Adaptive and Radio-Agnostic QoS for Body Sensor Networks

Gang Zhou, Qiang Li†, Jingyuan Li†, Yafeng Wu†, Shan Lin†, Jian Lu†, Chieh-Yih Wan‡, Mark D. Yarvis‡, and John A. Stankovic†
Computer Science Department, College of William and Mary
gzhou@cs.wm.edu
†Computer Science Department, University of Virginia
†{*lq7c,jl3sz,yw5s,shanlin,jl3aq,stankovic*}*@cs.virginia.edu*
‡Intel Research, Oregon
‡{*chieh-yih.wan,mark.d.yarvis*}*@intel.com*

As wireless devices and sensors are increasingly deployed on people, researchers have begun to focus on wireless body-area networks. Applications of wireless body sensor networks include healthcare, entertainment, and personal assistance, in which sensors collect physiological and activity data from people and their environments. In these body sensor networks, quality of service is needed to provide reliable data communication over prioritized data streams. This paper proposes BodyQoS, the first running QoS system demonstrated on an emulated body sensor network. BodyQoS adopts an asymmetric architecture, in which most processing is done on a resource rich aggregator, minimizing the load on resource limited sensor nodes. A virtual MAC is developed in BodyQoS to make it radio-agnostic, allowing a BodyQoS to schedule wireless resources without knowing the implementation details of the underlying MAC protocols. Another unique property of BodyQoS is its ability to provide adaptive resource scheduling. When the effective bandwidth of the channel degrades due to RF interference or body fading effect, BodyQoS adaptively schedules remaining bandwidth to meet QoS requirements. We have implemented BodyQoS in NesC on top of TinyOS, and evaluated its performance on MicaZ devices. Our system performance study shows that BodyQoS delivers significantly improved performance over conventional solutions in combating channel impairment.

Categories and Subject Descriptors: C.3 [**Special-purpose and Application-based Systems**]: Real-time and Embedded systems; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; D.2.8 [**Software Engineering**]: Metrics—*complexity measures,performance measures*

General Terms: Algorithms, Design, Performance, Experimentation

Additional Key Words and Phrases: Body Sensor Networks, Asymmetric Architecture, Adaptive QoS, Virtual MAC, Bandwidth

1. INTRODUCTION

In the last few years commercial sensor network applications have emerged that require pervasive sensing of people and the environment, e.g., assisted living [Ganti et al. 2006] [Harvard CodeBlue] [MIThril] and smart homes [UVA Smart House] [Kidd et al. 1999] [UFL Smart House]. For these applications, it is essential to be able to reliably collect physiological readings from humans via Body Sensor Networks (BSN). Such networks could benefit from Quality of Service (QoS) mechanisms that support prioritized data streams, especially when the channel is

impaired by interference or fading. For example, heart activity readings (e.g., EKG waveforms) are often considered more important than body temperature readings, and hence can be assigned a higher priority in the system. In another example, a glucose data stream might be assigned a low priority when the readings are in normal range, but a higher priority might be re-assigned to the data stream by user applications when readings indicate hyper- or hypo-glycemia. QoS support is needed to ensure reliable data collection for high priority data streams and to dynamically reallocate bandwidth as conditions change, especially when the effective channel bandwidth is reduced by interference or fading.

QoS research is not new; prior research has focused on managing and reserving resources [Zhu and Cao 2005] [Aad and Castelluccia 2001] [Garg et al. 2003] [G. S. Ahn and A. Campbell and A. Veres and L. H. Sun 2002] in the Internet, wireless networks, and ad hoc networks. However, for body sensor networks, three new QoS challenges [Zhou et al. 2008] arise: (1) A body sensor network consists of two levels of devices: multiple simple sensor nodes and a more powerful aggregator. A sensor node can be very resource constrained. For example, a sweat sensor can be designed as a Band-Aid, which uses a film battery and is attached to the wrist to obtain sweat levels. In contrast, an aggregator is comparatively more powerful. An aggregator might be a cell phone or an electronic watch that includes a 32-bit processor and rechargeable batteries. This asymmetric, star-shaped architecture requires an asymmetric QoS solution, not typical of prior QoS research. (2) Existing medical sensor devices often use different radio technologies, such as CC1000 [ChipconCC1000], IEEE 802.15.4 [IEEE 802.15.4 2003], or Bluetooth [IEEE 802.15.1 2002]. There is a need for a radio-agnostic QoS solution, which can be easily ported from one radio platform to another. (3) Low power radios are susceptible to channel fading and interference. This issue is especially evident in BSN due to human body factor. A number of studies [Shah et al.] [Natarajan et al. 2007] [Roelens et al. 2006] [Johansson 2002] showed that the human body presents various adverse fading effects to wireless communication channels that are dependent on body size and posture. As a result, the communication channel characteristics in a BSN are highly variable and difficult to predict. Furthermore, when co-existing networks or RF emitting devices such as microwaves are present, packets may be lost due to interference, reducing the effective channel bandwidth [Zhou et al. 2006]. In both cases, the QoS software needs to re-allocate resources from lower priority streams to higher priority streams. Adaptive QoS scheduling is thus needed to provide statistical bandwidth guarantees, which are essential for reliable data collection in body sensor networks.

Driven by these new challenges, we propose BodyQoS, an adaptive and radio-agnostic QoS solution for body sensor networks. The main contributions of this work are:

—*Prioritized Data Stream Service:* Through well-defined interfaces, applications submit their QoS requirements for each stream, assigning a specific priority according to the data type and level of criticality. If the available wireless resource is not sufficient to serve all QoS reservations, the higher priority streams are served first.

—*Asymmetric QoS framework:* Most admission control functions and resource

scheduling computations in BodyQoS are managed by the aggregator, while little processing is required at sensor nodes.

—*Radio Agnostic QoS:* A virtual MAC is implemented in BodyQoS to represent and schedule channel resources. It separates the QoS scheduler from the underlying MAC implementation. The virtual MAC design allows the QoS system to be ported from one radio platform to another (less than 100 lines of modified code, in one instance).

—*Adaptive Bandwidth Scheduling:* Besides conventional RSVP-Light [RSVP] QoS scheduling, we also implement and evaluate adaptive QoS scheduling in BodyQoS that provides statistical bandwidth guarantees.

—*Testbed Implementation:* BodyQoS has been implemented in TinyOS [Hill et al. 2000] and evaluated on the MicaZ [CROSSBOW] platform. Our evaluation shows that BodyQoS exhibits improved performance compared to conventional solutions.

The rest of the paper is organized as follows. In section 2 we present an overview of the BodyQoS architecture. We then explain design details of the BodyQoS components: the Virtual MAC in section 3, the QoS scheduler in section 4 and admission control in section 5. We also build a statistical model for BodyQoS and theoretically compare the energy efficiency with or without BodyQoS in section 6. Implementation issues in TinyOS are discussed in section 7, and the performance evaluation and comparison with existing solutions are presented in section 8. Section 9 describes related work. Finally, in section 10, we present the conclusions.

2. BODYQOS OVERVIEW

A BSN is a collection of small, low power sensing devices (such as EKG, pulse oximeter, temperature, sweat) wirelessly connected to a resource-rich aggregation device (such as a watch or cell phone), all within one communication hop of each other. BodyQoS is a set of software modules that sits between the transport and the MAC layers. BodyQoS receives QoS and data transmission requests from the transport layer and uses the underlying MAC protocol to transmit data. BodyQoS consists of three components: Admission Control, QoS Scheduler and Virtual MAC (VMAC). BodyQoS adopts an asymmetric architecture. The Admission Control and Scheduler components are implemented as a master and slave module on the aggregator and sensor nodes respectively. As shown in Figure 1, the slave admission control and slave QoS scheduler execute on the sensor motes, while the main part of admission control and the QoS scheduler execute on the aggregator.

The admission control module has two main functions. The first function is to accept or reject new QoS reservations. The decision is based on the requested bandwidth, time delay requirement, and priority, as well as the available wireless resources, previously accepted reservations, and the effective channel bandwidth. The second function is to continuously monitor and estimate the effective bandwidth, which varies according to interferences in the environment and body fading effect. QoS reservations are dynamically adjusted as necessary.

The main function of the QoS scheduler is to control use of the channel such that reservation requirements are attained. It schedules wireless resource for all admitted streams and provides admission control signaling between sensor nodes

and the aggregator. The scheduler also allocates unused wireless resources for best-effort communications. The QoS scheduler is also responsible for measuring the effective channel bandwidth at runtime and for reporting it to the admission control module. Finally, the QoS scheduler can schedule a sensor node to sleep if it predicts a lull in the communication schedule.

The VMAC isolates BodyQoS from the details of a specific MAC protocol. The VMAC design is challenging because it must control and reserve the wireless resource to support QoS scheduling, while abstracting away the details of the underlying MAC, including both CSMA-based and TDMA-based MACs. To achieve this goal, the VMAC exposes an abstract representation of wireless resource. It also provides an interface for the QoS scheduler to schedule wireless resource.

As illustrated in Figure 1, both admission control and the QoS scheduler consist of two parts that are distributed across sensor nodes and the aggregator. The aggregator makes admission control decisions, computes the communication schedule, and polls individual sensor nodes for data. Use of polling simplifies resource scheduling and supports an efficient asymmetric QoS design. Polling also simplifies coordination among multiple aggregators, allowing possible extension to co-existing body sensor networks.

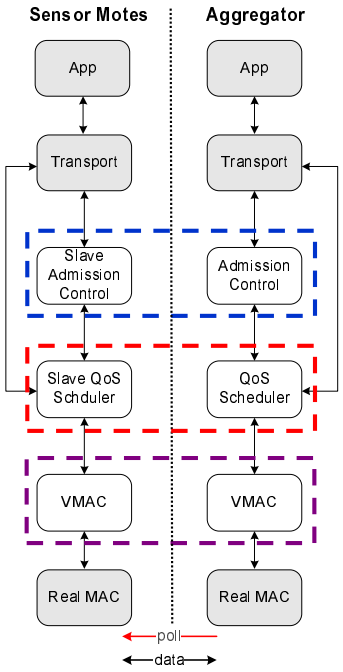


Fig. 1. BodyQoS Architecture

3. VMAC DESIGN

Today, several different radio platforms are available for sensor devices, e.g., the CC1000 [ChipconCC1000], the CC2420 [ChipconCC2420] or Bluetooth [IEEE 802.15.1 2002]. In some cases, multiple radio platforms may exist within a single system. To prevent QoS scheduling from being tied to a specific radio implementation, a virtual MAC design is required.

The VMAC design is particularly challenging. We first identify three MAC features that are essential for QoS support and commonly available in most MAC protocols. First, a MAC should be able to send a packet when requested. It should return control of the radio back to the scheduler when the packet is either transmitted successfully or fails after exceeding the maximum number of backoffs or retransmissions. As a result, there is a maximum time period for a MAC to handle a packet transmission request, after which the MAC should return control back to the QoS scheduler. Second, when a packet is received, the MAC should notify the QoS scheduler with the received packet. Third, it should track the time and energy overhead for transitioning between sleeping/waking states, to allow efficient sleep scheduling for sensor nodes.

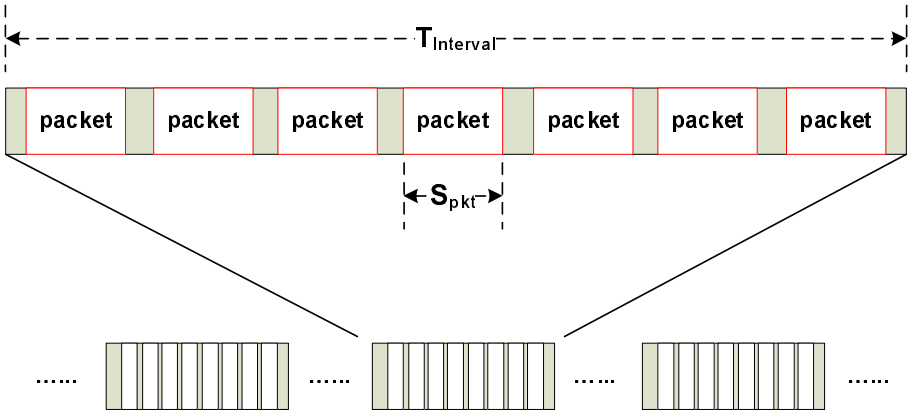


Fig. 2. VMAC Abstraction

Figure 2 presents our VMAC abstraction for representing and scheduling wireless resources. Time in VMAC is divided into intervals. Within each interval, VMAC is able to send out a certain number of packets with the specified data payload length, each within a specified time period. The following parameters are essential for the VMAC abstraction, and their values should be configured according to measurements of underlying MAC implementations.

—$T_{interval}$: the length of each interval in seconds.

—N_{pkt}: the maximum number of packets that the QoS scheduler can transmit or receive within each time internal, assuming a clean channel.

—S_{pkt}: the effective data payload size in bytes.

—T_{minPkt}: the minimum time duration for the MAC to transmit a packet assuming a clean channel, i.e., the minimum MAC response time for handling a packet transmission request, after which the MAC returns radio control to the QoS scheduler.

