2LQoS– Two-Layered Quality-of-Service Model for Routing in Mobile Ad Hoc Networks

Navid Nikaein, Christian Bonnet, Yan Moret, and Idris A. Rai
Institut Eurécom
06904 Sophia Antipolis, France

ABSTRACT

This paper presents a quality of service routing model for mobile ad hoc networks. Our model considers the characteristics of manet and emulates Intserv and Diffserv in that at application layer it uses end-to-end mechanisms for path establishment and at the network layer it utilizes a coding scheme to guarantee network resources on per-hop basis respectively. Furthermore, the model applies a coding method which provides faster routing decision for protocols, and it decreases the complexity of the QoS routing.

Keywords: Mobile ad hoc networks, quality of service (QoS), DiffServ, IntServ, routing, metrics, coding.

1. INTRODUCTION

Quality of service support in mobile ad hoc networks is a challenging task due to the fully mobile infrastructure and limited resources provided by the network. *Integrated services* (Intserv) [1] and *differentiated services* (Diffserv) [2] are the two basic architectures proposed to deliver QoS guarantees in the Internet. The Intserv architecture allows sources to communicate their QoS requirements to routers and destinations on the data path by means of a signaling protocol such as RSVP [3, 4]. Hence, Intserv provides per-flow end-to-end QoS guarantees. As in the case of the Internet, Intserv is not appropriate for mobile ad hoc networks, because the amount of state information increases proportionally with the number of flows, which results in scalability problems. Diffserv architecture on the other hand avoids the problem of scalability by defining a small number of per-hop behaviors (PHBs) at the network edge routers and associating a different Diffserv Code Point (DSCP) in the IP header of packets belonging to each class of PHBs. Core routers use DSCP to differentiate between different QoS classes on per-hop basis. Thus, DiffServ is scalable but it does not guarantee services on end-to-end basis. This is a drawback that hinders DiffServ deployment in the Internet, and remains to be a drawback for manet as well, since end-to-end guarantees are also required in manet. In this paper, we suggest an approach which considers the characteristics of manet and tries to emulate both end-to-end service management of Intserv while maintaining the scalability and per-hop service differentiation of Diffserv.

Quality-of-service means a set of service requirements to be met by the network while transporting a flow from the source to the destination. The QoS routing model specifies the architecture in which different routing protocols can be extended with the QoS support. The quality of service routing is a routing mechanism under which paths for flows are determined based on some knowledge of resource availability in the network as well as the quality of service requirements of flows [5]. Therefore, the main objective of quality of service routing is to optimize the network resource utilization while satisfying specific application requirements.

In this paper, we suggest a lightweight approach called 2LQoS– two-layered quality of service model to decrease overhead of QoS support in ad hoc routing through a *coding* method. This method associates a code to available network resources, which provides faster routing decisions for protocols. The code is set at the source node and is updated at each intermediate nodes. Destination node uses this code in conjunction with the desired QoS class to extract the most suitable path according to the application requirements. The code of the extracted path is re-used at the source node to *shape* the packet streams or flows according to the available network resources. However, the source may dismiss the path if the code does not satisfy the application requirement.

The organization of this paper is as follows. In section II, we describe in detail different phases of our QoS model called 2LQoS– two-layered quality-of-service. Section III presents the related work. Finally, section IV provide concluding remarks and highlights some future work.

2. TWO-LAYERED QUALITY OF SERVICE MODEL

In this section, we present a global view of our approach. Then, we highlight our two-layered QoS metrics: network and application layer metrics. The network layer metrics used in the path discovery procedure, while the application layer metrics used in the path selection procedure. After that, we outline QoS classes and their mapping with IntServ and DiffServ service architectures. We also give the algorithm of 2LQoS– two-layered quality-of-service. Next, we propose an analogy of DiffServ architecture which extends our service differentiation model.

