QoS-aware Dynamic Source Routing Protocol using Diffserv Principles

A Project Report Submitted in the partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science

By
Sravanthi Bhavanam

Department of Computer and Information Sciences
University of Hyderabad
Hyderabad, India

June, 2008

CERTIFICATE

This is to certify that the project work entitled "**QoS-aware Dynamic Source Routing Protocol using Diffserv Principles**" being submitted to University of Hyderabad by **Sravanthi Bhavanam** (Reg. No. 06MCMT29), in partial fulfillment for the award of the degree of Master of Technology in Computer Science, is a bona fide work carried out by her under my supervision.

Ms. Anupama Potluri
Project Supervisor,
Department of CIS,
University of Hyderabad

Head of Department,
Department of CIS,
University of Hyderabad

Dean,
School of MCIS,
University of Hyderabad

To,

My Parents and Supervisor.

Acknowledgments

I take this opportunity with immense pleasure to convey my gratitude to my supervisor, **Ms. Anupama Potluri** who acted as a challenging captain coordinating all of our batch with great patience and invaluable suggestions. They really helped me to complete this task well before the stipulated time. She has always encouraged, supported, corrected and guided me during the project. This project happened to be one of my best journeys through core concepts of Computer Networks. The constant guidance of our supervisor helped me to develop both individually and professionally.

I can not get a better opportunity than this to thank profusely my senior **Y. Jaya Lakshmi** with whose hard work my project literally became a cake walk. I would like to thank **Tholoana Masupha** who is also part of this project. I am extreamly thankful to my lab mates **B. Saritha, B.Vishala and L. Ramprasad** for continuously inputting me with their valuable suggestions and being a source of encouragement at all possible times. I would also like to thank the developers of **ns2** for providing such a wonderful and robust software to work along with detailed documentation.

I am extremely grateful to our Head of Department, **Prof. Arun Agarwal**, for providing excellent computing facilities and such a nice atmosphere for doing my project. I convey my heartfelt thanks to **AI Lab staff** for their help in completing the project work successfully.

Last but not the least, I would like to express my heartful wishes to my beloved parents and my dear friends whose gracious solicitude helped me to complete this project successfully.

Sravanthi Bhavanam

Abstract

Mobile Adhoc Networks(MANETs) allow for infrastructure-less mobile wireless networks to be established with minimal delay for deployment. The trend seen in applications of wireless services is that they make increasingly complex demands for Quality of Service(QoS). Thus, providing QoS in a MANET environment is becoming more important. However, QoS in MANETs is a challenging task, because of the inherent MANET limitations and properties. Some of the existing solutions are not scalable and have high signaling overhead. Some solutions are scalable but do not use the resources efficiently. Some solutions are simple but they do not even guarantee QoS and typically do not support multiple classes of service.

Our proposed scheme is a cross-layer QoS-aware routing protocol that uses Diffserv model for data plane operations. It supports multiple classes of service and dynamically allocates resources for each QoS class. It reserves the resources for each flow and each flow is mapped to one of the QoS classes. The number of meters, policers and queues maintained per node are restricted to number of QoS levels only. So in this way it is scalable. As it reserves resources during route discovery process itself, it has less signaling overhead and has less latency to start data plane operations. To evaluate the performance of our scheme, we implemented it in the network simulator *ns-2.29* by extending Dynamic Source Routing (DSR) protocol.

Simulation results show that our scheme has achieved throughput close to ASAP while using fewer meters, policers and queues. Results also show that call acceptance ratio of our scheme is higher than ASAP. From the results we can also observe that our scheme gives QoS guarantee to the flows when congestion occurs whereas SWAN does not. Average end-to-end delay for our scheme is less than that of both ASAP and SWAN. Latency to start data plane operations is also acceptable in our scheme. Overall, we are performing better than ASAP and SWAN.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

1.1 QoS Models for Wired Networks 1

1.1.1 Integrated Services(Intserv) 1

1.1.2 Differentiated Services(Diffserv) 2

1.2 Mobile Adhoc Networks(MANETs) 2

1.3 Challenges for Providing QoS in MANETs 3

1.4 QoS Models for MANETs 3

1.4.1 Cross-layer Solutions 3

1.4.2 Independent Solutions 4

1.5 Motivation 4

1.6 Organization of the Project Report 5

2 Related Work **6**

2.1 ASAP[1] 6

2.1.1 Overview 6

2.1.2 Advantages and Limitations 8

2.2 AQOR[2] 9

2.2.1 Admission Control 9

2.2.2 Route Discovery 10

2.2.3 Adaptive Route Recovery 10

2.2.4 Advantages and Limitations 11

2.3 SWAN[3] 11

2.3.1 Source Based Admission Control of Real-Time Traffic 12

2.3.2 Rate Control of Best-Effort Traffic 12

2.3.3 Dynamic Regulation of Real-Time traffic . . . 13
2.3.4 Advantages and Limitations . . . 13
2.4 QoS Framework using Differentiated Services[4] . . . 13
2.4.1 Overview . . . 14
2.4.2 Advantages and Limitations . . . 15
2.5 CEQMM[5] . . . 15
2.5.1 Architecture . . . 15
2.5.2 QOLSR Protocol . . . 15
2.5.3 Call Admission Control . . . 16
2.5.4 Congestion Avoidance and Control . . . 17
2.5.5 Advantages and Limitations . . . 17
2.6 Summary . . . 18

3 Proposal and Implementation **19**
3.1 QoS aware Routing using Diffserv Principles[6] . . . 19
3.1.1 Routing Algorithm . . . 20
3.1.2 Data Plane Operation . . . 20
3.2 Design of the Protocol . . . 21
3.3 Request Table Modification . . . 23
3.4 Admission Control . . . 23
3.5 New Route Cache Implementation . . . 23
3.6 Route Maintenance . . . 28
3.7 Session Record Maintenance . . . 28
3.8 Dynamic Resource Allocation . . . 29
3.9 Data Plane Operation . . . 30
3.10 Analysis of DiffQ-DSR . . . 31
3.11 ASAP[1] Porting . . . 32
3.12 SWAN[3] Porting . . . 33

4 Results and Analysis **34**
4.1 Simulation Environment and Scenarios . . . 34
4.1.1 Simulation Environment . . . 34
4.1.2 Performance Metrics . . . 35
4.1.3 Scenarios . . . 36
4.1.3.1 Basic QoS Scenarios . . . 36
4.1.3.2 Effect of Interference on QoS . . . 37
4.2 Simulation Results and Discussion . . . 38

4.2.1 Throughput and Packet Delivery Ratio 38
4.2.1.1 Low Density Scenario 38
4.2.1.2 Medium Density Scenario 41
4.2.1.3 High Density Scenario 41
4.2.2 Comparison of guaranteed QoS 44
4.2.3 Call Acceptance Ratio(CAR) 44
4.2.4 Latency to Start Data Plane Operations 45
4.2.5 Scalability 47
4.2.6 Average End-to-End Delay 48
4.2.7 More Simulation Results 49
4.3 Summary 52

