

A MIDDLEWARE FOR SERVICE ADAPTATION IN DIFFERENTIATED 802.11 WIRELESS NETWORKS

Chui Sian Ong*, Yuan Xue*, Klara Nahrstedt*

Department of Computer Science
University of Illinois At Urbana Champaign
Email: {chuiong, xue, klara}@cs.uiuc.edu

Abstract—Currently there is a growing interest to support Quality of Service(QoS) for multimedia application delivery over bandwidth limited and varying wireless network. Existing work either only focus at network level QoS via MAC scheduling or provide application level QoS via per-application admission control. Due to the lack of application information, the former may not meet the specific QoS requirements of different applications, while the latter results in poor scalability. In light of the limitations of existing approaches, we present a novel QoS architecture, which incorporates a two-level design. At the network level, the cross-layer scheduling and queue management provide *service differentiation* for wireless network. At the middleware level, a monitor maintains a global view of the application performance over the whole network, and an adaptor coordinates *service adaptation* to achieve the required QoS for multimedia applications. In this paper, we present the whole QoS architecture, with a focus on the middleware design, which adopts a control-based adaptation model. To validate our design, applications with different QoS requirements are built on top of the middleware. Experimental results show that, the specific QoS levels for multimedia applications can be successfully achieved in IEEE 802.11-based wireless environment.

I. INTRODUCTION

A key vision of next-generation wireless systems is to support new emerging distributed multimedia services such as voice-over-IP and video-on-demand anytime anywhere. Multimedia applications are very sensitive to the QoS provided by the communications environment and the support of multimedia services over 802.11 wireless networks presents a number of technical challenges. First, the time-varying error characteristics and time-varying channel capacity at the physical layer makes it difficult, if not impossible to provide hard Quality-of-Service (QoS) guarantees. Second, user mobility may induce signal fading that in turn can trigger rapid degradation in the delivered service quality. Third, wireless networks are typically bandwidth constrained in comparison with their traditional wireline peers.

Current work on developing QoS support for wireless networks are mostly focused at the MAC layer. For example, the work of [1], [2] have focused on differentiated MAC layer scheduling under IEEE 802.11 DCF. And the work of [3] has studied the fair scheduling in wireless networks. These MAC layer solutions can only provide certain QoS at the network level. They lack the architectural flexibility to accommodate end-to-end application-specific QoS requirements in time-varying wireless networking environments.

*This work is supported by Motorola(University Research Program), MURI ONR and Vodaphone fellowship.

Some works provide an end-to-end QoS support by adapting traditional QoS models for wireless network. For example, based on the IntServ model, INSIGNIA [4] utilizes an in-band resource signaling protocol to reserve per-flow resources. Alternatively, SWAN [5] follows the absolute DiffServ QoS model by defining two service classes: *real-time* and *best-effort* traffic. Both these approaches use per-application admission control, thus they have poor scalability and may suffer false admission due to the imprecise resource estimation in the highly dynamic wireless environments.

In light of the limitations of existing approaches, it is clear that a complete and efficient QoS architecture for wireless network would require (1) lightweight and scalable QoS support with minimum or no negotiation overheads; (2) agile and responsive QoS management tools to monitor network resources and make appropriate adaptation decisions for applications so that their individual QoS requirements can be satisfied. Hence, we propose a QoS architecture with a two-level design. At the network level, the cross-layer scheduling and queue management provide *service differentiation* in the wireless network to provide scalable QoS support. At the middleware level, a *monitor* maintains a view of the application performance, and an *adaptor* makes application-specific *service adaptation* decisions to achieve the desired QoS requirements for multimedia applications.

In [6], *proportional differentiation model* can be used to provide service differentiation at the network level in IEEE 802.11 wireless network. The focus of this paper is on the middleware design. The middleware understands both the lower level network resource dynamics and higher level application specific QoS requirements. It comprises of monitoring tools to detect QoS violations. Upon detection, the middleware is responsible for adjusting the service class of each application, to ensure non-conflicting QoS adaptations in the wireless environment. To achieve this goal, the middleware is build on a control-based adaptation framework.

Additionally, we note that recent research has made considerable progress for cross-layer adaptation in resource constraint environment [7] and control-based adaptation [8] for applications with multiple QoS parameters. However, in these models, the QoS adaptations are performed for single localized nodes in a network, typically to save power and CPU cycles. For a wireless environment, in order for the limited bandwidth to be used efficiently, we require a different model for multiple nodes

in the wireless network to adapt in a coordinated manner.