—T_{maxPkt}: the maximum time duration for the underlying MAC to transmit a packet or report giving up after exceeding the maximum backoffs or number of retransmissions. This is the maximum MAC response time for handling a packet transmission request, after which the MAC returns radio control to the QoS scheduler.

—$T_{minSleep}$: the minimum sleep duration for radio duty cycling, taking into account the delay to transition the radio between sleep and normal mode, as well as a user specified value to account for other energy costs.

The above parameters define a virtual MAC that the QoS scheduler can directly use for scheduling wireless resource. Setting these parameter values, however, is not trivial and is MAC dependent[1]. For example, S_{pkt} is the actual data payload length used in the underlying MAC. Likewise, T_{maxPkt} needs to account for both the worst-case backoff time and the maximum number of MAC-layer retransmissions. One

[1]Notice that the MAC dependency is encapsulated entirely in the VMAC.

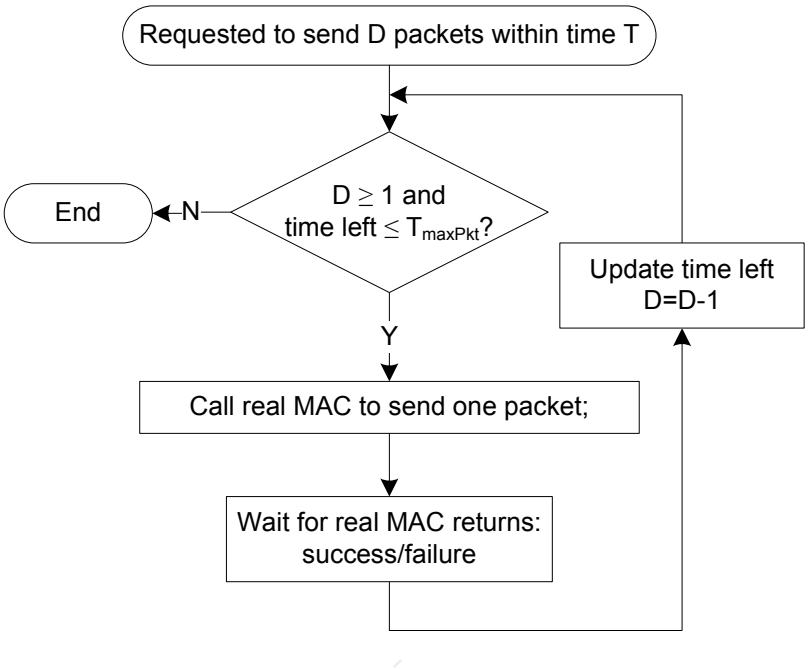


Fig. 3. Packet Transmission in VMAC

could obtain these values either analytically or experimentally. Once the above parameters are configured, VMAC is able to represent and reserve the wireless resource regardless of the underlying MAC implementation.

Figure 3 presents the packet transmission process in VMAC. In each interval, the VMAC attempts to transmit D packets iteratively within the time period T, until either all packets have been sent or the remaining time is less than T_{maxPkt}.

VMAC design is generic, refraining from functions that are only available in a specific MAC. For example, overhearing is possible in CC1000 and IEEE 802.15.4, but not in Bluetooth. This design choice allows the QoS implementation to be ported from one radio platform to another with minimum code modification. Researchers from U.C. Berkeley have developed a virtual MAC [Polastre et al. 2005], in which priorities and reliability can be configured for a set of underlying MAC protocols. However, this solution is designed for general use, without specific features for QoS reservation. Specifically, there is no support for bandwidth specification or reservation in their solution. In our BodyQoS design, as we will discuss later in Section 4 and 5, users' high level bandwidth specification can be translated into low level VMAC parameter configuration, which as shown in Figure 3 can be enforced regardless of the underlying real MAC implementation.

4. QOS SCHEDULER DESIGN

The QoS scheduler is the core of BodyQoS. We first describe the basic scheduling framework that BodyQoS uses to reserve the channel for different types of data

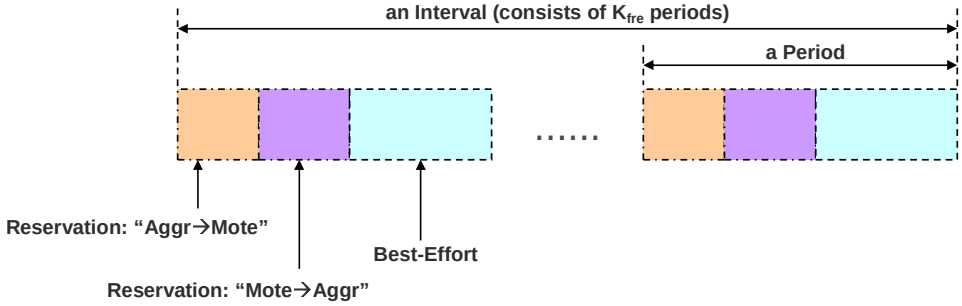


Fig. 4. QoS Traffic Scheduling

traffic. Then we describe a mechanism to measure the effective bandwidth in a radio-agnostic manner. Finally, we present two QoS scheduling algorithms: RSVP-Light [RSVP] QoS scheduling and adaptive QoS scheduling. We also consider the impact of different delay requirements on scheduling.

4.1 Basic Scheduling Framework

There are three types of traffic in our system: (1) reserved aggregator→mote communication, (2) reserved mote→aggregator communication, and (3) best-effort communication. Each QoS reservation has a delay requirement that is either high or low sensitivity. A reservation that has low delay sensitivity is only scheduled once for communication within an interval. For a reservation with high delay sensitivity, a data communication slot is scheduled K_{maxFre} times within each interval, where K_{maxFre} is a system-wide parameter[2].

The average scheduling delay for a low delay sensitivity reservation is $\frac{T_{interval}}{2}$, and the worst case scheduling delay is $T_{interval}$. For a high delay sensitivity reservation, the average scheduling delay is $\frac{T_{interval}}{2\times K_{maxFre}}$, and the worst case scheduling delay is $\frac{T_{interval}}{K_{maxFre}}$. Depending on delay sensitivity requirements, an interval can be divided into either 1 or K_{maxFre} periods. Within each period, each high delay sensitivity reservation is scheduled once for data communication. For ease of explanation, assume that an interval consists of K_{fre} periods, which can be either 1 or K_{maxFre}.

For a mote→aggregator QoS reservation, the aggregator generates polling packets to poll each sensor mote for data within each time interval. If the number of packets returned for each poll is too small, the control overhead will be high. Hence, we aggregate multiple short polling packets into a larger one. However, the requested number of packets can not be arbitrarily large within a single polling packet because losing such a polling packet leads to significant wireless bandwidth loss. In BodyQoS, a dynamically configurable parameter PL is used to set the maximum number of packets requested within a single polling packet.

To take advantage of polling aggregation, we group all QoS reservations that request mote→aggregator traffic, as shown in Figure 4. Within each period, the wireless bandwidth is first assigned to aggregator→motes traffic. If the requested

[2]In the current implementation, K_{maxFre} is configured during system initialization. Dynamic configuration of the K_{maxFre} value can be added in the future.

data packets are successfully delivered before the assigned bandwidth is consumed, the unused bandwidth is used for scheduling motes→aggregator traffic. The remaining bandwidth is used for best-effort communication.

For each QoS reservation R_i, the aggregator knows how many QoS reservations remain to be scheduled before serving the best-effort traffic. Thus, the aggregator can assign reservation R_i a sleep period T_{sleep}, during which R_i will not be scheduled for data communication and the corresponding radio can be scheduled for sleeping. The T_{sleep} value is piggybacked in each polling packet to notify corresponding sensor motes. Each sensor mote checks all of its QoS reservations for the overlapping sleep period. If the overlapping period is greater than the configured sleep overhead $T_{minSleep}$, the radio is put into sleep mode for the overlapping period.

4.2 Measuring Effective Bandwidth

In order for BodyQoS to perform efficient QoS scheduling, it is essential to measure the effective bandwidth that the QoS system can use during runtime. Conventional wisdom for measuring effective bandwidth usually depends on underlying MAC functions. For example, in CODA [Wan et al. 2003], each node samples the channel load periodically to compute how busy the channel is. This scheme works for radios that have the carrier sense ability, and may not work for frequency hopping spread spectrum radios such as Bluetooth [IEEE 802.15.1 2002]. The effective bandwidth can also be measured through utilizing ACK packets such as in B-MAC [Polastre et al. 2004] or the RTS-CTS-DATA-ACK handshake in IEEE 802.11b [IEEE 802.11 1999].

In BodyQoS, we propose a radio-agnostic method to measure effective bandwidth by taking advantage of the global knowledge of the QoS scheduler. In BodyQoS, if QoS reservation R_i requests bandwidth b_i, the QoS scheduling algorithm, discussed later in this section, interprets the request as: within each time interval, request VMAC to send/receive D_i packets within time $T_i \times D_i$, where T_i is the time for MAC to send a packet. The D_i and T_i information is included in a polling packet and sent from the aggregator to a mote. When the mote receives this packet, it continues sending the requested D_i packets until the specified time period $T_i \times D_i$ expires. When the requested reservation has no packet to send, a "NoData" packet is returned, which ensures that the mote tries its best to send back D_i packets within the time period $T_i \times D_i$.

On the aggregator side, after a polling packet is sent out, it waits for the mote response for a time period of $T_i \times D_i + T_{maxPkt}$, where T_{maxPkt} is used to account for the maximum backoff or retransmissions of the polling packet. The effective bandwidth computation needs to consider three possible cases:

(1) If the aggregator receives D_i packets within time $T_i \times D_i + T_{maxPkt}$, it stops waiting for more packets, and records the actual wait time $T_{waitTime}$. The number of packets received is D_i, and the effective bandwidth is $BW_{effective} = \frac{D_i * Bytes\,per\,Packet + Polling\,Packet\,Size}{T_{waitTime}}$, within the recorded actual waiting time.

(2) If the aggregator does not receive D_i replies, it continues waiting for packets until the specified time period $T_i \times D_i + T_{maxPkt}$ expires. Then it records the actual number of received packets to compute the effective bandwidth within the time period $T_i \times D_i + T_{maxPkt}$ as:

$BW_{effective} = \frac{Num\,of\,Recived\,Packet * Bytes\,Per\,Packet + Polling\,Packet\,Size}{T_i \times D_i + T_{maxPkt}}$

(3) If the aggregator does not receive any packets before the time period $T_i \times D_i + T_{maxPkt}$ expires, it knows that either the polling packet was lost, or all replies were lost. The aggregator assumes that the single polling packet was lost (since this is far more likely), and the effective bandwidth is 0 within the time period T_{maxPkt}.

As discussed above, whenever a polling packet is transmitted, the aggregator gets a chance to measure the effective bandwidth almost for free. Even though these bandwidth measurements are only samples of the channel when polling occurs, they are accurate enough for the QoS support since the time interval is relatively short (2 seconds in our system).

The aforementioned is a direct way to compute the effective bandwidth in BodyQoS. To reduce the processing load, we propose a second, simpler, and indirect scheme: $BW_{effective} = BW_{ideal} * \frac{Num\,of\,Delivered\,Packet}{Num\,of\,Requested\,Packet}$. For each polling packet, the QoS scheduler knows the number of requested packets. The number of delivered packets can also be easily obtained at the aggregator. The ideal bandwidth can be computed as follows:

$$BW_{ideal} = \frac{N_{pkt} \times S_{pkt} \times 8}{T_{interval}} \tag{1}$$

In our performance evaluation, we use this indirect scheme to compute effective bandwidth, and also to maintain a moving average of effective bandwidth with the decay factor of δ:

$$\begin{aligned} BW_{movAvg}(i+1) &= \delta \times BW_{effective} \\ &\quad +(1-\delta) \times BW_{movAvg}(i) \end{aligned} \tag{2}$$

4.3 Advanced Scheduling Algorithms

Next, we describe a way to compute T_i and D_i values at runtime. In BodyQoS, we provide two algorithms for computing these values: RSVP-Light QoS scheduling and adaptive QoS scheduling.

4.3.1 *RSVP-Light QoS Scheduling.* In the standard RSVP [RSVP] protocol, an audio/video stream reservation consists of a fixed bandwidth and time for data communication, and lost packets are not retransmitted. This is reasonable for audio/video streams because it does not make sense to play late or unordered portions of audio/video data. BodyQoS supports this conventional wisdom, by reserving a fixed wireless bandwidth according to the QoS reservation, disregarding the current channel condition and measured effective bandwidth.