Global View

Quality of service metrics are the criteria according to which paths are generated and then selected based on network and application requirements. Our quality-of-service model imposes both network and application requirements *on* path determination. For this purpose, the network and application requirements have to be meaningfully translated into metrics. Then, a reasonable combination of metrics have to be mapped onto *QoS classes* in such a way that the path computation complexity does *not* make route determination impractical. That is why, we propose two-layered QoS metrics: network layer metrics and application layer metrics. The network layer metrics are used by the *path discovery procedure* in order to *generate* paths which can satisfy the network. Indeed, the objective of the network layer metrics is to avoid unbalanced network utilization while minimizing the network resource consumption. Therefore, there is a trade-off between load balancing and resource conservation. On the other hand, the application layer metrics are employed by the *path selection procedure* at the destination node. They attempt to extract/select the most suitable path according to application requirements. In this sense, the network layer metrics become a primary metric and the application layer metrics become a secondary metric. To attain both objectives, the path discovery procedure determines the QoS *state* associated with the network layer metrics of paths. This procedure generates several paths between source and destination. Then, the destination node uses this QoS state together with the QoS class to extract the most suitable path according to the application requirements. The QoS state of the extracted path is *re-used* at the source node to *shape* the packet streams or flows according to the available network resources. Finally, the model differentiates services and provides soft guarantees to network resources for an admitted application by using a *class-based weighted fair queuing* (CB-WFQ) in the nodes.

Network Layer Metrics

As stated earlier, the network layer metrics are used during the path discovery procedure in order to indicate the current QoS state of paths. The main objective of the network layer metrics is to provide a trade-off between load balancing and resource conservation. Therefore, they control and maintain the network performance. We define four network layer metrics:

1) hop count – which is defined as the number of intermediate nodes between the source and the destination. The hop count metric is related to resource conservation. Note that a path with smaller number of hops – that can support a requested QoS – is preferable, since it consumes fewer network resources.

2) power level – which represents the current amount of available battery (i.e. energy). This metric is related to load balancing. This metric is translated into a two-bit code which indicates the QoS state of a node in terms of available battery. As it is illustrated in Table 1, the *high* QoS state is shown by 11 which indicates that the corresponding node has a fully charged battery. The *selfish* QoS state shows that the available battery is less than 25%.

Table 1: Coding for power level

QoS State	Code	Available battery
high	11	75-100%
medium	10	50-75%
low	01	25-50%
selfish	00	0-25 %

3) buffer level – which stands for the available unallocated buffer. Like the power level, this metric is also related to load balancing. It represents a node's internal state, and we assume that a node is capable of determining its state. [1]Note that if the buffer level of a particular node is low, then this implies that a large number of packets are queued up for forwarding. This metric translates into a two-bit code which indicates the QoS state of a node in terms of available buffer. As is illustrated in Table 2, *high* QoS state is shown by 11 which indicates that the corresponding node has an available buffer. The *selfish* QoS state shows that the available buffer is less than 25 percent of its size.

Table 2: Coding for buffer level

QoS State	Code	Available buffer
high	11	75-100%
medium	10	50-75%
low	01	25-50%
selfish	00	0-25 %

4) stability level – which we define as the connectivity variance of a node with respect to its neighboring nodes over time. This metric is used to avoid unstable nodes to relay the packets. We estimate the stability of a node x as:

$$stab(x) = \frac{|N_{t_1} \cap N_{t_0}|}{|N_{t_0} \cup N_{t_1}|}$$

N_{t_1} and N_{t_0} represent the nodes in the neighborhood of x at times t_1 and t_0 respectively. Note that, $t_1 - t_0$ denotes the time period in which nodes exchange beacons. A node is unstable if a large number of its neighbors change. Further, if most (or all) of the

[1]Note that it is trivial to determine the power and buffer level since a node can directly read from its battery and its buffer.

neighbors remain the same at the two times t_1 and t_0, then we call this node stable. Note that $N_{t_1} \cap N_{t_0}$ (the numerator of $stab(x)$) denotes the set of nodes that have remained in the neighborhood of x between time t_0 and t_1. The denominator of $stab(x)$ is a normalization term. A node has *high* stability if none of its neighbors change ($N_{t_1} = N_{t_0}$) , in this case we have $stab(x) = 1$. A node is *unstable* (no stability), if all its neighbors change ($N_{t_1} \cap N_{t_0} = \phi$), in this case we have $stab(x) = 0$. We say that a node has *low* stability if $0 < stab(x) \leq 0.5$ and that it has *medium* stability if $0.5 < stab(x) < 1$. Similarly, Table 3 shows the code associated with the stability.

Table 3: Coding for stability level

QoS State	Code	stab.
high	11	1
medium	10	0.5 <stab(x) < 1
low	01	0 < stab(x) ≤ 0.5
selfish(unstable)	00	0

It has to be mentioned that one can combine these metrics into a single weighted metric and use that as the basis for route determination.