Furthermore, adaptations at the application level [9] are focused on higher level application specific semantics, such as the adaptation time-scale and adaptation policies, and are often optimized for individual application's own performance. Without the recognition of global system states, such models tend to make conflicting adaptation decisions and yield unstable system. On the other hand, adaptations at the network level [10], [11] are inadequate to support diverse and complicated application QoS requirements.

This paper is organized as follows: Section II gives an overview of our QoS architecture. Section III presents the design details of the middleware component. We present our implementations and results in Section IV and conclude the paper in Section V.

II. QoS ARCHITECTURE OVERVIEW

Our QoS architecture is presented in Fig. 1. The architecture operates from the MAC layer up to 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.

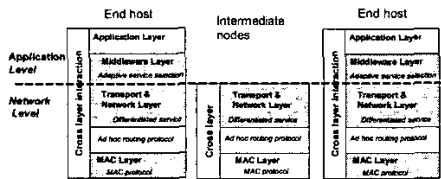


Fig. 1. QoS Architecture: Overview

A detailed diagram of our QoS architecture, which shows the key components of this architecture, is illustrated in Fig. 2. In order to provide QoS support in wireless networking environments, these components interact in the following way.

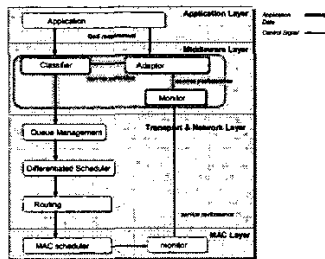


Fig. 2. QoS architecture: Details

1) *At application level in the end hosts,*

- The application notifies the *Adaptor* 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 *Adaptor* 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 *Adaptor* 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 *Adaptor* 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 well balances 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 our previous work [6], A service differentiation mechanism at the network level is presented. In this paper, we focus on the middleware design and implementation.

III. MIDDLEWARE FRAMEWORK FOR QoS-AWARE ADAPTATION

A. Overview

In this section, we describe the adaptation services to be provided by the middleware framework. The adaptation services have three major objectives: First, they 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.

Second, the adaptation services strengthen the adaptation awareness and effectiveness within flexible applications by making decisions to control their adaptive behavior. The adaptation

awareness includes when, how and to what extent adaptation is carried out in the applications.

Third, the middleware achieves coordinated adaptation as each end host continually updates the service class for each application until a stable value is achieved in the distributed environment. Stability is achieved when the application can gain no further improvement in the QoS obtained by further increasing its service class. This occurs either when the required QoS has been met by the available resources or when the resources usage is already saturated and hence any service priority increment does not yield further user improvement.

The design of our middleware adaptation framework is based on a task control model [8] as shown in Fig. 3(a). Within the middleware control framework, the *Adaptation Task* and the *Observation Task* are represented in two respective components: the *Adaptor* and the *Monitor*. And the *Target System* is the differentiated network, represented by the *Classifier* in the middleware layer, as shown in Fig. 3(b). The *Control Action* is the service class selection; and the *Task States* are the end-to-end performance of the multimedia application.

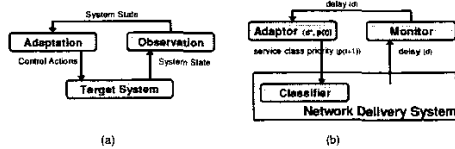


Fig. 3. Middleware Control Framework

B. Adaptor Design

We consider delay as an example QoS parameter, the control framework in the middleware detect the delay changes of the application and determines the necessary service class changes. The middleware in turn passes this information to the classifier which marks the packets with their service classes so that the cross-layer scheduling at the network and MAC layer enforces corresponding QoS changes.

In particular, the *Adaptor* 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.

Now we present the detailed example of adaptation design in the setting of delay differentiation service so that bounded delay can be achieved for audio service. Let us denote the delay bound specified by the application as d^* and the observed delay for this application via monitor as d . The service class of the audio application is represented by its service priority p , which will be used as a *service differentiation parameter* for the underlying delay differentiation scheduling. Then the control-based adaptation can be modelled as follows.

$$\frac{dp(t)}{dt} = \dot{p}(t) = f(p(t), d(t), d^*) \quad (1)$$

In discrete time form,

$$p(t+1) = p(t) + f(p(t), d(t), d^*) \quad (2)$$

In our implementation, the adaptation decision is made based on a linear relationship of d and d^* and followed by a linear priority adjustment. Thus function $f(\cdot)$ is given as,

$$f(p(t), d(t), d^*) = (\alpha_2 - 1) \cdot p(t) + \beta_2 \text{ if } \alpha_1 \cdot d + \beta_1 > d^* \quad (4)$$

where α_1, β_1 are parameters that determine the conditions where QoS violation occurs when the application either cannot meet its QoS requirement or has over-utilized the resource. Parameters $\alpha_2 \geq 1, \beta_2 \geq 0$, determine how the service class is adapted. Different parameters are evaluated and compared in the experiment.