If a QoS reservation R_i requests b_i Kbps bandwidth, the number of bits sent within each time interval $T_{interval}$ is $b_i * T_{interval}$. Since the effective application data payload size for each VMAC packet is S_{pkt} bytes, the number of data packets that should be scheduled for R_i is:

$$D_i = \lceil \frac{b_i * T_{interval}}{S_{pkt} \times 8} \rceil \tag{3}$$

In section 3, we defined T_{minPkt} as the minimum time period for the real MAC

to send out a packet when the channel is clean. For VMAC to send out D_i packets, BodyQoS should reserve a time period of $T_{minPkt} \times D_i$

4.3.2 *Adaptive QoS Scheduling.* To cope with channel impairment, it is essential for the QoS system to manage wireless resources adaptively, based on channel conditions. For example, an EKG sensor continuously samples heart activities and places data into a limited size buffer in a sensor mote. During times of channel impairment, the effective bandwidth reduces and packets are lost. The lost packets should receive more opportunities to be retransmitted. The question is how much time should the lost packets get for retransmission. For example, if the packet needs 4ms to be sent when the effective bandwidth is 100%, then 40ms is needed to service the reservation when the effective bandwidth reduces to 10%.

Now we must determine how to allocate the 40ms. One approach is to give the MAC more time for retransmission. However, MAC protocols typically limit the number of backoffs and retransmissions. Radio control is returned to QoS scheduler after time T_{maxPkt}. If the scheduler waited any longer, the radio would remain idle, wasting precious resources. Instead, a portion of the bandwidth should be given to the transport layer so that the transport layer can retransmit the lost packet. Traditional transport layers do not ask for more resources when packets are lost. For example, communication failure in TCP is considered a result of congestion. TCP actually backs off and uses less bandwidth when packets are lost. However, packet losses in a BSN are mainly due to channel impairment, rather than congestion, since the radio range is small and BSNs typically have a single use. As a result, transport layer retransmissions make sense in BSN. Without the adaptive bandwidth provision, buffer space in sensor motes may be depleted and data samples could be lost.

In order to schedule adaptive bandwidth, we need to compute D_i and T_i values dynamically, according to the effective bandwidth BW_{movAvg} from Formula 2. For ease of explanation, we denote the dynamically computed new values of D_i and T_i as $D_i^\star$ and $T_i^\star$. D_i and T_i are used to denote the values obtained in the ideal case when there is no channel impairment. Therefore, T_i in the ideal case is the minimum time for the MAC to send out a packet, that is, T_{minPkt}. When there is channel impairment and the effective bandwidth is BW_{movAvg}, the MAC needs to increase the time for sending one packet to $T_i^\star$:

$$T_i^\star = min\{T_{minPkt} \times \frac{BW_{ideal}}{BW_{movAvg}}, T_{maxPkt}\} \tag{4}$$

Where BW_{ideal} is computed according to Formula 1.

When the channel degrades to the point at which the maximum MAC retransmissions fails to deliver most packets, some additional time must be allocated so that the transport layer can retransmit lost packets. The total time assigned to both the transport and MAC layers should be the adaptive bandwidth the reservation requests under the current channel condition. In the ideal case, the time assigned to the reservation is $D_i \times T_i$. When there is channel impairment, the time assigned to it should be $D_i^\star \times T_i^\star$. So we have the following equation:

$$\frac{D_i^\star \times T_i^\star}{D_i \times T_i} = \frac{BW_{ideal}}{BW_{movAvg}}$$

From this equation, we can obtain the following formula:

$$D_i^\star = D_i \times \frac{BW_{ideal}/BW_{movAvg}}{T_i^\star/T_{minPkt}} \quad (5)$$

4.3.3 *High Delay Sensitivity.* Thus far we have only considered QoS reservations that require low delay sensitivity. For high delay sensitivity QoS reservations, the algorithms are slightly different.

For RSVP-Light QoS scheduling, we modify Formula 3 and calculate D_i as follows:

$$D_i = max\{\lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil, K_{maxFre}\} \quad (6)$$

For adaptive QoS scheduling, we can still use Formula 3 to compute D_i, and use Formula 5 to compute $D_i^\star$. However, the minimum allowable value for $D_i^\star$ is now K_{maxFre}.

5. ADMISSION CONTROL DESIGN

This section discusses three important aspects of admission control: (1) Computing the required bandwidth, including both data and signaling bandwidth for new QoS reservations; (2) Making decisions to admit/reject new QoS reservations. The same algorithm is used to adjust existing reservations when the effective bandwidth is reduced as a result of channel impairment; and (3) Coordinating slave admission control modules on the motes from a master module on the aggregator.

5.1 Computing Bandwidth Requirements

For each QoS reservation, the requested data bandwidth is directly obtained from its QoS specification. We also compute the control overhead that is needed for scheduling reservations, which varies depending on the type of QoS reservation.

For aggregator→mote QoS reservations, no polling packets are needed. The required data bandwidth is given by:

$$D_{AM} = \sum_i D_i, \text{ where } D_i = \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil \quad (7)$$

For mote→aggregator QoS reservations with low delay sensitivity, the required data bandwidth is given by:

$$D_{MAL} = \sum_i D_i \quad (8)$$

For mote→aggregator QoS reservations with high delay sensitivity, the required data bandwidth is given by:

$$D_{MAH} = \sum_i max\{D_i, K_{maxFre}\} \quad (9)$$

For mote→aggregator QoS reservations, regardless of the delay requirements, polling packets are needed. As discussed in Section 4, neither long polls nor short polls are optimal in terms of bandwidth efficiency. As a result, a standard polling length PL is used, which denotes the standard packet size that is desired to be

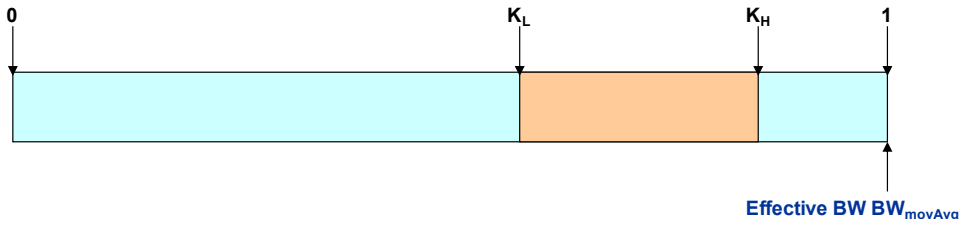


Fig. 5. Water Mark and Admission Decision

polled for each polling. Since all mote→aggregator traffic is grouped together for scheduling, the (polling) overhead bandwidth[3] can be computed as:

$$P = K_{fre} \times \lceil \frac{(D_{MAH} + D_{MAL})/K_{fre}}{PL} \rceil \tag{10}$$

Combining all aforementioned data and polling overhead bandwidths together, the total required bandwidth for all QoS reservations is:

$$D_{required} = D_{AM} + D_{MAL} + D_{MAH} + P \tag{11}$$

5.2 Admission Decisions

As shown in Figure 5, a high water mark K_H and a low water mark K_L are set within the resource range [0, 1], where 1 corresponds to the effective bandwidth BW_{movAvg}. If the total required bandwidth, including the new reservation, is no greater than the low water mark $K_L \times BW_{movAvg}$, then the new reservation is accepted.

The total required bandwidth may also fall between the low water mark and the high water mark $K_H \times BW_{movAvg}$. In this case, if the new QoS reservation's priority is no less than the highest priority among all admitted reservations, this new reservation is admitted. Otherwise, it is rejected.

Finally, if the required bandwidth is greater than the high water mark, BodyQoS considers removing existing, lower priority reservations to make room for the new reservation. If enough bandwidth can be reclaimed by removing lower priority requests, the new reservation is admitted. Otherwise, it is rejected. By removing existing reservations, the total bandwidth requirement may be pushed below the high water mark, which can prevent thrashing when the effective bandwidth oscillates. When less important reservations are removed, they are temporarily treated as best-effort communication, and the requesting applications are notified of the QoS termination. Applications can choose to resubmit their request with different QoS requirements, perhaps setting a higher priority or a lower bandwidth requirement.

The ejection policy is now described in more detail. First, admission control decides how much bandwidth to reclaim. Then, it sorts all admitted reservations in increasing priority order. When two QoS reservations tie in priority, the one

[3]We assume in this paper the resources consumed by polling and data packets is roughly the same, at a first order approximation. We are aware that large data packets do exist, and we will consider an additional variable for controling packet size in future work.

with the higher bandwidth requirement is considered first. The sorted list is then evaluated from head to tail. As the reservation list is processed, three situations may occur:

(1) If the aggregated bandwidth of all QoS reservations to be removed, including the current reservation being checked, is greater than the bandwidth that is needed to be reclaimed, then the current reservation is ignored; admission control checks the next reservation in the list.
(2) If the aggregated bandwidth of all QoS reservations to be removed, including the current reservation being checked, is smaller than the bandwidth that is needed to be reclaimed, then the current reservation is removed; admission control checks the next reservation.
(3) If the aggregated bandwidth of all QoS reservations to be removed, including the current reservation being checked, is equal to the bandwidth that is needed to be reclaimed, then the current reservation in the list is removed and the algorithm stops.

Once admission control determines which QoS reservations to remove, it notifies the QoS scheduler to stop reserving resources for each.

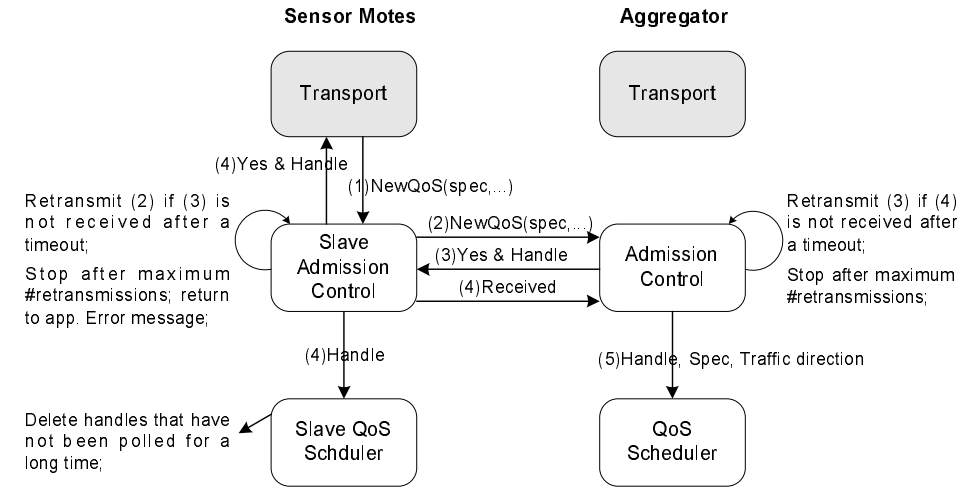
The above algorithm executes when a new reservation request is received and when the effective bandwidth drops below the amount needed for admitted QoS reservations.

5.3 Admission Control Signaling

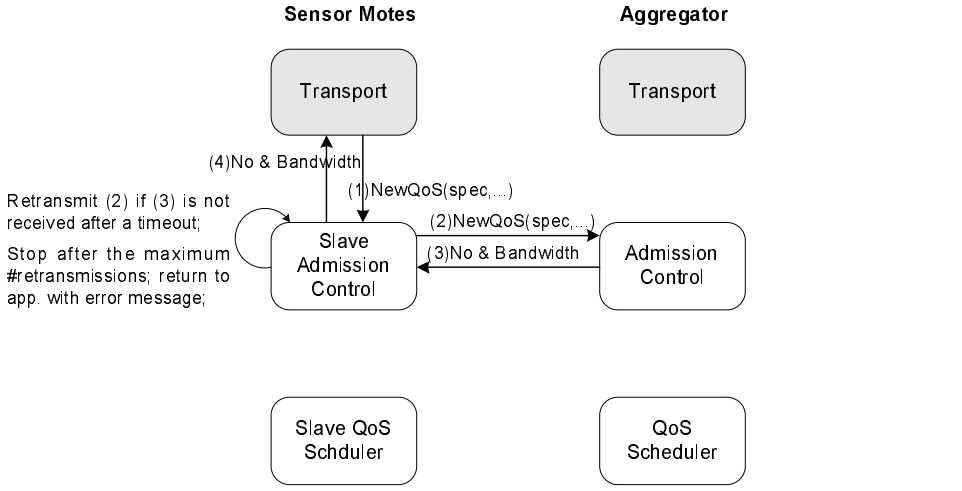
BodyQoS design is based on an asymmetric architecture. This drives the need for collaboration between the two components of admission control that execute on the sensor motes and the aggregator. Admission control signaling is necessary in three cases: when a new QoS request is submitted through the "NewQoS" interface, when an existing QoS is requested to stop through the "StopQoS" interface, and when an admitted QoS is removed through the "RemoveQoS" interface.

NewQoS Signaling: There are four cases for new QoS signaling, depending on whether the new QoS is submitted from a mote or the aggregator, and also depending on whether the admission result is yes or no.

Figure 6 presents the cases when a new QoS is submitted from a mote. As shown in Figure 6 (a), the signaling starts when a new QoS is submitted from the transport layer at a mote. Upon the NewQoS reception, the slave admission control component forwards this request to the admission control component at the aggregator. In this case, the aggregator accepts this request and replies with a handle for data submission in the future. When the slave admission control receives this handle, it does three things: notifies the transport layer of the acceptance and the handle, notifies the slave QoS scheduler of the handle, and replies to the aggregator that the yes decision has been received. The slave admission control needs to retransmit (2) if (3) is not received, and also the admission control needs to retransmit (3) if (4) is not received. These retransmissions are stopped after the maximum number of retries. Finally, when the aggregator receives the confirmation from the mote, it notifies the QoS scheduler to schedule resources for the reservation, and the NewQoS signaling ends.



