares these two mapping algorithm.

We simulate the CWTP algorithm under both linear mapping and piecewise linear mapping schemes on a variety of network settings in ns-2 [41]. In the simulation, the number of nodes ($N$) is a parameter to show how CWTP scales to the network size. Each node in the wireless LAN sets up a connection. The transmission rate of each flow is configured to give the network an aggregated load about 1500Kbps.

We first show the impact of network size on CWTP algorithm. In this experiment, 2 service classes with $\delta_2/\delta_1 = 2$ are supported in the network. Fig. 13.19 shows the differentiation index $I$ with different numbers of nodes in the network. The differentiation index ($I$) is defined as the ratio of the average delay of the two service classes. That is
Figure 13.18: Linear mapping and piecewise linear mapping: a comparison.
\[ I = \frac{\bar{d}_1}{\bar{d}_2} \] (13.19)

where \( \bar{d}_i \) is the expected packet delay of service class \( i \). This metric shows the effectiveness of the service differentiation – how close the differentiation result matches the differentiation goal. Ideally, in these experiments \( I = 2 \). We observe that both linear mapping and piecewise linear mapping schemes can lead the CWTP scheduling algorithm to achieve a delay differentiation index very close to the target value, when the network size is relatively small (the number of nodes \( N < 20 \)). When the network size is large (e.g. \( N = 50 \)), the piecewise linear mapping scheme performs much better than the linear mapping scheme.

In Fig. 13.20, we show the instantaneous delay behaviors under these two schemes when \( N = 10 \). From these results, we observe that piecewise linear mapping scheme gives much more consistent and smooth delay behavior than linear mapping scheme. This is because with the consideration of normalized waiting time distribution, piecewise linear mapping significantly reduces the possibility of packet collision at the MAC layer.

**Middleware-based Adaptation Services:** Now we describe the adaptation services to be provided by the middleware framework. The adaptation services work with the network level service differentiation mechanism to provide an *absolute* QoS level for applications. At the network level, service differentiation provide differentiated quality for packets from different classes. However, applications usually require a QoS level with an absolute value hence the middleware is responsible for mapping the required QoS level to the correct service class. Our middleware achieves this goal by continually monitoring the performances of each applications and adaptively adjusts their service class to meet their required QoS level.
Figure 13.20: Instantaneous delay behavior.
The design of our middleware adaptation framework is based on a task control model as shown in Fig. 13.21(a). Within the middleware control framework, the Adaptation Task and the Observation Task are represented in two respective components: the Adapter and the Monitor. And the Target System is the differentiated network, represented by the Classifier in the middleware layer, as shown in Fig. 13.21(b). The Control Action is the service class selection; and the Task States are the end-to-end performance of the multimedia application. In particular, the Adapter takes the end-to-end delay observed by the Monitor as its input, make the service class selection decision based on the input values and sets the service class at the classifier as its output. It is controlled by a set of conditional statements in the form of if-then rules. In [32], we presented the detailed design of rules. An example rule is illustrated as follows, where $d$ is the current observed delay of the application, $d^*$ is its delay bound, $\delta(t)$ is its service differentiation parameter at time $t$.

\[
\begin{align*}
    \text{if} & \quad (d > 2.5d^*) \\
    \text{then} & \quad \delta(t + 1) = 2\delta(t)
\end{align*}
\] (13.20) (13.21)

We show the performance of the adaptation service integrated with the delay differentiation service over an IEEE 802.11-based wireless ad-hoc testbed implementation. In the experiment, we first start an audio application which has a QoS requirement in terms of maximum packet delivery delay. Then background UDP traffic with 15000 Bytes/s is started. From the results in Fig. 13.22, we see that the average delay increases quickly from 70ms to 800ms without service differentiation and adaptation. Using the service adaptation policy in the example and the underlying delay differentiation support, we observe that the average delay for the audio application was successfully bounded to $< 150$ms.
13.4.4 Comparison of QoS Architectures

The two QoS architectures described above support different QoS models. BM supports the IntServ model by admission control, bandwidth reservation, traffic shaping and bandwidth re-negotiations. Proportional delay differentiation supports the DiffServ model by a special per-hop forwarding behavior which relies on a joint scheduling algorithm at the MAC and network layers.