Alternatively, we express the adaptation decision in the form of if-then rules.

$$\text{if } \alpha_1 \cdot d + \beta_1 > d^* \quad (5) \\ \text{then } p(t+1) = \alpha_2 \cdot p(t) + \beta_2 \quad (6)$$

Application may gain no further decrement in delay obtained by further increasing the service class when the available scheduling resource is already saturated. So we need to bound the service priority in this case.

$$\text{if } d > d^* \text{ and } p(t) > p_{max} \text{ then } p(t+1) = p(t) \quad (7)$$

C. Monitor Design

The main goal of the *Monitor* is to measure, for each application, the average round trip delay incurred to deliver data packets in the wireless network. In particular, the measured delay values are used by the *Adaptor* to update service priority for the applications in order to maintain the desired QoS values for each application.

We take into account the need to minimize “delay spikes” in network applications. Therefore we take an average of N , (d_1, d_2, \dots, d_N), number of round-trip delay measurements and use, $d_{avg} = \frac{1}{N} \sum_{i=1}^N d_i$, as the *control value* to be used by the *adaptor* to update the service priority appropriately. The value of N (number of delay values to average over) should be varied depending on the network conditions and QoS requirements in order to achieve a stable *control value*. A stable control value is achieved when increasing service priority does not provide further improvement to the received QoS or the require QoS is achieved.

IV. IMPLEMENTATION AND RESULTS

A. Implementation

We implemented a IEEE 802.11-based wireless adhoc testbed to incorporate the service adaptation design we have described. The setup is shown in Figure 4. The *AudioSender* is an audio application that capture audio from a microphone and sends

out the audio packet in real time at a rate of 8KB/s. The *AudioSender* has a QoS requirement for a minimum delay bound. The *UdpSender* is a data application that sends out data at rates up to 8Mbps/s. The *UdpSender* has elastic QoS requirement and does not have a strict minimum delay bound. The EXACT router [12] forwards each packet to the next hop in the delivery route.

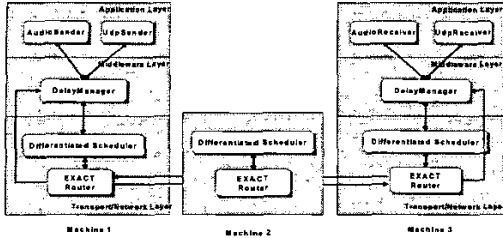


Fig. 4. The adhoc testbed implementation over IEEE 802.11b wireless network

The *DelayManager* has a view of all applications that are running on the node. For each application, the *DelayManager* maintains a *DelayObject* that keep tracks of (1)sequence number, (2)sent time and (3)acknowledgement received time. The *DelayManager* interacts with its peer at the destination to obtain these values. The *DelayManager* contains a *DelayWindowObject* to maintain the history of per packet round trip delay. The average round-trip delay for each application is calculated as an average value over a specified N number of values.

Based on the chosen priority adaptation policy and the *AudioSender* application's QoS specifications, the *DelayManager* determines the new priority value and updates the priority marking for each application's data packet accordingly. The marked packets are forwarded to the *Differentiated Scheduler* which selects the appropriate packets to transmits.

B. Experimental Setup

We used the equipment in Table I for our experiments. We have three computing nodes in our wireless ad-hoc network (Figure 4). Each node uses the EXACT router to forward packets to its destination. In order to validate our adaptive QoS architecture, we use the data application (*UdpSender*) to saturate the wireless network.

We show that as the network resource availability changes, our adaptive framework is able to bound the audio packet delays to a level that maintained an acceptable QoS for the audio application. In our experiment, the *AudioSender* and *UdpSender* are executed on machine 1, while the *AudioReceiver* and *UdpReceiver* execute on machine 3. Machine 2 is used as an intermediate router between machine 1 and 3.