(a) Mote submits NewQoS and reply is Yes



(b) Mote submits NewQoS and reply is No

Fig. 6. Admission Control Signaling for NewQoS

There is a possibility that message (4) from the mote to the aggregator is never successfully delivered, which leads to an inconsistency that the slave QoS scheduler has the handle, but the QoS scheduler does not. This can be resolved by timing out the handle at the mote, since it will never be polled by the aggregator.

Figure 6 (b) presents a different case when admission control sends a rejection. Similar to case (a), the slave admission control forwards the received NewQoS request to the aggregator. But different from case (a), when the slave admission

control receives the rejection, it does not send confirmation to the aggregator. It notifies the transport layer of the rejection decision, together with a bandwidth that the aggregator suggests for future QoS reservations. Also, retransmissions of (2) are necessary before (3) is received, and a maximum number of retries is set.

The signaling processes are similar when a NewQoS request is submitted from the aggregator. If admission control returns yes and a handle, it just informs the decision to the transport layer and the QoS scheduler at the aggregator. There is no need for handshaking with motes, since traffic is from the aggregator to motes. In the case when admission control returns no and the suggested bandwidth, there is also no need for handshaking with motes. Admission control just needs to return the rejection decision as well as the suggested bandwidth to the transport layer at the aggregator side.

StopQoS Signaling: The transport layer can submit a StopQoS request with "handle" as the parameter. When a StopQoS is submitted at the mote side, slave admission control notifies the slave QoS scheduler to delete the corresponding handle. When this handle is requested by the aggregator for further data, the slave QoS scheduler replies with a "HandleInvalid" message, so that the aggregator knows that this handle should be stopped. The QoS scheduler at the aggregator side removes the handle from its reservation list and also tells admission control to remove this handle. Then, the QoS reservation is stopped.

When a StopQoS is submitted at the aggregator side, admission control notifies the QoS scheduler to stop the corresponding handle. Then no more resources are reserved for this QoS reservation again.

RemoveQoS Signaling: Admission control removes lower priority QoS reservations when the available resource is not enough to serve all QoS requirements. If the reservation to be removed requires aggregator→motes traffic, admission control just tells the QoS scheduler to remove the handle, and then its data transmission requests are treated as best-effort traffic. It is also notified that it will no longer get QoS reservation service and a new QoS request should be submitted.

When the QoS reservation that must be removed needs motes→aggregator traffic, the mote needs to be notified since that is where data is submitted and transmitted. As shown in Figure 7, admission control informs the motes of the RemoveQoS message as well as the corresponding handle. Retransmissions are conducted, until either a confirmation is received from the mote or a maximum number of retransmissions have been tried. Since the handle is removed from the QoS scheduler, the reservation is not made. Instead, it is treated as best-effort communication. When communication fails and the handle is not removed from the slave QoS scheduler, it is automatically timed out.

6. BODYQOS MODELING AND ENERGY EFFICIENCY ANALYSIS

In this section, we build a statistical model for BodyQoS and analyze its energy consumption in a simple scenario. Especially, we study how the BodyQoS protocol affects the energy consumption of nodes.

To build the model, we assume that one aggregator and N motes perform two types of traffic in a body network: (1) The aggregator sends packets to motes with b_i bps bandwidth; (2) each mote sends packets to the aggregator with the QoS

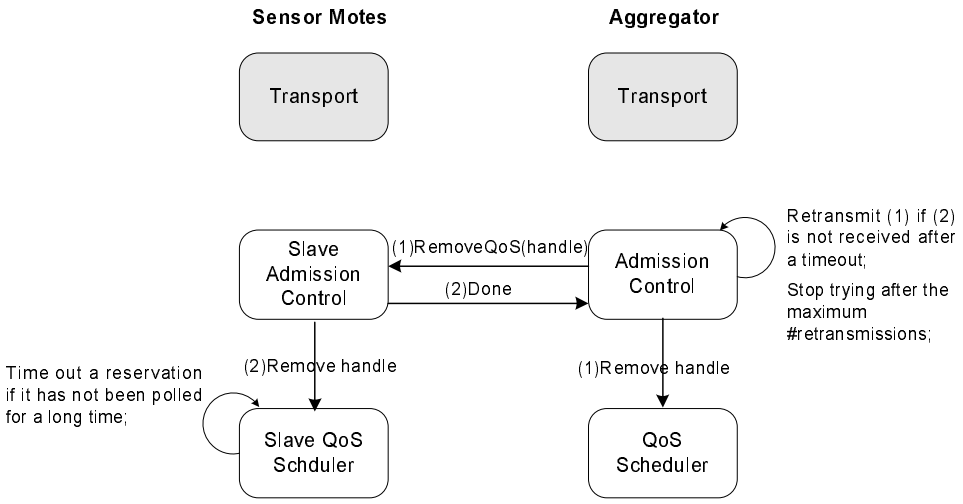


Fig. 7. Admission Control Signaling for RemoveQoS

Operation Mode	Current
Sleep	0.03mA
Idle Listen	26.1mA
Receive	26.5 mA
Transmit at -5dBm	21.6mA

Table I. Current Measurements for Different MicaZ Modes

reservation. Here, we do not consider the best-effort communication, because this type of traffic consumes the same energy with or without BodyQoS. In this simple scenario, we assume all motes request the same bandwidth, b_i bps. Furthermore, as we defined in section 3, we assume that it takes time T_{minPkt} for a sensor node to send or receive a packet when the channel is clear.

Next, we measure the energy consumption of a MicaZ node in different modes, using a power meter connected to the mote via the MicaZ's 51-pin connector [CROSSBOW]. During the measurement, the supply voltage of the node is 3.07 V. The current measurements are presented in Table 6. The power level is the computed as the product of the current and voltage readings.

From table 6, it is clear that MicaZ motes consume almost zero energy during sleep, comparing with current readings of other modes. So, for simplicity, we assume that nodes do not consume energy when sleeping.

6.1 Energy Consumption with BodyQoS

Now we compute the energy consumption of the aggregator and motes in one interval $T_{interval}$.

First, during the reserved Aggregator→motes communication, we have:

$$E^1_{Agg} = P_{tx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{Pkt\times 8}} \rceil \quad (12)$$

and

$$E^1_{Mote} = P_{rx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{Pkt\times 8}} \rceil \quad (13)$$

Here S_{pkt} is the effective application data payload size for each VMAC packet, as defined in section 3. P_{tx} is the MicaZ transmission power and P_{rx} is the MicaZ receiving power.

Second, during the reserved motes→Aggregator communication, the aggregator sends polling packets and receives packets from motes. According to Equation 10, the number of polling packets in one interval is:

$$P = K_{fre} \times \lceil \frac{N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt}\times 8} \rceil}{K_{fre} \times PL} \rceil$$

Therefore, the energy consumption of the aggregator in this stage is:

$$E^2_{Agg} = P_{tx} \times T_{minPkt} \times K_{fre} \times \lceil \frac{N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt}\times 8} \rceil}{K_{fre} \times PL} \rceil + P_{rx} \times T_{minPkt} \times N \times \lceil \frac{b_i \times T_{interval}}{S_{Pkt\times 8}} \rceil \quad (14)$$

According to the BodyQos protocol, each mote receives all polling packets, sends its own packets and goes to sleep after finishing its own transmission. Thus the energy consumption of the aggregator in this stage is:

$$E^2_{Mote} = P_{rx} \times T_{minPkt} \times K_{fre} \times \lceil \frac{N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt}\times 8} \rceil}{K_{fre} \times PL} \rceil + P_{tx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{Pkt\times 8}} \rceil \quad (15)$$

To summarize, when BodyQoS is used, the total energy consumptions of the aggregator and motes in one interval are as follows:

$$E_{Aggr} = E^1_{Aggr} + E^2_{Aggr} \text{ and } E_{Mote} = E^1_{Mote} + E^2_{Mote} \quad (16)$$

6.2 Energy Consumption without BodyQoS

In order to better understand the impact of the BodyQoS protocol on energy efficiency, we also compute the energy consumption of nodes when BodyQoS is not used. For fairness, we assume that the aggregator always sends packets first, and motes start to send after the aggregator is done. Here, we use the simple CSMA protocol, which is the standard MAC protocol provided in the TinyOS [Hill et al. 2000] library.

In this CSMA protocol, to send a packet, nodes first do the CCA to sense the channel. If the channel is clear, the packet is sent out immediately. Otherwise, the node randomly selects a time from [0, CW] to backoff, and then does the CCA again. In TinyOs implementation, CW = 0.32ms × 64 = 20.48ms. To ease the

analysis, we assume that this CSMA protocol eliminates all collisions. Thus, each mote still takes T_{minPkt} to send out one packet. Since all motes have the same packet incoming rate, we assume that a mote sends out the $(i+1)^{th}$ packet after all motes send out the i^{th} packets. Given that, the average backoff time of each packet can be approximated as $CW/2 \times min\{(N-1)/2, N_{try}\}$, where N_{try} is the maximum number of backoff retries. The total latency of all motes sending one packet is $CW/2 \times min\{(N-1)/2, N_{try}\} + T_{minPkt}$.

Therefore, when BodyQoS is not use, the energy consumption of aggregator and motes can be computed as follows:

$$E'_{Aggr} = P_{tx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil + P_{rx} \times T_{minPkt} \times N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil + P_{idle} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil \times (CW/2 \times min\{(N-1)/2, N_{try}\} - T_{minPkt \times (N-1)}) \quad (17)$$

and

$$E'_{Mote} = P_{rx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil + P_{tx} \times T_{minPkt} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil + P_{idle} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil \times CW/2 \times min\{(N-1)/2, N_{try}\} \quad (18)$$

6.3 Impact of BodyQoS on Energy Efficiency

To study the impact of the BodyQoS protocol on energy efficiency, we can compute the energy difference with or without BodyQoS. The energy difference in Aggregator and motes can be represented as follows:

$$\Delta E_{Aggr} = E_{Aggr} - E'_{Aggr} = P_{tx} \times T_{minPkt} \times K_{fre} \times \lceil \frac{N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil}{K_{fre} \times PL} \rceil - P_{idle} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil \times (CW/2 \times min\{(N-1)/2, N_{try}\} - T_{minPkt} \times (N-1)) \quad (19)$$

and

$$\Delta E_{Mote} = E_{Mote} - E'_{Mote} = P_{rx} \times T_{minPkt} \times K_{fre} \times \lceil \frac{N \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil}{K_{fre} \times PL} \rceil - P_{idle} \times \lceil \frac{b_i \times T_{interval}}{S_{pkt} \times 8} \rceil \times CW/2 \times min\{(N-1)/2, N_{try}\} \quad (20)$$

Let $T_{minPkt} = 2ms$, $K_{fre} = 10$, $b_i = 20Kbps$, $N_{try} = 8$, $S_{pkt} = 25bytes$, $T_{interval} = 2s$, and $PL = 5$, we compute ΔE_{Aggr} and ΔE_{Mote}, with different numbers of motes in this body network. Results are shown in Figure 8.

As shown in Figure 8, the BodyQoS protocol brings more energy consumption

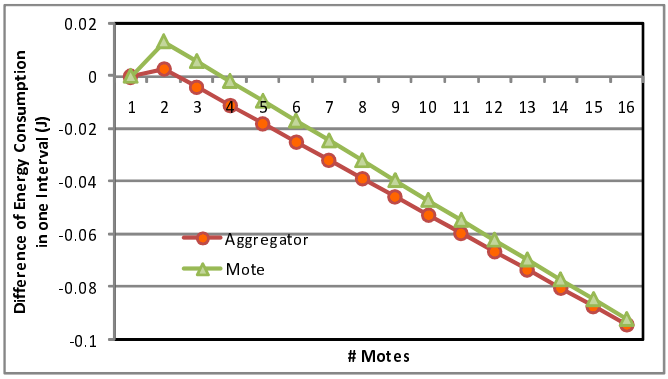


Fig. 8. BodyQoS Energy Efficiency Analysis

to the aggregate when there are one or two motes, mainly because the aggregate has to send extra polling packets, 40 packets in this case. However, When there are more than two motes, the BodyQoS actually reduces the energy consumption, because those polling packets schedule valid motes transmissions, and also reduces the backoff time of the CSMA protocol, and hence reduces the idle listening time of the aggregator. With more motes, more energy is saved. This trend is more obvious for motes' energy consumption. For example, with 15motes, the BodyQoS can save 0.09J in a 2 second interval.

Therefore, we can conclude from our analysis that our BodyQoS design can actually save a lot of energy compared with existing techniques, when there are more than two sensors deployed inside a Body Sensor Network, which is the typical case in real applications. We also observe that even when only one or two sensors are deployed in a body network, the control overhead that BodyQoS brings is still very small and hence can be ignored.