5 Conclusions and Future Work **53**
5.1 Conclusions 53
5.2 Future Work 54

Bibliography **55**

List of Figures

2.1 SWAN Network Model[3]. 12
2.2 Architecture of CEQMM[5]. 16

3.1 Class diagram. 22
3.2 Flow f1 starts between s1 and 2 and corresponding cache entries. . . . 26
3.3 Flow f2 starts between s1 and D and corresponding cache entries. . . 26
3.4 Flow f3 initiates between s1 and 3 and corresponding cache entries . . 26
3.5 After processing of RERR cache entries. 27
3.6 ASAP Architecture in *ns*. 33

4.1 Interference Experiment 38
4.2 Average Throughput for each QoS level for the density of 50 nodes . 40
4.3 PDR for each QoS level for the density of 50 nodes 40
4.4 Average Throughput for each QoS level for the density of 75 nodes . 42
4.5 PDR for each QoS level for the density of 75 nodes 42
4.6 Average Throughput for each QoS level for the density of 100 nodes . 43
4.7 PDR for each QoS level for the density of 100 nodes 43
4.8 Call acceptance ratio by varying number of the flows 45
4.9 Latency to start data plane operations 46
4.10 Average Throughput for each QoS level for the density of 50 nodes with 15 connections 47
4.11 Average end-to-end delay for 5 connections 49
4.12 Average Throughput for each time interval(15sec) with 5 connections and 30 pause time 51

List of Tables

4.1 Simulation Environment 35
4.2 QoS class levels and corresponding DSCP values 36
4.3 QoS class levels 47
4.4 QoS class levels 50

Chapter 1

Introduction

In computer networks the goal of QoS support is to achieve a more deterministic communication behavior, so that information carried by the network can be better preserved and network resources can be better utilized. QoS is an agreement or guarantee by the network to provide prespecified service attributes to the user in terms of delay, jitter(variance of delay), available bandwidth and probability of packet loss etc. QoS is essential to support real-time applications like VoIP, video conferencing, multimedia streaming etc. The Internet is, however, best-effort and does not provide any QoS. Many QoS models have emerged in recent times to support QoS in the Internet.

1.1 QoS Models for Wired Networks

The two most popular QoS models for wired networks are

- Integrated services(Intserv).
- Differentiated services(Diffserv).

1.1.1 Integrated Services(Intserv)

The Integrated services model is the first standardized QoS model for the Internet. This model provides per flow QoS granularity. It reserves resources for each flow on the path by using Resource Reservation signaling protocol(RSVP) before commencing of the flow. The signaling messages carry QoS requirements of the flows. In this model per flow state information is maintained at the nodes and this information is refreshed periodically. At every node four basic components should be implemented. Those

are signaling protocol, admission control, classifier and scheduler. It is well suited for meeting the dynamically changing needs of applications but it suffers from scalability problem.

1.1.2 Differentiated Services(Diffserv)

Diffserv is a fully distributed and stateless model. Instead of providing QoS at per flow granularity, Diffserv differentiates the traffic into a fixed number of classes. Diffserv aggregates a set of flows and applies a pre-defined behavior to that aggregate. So no state information for each flow is required to be maintained at any node. The network is divided into edge network and core network. The nodes at the edge of the network are responsible for classification of flows, policing them to ensure that the traffic complies with the agreement made by the service provider and marking the packets so that they can be differentiated in the core network. ToS field of the IP header is used for carrying marked codepoint. The nodes in the core network just check the codepoint of the packets and forward according to the per-hop behavior defined for that codepoint, making dataplane operations very efficient.

The raising popularity of multimedia applications among end users and the potential use of Mobile Adhoc Networks(MANETs) in civilian life have led to research interest in providing Quality of Service support in MANETs. We are also concentrated on this research area. In the following sections we discuss about MANETs and their features, what are the challenges for providing QoS in MANETs and the existing solutions and their drawbacks.

1.2 Mobile Adhoc Networks(MANETs)

A Mobile Ad hoc Network (MANET) is a collection of wireless mobile nodes dynamically forming a temporary network without the use of any existing network infrastructure or centralized administration. It is a multihop wireless communication network in which each node can either act as a host or a router. MANETs represent future generation wireless networks, capable of being deployed quickly and economically at places lacking any infrastructure. These characteristics make MANETs more suitable for defense based applications, disaster relief operations and commertial applications.

1.3 Challenges for Providing QoS in MANETs

Providing QoS in MANETs is challenging because of the following reasons:

- *Dynamic network topology:* Nodes are mobile in the network. Link breaks occur due to mobility of nodes. The flows which are using these links are disturbed and latency is involved in finding a new route. It is also possible that a new route may not be found. Thus QoS can be violated for such flows.
- *Interference and Noise:* Because of the wireless medium there is more interference and noise. Due to this packet losses may be more. So we can not guarantee QoS.
- *Limited battery life:* Nodes are power constrained. So QoS solutions should not be too complex which consume more processing power of nodes.
- *Bandwidth-constrained:* Bandwidth is very limited in wireless networks. The estimation of available bandwidth is very difficult because it not only depends on admitted flows in the channel, but also on the neighborhood nodes.

1.4 QoS Models for MANETs

All the proposed models either use Intserv or Diffserv properties and these are mainly categorized as

- Cross-layer solutions.
- Independent solutions also known as frameworks.

1.4.1 Cross-layer Solutions

QoS solutions can be provisioned at any layer in the protocol stack. Most existing schemes are at network layer. QoS-aware routing protocols will take QoS metrics as constraints to be satisfied, rather than trying to find the shortest path. Examples of these protocols are AQOR[2], ACOR[7], QAODV[8] etc. A brief description of AQOR is presented in the next chapter.

1.4.2 Independent Solutions

Instead of adding QoS to the routing or MAC layer, these schemes define separate QoS modules for signaling, admission control and scheduling etc. The main idea behind this is to separate functionality of each module so that these are independent to the routing or MAC protocols. Examples for these schemes are INSIGINA[9], ASAP[1] which are per flow QoS provisioning schemes and SWAN[3], Diffserv framework[4] which are per class QoS provisioning schemes. We will discuss some of these frameworks in the next chapter.

1.5 Motivation

From our literature survey, we find that the problems with existing QoS solutions for MANETs are

- Some are not scalable [1], [2], [9].
- A few have high signaling overhead [1], [5].
- There is no QoS guarantee[3], [1].
- Many of the scalable schemes use static resource allocation which leads to waste of resources if there are no flows of tat QoS level[4], [5], [10].
- Some of the schemes do not support multiple classes of services[3], [4].