Each of these QoS architectures has its own strength and weakness. For example, it is convenient for BM to provide per-flow “soft” bandwidth guarantee, but a flow may be rejected in admission or terminated during transmission. In the delay differentiation architecture, every flow can always send out packets, but the quality protection between different classes of packets are only “relative”. Each application takes the risk and burden of choosing an appropriate service class to meet its own needs. Therefore, BM is more suitable for a small number of concurrent flows with stringent QoS requirements, while delay dif-
ferentiation is better for a large number of flows where a few of them are QoS sensitive while the rest are not.

Despite the differences in their QoS models, there is a common trait in these two architectures, which is the cross-layer design principle. In both architectures, there is a close interaction between application, middleware, network and MAC layers. Together they provide an agile adaption framework for QoS applications in wireless networks.

13.4.5 Beyond Single-hop Wireless Networks

The two QoS architectures discussed above assume a single-hop ad hoc network (or wireless LAN) where each node can talk to each other directly. In this section we discuss how to support multimedia applications in a multi-hop ad hoc network (or “MANET”).

Running multimedia applications over a MANET has even more challenges: 1) the network topology is dynamic which often results in route breakage and re-routing; 2) wireless resource usage is very dynamic and complex due to location-dependent wireless contention and spatial reuse. Examples of QoS support architectures in this network include INSIGNIA [20] and SWAN [2]. INSIGNIA supports the IntServ model by reserving bandwidth over a multi-hop path and continually re-negotiating the reservations via signaling. SWAN supports the DiffServ model by differentiating two classes of traffic: real-time and best-effort. Real-time traffic needs to go through a distributed admission control process at a flow’s start-up, and needs to monitor the available bandwidth of the path continuously.

Due to the dynamic nature of multi-hop ad hoc network, robust QoS support is very difficult. Hence we ask another interesting question: how can we better support multimedia flows as part of the best-effort traffic in MANET? We are not concerned about any QoS model, but we are interested in how flow control at the transport layer can facilitate the transmission of multimedia traffic. Traditional flow control such as TCP relies on “probing” the network until packet lost is observed. This is certainly not an appealing method to carry multimedia traffic because frequent and large rate fluctuations are inevitable especially a wireless environment.

To this end, we study a special explicit flow control scheme called “EXACT” [11] where the transport layer gives explicit rate signals to the application layer. Its design rationales are as follows:

- **Router Assisted Flow Control**: in our framework, router explicitly gives rate signals to the flows that are currently passing it, since routers are in a better position to react to network bandwidth variation and route changes in MANET.

- **Rate-based Transmission**: in our framework, the sender follows the rate information set by the routers, and hence the packet transmission is rate-based.
Feasibility in MANET: our framework incurs additional complexity and overhead at the routers. It is not targeted for the large scale Internet (where core routers have to process huge number of concurrent flows), but rather as a solution for the smaller scale MANET environment.

Overview: An overview of the EXACT framework is shown in Figure 13.23(a). Each data packet carries a special IP header, called flow control header, which is modified by the intermediate routers to signal the flow’s allowed sending rate. When the packet reaches the destination, the explicit rate information is returned to the sender in a feedback packet. As a result, any bandwidth variation along the path will be returned to the sender within one RTT.

In the event of re-routing (Figure 13.23(b)), the first data packet traveling through the new path \((R_1, R_2, R'_3)\) collects the new allowed rate of the flow. As a result, the sender learns the exact sending rate after only one RTT of delay after re-routing, without having to go through the additive probing phase of TCP.

A packet’s flow control header includes two fields: ER (Explicit Rate) and CR (Current Rate). ER is the allowed sending rate of a flow. It is initially set at the sender as its maximum requested rate, and subsequently reduced by the intermediate routers to signal its allowed data rate. CR is initially set at the sender as its current sending rate, and modified by the intermediate routers to signal possible rate reduction along the path. Each router remembers the CR of the current flows in its flow table, in order to compute each flow’s fair share of bandwidth.