7. TINYOS IMPLEMENTATION

We have implemented BodyQoS in TinyOS [Hill et al. 2000] with the CC2420 [IEEE 802.15.4 2003] and Bluetooth [IEEE 802.15.1 2002] radio platforms. Three important implementation issues are addressed in this section: how to integrate BodyQoS with other modules and configurations in TinyOS, how to implement VMAC above the CC2420 radio platform, and how to implement VMAC above the Bluetooth radio platform.

7.1 Modules and Configurations

Figure 9 illustrates how BodyQoS is integrated with applications and the CC2420 radio components. In the TinyOS-1.x implementation, all packets from applications arrive at the RadioCRCPacket configuration module and then get wired to different radio platforms depending on the sensor devices that are used. When the MicaZ devices are used, RadioCRCPacket is wired to the CC2420RadioC configuration module. So the best way for BodyQoS to intercept all packets is to place it between the RadioCRCPacket and CC2420RadioC configurations. All "send" commands and "sendDone/receive" events between RadioCRCpacket and CC2420RadioC are

intercepted and scheduled by BodyQoS and used as best-effort communication.

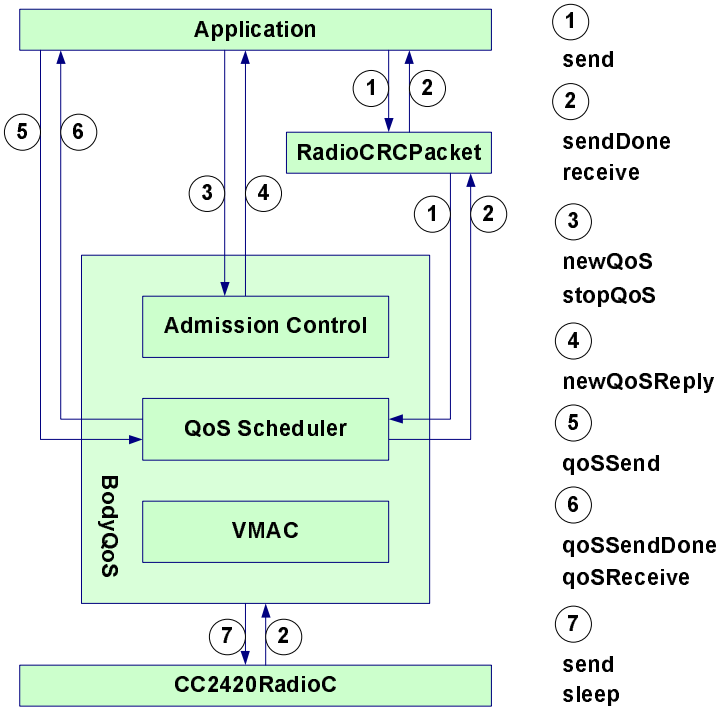


Fig. 9. VMAC materialization in CC2420

For QoS communicaiton, "newQoS/stopQoS" commands are provided by admission control, and the "newQoSReply" event is generated to inform applications of admission decisions. QoS data communication is done through the "qosSend" command and the "qosSendDone/qosReceive" events. VMAC communicates with the CC2420RadioC through the default "send" command and "sendDone/receive" events. It also calls the command "sleep" to schedule the CC2420RadioC for sleeping to save energy.

Intel provides TinyOS implementation of its IMote1 Bluetooth platform, which can be configured in a similar way and hence is not elaborated here.

7.2 VMAC Implementation with CC2420 and Bluetooth

In Section 3, Figure 3 explains how VMAC responds to a packet transmission request. In fact, in the whole response of VMAC, only a very little part is platform dependent. That is, only the "call real MAC to send the next packet" and the "wait for real MAC returns" in Figure 3 are radio platform dependent. So in an implementation, only this part needs to be separated for modification, if the system is ported from one radio platform to another. To demonstrate the code modifications for different radio platforms, the VMAC implementation in TinyOS

```
bool pending;
TOS_Msg sendPkt;
TOS_Msg recvPkt;

command result_t VMACSendPkt.send(TOS_MsgPtr msg){
    bool success;
    if(pending==TRUE)   return FAIL;
    memcpy(&sendPkt, msg, sizeof(TOS_Msg));
    success=call CC2420Send.send((TOS_MsgPtr)(&sendPkt));
    if(success)   pending=TRUE;
    return success;
}

event result_t CC2420Send.sendDone(TOS_MsgPtr msg, result_t success){
    pending = FALSE;
    return signal VMACSendPkt.sendDone(msg, success);
}

event TOS_MsgPtr CC2420Recv.receive(TOS_MsgPtr m){
    /* begin: drop noise packets*/
    if(HEADER_GetMsgType(m->info)>=TYPE_INVALID_UP
    || HEADER_GetMsgType(m->info)<=TYPE_INVALID_DOWN)   return m;
    /* end: drop noise packets*/
    memcpy(&recvPkt, m, sizeof(TOS_Msg));
    signal VMACRecvPkt.receive(&recvPkt);
    return m;
}
```

(a) VMAC Implementation on CC2420

```
bool pending;
TOS_Msg sendPkt;
TOS_Msg recvPkt;

command result_t VMACSendPkt.send(TOS_MsgPtr msg) {
    bool success;
    if(pending==TRUE) return FAIL;
    memcpy(&sendPkt, msg, sizeof(TOS_Msg));
    success = call BluetoothSend.send(msg->addr, (TOS_MsgPtr)(&sendPkt), sendPkt.length);
    if(success) pending = TRUE;
    return success;
}

event result_t BluetoothSend.SendDone(char *data, result_t success){
    pending = FALSE;
    return signal VMACSendPkt.sendDone(data, success);
}

event result_t BluetoothRecv.receive(uint32 Source, uint8 * Data, uint16 Length) {
    /* begin: drop noise packets */
    if(HEADER_GetMsgType(Data->info)>=TYPE_INVALID_UP
    || HEADER_GetMsgType(Data->info)<=TYPE_INVALID_DOWN) return FAIL;
    /* end: drop noise packets */
    memcpy(&recvPkt, (TOS_MsgPtr)Data, Length);
    signal VMACRecvPkt.receive(&recvPkt);
    return SUCCESS;
}
```

(b) VMAC Implementation on Bluetooth

Fig. 10. VMAC Implementation on Real MAC Protocols

on CC2420 radio and bluetooth radio are shown in Figure 10, from which we can see that the VMAC design really makes BodyQoS radio-agnostic.

8. PERFORMANCE EVALUATION

This section presents the performance evaluation of BodyQoS. We first explain our evaluation design and then go into details of performance discussion.

8.1 Evaluation Design

Our experimental setup mimics a typical assisted living facility, where a body sensor network is used for monitoring physiological readings of a patient at home. A MicaZ mote is configured to report heart activity readings generated by an EKG sensor to the aggregator. A second MicaZ mote is configured to report patient's location information, and a third MicaZ mote is configured to report body temperature readings. The aggregator is emulated by a MicaZ mote, which is connected to a serial port of a laptop for data collection. Within the body sensor network, we compare the performance of three solutions: adaptive QoS that we propose in BodyQoS, the conventional RTP-Like QoS that we also implement in BodyOoS, and the best-effort communication that requires no resource reservations. In different groups of experiments, which are presented later in this section, the EKG/location/temperature data streams use different services: adaptive QoS, RTP-Like QoS or best-effort communication.

Four metrics are measured for performance evaluation. First, we measure the ratio of the delivered bandwidth over requested bandwidth, which demonstrates the reliability of data collection. Second, we measure the average time delay of data communication, which starts when a packet is fetched from a sensor's data buffer and ends when the packet is actually delivered to the aggragator. Third, we compare the speed that the sampled data is removed from the buffer for transmission, since sensor motes have very limited storage space for buffering sampled data. Finally, we measure the energy consumption per delivered data byte.

Six groups of experiments are conducted for performance evaluation, and each of them is repeated for five times. We observed similar performance results in repeated experiments and present one representative result for each of them. Even though detailed parameter values of BodyQoS and experiment configurations can be found in the following subsections, here we list representative values of important BodyQoS parameter that we found out through performance evaluation: polling length $PL = 20$, decay factor $\delta = 0.2$, buffer size=50 packets.

8.2 Performance with Different Interference

Since most sensor nodes use the unlicensed 2.4GHz ISM bandwidth, a body sensor network may suffer interference from co-existing sensor networks, wireless mesh networks, cell phones, Bluetooth headsets, wireless routers or even microwaves. It is important to measure BodyQoS performance when different levels of interference is present. For this purpose, we design and conduct this experiment. In the first time period, 0s∼135s, there is no explicit noise. In the second time period, 135s∼225s, a MicaZ node is turned on to generate noise signals. It sends out a noise packet every 30ms. In the third time period, 225s∼315s, the noise node increases its noise

level by sending out a packet every 25ms. In the fourth time period, 315∼400s, the noise node increases its noise level again by sending out a packet every 20ms.

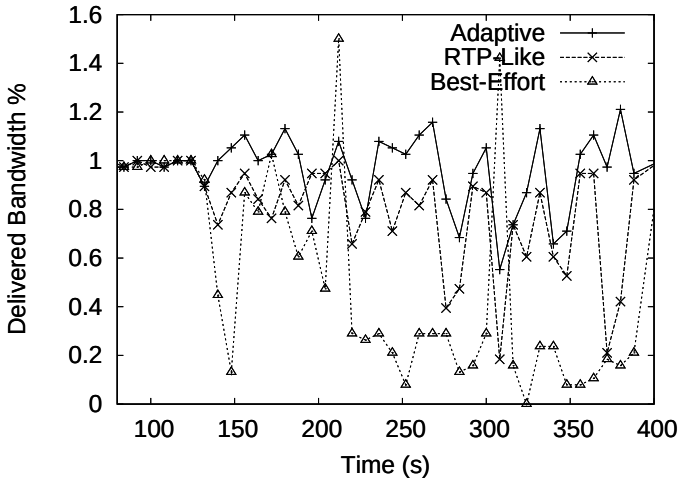


Fig. 11. Delivered Bandwidth with Different Interference

As presented in Figure 11, before the noise node is turned on at the end of the first time period, all three solutions work well, maintaining 100% bandwidth delivery ratio. But, when different levels of interference are present, both RTP-Like QoS and best-effort communication suffer severe packet loss and their bandwidth delivery ratios decrease. For example, the bandwidth delivery ratio of RTP-Like QoS decreases to about 90% in the second time period, and then to about 80% in the third time period, and then to about 60% in the fourth time period. Similarly, the bandwidth delivery ratio of best-effort communication decreases to 45∼90% in the second time period, and then 0∼30% in the third and fourth time periods. The reason for the performance decrease for RTP-Like QoS is that it always requests reserving fixed wireless resource, disregarding existing interference. The higher the interference level, the more the performance decreases. For best-effort communication, the performance decrease is because of two reasons. First, it does not get any promise of wireless resource reservation. When interference is present, more wireless resources go to adaptive QoS and less goes to best-effort communication. Second, when interference increases, it only relies on the MAC layer for more backoffs and retransmissions, but does not benefit from the adaptive bandwidth scheduling, in which collaborative effort from both MAC and Transport layers is made.

However, even when different interference levels are present in time period 2∼4, our adaptive QoS solution still maintains close to 100% bandwidth delivery ratio. Even though the bandwidth delivery ratio varies with time, which is due to the adaptation to interference, statistically speaking, the delivered bandwidth meets the requested bandwidth. This reliable data collection provided by adaptive QoS is mainly due to our novel bandwidth scheduling. When the body network suffers interference, the QoS scheduler allocates more resources for supporting more retransmissions, so that statistically the delivered bandwidth is able to meet the

requested bandwidth. We are also aware that sometimes the bandwidth delivery ratio for adaptive QoS exceeds 1. This is because when more time is allocated for adapting to the interference, the sensor node sends out the buffered data that has not received enough time to be sent out in the previous time interval.

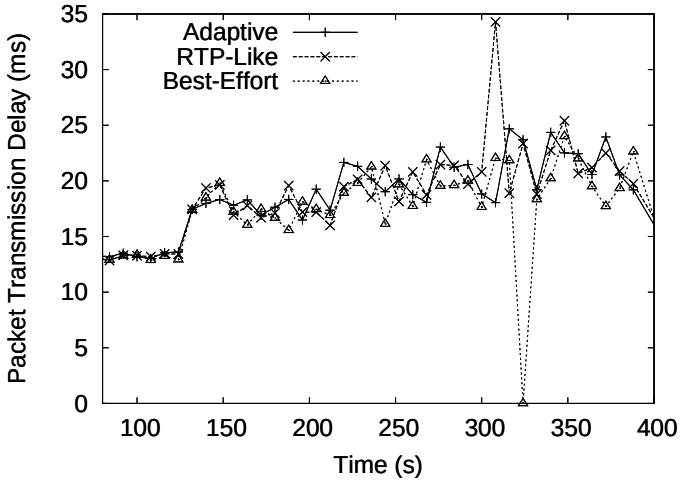


Fig. 12. Average Channel Access Delay with Different Interference

As shown in Figure 12, BodyQoS does not pay significantly more time delay for the impressive reliable data communication it provides. We do realize that the average time delay for all three services is a little higher than that when there is no QoS reservation. As we discussed in section 4, this time delay can be significantly reduced when we set the K_{fre} parameter greater than 1.