So, we proposed a cross-layer QoS aware routing protocol that supports multiple classes of service. It dynamically allocates resources for each QoS class. It reserves the resources for each flow and each flow is mapped to one of the QoS classes. The number of meters, policers and queues maintained per node are restricted to number of QoS levels only. So in this way it is scalable. As it reserves resources during route discovery process, it has less signaling overhead and the latency to start data plane operations is reduced. In addition, we use Diffserv principle of marking packets at the source(which is treated as an edge router) and simply enqueueing in the right queue at the intermediate routes speeding up processing in the data plane. To evaluate the performance of our scheme, we implemented the scheme in the network simulator *ns-2.29* by extending Dynamic Source Routing (DSR) protocol which is an on-demand routing protocol for MANETs. We also ported the freely available implementations ASAP and SWAN to *ns-2.29* for comparison of the performance of our scheme with them.

Our simulation results show that the objectives with which we proposed our solution like scalability, high call acceptance ratio and QoS guarantee etc. are achieved.

1.6 Organization of the Project Report

The rest of the project report is organized as follows: In Chapter 2, we present some of the existing QoS solutions for MANETs with their advantages and disadvantages. In chapter 3, we present our proposal and implementation details of our proposal in network simulator *ns-2.29*. Actually we have extended Dynamic Source Routing Protocol(DSR) to be QoS-aware. So in this chapter we present what are the extensions required and how we achieved them. Simulation results of our scheme, ASAP[1] and SWAN[3] are given in chapter 4. Simulation results show that our scheme performs better than ASAP and SWAN. Finally we conclude with chapter 5.

Chapter 2

Related Work

Providing QoS in MANETs is a challenging task because of their inherent properties like dynamic topology, bandwidth constraints and limited battery life. Many solutions exist in literature for providing QoS in MANETs and these are mainly categorized as per-flow and per-class QoS provisioning schemes. Examples of per-flow QoS provisioning schemes are INSIGNIA[9], ASAP[1], AQOR[2] and QAODV[8]. Some of the per-class QoS provisioning schemes are SWAN[3], QoS Framework with Diffserv[4] and CLAD[10]. Some solutions integrate both the schemes - FQMM[11] and CEQMM[5]. In this chapter we discuss a few of these schemes - ASAP[1], AQOR[2], SWAN[3], [4], CEQMM[5] - and we present their advantages and limitations. We summarise the issues with these schemes and the motivation for developing a new scheme in the last section of this chapter.

2.1 ASAP[1]

ASAP is a QoS provisioning framework with the following features:

- Per flow QoS provisioning scheme.
- Two phase bandwidth reservation mechanism.
- Hybrid signaling protocol.

2.1.1 Overview

ASAP assumes that routes to destination are already present in the nodes using any of the existing on demand routing protocols[12], [13]. ASAP introduces its own signaling system to reserve the resources. ASAP adapts the admitted real-time flow's

sending rate according to the feedback given by the signaling messages. It even treats the admitted real-time flows as best-effort flows if there is no sufficient bandwidth on the new path after rerouting.

ASAP signaling ASAP introduces two messages - SR and HR - to do soft reservation and hard reservation respectively. SR message soft-reserves the resources on the path. Resources which are in this state can be used by other real time or best effort flows but can not be reserved by any other real time flow. The HR message switches soft reserved bandwidth to hard by forcing out the traffic which uses this soft reserved bandwidth temporarily. In this way, this two phase bandwidth reservation improves the efficiency of bandwidth usage. SR messages use in-band signaling and HR messages use out-of-band signaling. The fields in SR message are MinBW, MaxBW, SoftBW and HardBW. MinBW and MaxBW are minimum and maximum bandwidth requirements of the application respectively. SoftBW and HardBW values represent how much bandwidth has been soft reserved and hard reserved on the path respectively. The fields in HR message are SetBW, SoftBW and HardBW. SetBW represents the amount of bandwidth to be reserved as hard.

Flow setup To start a real time flow, the sender sends a unicast SR message, with minimum and maximum bandwidth requirements filled in the corresponding fields of the packet, towards the receiver. Intermediate nodes on the path which receive this message will do admission control and if it succeeds, they soft reserve within the range Min-Max. Then, they add an entry for that flow and fill the SoftBW field in the packet with reserved bandwidth and forward to the next hop. At the receiver SoftBW field represents the bottleneck bandwidth of the path. The receiver hard reserves this much bandwidth and sends HR message so that each intermediate node switches its soft reserved bandwidth to hard reserved. After receiving HR message, the sender starts its flow with the rate matching the reserved bandwidth.

QoS monitoring and adaptation After flow setup, SR messages are periodically sent on the path by the sender to collect the QoS information. As part of adaptation, the SR packet is used to try to make extra reservations on its way to the destination. If every node on the way has increased its soft reservation, then the receiver will send an HR packet to hard reserve the newly available bandwidth. After receiving this HR message, the sender increases the flow transmission rate to the new hard reserved bandwidth. Similarly, a flow can be scaled down on re-routing of the flow when a

route breaks. In this case, HR message releases extra reservations.

Local path repair Whenever a route break occurs, the node which precedes the broken link will find the new path. After finding the new path, the node sends SR message to reserve the bandwidth on the new path. The SoftBW and HardBW fields in the SR message are filled with the soft reserved and hard reserved values of that flow which are stored in the node's flow table. Whenever the nodes on the new path receive this message, they will follow the same procedure as described above. If the nodes on the new path do not satisfy the bandwidth requirements of the flow, then that flow is treated as best-effort flow.

Flow releasing Flow releasing can be done by soft state or by explicitly sending HR message with setBW zero. Periodic SR messages are used to refresh the flow entries. So the sending rate of SR messages is correlated with soft state timeout period. The adaptability and signaling overhead also depends on sending rate of SR messages.

2.1.2 Advantages and Limitations

Advantages of this scheme are

1. Supports any QoS level between MIN-MAX range.
2. Does not have any excess reservations.
3. Efficient bandwidth usage.

Limitations are

1. No scalability as it maintains per flow state information.
2. Adaptability might result in loss of QoS.
3. Signaling overhead is high.
4. Adaptation depends on signaling frequency.

2.2 AQOR[2]

AQOR is a QoS provisioning protocol with the following features:

- It is an on-demand routing protocol.
- It has in-band signaling for resource reservation and maintenance.
- It incorporates an efficient method of estimating available bandwidth based on the traffic not only at the node but also at its neighbors.
- Destination initiated route recovery.

In AQOR, every node maintains a list of neighbors, N(I), which includes all its neighbors with their corresponding traffic. This information is useful to calculate the available bandwidth at that node.

2.2.1 Admission Control

To accept a flow, the node should know the available bandwidth at that node and the bandwidth to be consumed by the requesting flow. Because of the interference of the neighborhood nodes, the bandwidth to be consumed by the requesting flow is not equal to the minimum bandwidth requirement of that flow.

$B_{consume}$estimation The bandwidth to be reserved for the flow j at any node I, $B_I(j)$ is given by

$B_I(j) = B_{min}$,*if I is source node or destination node for the flow j.*
$B_I(j) = 2B_{min}$, *otherwise*

The consumed bandwidth at node I can be calculated by taking into consideration the new self traffic and the new boundary traffic introduced by flow j

$$B_{consumed}(I, j) = B_{uplink(I)}(j) + B_{downlink(I)}(j) \tag{2.1}$$

where $B_{uplink(I)}(j)$ and $B_{downlink(I)}(j)$ are either equal to B_{min} or $2B_{min}$.