Router’s Behavior: Router plays the central role in EXACT. A router has four major tasks: 1) keep track of current flows and their sending rates; 2) measure the current bandwidth of the outgoing wireless links; 3) compute rates for the current flows; and 4) update the header of each passing data packet.

The core part of each router is its rate computation algorithm to allocate sending rates for the competing flows. The rate computation, performed locally, is based on the current measured bandwidths of the outgoing links, as well as
the current rates of the flows going through the router. Efficiency is achieved by making sure that the flows can fully occupy the outgoing wireless links. Fairness can be achieved by allocating the bandwidth “fairly” to each flows. A common fairness criterion is max-min fairness [5]. In max-min fairness, flows with minimum requests are granted their requests first; the remaining bandwidth resource is then evenly divided among the higher demanding flows.

Here we propose to maintain fairness among competing flows according to their channel time demands to access the wireless channel. The wireless link’s bandwidth at the MAC layer is measured using the monitor as described above. To represent a flow’s resource request, we normalize a flow’s requested rate to its next-hop link’s bandwidth as $TF_i = r_i/b_i$, where $r_i$ is the flow’s data rate, and $b_i$ is the current bandwidth of the link. The max-min allocation is then performed on top of the requests of the flows: $TF_i, i = 1$ to $N$. Since each flow obtains a throughput proportional to its next-hop link’s bandwidth, we call it bandwidth-proportional max-min fair. For details of the rate computation, interested readers are referred to [11].

**Multimedia Streaming using EXACT:** EXACT provides explicit rate signals for the flows, but these rate signals may be very fluctuating. In order to support multimedia streaming on top of EXACT, our framework is flexible to support split-level adaptations. At the transport layer, EXACT provides explicit rate signals to the upper applications. It serves as the upper-bound of the application’s sending rate. Within this upper-bound, each application may adjust its own sending rate based on its adaptation policies, for example to maintain smooth rate changes for multimedia flows.

Such informed adaptation is possible only with EXACT’s explicit rate signals. Although all the flows are treated as best-effort at the transport layer, using EXACT as the flow control scheme facilitates running multimedia applications over MANET.

**Evaluations:** Here we show the efficiency of EXACT compared to traditional TCP flow control. Using the ns-2 simulator, we create a MANET with 30 nodes moving in a 1500m by 300m space with maximum speed of 20m/s and different pause times (0s, 5s, 10s, 15s, and 20s) to create different levels of network dynamics. Under these mobility patterns, we compare EXACT with TCP-Reno and TCP-SACK. For each scenario, we average the total number of reliably transmitted packets over 10 runs for each scheme. The results in Figure 13.24 show that under all mobility scenarios, EXACT overall outperforms TCP-Reno and TCP-SACK by 42% and 36% more packets, respectively. This demonstrates the effectiveness of the EXACT flow control scheme in a dynamic MANET environment.
13.5 Design Principles Learned

In this section we summarize various design principles we have learned in our study.

13.5.1 Cross-Layer Strategies

Cross-layer resource management strategies in wireless networks have attracted increasing attentions in recent years\(^1\). The need for cross-layer design is based on two characteristics of wireless networks. First, wireless medium is a shared medium. Sending a packet from a node to another creates interference to other nodes in the same neighborhood. Therefore, in designing network packet scheduling algorithms, we have to consider the interaction between network layer and MAC layer due to wireless interference. At the network layer, only packets within the same host are scheduled; while inter-host packet transmissions can only be enabled at the MAC layer.

Second, resources are generally scarce and variable in wireless environments, and hence they must be managed carefully. For example, we translate multimedia application requirements into bandwidth and CPU resource requirements, and use controllers at the lower-level to monitor the load on the resource. The feedbacks from the controllers are then used to tune the network and CPU schedulers, so as to satisfy multimedia application requirements. In this picture, the control flow of resource management is two-way. Variations in the load on the resource are fed by the lower-level controllers to the application, so that the application can adapt. The chosen operating quality level of the multimedia application is fed to the lower-level controllers so that the schedulers can be tuned accordingly.