Path Discovery

A sender willing to communicate with the destination node sends a QoS *path* request to the neighboring nodes. Fig. 1 depicts the format of a *path* message used to discover a QoS path. In addition to the source address *src*, destination address *dst*, and port number *port* which represents the *traffic ID* number, each packet is identified by a *class*; a QoS *state*; and the hop count.

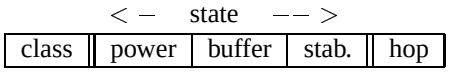

Figure 1: Fields of a path message

The QoS class identifies the desired service that is required by the application (refer. 4). It is also used to classify the packets in their appropriate queues (refer 2). This field is set by the source node and remains unchanged during path discovery procedure. The QoS state consists of three sub fields: power, buffer, and stability. They are updated as the path message is forwarded from node to node. Theses metrics define whether a node is *forced* to be *selfish* or not. A node switches to the selfish mode if it lacks the power or the buffer or the stability. For example in case of buffer level, the selfish mode avoids an overloaded node to be in the router mode, and lets an under-loaded node relay the network connectivity. Hence, leading to load balancing throughout the network. The power and stability level are computed by a concave function during the path discovery procedure, as they are shown below:

$$path.power = min(path.power, power)$$

$$path.stab = min(path.stab, stab)$$

The buffer level is also a concave function, but it represents the *average* buffer level of path from source up to the current node. It is computed by means of:

$$path.buffer = \frac{hop \times path.buffer + buffer}{hop + 1}$$

The last field, hop count, is an additive function– i.e. $hop = hop + 1$. Indeed, the hop count is incremented as the QoS request traverses a node. Upon receiving this message, a node checks whether its current QoS state is *forced* to be *selfish*. It drops the message if it is in the selfish mode; otherwise it updates the message and forwards it to its neighbors. The destination node will eventually receive the QoS path request. Then, it carries out the path selection procedure on the discovered paths (refer 2).

Application Layer Metrics

The application layer metrics are employed by the path selection procedure which is carried out at the destination node. They examine paths according to the application requirements in order to select the most suitable one. They include:

- delay– which is defined as the elapsed time for a packet to traverse the network from the source to the destination. At the network layer, the end-to-end packet latency is the sum of processing delay, packetization, transmission delay, queuing delay and propagation delay. As the QoS state represents in part the available bandwidth, therefore delay metric can be determined by the allocated bandwidth and hop count of the route [6, 7].

- throughput– which is defined as the rate at which packets are transmitted in the network. It can be expressed as a peak rate or an average rate.

- cost– which determines the amount of credits needed to establish a connection and access to network resources. The cost metric is a function of the power and the buffer level of a node. The cost metric is calculated during the path discovery procedure. Basically the cost metric is a additive function like hop count, and each node adds the amount of credit it needs to forward the traffic to the value of the cost metric. The cost metric has to be calculated on per user-basis; because the cost is strictly related to the power and the buffer level of a node, which are variable during the life time of a node. But for the simplicity reason; we assume that for a given QoS class, this amount is equal for each node in the network. So, hop count also represents the cost metric.

QoS Classes

We define three QoS classes for the destination to select the best available path. The first class has the highest priority and corresponds to applications with real-time traffic such as voice. This class is for applications with high delay constraints. The corresponding service of this class in Diffserv is referred to *expedited forwarding* and in Intserv to *guaranteed service*. The well-known port for this class is VAT. The second class has less priority than the first class. It is suitable for applications requiring high throughput such as video or transaction-processing applications. The service of this class is referred to *assured forwarding* in Diffserv and *controlled load* in Intserv. FTP and HTTP are the well-known port for the second class. The least priority class has no specific constraint (best-effort). This class is referred to the *best effort* in both architectures. Table 4 shows the defined QoS classes together with their mapping to Intserv and DiffServ services.

Table 4: QoS Classes & Mapping

Priority Class	IntServ	DiffServ
First Class e.g. voice low delay	Guaranteed	Expedited Forwarding
Second Class e.g. video High throughput	Controlled Load	Assured Forwarding
Third Class e.g. date No Constraint	Best Effort	Best Effort

Now, each QoS class has to be mapped to a code. This code represents which metric has to be considered for each class. This enables the destination node to select the most suitable path (path selection procedure). The first class is coded to 01 which associates the delay metric to this class. The second class is coded to 10 which in this case associates the throughput to the class. The third class is coded to 11 which represents that the application has no specific requirements. In this case, the hop count metric has to considered to reduce the network resource utilization. Table 5 depicts the mapping between the QoS class, code and metric.