$B_{available}$ estimation The available channel bandwidth $B_{available(I)}$ of node I can be determined by the aggregated traffic in the channel and maximum transmission rate of the node.

$$B_{available}(I) = B - \sum_{J \in N(I)} B_{self}(J) \tag{2.2}$$

where B is maximum transmission rate of that node. N(I) is the set of neighbors of I. $B_{self}(J)$ is J's self traffic, i.e., the total reserved bandwidth of all existing flows at that node.

If $B_{available}(I)$ is greater than $B_{consumed}(I, j)$, then the node I will accept the flow j.

2.2.2 Route Discovery

Route discovery algorithm is implemented by route exploration from source to destination and route registration in reverse path.

Route exploration When the destination is not in the source's neighbors list, the source broadcasts a RREQ which includes requested bandwidth and end to end delay. Intermediate nodes which receive this RREQ will do admission control and if it succeeds they will add a route entry with status explored and rebroadcast. If no reply comes at explored node in time, the route entry will be deleted and late coming replies will be ignored.

Route registration For each RREQ it gets, the destination will give RREP in the reverse path. The intermediate nodes will do admission control again and if then also it succeeds, they change the explored status to registered status. The source will take one of the paths among received RREPs and sends the data through that path. The reservation of resources on the path happens when the first data packet is sent along the chosen path. All registered routes that are not used for data plane operations will be deleted after some time.

2.2.3 Adaptive Route Recovery

The destination can detect end-to-end delay by observing arrival time of packets. It can also detect route break when the soft state timer for reserved bandwidth of the flow expires. If the end-to-end delay is more than the requested delay or a route break

is observed, the destination tries to rediscover a new route satisfying the application constraints. It initiates the route discovery process in the reverse direction, by sending a route update message. Whenever the source gets the first route update message it switches the flow to the new path.

2.2.4 Advantages and Limitations

Advantages are

1. Provides QoS guarantee.
2. No signaling overhead.
3. Estimates accurate available bandwidth.

Limitations are

1. Not scalable.
2. Route requests are forwarded until reservation happens on a path, i.e., even after registration. This can lead to more overhead when the available bandwidth after reservation does not satisfy the requested bandwidth and the route discovered for the new flow includes such nodes on its path.

2.3 SWAN[3]

The main features of SWAN are

- It is a stateless network protocol.
- It uses distributed control algorithms to provide service differentiation.
- It uses Explicit Congestion control Notification (ECN) to dynamically regulate admitted real time traffic.

Figure 2.1 shows the SWAN network model.

Figure 2.1: SWAN Network Model[3].

To admit real time flows SWAN uses source based admission control algorithm and to ensure that the bandwidth and delay requirements of real time traffic are met, the rate control of best-effort traffic is performed at every node. A classifier is used to differentiate real time traffic and best-effort traffic. Best-effort packets are given to the shaper to delay them according to the rate calculated by the rate controller.

2.3.1 Source Based Admission Control of Real-Time Traffic

The admission controller which is present at each node estimates the local available bandwidth by listening on the shared channel. To admit a real time flow, the admission controller of the source sends a bandwidth probe request which collects the bottleneck available bandwidth on the path instantaneously. The destination sends bandwidth probe reply to the source with that bottleneck bandwidth. Then source will compare the bottleneck bandwidth with the bandwidth requirement of the real time flow. If it is satisfied, that flow will be admitted and its packets are marked as RT. No resources are reserved; so intermediate nodes do not maintain any state information.

2.3.2 Rate Control of Best-Effort Traffic

The rate controller takes feedback of packet delays from MAC layer and if this packet delay exceeds threshold delay, then it controls the transmission rate of best-effort traffic. For this, it uses Additive Increase and Multiplicative Decrease(AIMD) algorithm. The shaping rate is determined by feedback of the rate controller.

2.3.3 Dynamic Regulation of Real-Time traffic

QoS violation may occur due to mobility or false admission. False admission occurs because multiple sources read the state of the network at the same time and may admit more flows than intermediate nodes can support. To regulate traffic in such cases two algorithms are proposed in SWAN:

Source based regulation When congestion occurs the intermediate nodes mark the packets with Congestion Experienced(CE) bit. Whenever the destination gets these marked packets, it sends the regulate message to the corresponding sources. Upon receiving regulate message, the sources will wait for a random amount of time and then try to re-establish the flows with the required constraints. A random waiting

time is chosen so that all sources will not drop their real time flows at the same time to regulate the traffic.

Network based regulation Instead of marking all flows with congestion experienced(CE) bit, congestion nodes will select a congestion set and only those flows will be marked until some period of time. After that time a new congestion set is calculated, if congestion persists. This way, only the flows which are marked only need to be re-established.

2.3.4 Advantages and Limitations

Advantages are

1. Scalable as no state information is maintained.

Limitations are

1. It does not guarantee QoS because it does not reserve any resources.
2. Only two classes of traffic are supported.

2.4 QoS Framework using Differentiated Services[4]

Features of this framework are

- It uses Quality of service Adhoc Routing protocol(QuaSAR).
- It uses diffserv principles.
- It reserves resources on per class basis.
- It releases resources automatically.

2.4.1 Overview

This framework defines the first downstream node from the source as the edge router which is responsible for classifying, conditioning and marking of traffic from the source. All the other nodes act as core routers and they support two PHBs - Expedited Forwarding(EF) and Best-Effort(BE) forwarding. To send traffic, the source node first initiates route discovery process. The RREQ packet carries service level and bandwidth requirement. Admission control will be done for EF traffic only.

The distributed max-min fair share approximation algorithm is used to estimate the fair bandwidth share of a link. Only a fraction of this bandwidth is assigned to EF traffic to avoid congestion due to mobility. Whenever the destination gets RREQ, it replies back on the reverse path and the edge node stores information regarding that flow like flow id, service level, bandwidth requirement and token bucket states in its policy table. After route discovery, the source node starts sending the traffic.

Intra node differentiation The edge router assigns corresponding traffic profile according to the entries in its policy table. Token bucket is used for metering the EF traffic and its rate is determined by the requested bandwidth which is stored in policy table. When there are not sufficient tokens for arriving packets, these are out-of-profile packets and marked as BE; otherwise, they are marked as EF. After marking, each packet is enqueued in the corresponding queue- BE queue or EF queue. Priority queuing is used to schedule the packets.

Inter node differentiation 802.11 MAC layer is slightly modified to give priority to EF traffic in accessing the medium. With this modification EF data packets wait for DIFS period and BE data packets wait for double the DIFS period when transmitting unicast packets.

Automatic resource release There is no explicit release message nor even soft state release. A core router does not know how much bandwidth to release as it does not maintain any state information. So a node periodically estimates its average queue length and compares it with the long term average queue length. If it is smaller, then the node knows that some of the reserved resources are not used and it releases that unused banwidth gradually.