Name	CPU	Memory	OS
Machine 1	PIV 1.8GHz	256MB	Windows XP Pro
Machine 2	PIV 366MHz	128MB	Windows XP Pro
Machine 3	PI 266MHz	256MB	Windows 2000

TABLE I
EQUIPMENT USED

C. Results

In this section, we evaluate our implementation to see how adaptation mechanism performed. In particular, we varied (i)*Adaptation Parameters* ($\alpha_1, \beta_1, \alpha_2, \beta_2$), (ii)*Number of Delay Value to average over*, $N(N)$ and (iii)*Amount of Background Traffic(BT)* to understand how the *AudioSender*'s performance could be affected. For the experiment, the audio application is initially given a priority level of 3 or 5 while the data application is given a priority level of 1. A larger numerical value implies a higher service priority level. The delay bound of the audio application is specified as, $d^* = \text{mindelay}$.

Policy No.	α_1	β_1	α_2	β_2
1	1/3	0	2	0
2	1/2.5	0	2	0
3	1/3	0	7	0
4	1/7	0	2	0
5	1/7	0	7	0

TABLE II
ADAPTATION POLICIES AND PARAMETERS.

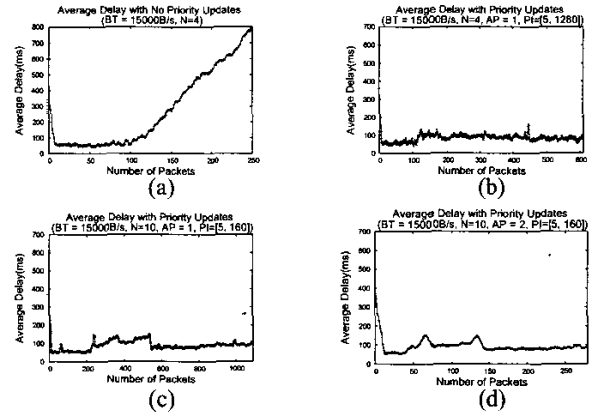


Fig. 5. Adaptation Results where Background Traffic = 15000 Bytes/s. (BT = Background Traffic, N = No. of Delay values to average over, AP = Adaptation Policy, PI = Priority Interval)

1) *Performance Effects of Adaptation*: In Figure 5(a), we execute the *UdpSender* after first 100 audio packets. From the graph, we see that the average delay increases quickly from 70ms to 800ms, when there is no priority adaptation. We compare this graph with Figure 5(b), where the *UdpSender* is also started approximately after first 100 audio packets. Using the priority adaptation policy 1, we observe that the average delay for the audio application was successfully bounded to < 150ms. The priority value was increased from 5 to 1280 based on adaptation policy 1.

Clearly, as the priority of the audio packets is increased, they are given higher service priority over the data packets. In particular, our middleware queue management design ensures that the audio packets are quickly forwarded to the head of the network queue in preference over the data packets. In this way, the delay becomes comparable to when there is only audio traffic in the network.

2) *Varying N*: In Figure 5(c), we increased N , from 4 to 10 packet delay values and observe high delays between 210th and 550th packets. The minimum average delay observed was increased from 40ms to 55ms. Hence, when $d^* = 55ms$ in our adaptation policy 1, the priority is not increased fast enough for the audio packets to be pushed to the front of the network queue. Instead, more lower priority data packets are being sent. The delay on the audio packets were cascaded until the observed delay, d , is $> 3 \times d^*$, before we see the falling edge of the plateau at around the 550th packet.

We lowered the control threshold in adaptation policy 1 from 3 to 2.5 in adaptation policy 2. The results in Figure 5(d) show that the observed delay has decreased as the priority were increased at a much faster rate.

3) *Varying Adaptation Parameters and Network Saturation*:: We saturated the network by sending data from the *UdpSender* at a rate of 1000000Bytes/s. Figure 6(a) shows a significant increase in the observed delay, and there was also significant degradation in the audio quality. However, once our adaptation mechanism was turned on, we were able to bound the delay on the audio packets to $< 150ms$ over time.

We further evaluate the performance of our implementation for different adaptation parameters based on Equation (5) and (6). The parameters for each adaptation policy is shown in Table II and the results are shown in Figure 6.

Comparing Figure 6(a) and Figure 6(b), we see that the adaptation reached a steady state much more quickly when we increase the priority levels at a faster rate. Figure 6(c), shows that if we set the adaptation threshold too high, ($d > 7d^*$), the adaptation becomes less smooth than in Figure 6(a) and (b). In Figure 6(d), we see that the negative effects observed in Figure 6(d) can be offset when we start to increase the priority levels at a faster rate.

Hence, depending on the criticality and the number of simultaneously running applications, it is possible to adjust the various adaptation parameters to achieve desired performance. In general a mid-range delay threshold will bound the initial increase in delay to a smaller value. Additionally, a fast priority increase rate will allow a more critical application to adapt to steady state more quickly.