Table 5: QoS Classes & Mapping

QoS class	code	ALM
First class	01	delay
Second class	10	throughput
Third class	11	hop count

Path Selection

At this stage, destination should select the most suitable path to satisfy the application requirements. For this purpose, it requires to determine the corresponding metric according to the Table 5:

- delay– in this context, the selected path should have a minimum end-to-end delay. The propagation delay being insignificant, the end-to-end transfer delay is closed to the queuing delay in the node buffer interface of the path. In stationary state, the queuing delay d in a node interface is:

  $$d = b/c$$

  , where b is the buffer level and c represents the total link throughput. Thus, if we assume that all nodes interface throughput is equal to c, the maximum end-to-end transfer delay $\bar{d}$ is:

  $$\bar{d} = N \times \bar{b}/c$$

  , where N is the number of hop, and $\bar{b}$ is the maximum buffer level. We select the path with the minimum $\bar{d}$. Thus, the selected path becomes:

  $$d = \min_{0 \leq i \leq p} \ (N_i \times \bar{b}_i))$$

  , where c is constant and p is the number of the paths received by the destination nodes.

- throughput– the selected path should have a maximum available bandwidth. If we assume that all node interface throughput is equal to c, the available bit rate for a path can be estimated by the following expression [8]:

  $$e = c + (r - \bar{b})/T$$

  , where r is buffer level threshold, $\bar{b}$ is the buffer level and T the end-to-end transfer delay. The estimation of T becomes:

  $$T = \bar{d} = N \times \bar{b}/c$$

  Thus, the selected path n is:

  $$e = \max_{0 \leq i \leq p} \ ((r - \bar{b}_i)/N_i)$$

- no constraints– in order to reduce the resource utilization, the selected path has to be the shortest path. That is:

  $$p = \min_{0 \leq i \leq p} \ (n_i))$$

If more than one path is found, then the destination node selects the most stable path. After that, the highest power level is desirable as a criteria for the path selection. Finally, if still more than one path is selected, one of them is randomly selected.

Shaping

At this stage, source node receives a reply message. This message indicates the QoS state of the selected path from

source to the destination. If the achieved QoS state corresponds to the *primary* application requirements, then data transmission occurs without any delay. Otherwise, source node verifies whether to *shape*[2] or *dismiss* its traffic. To do this, the stability level metric is used to capture the *duration* for which the communication between the source node and destination node may remain unbroken. This duration is called *stability period*. The rationale relies on the fact that a high stability indicates (with a large probability) a low state of node mobility, while a low stability indicates (with a large probability) a high state of mobility. If the stability period satisfies the shaping process, then source shapes the traffic. Otherwise, it dismisses the traffic.

For a path $s, n_1, n_2, \ldots, n_k, d$ between the source node s and the destination node d, one way to estimate the stability period is:

$$SP = \min_i\{stab(n_i)\} \cdot P$$

Hence, the stability period of a path is determined by the least stable (most unstable) node. Indeed, if a node x has low stability ($stab(x) \approx 0$), then this node should not be used to forwards packets from the source to the destination, and hence, this path should, in this case, be rejected. Note that if the stability period of a particular path is equal to the period P, then this implies that all nodes x on this path have $stab(x) = 1$. Furthermore, if the stability period of a particular path is 0, then this implies that there is some intermediate node x on this path for which $stab(x) = 0$, implying that this path should not be used because node x on it is unstable. Clearly, paths of higher stability are preferred.

Furthermore, each path can be associated with a stability period after which a new path discovery phase has to be triggered. This avoids path failure as the network topology may change after a certain time. Hence increasing the delay performance of the path failure.

2LQOS-Algorithm

Fig 2 depicts intuitively different stage of the 2LQOS–two-layer quality-of-service algorithm.

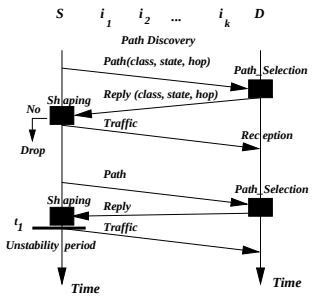

Figure 2: Signaling

[2]Shaping is the process of delaying packets within a traffic flow to cause them to conform to the QoS state of the selected path.