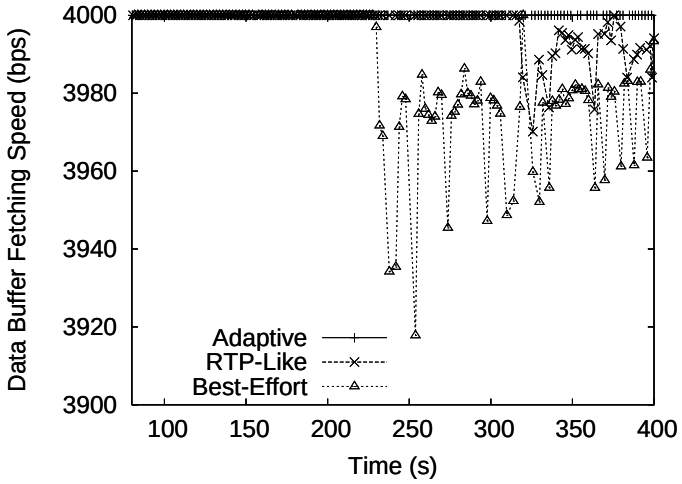


Fig. 13. Data Buffer Fetching Speed with Different Interference

In this group of experiments, the EKG/Location/temperature sensors all generate sensor readings at the same speed of 4Kbps. The data buffer fetching speed for the corresponding adaptive QoS/RTP-Like QoS/best-effort communication is illustrated in Figure 13. It is clear that adaptive QoS always fetches data from the buffer at the same speed as that the data is generated and put into the buffer, no matter whether there is interference or not. This feature is essential for the storage limited sensor nodes. Otherwise, when the buffer fetching speed is consistently lower than the data generating speed, the data buffer overflows, which leads to a blank area in the patient's EKG data. The reason that adaptive QoS can constantly provide the requested data buffer fetching speed, is that it adopts the adaptive bandwidth scheduling that adapts to the dynamic interference.

For RTP-Like QoS, the fixed wireless resource is reserved during all the four time periods. So it receives a fixed time for removing data from the buffer for transmission. When interference increases, the MAC layer uses more time for sending the same data packet, and hence the number of packets that can be fetched for transmission reduces within the fixed time period. That is why we observe in Figure 13 that the data buffer fetching speed of RTP-Like QoS reduces to 3.9∼4Kbps in the fourth time period. For best-effort communication, it gets less resources for data communication when more resources are given to the adaptive QoS in the presence of interference. This is why the data fetch speed goes below 4Kbps in the third and fourth time periods.

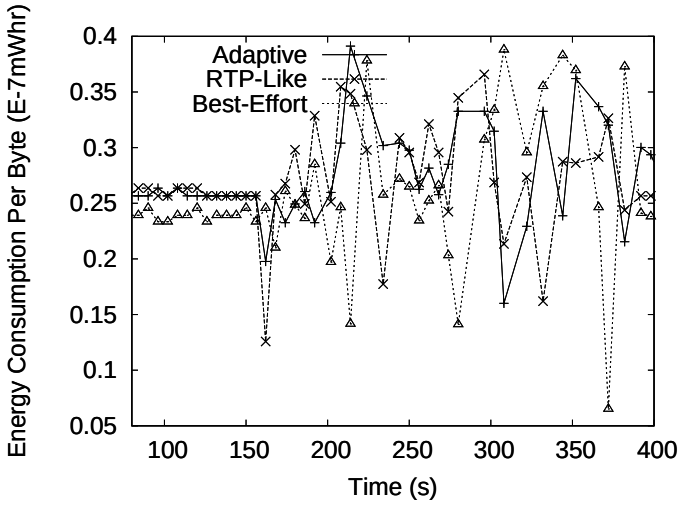


Fig. 14. Energy Consumption Per Byte with Different Interference

Figure 14 illustrates the energy consumption per delivered data byte. We can see that in the first time period, adaptive QoS and RTP-Like QoS have slightly more energy consumption compared to that of best-effort. This is because adaptive QoS and RTP-Like QoS have slightly higher polling overhead compared to that of best-effort. The second observation is that when there is interference in time period 2∼4,

there is no obvious difference for the energy consumption of adaptive QoS, RTP-Like QoS and best-effort. They have similar energy efficiency when interference is present.

Therefore, compared with the RTP-Like and Best-Effort methods, the adaptive QoS design achieves much better bandwidth delivery ratio as observed in Figure 11 and data fetching speed as observed in Figure 13, but pays similar time delay as observed in Figure 12 and energy efficiency as observed in Figure 14. Since the bandwidth delivery ratio is more interesting, we focus on the bandwidth comparison in the rest of performance comparison.

8.3 Performance with Different Polling Lengths

As we discussed in Section 4, neither long polls nor short polls are efficient designs and a standard polling length PL is used, which denotes the standard packet size that is desired for each polling. In the previous experiment, PL is set to 20. In this experiment, we also evaluate the performance when PL is 10 and 50. The performance is observed in three time periods. In the first time period, 0∼165s, the noise node is turned off. In the second time period, 165∼285s, the noise node is turned on and it sends out a noise packet every 30ms. In the third time period, 285∼400s, the noise node increases the noise level by sending out a noise packet every 20ms.

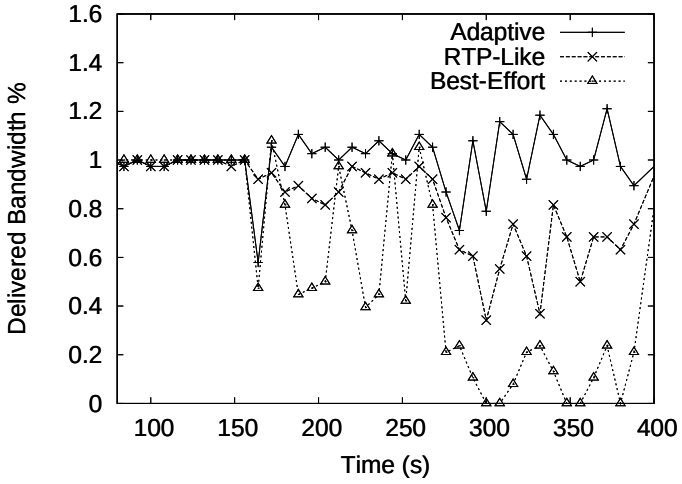


Fig. 15. Performance When Polling Length = 10

One interesting observation from Figures 15∼17 is that when the polling length increases, adaptive QoS and best-effort have more performance variation. For instance, the maximum difference of consecutive bandwidth delivery ratios for adaptive QoS is 35% points in Figure 15 when PL is 10, and then increases to 80% points in Figure 16 when PL is 20, and then increases to 120% points in Figure 17 when PL is 50. For best-effort communication, the maximum difference of consecutive bandwidth delivery ratios is 60% points in Figure 15, and then increases to 75% points in Figure 16, and then increases to 100% points in Figure 17. Adaptive QoS

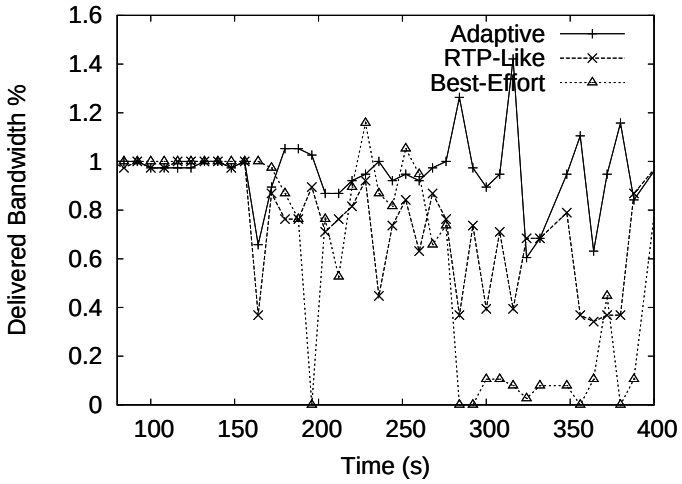


Fig. 16. Performance When Polling Length = 20

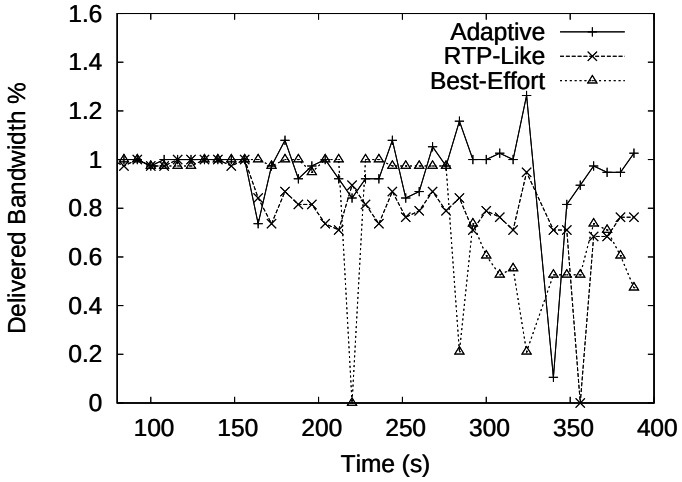


Fig. 17. Performance When Polling Length = 50

achieves less stable performance when PL increases, because the loss of a polling packet leads to the absence of transmission of all polled packets. The greater the polling length, the more the data packets that are prevented from being sent out, and the more the bandwidth delivery ratio variation is observed at the aggregator. Also, when more data packets are lost due to the loss of a polling packet, the measured effective bandwidth decreases more, and more resources go to the adaptive QoS in the next time interval. So best-effort gets less resources in the next time interval, since the total resources are fixed. This is why best-effort also illustrates increased performance variation when the polling length increases.

Another interesting observation in Figures 15∼17 is that when the polling length increases, the polling overhead decreases and, hence, more wireless resource is given

to best-effort communication. For example, in Figure 15, when the polling length is 10, best-effort can only achieve about 50% bandwidth delivery ratio in the second time period and 15% in the third time period. However, as shown in Figure 17, when the polling length increases to 50, the bandwidth delivery ratio for best-effort increases close to 100% in the second time period and 55% in the third time period. Therefore, increasing the polling length helps improve the whole system performance. Also considering that too large a polling length also leads to decreased performance stability, we use polling length 20 in the rest of the performance evaluations, which is proven, through experiments, to have the best tradeoff between performance stability and communication overhead in our evaluation. Please be aware that there is no single "best tradeoff", and it will vary with user preferences and detailed applications.

8.4 Performance with Different Effective Bandwidth Measurements

As analyzed in Formula 2, a decay factor δ is designed to control the moving average speed in effective bandwidth measurement. In previous experiments, δ is set to 0.2. In this experiment, we also evaluate the performance when δ is 0,05 and 0.8. In the first time period, 0∼165s, the noise node is turned off, and in the second time period, 165∼200s, the noise node is turned on and it sends out a noise packet every 30ms. The system performance is observed when interference is added at the beginning of the second time period.

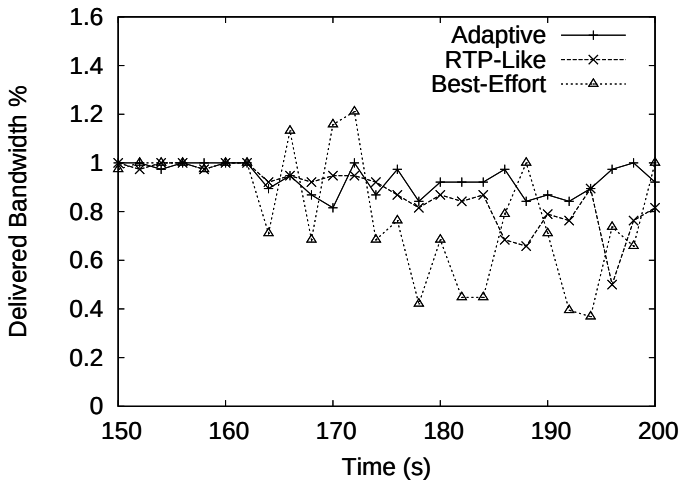


Fig. 18. Performance When Decay Factor = 0.05

From Figures 18∼20, we observe that when a greater decay factor is used, the system adapts to interference more quickly. As shown in Figure 18, when δ is 0.05, it takes about 30s for the system performance to be stabilized in the presence of interference. However, as shown in Figure 20, when δ is 0.8, it only takes 6s for the performance to become stabilized, which is an 80% reduction in system response time. The reason is that when δ increases, more weight is given to the most recently measured effective bandwidth.