2.4.2 Advantages and Limitations

Advantages are

1. Scalable as it does not maintain a lot of state information per-flow.
2. Supports multiple classes.

Limitations are

1. Static bandwidth allocation for classes.

2. Release of resources is not efficient. Hence, call acceptance rate can be affected even when there is unused bandwidth that can satisfy the new flow.

2.5 CEQMM[5]

Characteristics of CEQMM are

- It supports both per flow state property of Intserv and the service differentiation of Diffserv.
- It supports multi-constraint path QoS routing.
- It includes admission control, bandwidth reservation mechanisms and congestion avoidance, congestion control mechanisms.
- It supports multiple service classes.

2.5.1 Architecture

Figure 2.2 shows the architecture of CEQMM.

Figure 2.2: Architecture of CEQMM[5].

2.5.2 QOLSR Protocol

It is a proactive routing protocol. HELLO messages carry the QoS information which is estimated by metric measurement component and the neighbors information. Using these messages a node selects QMPR set which is a subset of the 1-hop neighbors. Only the nodes which are in the the QMPR set can be selected as intermediate nodes for QoS flows because they provide best QoS metrics. This set is re-calculated when there is a change in 1-hop or 2-hop neighborhood or QoS conditions. Each QMPR node in the network periodically sends Topology Control(TC) extension messages which consists of its QMPR selector set and QoS conditions. This information is used to calculate the routing table. Each node maintains two routing tables- one for Best Effort and another for QoS flows. Per flow state information is maintained for highest priority flows;otherwise per QoS level information is maintained.

2.5.3 Call Admission Control

To admit a real time flow, the sender unicasts check request(CREQ) packet with QoS requirements. It carries entire path for highest priority flow; otherwise only next hop. Every node which receives this calculates the new available bandwidth after adding the requested bandwidth to the current available bandwidth of the node. If the new available bandwidth is greater than zero, then the link to the next hop satisfies the required bandwidth and it soft reserves requested bandwidth for high priority flows. If the delay of the link is greater than requested delay, the node simply drops that CREQ packet. Otherwise the delay of the link is subtracted from requested delay and this value is placed in CREQ packet. Every intermediate node will do the same and when destination gets this packet, it sends check reply(CREP) packet back on the same path. Call admission control is done once again on the reverse path to ensure QoS guarantee. While forwarding CREP soft reserved bandwidth is switched to hard reserved bandwidth.

Priority classifier classifies the control, QoS and best-effort traffic and enqueues them in to the corresponding queue. Control packets are given highest priority to maintain latest information about network topology. It uses Random Early Detection(RED) queue management to avoid buffer overflows. Token balancing mechanism is used for scheduling packets to avoid starvation of low priority flows. In this mechanism each class is given some balance of tokens. whenever a packet is scheduled from a class s, then Ws amount of tokens are removed from that class's balance tokens and are added to the other classe's balance of tokens in equity share. The weight factor Ws value is high for low priority flows. The class which has high balance of tokens is given first preference to transmit packets.

2.5.4 Congestion Avoidance and Control

Call admission control, best-effort rate control and RED Queue management scheme are used to avoid congestion. In case congestion occurs, best-effort flows may be dropped or a new path is calculated and TC messages are sent with new congested metric so that other nodes calculate new path for their best-effort flows. For QoS flows, the originator node will wait for some backoff time and then change the route if the node which precedes the congestion node does not have any alternate route with that QoS requirements.

2.5.5 Advantages and Limitations

Advantages are

1. Scalability.
2. Supports multiple service classes.

Limitations are

1. Uses proactive routing protocol.
2. Lot of message overhead.
3. Static bandwidth allocation for QoS classes.

2.6 Summary

The issues with the existing schemes are

1. Some are Not scalable [1], [2], [9].
2. A few have more signaling overhead [1], [5].
3. There is no QoS guarantee[3], [1].
4. Many of the scalable schemes use static resource allocation[4], [5], [10].
5. Some of the schemes do not support multiple classes of services[3], [4].

So, a new protocol, QoS aware Routing using Diffserv Principles[6] was proposed to achieve the following

1. Scalability in terms of the number of meters and policers used per node.
2. Support for multiple classes of services.
3. Dynamic resource allocation into a QoS class of service.
4. Reduce signaling overhead and latency to start data plane operations by using cross-layer principles.

In the next chapter, I describe the design and working of this protocol, and my contribution to it.

Chapter 3

Proposal and Implementation

In this chapter, I describe the proposed protocol *QoS aware Routing using Diffserv principles[6]* and I present the design of the protocol. I also specify my contribution to that protocol in the implementation part. To compare our scheme with respect to parameters like call acceptance ratio, latency to start dataplane operations, packet delivery ratio and throughput, I have ported ASAP[1] and SWAN[3], which were already described in the previous chapter, to network simulator *ns-2.29*.

3.1 QoS aware Routing using Diffserv Principles[6]

This scheme is proposed by Jaya Laksmi et al.[14] to take advantage of both Intserv and Diffserv. It provides

1. Scalability in terms of meters and policers used.
2. Dynamic resource allocation per flow.
3. Per QoS-level policing and metering.
4. Multiple classes of service.
5. Soft state resource release.
6. Low signaling overhead.

In this scheme, the source acts as an edge router and others act as core routers. Service level agreements like Committed Information Rate(CIR), Peak Information Rate(PIR) are configured dynamically. An assumption in this scheme is that every node knows the existing QoS levels and their corresponding Diffserv Code Points(DSCP).

3.1.1 Routing Algorithm

Route discovery To admit a real time flow, the source first checks its routing table. If it does not have any route for this flow, then it will do admission control. If admission control succeeds, the source broadcasts QRREQ packet. QRREQ carries minimum and maximum bandwidth requirements of the application. Intermediate nodes which receive this QRREQ will do admission control and if it succeeds they will rebroadcast the QRREQ packet; otherwise they simply discard the packet. If admission control succeeds at the destination, it will give a reply, QRREP, to the first received QRREQ in the reverse path, after populating its routing table and session table. Session table is a table that contains session ID of that flow, source ID , the minimum bandwidth reserved, length of the path and initial DSCP value. The amount of reserved bandwidth is deducted from available bandwidth and added to the corresponding QoS level. Every intermediate node on the path checks for admission control once again when forwarding the reply. If it succeeds, the node adds the minimum bandwidth to the bandwidth for that QoS level and deducts it from its available bandwidth. Otherwise, the node at which admission control fails will set the admission control (ac) bit in the QRREP and just forwards it towards the source. The nodes downstream to that node will simply forward the reply to the source without processing it further. If the source gets QRREP with ac bit set, then it will initiate the route discovery process again. Otherwise, it populates its routing table and session table and starts transmitting data.

Resource maintenance Reserved resources are refreshed through data plane operations. If the flow is completed or a route break occurs and the flow no longer uses a node, the soft state timer on the reservation expires and the bandwidth of that flow is returned to available bandwidth.