In our approach, the network always strives to maintain the *connectivity* of a node with the neighboring nodes through the network layer metrics regardless of the application requirements. Routing protocols apply the network layer metrics as the primary metrics for path discovery to maximize the network connectivity and consequently the network lifetime. Then, destination node carries out the path selection algorithm on paths that have already satisfied the network. The path selection algorithm employs the application layer metric as the metric to select the most suitable path according to the application requirements.

The outline of the algorithm used in 2LQOS is formally stated in Table 6. This algorithm has to be executed at each node in the network.

Table 6: 2LQoS Algorithm

```
path Path_discovery/selection(path){
  if(QOS_state = low) /*selfish mode*/
    drop path;
  path.power = min(path.power, power);
  path.buffer = min(((hop×path.buffer+buffer)/(hop+1)));
  path.state = min(path.stab, stab);
  path.hop_count++;
  if(path.dst = myaddress){
    if(path_class = 01)
     path = select path with
       min (path.hop × path.buffer);
    if(path_class = 10)
     path = select path with
       max ((r − path.buffer)/hop);
    if(path_class = 11)
     path = min(path.hop);
    if(|path| = 1)
     reply = path;
    if(|path| > 1){
     reply = max(path.stab);
     if(|reply| > 1){
       reply = max(reply.power);
       if(|reply| > 1)
           reply = random(reply);
     }
    }
    return reply;
  }
  if(reply.src = myaddress)
    shape(traffic, reply.state, reply.hop_count);
  forward path;
}
```

Service Differentiation in Nodes

The model presented above first computes a path that has sufficient network resources for an application and then uses the path to send information to the destination. The

model, as described so far, does not reserve the resources along the path, hence it does not guarantee that an admitted application receives the resources that were available from the network during path establishment. In this section, we propose an analogy of DiffServ architecture proposed for the Internet, which extends our model to provide a mechanism that guarantees the network resources for an admitted application on per-hop basis.

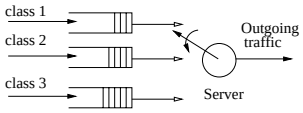

Figure 3: *Service Differentiation in an Adhoc Node*

To achieve this, we propose using the *class-based weighted fair queuing* (CB-WFQ) scheduling in adhoc nodes. Class-based WFQ is the extension of the standard weighted fair queuing (WFQ) [9, 10] functionality to provide support for user-defined traffic classes. In CB-WFQ, a queue is reserved for each class, and traffic belonging to a class is directly forwarded to the queue for that class, see Figure 3 for details. After packets are assigned to their corresponding queues, they receive prioritized service based on user-configured weights assigned to the queues. We define QoS classes in Tables 4. In our approach, Classification is performed by a source node. A source node assigns a QoS class to a packet by tagging a (two bit) code to the IP header of each packet belonging to an application. No further classification is required in the core nodes. Upon arrival to a core node, a packet is directly placed to the queue associated to the QoS code in its header. Hence, each queue buffers packets belonging to the same QoS class.In this model, the packets that reside in the same queue may belong to different applications with the same QoS class.

Finally, the server services packets from different queues based on the priority of the queue, which corresponds to the weights set for each queue in every node. Example of weights for each queue at the node can be set such that, class 1 service occupies 60% of the CPU times, class 2 service 30%, and class 3 service gets 10%. The weights in CB-WFQ are necessary to guarantee minimum bandwidth to each QoS class, this also prevents complete starvation of applications with lower priorities. Furthermore, the unused capacity in CB-WFQ is shared amongst other classes proportional to their weights. Hence, using CB-WFQ, a node guarantees QoS resources of an admitted application through scheduling.