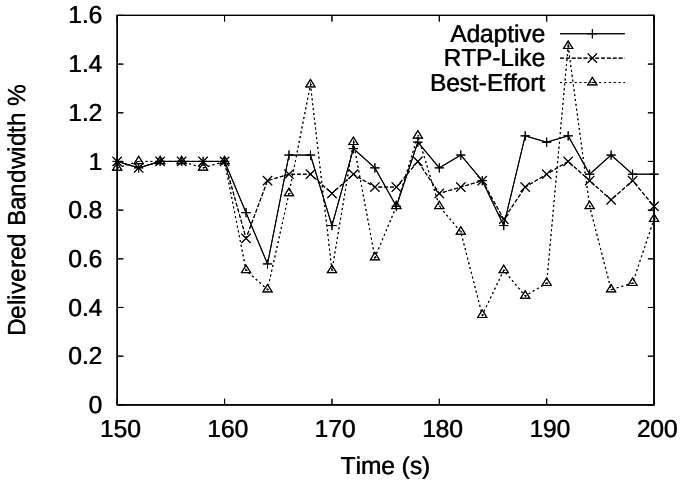


Fig. 19. Performance When Decay Factor = 0.2

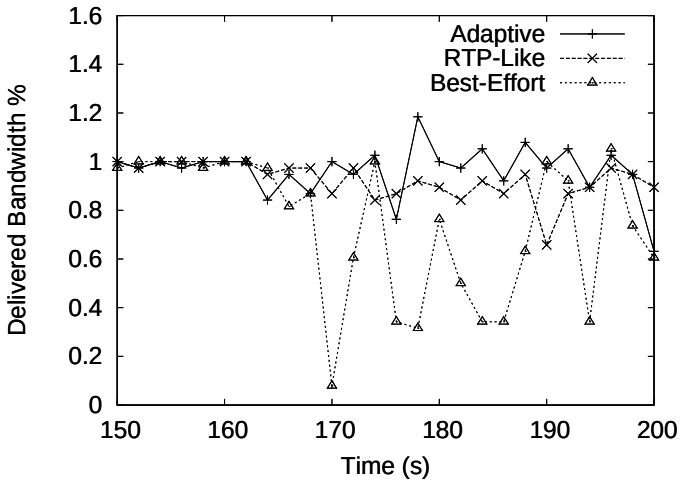


Fig. 20. Performance When Decay Factor = 0.8

On the other hand, even though a greater δ value leads to reduced system response time, a smaller δ value helps achieve smoother system performance. For example, the bandwidth delivery ratio varies about 70% points when δ is 0.8 in Figure 20. However, when δ decreases to 0.05 in Figure 18, the performance variation reduces to 35% points, half of what we observe in Figure 20. So we use the decay factor 0.2 in the rest of the performance evaluations, which is proven, through experiments, to have the best tradeoff between system response time and performance stability in our evaluation. Also, please be aware that there is no single "best tradeoff", and it will vary with user preferences and detailed applications.

8.5 Performance with Different Data Buffer Sizes

In this section, we measure the impact of data buffer size on BodyQoS performance. In previous experiments, the data buffer size is configured to be unlimited. In this experiment, we evaluate the system performance when the data buffer size is limited. In the experiment, the performance is evaluated with data buffer sizes of 25, 50 and 100. The performance is also observed in three time periods. In the first time period, 0∼165s, the noise node is turned off. In the second time period, 165∼285s, the noise node is turned on and it sends out a noise packet every 30ms. In the third time period, 285∼400s, the noise node increases the noise level by sending out a noise packet every 20ms.

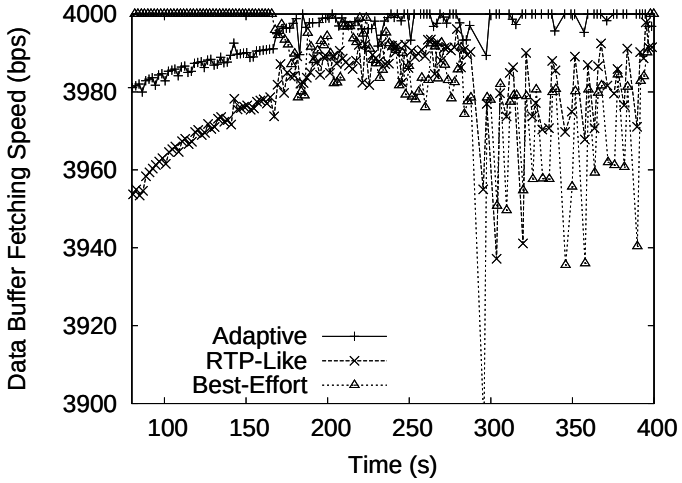


Fig. 21. Performance When Buffer Length = 25 Packets

As observed in Figures 21∼23, when interference is present, RTP-Like QoS and best-effort communication always suffer buffer overflow, no matter whether the data buffer size is set at 25, 50 or 100. This is because neither RTP-Like QoS nor best-effort communication adapts resources for dealing with interference. When interference happens, RTP-Like QoS suffers more MAC layer backoffs and retransmissions for each data packet. So it does not have enough time for delivering all sampled data packets, since the reserved channel control time is fixed. For best-effort communication, more time is given to adaptive QoS during interference, and hence less time is given to it for transmitting the sampled data packets.

However, for adaptive QoS, adaptive channel control time is assigned according to interference. So adaptive QoS almost always maintains the requested 4Kbps data buffer fetching speed, even when interference is present. We are also aware that when the buffer size is too small to hold data packets that are generated within 1 second (according to 4Kbps data sampling speed), all the three services suffer buffer overflow. When there is no enough data in the buffer, the data fetching and transmission speed may be below the requested 4Kbps. This is why in Figure 21, when the data buffer size is 25, adaptive QoS, RTP-Like QoS and best-effort all

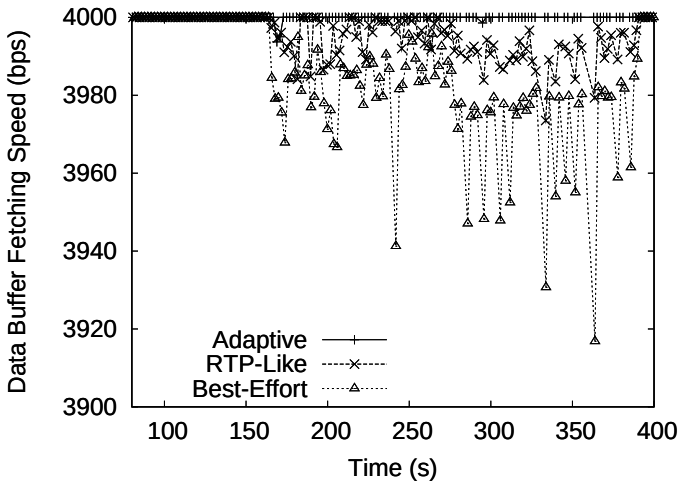


Fig. 22. Performance When Buffer Length = 50 Packets

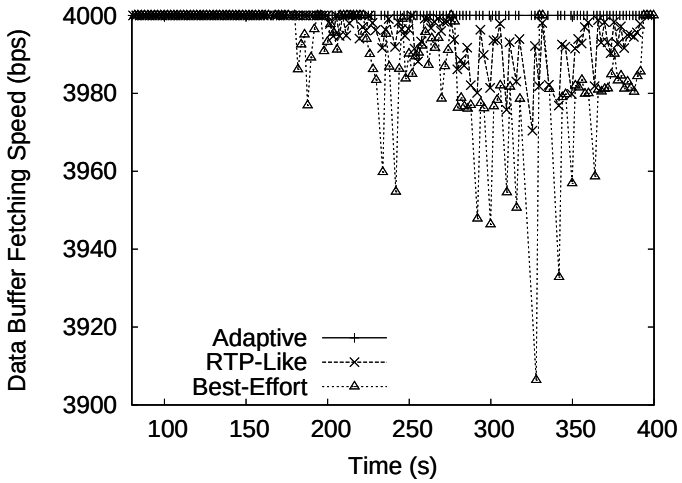


Fig. 23. Performance When Buffer Length = 100 Packets

suffer from the insufficient data fetching and transmission speed (<4Kbps) due to lack of data in the buffer. On the other hand, we also observe that this "false" buffer overflow is eliminated for adaptive QoS when the data buffer size is 50+. In real applications, it is reasonable to have a data buffer size of 50+ when the hardware is actually capable of generating data at the speed of 4Kbps.

8.6 Performance with Different Bandwidth Requirements

In previous experiments, all data streams request 4Kbps bandwidth. In this experiment, we evaluate system performance under different bandwidth requirements. Similar to the previous experiments, the performance is observed in three time periods. In the first time period, 0∼165s, the noise node is turned off. In the second

time period, 165∼285s, the noise node is turned on and it sends out a noise packet every 30ms. In the third time period, 285∼400s, the noise node increases the noise level by sending out a noise packet every 20ms.

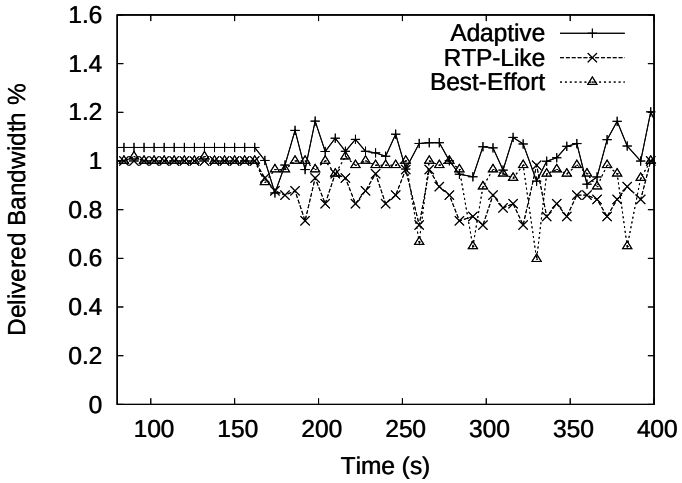


Fig. 24. Performance When Requiring 2Kbps Bandwidth

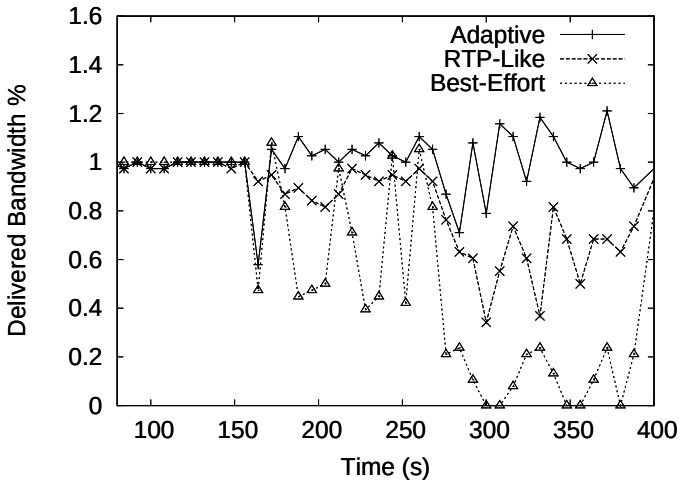


Fig. 25. Performance When Requiring 4Kbps Bandwidth

Comparing Figures 24 and 25, we observe that when the requested bandwidth increases, adaptive QoS still maintains the requested bandwidth, statistically speaking. The reason is that the adaptive resource scheduling provides adaptive QoS more resource for retransmitting the lost packets in the presence of interference. However, RTP-Like QoS and best-effort communication suffer decreased bandwidth delivery ratio when interference happens. For example, in Figure 25, the bandwidth

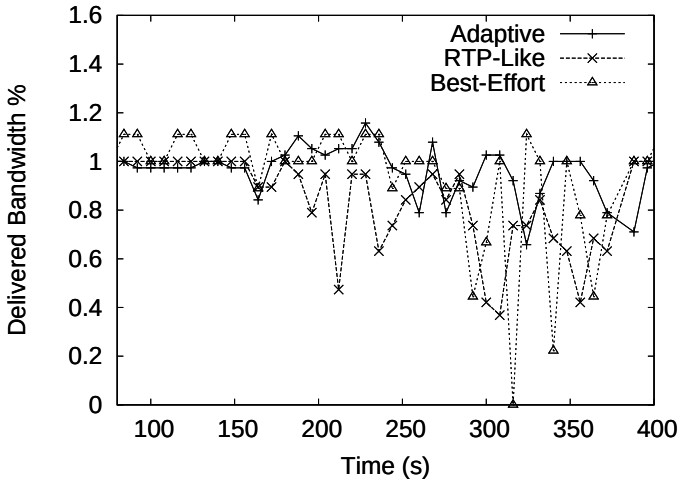


Fig. 26. Performance When Requiring 4/2/1Kbps Bandwidth

delivery ratio of RTP-Like QoS reduces to about 90% in the second time period, and then reduces further to about 60% when interference increases in the third time period. For best-effort communication, the bandwidth delivery ratio reduces to about 50% in the second time period, and then below 30% in the third time period. RTP-Like QoS has reduced performance because it requests a fixed wireless resource and hence has no extra resources for retransmitting lost packets. Best-effort communication suffers reduced performance because it gets reduced wireless resources when more resources are given to adaptive QoS for dealing with interference.