Route maintenance When a route break occurs while forwarding data, the node which precedes the broken link sends RERR message to the source after releasing resources of that flow and resources of all the other flows which are using that path. After receiving RERR, the source restarts the route discovery process.

3.1.2 Data Plane Operation

After finding the route, the source marks the data packets with the corresponding initial diffserv code point. After applying metering and policing these packets are

again remarked with different drop precedences based on whether they are in- or out-of-profile. Now these are enqueued into the corresponding queues. The marking and conditioning is done only at the source node as it is the edge router. The intermediate nodes, which are like core routers, just check the DSCP value in the packets and enqueue into the corresponding queues. When congestion occurs, the packets will be dropped according to drop precedence. Weighted Round Robin(WRR) mechanism is used to schedule the packets.

The protocol is implemented in the network simulator *ns-2.29* by extending the Dynamic Source Routing protocol[12].

3.2 Design of the Protocol

The class diagram of the whole protocol is given in fig3.1. All the classes are specific to *ns-2.29* implementation of DSR[12] as we are extending DSR protocol to implement our protocol. *DSRAgent* is the main class which implements all functionalities like route discovery and route maintenance.

Figure 3.1: Class diagram.

Jayalakshmi had defined QRREQ and QRREP packets by extending normal RREQ and RREP packets of DSR[12]. The extra fields in the QRREQ packet are minimum bandwidth, maximum bandwidth, bottleneck bandwidth and session ID. The QRREP packet contains bottleneck bandwidth, session ID and initial DSCP of the flow. She had implemented the generation and propagation of these QRREQs and QRREPs.

Tholoana Masupha had defined the new interface queue *dsrDiff* for our protocol. For this purpose she has defined a new class - *dsrDiff*. This class is also useful to differentiate the edge router functionality from the core router functionality. She has defined four QoS levels of service based on the application bandwidth requirements.

My contribution to the protocol is the implementation of:

1. Route request table modification.
2. Admission control functionality.

3. New route cache implementation.

4. Dynamic allocation of resources to each class.

5. Soft state management of session records.

6. Data plane processing.

The remaining sections of this chapter describe each of the above in detail. First, we describe the way DSR handles each of these. Then, we describe the modifications needed to support QoS extensions and how they have been designed and implemented.

3.3 Request Table Modification

In DSR, if many flows start at the same time from the same source to the same destination, route discovery is initiated for that destination only once. The source checks to see if any request is outstanding for this destination in its request table. If so, it does not initiate route discovery. But in our protocol the source should initiate route discovery for all flows because we need to reserve the resources for every flow. So to achieve this, we have added session ID field to the request table. The source checks its routing table and compares the destination ID and session ID of the current flow with those which are destined to the same node. If no matching entry is found, then the source initiates route discovery for the new flow. If an exact match exists, then, the data packets can be forwarded using that route cache entry.

3.4 Admission Control

DSR is a best-effort routing protocol. It does not require any available bandwidth information at that node. But our protocol is QoS-aware and it should guarantee the flows. It should not admit the flows for which nodes do not have sufficient bandwidth. So to do this admission control, every node should know the available banwidth at that node. We are maintaining the available bandwidth value in the MAC layer. As shown in the class diagram 3.1 we have added available bandwidth field in the MAC class. This value is initialized to the link bandwidth. As bandwidth is reserved and released from flows, this value is changed.

3.5 New Route Cache Implementation

In DSR, every node has its own route cache. The route cache can store the learned source routes. These routes can be used by intermediate nodes. Intermediate nodes can give reply by using these cached route entries. These route entries are refreshed through data plane operations. The node can also remove the routes when it learns about the broken link by getting RERR packet. If the route is not used by any flow, then, after timeout period these routes are deleted from the route cache.

Implementation Details In *ns-2.29* implementation of DSR, three types of caches are defined. Those are *SimpleCache*, *LinkCache*, *MobiCache*. By default *DSRAgent* chooses *MobiCache*. *MobiCache* stores the whole source route in it. It defines two caches - primary cache and secondary cache. Primary cache is used by the source and the destination nodes. Intermediate nodes use secondary cache to store the routes. As shown in the class diagram 3.1 *MobiCache* is derived from *RouteCache* which is an abstract class. Primary cache and secondary caches are objects of *Cache* class. *Cache* class is a set of objects of *Path* class. The *Path* class is an array of nodeIDs which represent the path for a destination that this node is aware of. Each *Path* object in the *Cache* object represents the route to all the various nodes that comprise this path.

In our protocol we need to store the session information, in addition to the route, in the cache for a flow. Session information tells how much bandwidth to release from which QoS class for that flow. So, this session information should be maintained per flow. As explained before, the cache maintains total route to the destination in one path object. So, if there are multiple flows to the same destination for which the same route is found after QoS route discovery, we need to maintain the list of associated session records for the same path object. Also, if there are data flows to any of the other nodes on the route, these are also part of the same path object and hence session records for all these flows are also maintained as part of the same path object. The algorithm for adding route and session record is given in Algorithm 1.

Whenever any intermediate node on the path moves away, the nodes beyond this node are no longer reachable. Hence in DSR implementation, when this happens, the path is truncated up to the node that has moved away. In addition to this, in DiffQ-DSR, every node should delete the session records of these nodes and reclaim the bandwidth to the available bandwidth. To achieve this, in each session record, we

Algorithm 1: ***addRoute() function***

[1] each entry in the cache compare the newpath with the cache entry. cache entry is NULL add path to the cache. $AddSRecordatBegining(sessionrecord)$ $return$ total newpath or some part of the new path matches the total or part of cachepath $newpath.length() < cachepath.length())$ $position \leftarrow GetSRecordPosition(sessionrecord)$ $AddSRecordatPosition(sessionrecord, position)$ $return$ $newpath.length() > cachepath.length())$ append remaining path to the cachepath. $AddSRecordatEnd(sessionrecord)$ $return$ $AddSRecordatEnd(sessionrecord)$ $return$

Figure 3.2: Flow f1 starts between s1 and 2 and corresponding cache entries.

maintain the length of the path to the destiantion of that session. If, after truncation, the length of the path is less than the length in a session record, the destination of that session is no longer reachable. So, we walk through the session list and delete all session records whose path length is greater than the current path length. The session list is maintained in a sorted order to accomplish this efficiently. The following figures show all this functionality clearly.

Figure 3.3: Flow f2 starts between s1 and D and corresponding cache entries.

Figure 3.4: Flow f3 initiates between s1 and 3 and corresponding cache entries

Figure 3.5: After processing of RERR cache entries.