\(^1\)Interested readers are referred to [1] for more research results in cross-layer design for wireless networks.

Figure 13.24: Comparison of EXACT with TCP under different mobility patterns.
13.5.2 Tightly-Coupled Resources

In wireless networks, resources are tightly coupled with each other. Therefore, we need to adopt coordinated resource management strategies because resources cannot be managed independent of each other.

For example, if a lot of bandwidth is available, a bandwidth management scheme may allot a high operating quality level to a media streaming application. But if CPU resources are scarce, a CPU management scheme will allot the same application a lower operating quality level. Obviously, this is a contradictory scenario, hence the resource management strategy must be coordinated between resources.

A direct result of the tight coupling of resources means that we need a multi-level resource management strategy. For example, some resources are global to the hosts (nodes) in a wireless network, e.g., network bandwidth. Other resources are global within a single host, e.g., energy. Thus, a multi-level resource management strategy is to let the network-wide resource management constrain the host’s resource availability, and let the host-wide resource management constrain each application’s resource availability.

13.5.3 Adaptation of Both Software and Hardware

Resource adaptation should not be limited to software only. To achieve greater flexibility, the middleware layer needs to consider the adaptability of both software and hardware. For example, adaptive hardware such as CPU and wireless network interface card can trade off performance for energy consumption; multimedia applications can trade off quality for resource demand.

Software and hardware adaptation and optimization can take place at different time granularity. At coarse time granularity, for example, when an application starts, we optimize to achieve high application utility and desired battery lifetime. At finer time granularity, for example, when the application renders a frame, we optimize to save more energy.

13.5.4 Suitable QoS Model

Selecting a suitable QoS model is the most important step in designing a QoS support architecture, because it has fundamental impact on the overall architecture. This is especially true in wireless networks due to the scarcity of bandwidth resources.

While selecting a QoS model, we need to keep in mind the unique characteristics of wireless networks. Internet’s QoS models, such as IntServ and DiffServ, should be carefully re-examined. For example, the main challenge of deploying IntServ over the Internet is the scalability problem in keeping per-flow states at the Internet routers. In contrast, the challenge of adopting IntServ in wireless networks is not the scalability problem; instead, it is the time-varying resource availability problem, which may result in repeated QoS setups. Relative DiffServ may be a more suitable QoS model here since it does not require the QoS
setup phase, and hence avoids the difficulty and overhead in doing repeated admission control and resource signaling.

13.6 Conclusion and Future Directions

We discussed in this chapter QoS support in mobile operating systems and mobile wireless networks for multimedia applications. We have shown that with careful OS design with respect to scheduling and dynamic voltage scaling, we can achieve deadline guarantees for wireless multimedia applications as well as energy-efficiency of mobile nodes to extend the application lifetime. Furthermore, we have shown two cross-layer networking architectures that support statistical bandwidth and delay guarantees in cooperative wireless single-hop environments. The Bandwidth Management architecture realizes the IntServ QoS model, while the proportional delay differentiation architecture realizes the DiffServ model. We have shown that by leveraging the cross-layer design principle, both of them can achieve different levels of QoS protection and are suitable in many situations in wireless networks.

Wireless network is in a critical junction of being widely accepted into everyday life by the proliferation of small wireless devices such as smart phones, as well as the maturing of VoIP softwares. Now many municipal governments are planning to roll out city-wide mesh 802.11 networks to the general public. Such networks are owned by a single entity, for example the city government, so that cooperation among the nodes can be assumed. It is likely that the IntServ QoS model can be implemented by per-user bandwidth provisioning based on their subscriptions, considering the fact that the number of users and flows should be manageable for a city-scale network. Higher speed 802.11 standards such as 802.11n using MIMO, is also going to alleviate the scarcity of wireless bandwidth. Wireless multimedia may become the next killer application, and QoS is certainly an important enabler in this picture.
Bibliography