Figure 26 illustrates a more likely setting, in which an EKG stream requests 4Kbps bandwidth, while the location stream requests 2Kbps bandwidth and the temperature stream requests 1Kbps bandwidth. As shown in the figure, when there is no interference in the first time period or when there is light interference in the second time period, EKG/location/temperature streams all achieve the requested bandwidth, even though the performance is not perfectly smooth. On the other hand, when interference increases in the third time period, adaptive QoS still statistically keeps the bandwidth promise, because it benefits from the adaptive resource scheduling. But RTP-Like QoS and best-effort suffer increased packet loss and can not meet the bandwidth requirements, because they lack adaptive techniques.

8.7 Performance when Adaptive QoS and Best-Effort have the Same Priority

Since Adaptive QoS is configured to have a higher priority than Best-Effort communication, one question may arise: what will be the comparative performance if Best-Effort is not configured to have a lower priority than Adaptive QoS and hence does not need to just use the space bandwidth left over by Adaptive QoS? In order to measure performance in such a scenario, we compare performance of an Adaptive QoS stream with a Best-Effort stream in the following scenarios: (1)In scenarios 1, the Adaptive QoS stream that we plan to study sends 4kbps data to aggregator, together with two other 4kbps Best-Effort streams, in the same way as we configured in subsection 8.2; (2) In scenario 2, the Best-Effort stream that

we plan to study sends 4kbps data to aggregator, together with two other 4kbps Best-Effort streams. All other experimental configurations are the same for these two scenarios for fair comparison: the same sensor motes are used and deployed at the same location and time.

We also introduce noise to check how the performance varies with time. In the first time period, 0s∼135s, there is no explicit noise. In the second time period, 135s∼225s, a MicaZ node is turned on to generate noise signals. It sends out a noise packet every 30ms. In the third time period, 225s∼315s, the noise node increases its noise level by sending out a packet every 25ms. In the fourth time period, 315∼400s, the noise node increases its noise level again by sending out a packet every 20ms.

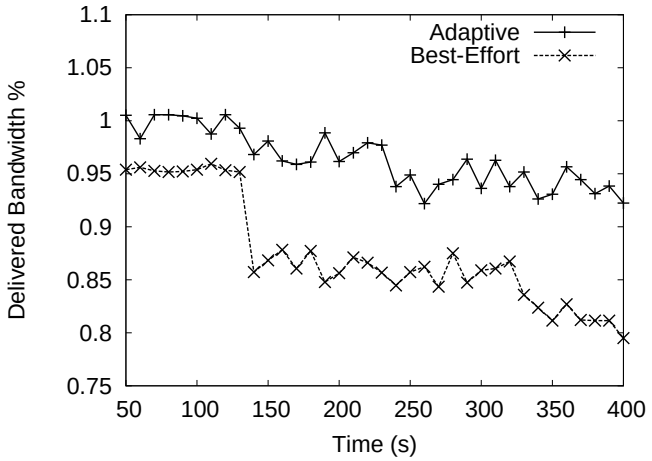


Fig. 27. Performance When Adaptive QoS and Best-Effort have the Same Priority

As shown in Figure 27, when the background noise node is not turned on during the 0s∼135s time period, Adaptive QoS can deliver the requested bandwidth while Best-Effort method can not. This is because adaptive radio resources is scheduled in the Adaptive QoS method to deal with the interference from the other two running Best-Effort data streams, while such adaptive resource scheduling is not available in the Best-Effort method. From Figure 27, we also observe that the increased noise level from the noise node has much great impact on the communication performance of the Best-Effort method than that of the Adaptive QoS method. This is also because efforts are made in Adaptive QoS to schedule more radio resource to retransmit lost packets when congested, but such efforts are not present in Best-Effort method. We are also aware that when the noise is very high, even the Adaptive QoS method can not guarantee the requested bandwidth. In such a case, the "RemoveQoS" interface that we have discussed in section 5.3 can be used to notify upper layer users that the requested QoS can no longer be guaranteed and hence the QoS requirement should be reduced and submitted again.

9. RELATED WORK

QoS research has been extensive in the Internet and in general wireless ad hoc networks. A lot of research, such as [Zhu and Cao 2005] [Aad and Castelluccia 2001] [Garg et al. 2003] [G. S. Ahn and A. Campbell and A. Veres and L. H. Sun 2002], has been performed on how QoS can be used to manage and reserve communication resources. However, those QoS solutions are designed for much more powerful devices, such as Internet routers and wireless access points, which are often line-powered. Most of these solutions do not apply to BSN applications, which use resource constrained sensor devices powered by small form factor batteries (e.g., AA, coin, or film).

There are several sensor network protocols that provide QoS features. For example, a VMAC like solution [Polastre et al. 2005] is developed at Berkeley, in which priorities and reliability can be configured for a set of underlying MAC protocols. However, bandwidth specification and reservation are not available in this protocol and hence users can not submit their bandwidth requirements to it for reserving specified wireless bandwidth. This is also true for Bluetooth [IEEE 802.15.4 2003] platform and a large number of other sensor network protocols like [Kim et al. 2007] [Hull et al. 2004] [Rangwala et al. 2006] [Ee and Bajcsy 2004] [Werner-Allen et al. 2006] [Sankarasubramaniam et al. 2003] [Stann and Heidemann 2003] [Paek and Govindan 2007] [He et al. 2003] [Liu et al. 2005] [Biswas and Morris 2005] [Zhao and Tong 2003] [Wan et al. 2002]. These protocols provide some QoS features but can not be called QoS systems. For example, different time delay features are considered in a geographic routing design in [He et al. 2003] to minimize the end-to-end time delay, but it does not provide a QoS system so that end-to-end communication can request and reserve resources for achieving the specified time delay. For body networks, real system experiences can be found at [Harvard CodeBlue] [MIThril], and a number of initial results are available in the BSN proceedings [BSN Proceedings]. The SATIRE paper [Ganti et al. 2006] describes a wearable personal monitoring service that is transparently embedded in user garments. However, none of these systems support QoS.

10. CONCLUSIONS AND FUTURE WORK

This paper presents the design and implementation of a QoS system for body sensor networks, called BodyQoS. Different from conventional QoS designs, BodyQoS addresses three unique challenges brought by BSN applications. First, BodyQoS adopts an asymmetric architecture, in which most processing is done at the resourceful aggregator while little is done at the resource limited sensor nodes. Second, a virtual MAC is developed in BodyQoS to make it radio-agnostic, so that it can control and schedule wireless resources without knowledge of the implementation details of the underlying MAC protocol. This approach supports a wide variety of different MACs, including CSMA, TDMA, and hybrid approaches. Third, BodyQoS adopts an adaptive resource scheduling strategy during times of channel impairment, either due to RF interference or fading effects. This makes it possible to provide statistical bandwidth guarantees as well as reliable data communication in BSN. BodyQos has been implemented in NesC on top of TinyOS, and evaluated in a MicaZ testbed. Our performance evaluation demonstrates that BodyQoS

achieves greatly improved performance as compared with conventional solutions, with minimal overhead.

REFERENCES

Aad, I. and Castelluccia, C. 2001. Differentiation Mechanisms for IEEE 802.11. In *IEEE INFOCOM*.

Biswas, S. and Morris, R. 2005. ExOR: opportunistic multihop routing for wireless networks. In *ACM SIGCOMM*.

BSN Proceedings. BSN Proceedings. http://www.bsn-web.org.

ChipconCC1000. Chipcon CC1000 Low Power Radio Transceiver. http://www.chipcon.com.

ChipconCC2420. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. http://www.chipcon.com.

CROSSBOW. XBOW Mote Specifications. http://www.xbow.com.

Ee, C. T. and Bajcsy, R. 2004. Congestion control and fairness for many-to-one routing in sensor networks. In *ACM SenSys*.

G. S. Ahn and A. Campbell and A. Veres and L. H. Sun. 2002. Supporting service differentiation for real-time and best effort traffic in stateless wireless ad hoc networks (swan). In *IEEE Transactions on Mobile Computing*.

Ganti, R. K., Jayachandran, P., Abdelzaher, T. F., and Stankovic, J. A. 2006. SATIRE: A Software Architecture for Smart AtTIRE. In *ACM MobiSys*.

Garg, P., Doshi, R., Greene, R., Baker, M., Malek, M., and Cheng, X. 2003. Using IEEE 802.11e MAC for QoS over Wireless. In *IEEE IPCCC*.

Harvard CodeBlue. CodeBlue: Sensor Networks for Medical Care. http://www.eecs.harvard.edu/ mdw/proj/codeblue/.

He, T., Stankovic, J. A., Lu, C., and Abdelzaher, T. F. 2003. SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks. In *IEEE ICDCS*.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. 2000. System Architecture Directions for Networked Sensors. In *ASPLOS-IX*.

Hull, B., Jamieson, K., and Balakrishnan, H. 2004. Mitigating congestion in wireless sensor networks. In *ACM SenSys*.

IEEE 802.11 1999. IEEE 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. ANSI/IEEE Std. 802.11.

IEEE 802.15.1 2002. IEEE 802.15.1: Wireless medium access control (MAC)and physical layer (PHY) specifications for wireless personal area networks (WPANs). http://www.ieee802.org/15/pub/TG1.html.

IEEE 802.15.4 2003. IEEE 802.15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). http://www.ieee802.org/15/pub/TG4.html.

Johansson, A. J. 2002. Wave-Propagation from medical implants-influence of body shape on radiation pattern. In *2nd Joint EMBS/BMES Conference*.

Kidd, C. D., Orr, R. J., Abowd, G. D., Atkeson, C. G., Essa, I. A., Macintyre, D., Mynatt, E., Starner, T. E., and Newstetter, W. 1999. The aware home: A Living Laboratory for Ubiquitous Computing Research. In *CoBuild*.

Kim, S., Fonseca, R., Dutta, P., Tavakoli, A., Culler, D., Levis, P., Shenker, S., and Stoica, I. 2007. Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In *ACM SenSyS*.

Liu, Y., Elhanany, I., and Qi, H. 2005. An energy-efficient QoS-aware media access control protocol for wireless sensor networks. In *IEEE MASS*.

MIThril. MIThril. http://www.media.mit.edu/wearables/mithril/.

Natarajan, A., Motani, M., de Silva, B., Yap, K., and Chua, K. C. 2007. Investigating Network Architectures for Body Sensor Networks. In *HealthNet 2007*.

Paek, J. and Govindan, R. 2007. Rate-Controlled Reliable Transport for Wireless Sensor Networks. In *ACM SenSys*.

Polastre, J., Hill, J., and Culler, D. 2004. Versatile Low Power Median Access for Wireless Sensor Networks. In *ACM SenSys*.

Polastre, J., Hui, J., Levis, P., Zhao, J., Culler, D., Shenker, S., and Stoica, I. 2005. A unifying link abstraction for wireless sensor networks. In *ACM SenSys*.

Rangwala, S., Gummadi, R., Govindan, R., and Psounis, K. 2006. Interference-aware fair rate control in wireless sensor networks. In *ACM SIGCOMM*.

Roelens, L., den Bulcke, V., Joseph, S., Vermeeren, W., and G. Martens, L. 2006. Path loss model for wireless narrowband communication above flat phantom. *IEEE Electronics Leters*.

RSVP. Resource ReSerVation Protocol. http://www.ietf.org/rfc/rfc2205.txt.

Sankarasubramaniam, Y., Ö. B. Akan, and Akyildiz, I. F. 2003. Esrt: Event-to-sink reliable transport in wireless sensor networks. In *ACM MobiHoc*.

Shah, R. C., Nachman, L., and Wan, C.-Y. On the performance of Bluetooth and IEEE 802.15.4 radios in a body area network. Third International Conference on Body Area Networks (BodyNets'08).

Stann, F. and Heidemann, J. 2003. RMST: Reliable data transport in sensor networks. In *The First International Workshop on Sensor Net Protocols and Applications*.

UFL Smart House. UFL Smart House. http://www.rerc.ufl.edu.

UVA Smart House. UVA Smart House. http://www.marc.med.virginia.edu.

Wan, C., Eisenman, S., and Campbell, A. 2003. CODA: Congestion Detection and Avoidance in Sensor Networks. In *ACM SenSys*.

Wan, C.-Y., Campbell, A. T., and Krishnamurthy, L. 2002. PSFQ: a reliable transport protocol for wireless sensor networks. In *ACM WSNA*.

Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M. 2006. Fidelity and yield in a volcano monitoring sensor network. In *ACM OSDI*.

Zhao, Q. and Tong, L. 2003. QoS Specific Medium Access Control forWireless Sensor Networks with Fading. In *SPSC*.

Zhou, G., Lu, J., Wan, C.-Y., Yarvis, M. D., and Stankovic, J. A. 2008. BodyQoS: Adaptive and Radio-Agnostic QoS for Body Sensor Networks. In *IEEE INFOCOM*.

Zhou, G., Stankovic, J. A., and Son, S. F. 2006. Crowded Spectrum in Wireless Sensor Networks. In *IEEE EmNets*.

Zhu, H. and Cao, G. 2005. On Supporting Power-efficient Streaming Applications in Wireless Environments. In *IEEE Transactions on Mobile Computing*.