In Figure 3.2, consider that there is a flow f1 between s1 and 2. If s1 finds the path *s1-1-2* for the flow f1, then it stores the path *s1-1-2* and the associated session record s2 in its cache. Now after some time, a flow f2 starts between s1 and D and consider that s1 finds that the path *s1-1-2-3-D* satisfies the QoS requirements for the flow f2. Then while storing this path in its cache, s1 finds that it has already some part of the path in its cache and it just appends the remaining path *3-D* to that path. S1 adds the session record *sd* at the end of the session list as its length of the path value is greater than the previous session record s2's length of the path value. This

is shown in Figure 3.3. Now again consider that there is a flow f3 between s1 and 3 and the path satisfying the flow f3's QoS requirements is *s1-1-2-3*. Whenever s1 tries to store this path, it finds that it has already that path in its cache. Then s1 inserts the session record s3 in between s2 and sd in the session list as s3's length field value is 4. This is shown in Figure 3.4. Now if the node 3 moves away, the source gets the route error and it truncates the path from *3-D*. Now the updated path length is 3. So s1 deletes all the session records whose length field is greater than this updated path length. This is shown in Figure 3.5. So by getting RERR for the flow f2 only, s1 initiates the route discovery for f3 even before it gets RERR for this flow. By doing this we can reduce the packet loss.

To implement the above functionality we defined two new classes: *QPath* and *SessionRecord*. *QPath* class is derived from *Path* class because we need all the functionality that *Path* class provides. To associate the session list with the path, we have defined container relationship between the SessionRecordList class and QPath class. The class *SessionRecord* contains the fields source Id, session ID, bottleneck bandwidth, DSCP and length of the path. All these classes and the relationships between them is shown in Figure 3.1

To store the session records, we need to know the position where to insert because we are maintaining session records in the sorted order. The algorithm for this functionality is given in Algorithm 2.

Algorithm 2: ***GetSRecordPosition() function***

[1] each SRecord in SessionList $SRecord.length <= sessionrecord.length$
$position \leftarrow position + 1$ return position

3.6 Route Maintenance

Whenever a route break occurs, the node which precedes the broken link will send RERR to the source. This RERR packet contains the IP addresses of the two nodes between which the link is broken. The processing of RERR is given in Algorithm 3. In this algorithm $DeleteSessionList(n + 1)$ function deletes all the session records whose length fields are greater than $n + 1$ value. $n + 1$ represents the updated path length.

Algorithm 3: ***processBrokenLink() function***

[1] each path in the cache $n \leftarrow 0$ $n < path.length()$ path[n] = RERRpacket.fromnode and path[n+1] = RERRpacket.tonode $n = 0$ delete the whole path truncate the path from n+1 node $DeleteSessionList(n+1)$ break $n \leftarrow n+1$

3.7 Session Record Maintenance

SessionRecordTimer class maintains the soft state of the session records. It checks the sessionrecord list periodically. If any session record timer expires that session record will be deleted and resources are released. This functionality is given in Algorithms 4 and 5.

Algorithm 4: ***sessionListCheck() function***

[1] each entry in the cache $temp \leftarrow GetFirstSRecord()$ $temp \neq NULL$ $(currenttime - temp.SRTime) > SessionTimeout$ $temp1 \leftarrow GetNextSRecord()$ $releaseResources(temp)$ $DeleteSRecord(temp)$ $temp \leftarrow temp1$ $temp \leftarrow GetNextSRecord()$

Algorithm 5: ***releaseResources() function***

[1] $addtoAvailBW(sessionrecord.SRbbw)$ $deductFromQoSlevel(sessionrecord.SRbbw)$ $minbw \leftarrow getminBW(sessionrecord.SRdscp)$ $maxbw \leftarrow getmaxBW(sessionrecord.SRdscp)$ $deductFromCIR(minbw)$ $deductFromPIR(maxbw)$ return

These session records are refreshed through the dataplane operations. For this purpose we have added new DSR option to hold session ID in data plane packets. The algorithm for this is presented in Algorithm 6.

Algorithm 6: ***refreshSessionRecord() function***

[1] each entry in the cache $temp \leftarrow GetFirstSRecord()$ $temp \neq NULL$ $temp.SRsrc = src$ and $temp.SRsid = sid$ $temp.SRTime = currenttime$ $return$ $temp \leftarrow GetNextSRecord()$

3.8 Dynamic Resource Allocation

In general Diffserv implementation, the metering values Committed Information Rate(CIR) and Peak Information Rate(PIR) for the QoS classes are fixed and these are assigned to each QoS class at the begining. But in our protocol, whenever a flow of that particular QoS class is admitted, then the CIR and PIR values are assigned to that class. These metering values are changed dynamically according to the flows admitted to that class. While processing QRREP every node adds the minimum bandwidth requirement of the flow to the CIR of the QoS class to which that flow belongs. In the same way maximum bandwidth requirement of the flow is added to the PIR.

3.9 Data Plane Operation

In DSR, the data packet carries the whole source route from the source to the destination which is copied from the route cache. Intermediate nodes which get this data packet add the source route to their cache if they do not have it. Otherwise, they simply refresh the route. Instead of carrying the whole source route, the data packet carries flow ID in the new extension of DSR. DSR uses only one interface queue for all the flows to enqueue the packets before they are transmitted by the network interface.

In our protocol, every data packet carries the whole source route and to refresh the session records we have added session ID option to the DSR header. At the source node each data packet is metered, marked and policed. Every intermediate node just checks the marked value and enqueues it to the corresponding queue. We have defined four QoS classes of service and three drop precedences for each QoS class. So we need to have four physical queues corresponding to the four QoS classes and each physical queue should have three virtual queues corresponding to the three drop precedences. We are using RED queue management.

To implement the above functionality, we are using the classes *DSRPolicy*, *dsrredQueue*, *dsrREDQueue*, *QOSClass* and *dsrDiff*. The relation between all these classes is shown in Figure 3.1. *dsrDiff* class differentiates the edge router functionality and core router functionality. *dsrREDQueue* class provides four physical queues and the class *dsrredQueue* provides three virtual queues. *DSRPolicy* class implements all the

functionality of marking, metering and policing.

Algorithm 7 gives the data plane operation in our protocol

Algorithm 7: ***Data plane operation***

[1] $packet \leftarrow recv(dataPacket)$ $nodeID = packet.dest$
$refreshSessionRecord(packet.src, packet.sessionID)$ $acceptPacket(packet)$
$nodeID = packet.src$
$path \leftarrow findroute(packet.dest, packet.src, packet.sessionID)$ $path \neq NULL$
$packet.path \leftarrow path$ $mark(packet)$ $applyMetering(packet)$
$applyPolicer(packet)$ $enqueue(packet)$ $getQRouteforPacket(packet)$ /* At intermediate nodes */ $refreshSessionRecord(packet.src, packet.sessionID)$
$enqueue(packet)$

3.10 Analysis of DiffQ-DSR

The main advantages of our scheme are

- Even though per flow state information is maintained at each node, we are providing per QoS level provisioning. So the number of meters and policers used are reduced to number of QoS levels.
- Dynamic resource allocation allows arbitrary number of flows of a class of service to be supported as long as there is bandwidth available at the node rather than be restricted by the static allocation for that class. This should increase the call acceptance rate while also achieving QoS.
- Reduce signaling overhead by adopting a cross-layer routing protocol that carries the QoS information during route discovery. Resource reservation is also done at the time of route discovery. This is expected to result in less latency to start data-plane operations than non cross-layer schemes.