V. CONCLUSION AND FUTURE WORK

Through our experiments, we have shown that our middleware adaptation framework can successfully improve QoS for applications appropriately. Using our integrated middleware approach, we are able to coordinate system-wide requirements and application specific requirements to ensure that applications do not adapt in a conflicting or unfair manner.

We have only implemented a preliminary and coarse-grain control system and adaptation policy for our experiments. In our experiments, we manually fine-tune the various parameters, N , the adaptation policy and extent of network saturation for optimal performance. Future work for a more refined control system with better resource monitoring mechanisms will allow us to adapt the control algorithms automatically for better performance.

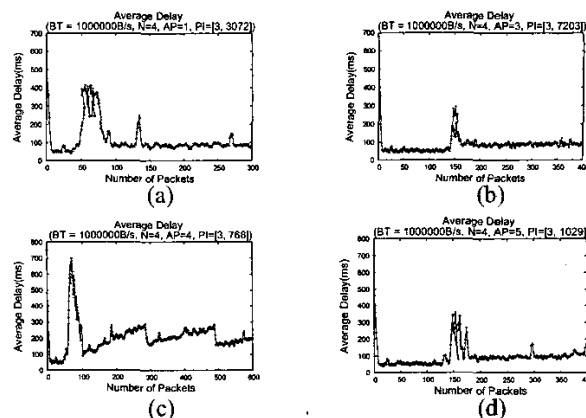


Fig. 6. Adaptation Results where Background Traffic = 1000000 Bytes/s. (BT = Background Traffic, N = No. of Delay values to average over, AP = Adaptation Policy, PI = Priority Interval)

Our current middleware framework is only focused on delay sensitive audio applications. In order to support other multimedia applications such as video-conferencing, our framework needs to be extended for a more complex, distributed control system in order to manage the priority levels for multiple streams. Furthermore, a multi-dimensional control system will be needed in order to adapt different QoS characteristics such as *framerate* and *resolution* in addition to delay.

REFERENCES

- [1] I. Aad and C. Castelluccia, "Differentiation mechanisms for IEEE 802.11," in *Proc. of IEEE INFOCOM*, 2001.
- [2] D. Gu and J. Zhang, "Qos enhancement in IEEE 802.11 wireless local area networks," *IEEE Communications Magazine*, June 2003.
- [3] H. Luo, S. Lu, and V. Bharghavan, "A New Model For Packet Scheduling in Multihop Wireless Networks," in *Proc. of ACM Mobicom*, 2000.
- [4] S.B. Lee, A. Gahng-Seop, X. Zhang and A.T. Campbell, "INSIGNIA: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks," *Journal of Parallel and Distributed Computing, Special issue on Wireless and Mobile Computing and Communications*, vol. 6, no. 4, 2000.
- [5] A. Veres G.-S. Ahn, A.T. Campbell and L. Sun, "Supporting service differentiation for real-time and best effort traffic in stateless wireless ad hoc networks (swan)," *IEEE Transactions on Mobile Computing*, September 2002.
- [6] Y. Xue, K. Chen and K. Nahrstedt, "Distributed End-to-End Proportional Delay Differentiation in Wireless LAN," in *IEEE International Conference on Communications*, 2004.
- [7] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets, "Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems," in *SPIE/ACM Multimedia Computing and Networking Conference*, 2003.
- [8] B. Li and K. Nahrstedt, "A Control-Based Middleware Framework for Quality of Service Adaptations," *Proceedings of Sixth International Workshop on Quality of Service*, 1999.
- [9] A. Hafid and G. Bochmann, "Quality of Service Adaptation in Distributed Multimedia Applications," *ACM Springer-Verlag Multimedia Systems Journal*, 1998.
- [10] V. Bharghavan, K.-W. Lee, S. Lu, S. Ha, J. Li, and D. Dwyer, "The TIMELY Adaptive Resource Management Architecture," *IEEE Personal Communications Magazine*, 8 1998.
- [11] G. Bianchi, A. Campbell, and R. Liao, "On Utility-Fair Adaptive Services in Wireless Networks," in *Proceedings of Sixth International Workshop on Quality of Service*, 1998.
- [12] K. Chen, K. Nahrstedt and N. Vaidya, "The Utility of Explicit Rate-Based Flow Control in Mobile Ad Hoc Networks," in *IEEE Wireless Communications and Networking Conference*, 2004.