3. RELATED WORK

Because of the fully mobile infrastructure and limited resources provided by the mobile ad hoc networks, conventional QoS model can not be directly used in the mobile ad hoc networks [11, 12, 13]. The QoS model specifies the architecture in which certain service could be provided in the network. Intserv [1] and Diffserv [2] are the two basic model proposed to deliver QoS guarantees in the Internet. IntServ defines two service classes: *guaranteed service* [14] and *controlled load* [15], in addition to the best effort service. The guaranteed service guaranteed to provide a maximum end-to-end delay, and is intended for audio and video applications with strict delay requirements. Controlled load, on the other hand, guarantees to provide a level of service equivalent to best effort service in a lightly loaded network, regardless of network load. This service is designed for adaptive real-time applications (e.g. application that can modify their play-out buffer as the end-to-end delay varies). The philosophy of this model is that there is an inescapable requirement for routers to be able to reserve resources in order to provide specific user packet streams, or flows [11]. Diffserv defines a limited number of aggregated classes in order to avoid the scalability problem of IntServ. In general, we can identify three different classes: *expedited forwarding*, *assured forwarding*, and *best effort*. Expedited forwarding provides a low delay, low loss rate, and an assured bandwidth. Assured forwarding provides guaranteed/expected throughput for applications, and best effort which provides no guarantee. Table 7 provides a comparison between integrated services and differentiated services architecture. There are several problems related to the IntServ model

Table 7: Comparison of IntServ and DiffServ

criteria	IntServ	DiffServ
granularity	individual flow	aggregate of flows
state in routers	per-flow	per-aggregate
classification	header fields	DS field
signaling	required(RSVP)	not required
coordination	end-to-end	per-hop
scalability	$<$ # of flows	$<$ # of classes

[12]. The amount of state information increases with the number of flows since IntServ architecture works on a per-flow basis. Furthermore, the RSVP reservation and maintenance process consumes lots of network resources. Finally, the heavy burden of admission control, classification, and scheduling again for network resources. DiffServ, on the other hand, tries to overcome the limitation of IntServ by aggregating the individual flows into a few classes. QoS decisions are taken in a hop-by-hop basis.

A new QoS model called FQMM - a flexible QoS model for manet - is proposed for mobile ad hoc networks [16]. The basic idea of this model is that it uses both per-flow state property of IntServ and the service differentiation of DiffServ. This model proposes that the highest priority is assigned per-flow provisioning and other priority classes are given per-class provisioning.

4. CONCLUSION

This paper presents a quality of service routing model which benefits from the advantage of both Intserv and Diffserv, and tries to improve the communication and time complexity of the algorithm via a coding method. We suggest network and application layer QoS metrics to address two objectives: firstly to avoid unbalanced network utilization while minimizing the network resource consumption; and secondly to select routes that can meet the specific application requirement. In our future work, we will evaluate the performance of our model under various routing protocols.

5. ACKNOWLEDGMENT

The authors wish to thank Ashish Rastogi for his useful discussions.

6. REFERENCES

[1] R. Braden, D. Clark, and S. Shenker, "Integrated services in the Internet architecture: an overview," 1994, IETF RFC 1633.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," 1998, IETF RFC 2475.

[3] Lixia Zhang, Stephen Deering, and Deborah Estrin, "RSVP: A new resource reservation protocol," *IEEE network*, 1993.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (RSVP) – version 1 functional specification," Sept. 1997, IETF RFC 2205.

[5] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the internet," Aug. 1998, RFC 2386.

[6] A. K. Parekh, *A generalized processor sharing approach to flow control in integrated services networks*, Ph.D. thesis, MIT, 1992.

[7] C. Partridge, *Gigabit Networking*, Addison-Wesley, 1994.

[8] Y. Moret and S. Fdida, "ERAQLES an efficient explicit rate algorithm for ABR," in *GLOBECOM*, 1997.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of the Sigcom'89, Symposium on Communications Architecture and Protocols*, 1989, pp. 1–12.

[10] D. Clark, S. Shenker, and L.Zhang, "Supporting real-time applications in an integrated service packet networks: Architecture and mechanism," in *ACM SIGCOM'92*, 1992, pp. 14–26.

[11] X. Xiao and L. M. Ni, "Internet QoS: A big picture," *IEEE Network*, pp. 8–18, Match/April 1999.

[12] K. Wu and J. Harms, "QoS support in mobile ad hoc networks," 2001.

[13] D. Chalmers and M. Sloman, "A survey of QoS in mobile computing environments," *IEEE Communications Surveys*, 1999.

[14] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," 1997, RFC 2212.

[15] J. Wroclawski, "Specification of the controlled-load network element service," 1997, RFC 2211.

[16] H. Xiao, W. K. G. Seahand A. Lo, and K. C. Chua, "A flexible quality of service model for mobile ad-hoc networks," in *IEEE VTC2000-spring*, Japon/Tokyo, 2000.