3.11 ASAP[1] Porting

I have already explained the ASAP[1] protocol in the previous chapter. To compare the performance of our scheme with ASAP with respect to parameters like call acceptance ratio, throughput, packet delivery ratio and latency to start dataplane operations, I have ported ASAP[1] which is implemented in *ns-2.27* to *ns-2.29* version.

Implementation Details

I describe, briefly, the implementation of ASAP as implemented in *ns-2.27*. The source code for this is available on *http://www.iks.inf.ethz.ch/asap*. Basically four components are implemented. Those are

- ASAP Agent for generating and processing of SR and HR messages.
- Adaptation Controller for controlling the end-to-end adaptation and providing an interface to the application.
- Interface Queue to get separate subqueue for each flow as ASAP reserves the resources for each flow.
- An ASAP-aware extension of the hash classifier. Hash classifier is used to classify a packet to which flow it belongs. And it also classifies whether that packet is data packet or control packet. To differentiate ASAP signaling packet this classifier is extended.

Figure 3.6 shows how these components fit into the *ns* architecture.

Figure 3.6: ASAP Architecture in *ns*.

The node entry is the point where packets first arrive. The classifier checks for the address field of a packet to verify whether that packet is destined for the current node. If the packet is a signaling packet it is passed to the ASAP agent. If the QoS state of a flow changes due to mobility or a lot of interference in the MANET, then the ASAP agent informs the Adaptation Controller which is responsible to adapt to the current state.

3.12 SWAN[3] Porting

SWAN[3] is a very light-weight protocol. It does not reserve any resources on the path. So we want to compare our scheme with this with respect to how much QoS guarantee we are providing. Actually this is implemented in *ns-2.1b9a*. So I have ported this to *ns2.29* version.

They had implemented two major components. Those are

1. SWAN admission controller agent for sending and processing of bandwidth probe requests, probe replies and regulate messages.
2. SWAN rate controller to adjust the rate of the best-effort flows.

For more implementation details refer *http://comet.columbia.edu/swan/sourcecode.html*

In the next chapter we present the various scenarios that we simulated and present the results of our simulation for all the three schemes - DiffQ-DSR, ASAP and SWAN. We analyze the results using our understanding of the protocols and some simple experiments we did to understand the results.

Chapter 4

Results and Analysis

We have implemented our protocol in network simulator *ns-2.29* to prove its efficiency. In this chapter we present the simulation results of our scheme. We also compare our results with that of the schemes like ASAP[1] which maintains per flow state information and SWAN[3] which does not maintain any state information. These are already implemented in another *ns* version. I have ported these to *ns-2.29* version. Our scheme has features of these two schemes like QoS guarantee feature of ASAP[1] and scalable property of SWAN[3]. For ASAP we did simulations with adaptation and without adaptation. In the first section we present the simulation environment and also some of the tests done to confirm the correctness of implementation of our protocol. We also present some other experiments we did to understand the results we got. In the second section we present the simulation results and discuss them. In the last section we summarize the results and analysis.

4.1 Simulation Environment and Scenarios

4.1.1 Simulation Environment

The Simulation environment is as described in table 4.1. All the simulations are done in *ns-2.29*. A MANET with variable density of 50 nodes, 75 nodes, 100 nodes was used with a simulation time of 150 sec. The mobile nodes were placed in a 1000 $\times$ 1000 m flat grid. The mobility of the nodes was in the range of 0-5m/s. We have varied the pause time for the nodes as 5 sec, 15 sec and 30 sec. We have used a random way point mobility model. In this model, each mobile node starts its journey from current location to a random destination location with a specified speed. Once the destination is reached, the mobile node will stay there for the amount of pause

Table 4.1: Simulation Environment

S.No.	*Parameters*	*Values*
1	*Area*	*1000 X 1000*
2	*Transmission range*	*250m*
3	*Nodes*	*50, 75, 100*
4	*Pause time*	*5, 15, 30 sec*
5	*Mobility*	*0- 5m/s*
6	*Simulation time*	*150 sec*
7	*Link capacity*	*2 Mb/sec*
8	*Mobility model*	*Random way point*
9	*Traffic type*	*Constant Bit Rate(CBR)*

time specified and then starts moving towards the new destination location. Each scenario is repeated ten times by varying different source-destination pairs.

4.1.2 Performance Metrics

To evaluate the performance of the schemes, we have measured the following metrics.

Call acceptance ratio This is the ratio between the number of flows accepted in the network to the number of flows initiated at the sources. But ASAP, instead of rejecting flows whenever there is no sufficient minimum bandwidth on the links, treats them as best-effort flows. So, when calculating call acceptance ratio for ASAP we are treating these best-effort flows as rejected flows.

Throughput It is defined as the number of bits received at the destination per second. We calculated average throughput for each QoS level and compared it with the corresponding flow's committed rate.

Packet delivery ratio This is the ratio between the number of data packets received at the destinations and the number of data packets generated at the sources. This metric indicates the reliability of admitted flows.

Latency to start data plane operations This is the time duration between the time when the first data packet was generated to when it was transmitted in the network at the sources. Actually for cross-layer schemes like DiffQ-DSR it is the time

Table 4.2: QoS class levels and corresponding DSCP values

Classes	*Range(kbps*	*Initial codepoint*	*Downgrade-1*	*Downgrade-2*
Class-1	*best-effort*	*0*	*0*	*0*
Class-2	*8-16*	*10*	*12*	*14*
Class-3	*16-32*	*18*	*20*	*22*
Class-4	*64-128*	*26*	*28*	*30*

taken to find the route to the destination and reserve the resources on that path. For non-cross-layer schemes, it includes the route discovery time followed by resource reservation signaling time.

Average end-to-end delay It is defined as latency incurred by the packets between their generation time and their arrival time at the destination.

Scalability It is the one of the important parameters for MANETs. As the number of flows increase, the processing and storage overhead should not be increased. So in this perspective we compare our scheme with other schemes.

4.1.3 Scenarios

We have done some test experiments to test the correctness of our protocol and to fix the bugs. The QoS levels and their corresponding DSCPs are shown in Table 4.2.

4.1.3.1 Basic QoS Scenarios

The scenarios we have done are

1. As shown in Figure 4.1(a) we have taken three static nodes 1, 2 and 3. We have started a flow which belongs to class-3 and send data at the rate of 64kbps which is its committed rate i.e at its minimum bandwidth requirement. We calculated its throughput and we got 64kbps and all its packets are marked with code point 26. Now we have started this flow at the rate 90kbps which is in between its committed rate and peak rate. Now the packets which are coming above committed rate are marked as 28. Again I have started the flow

at 150kbps which is above its peak rate. In this case the packets which are sent at above peak rate are marked as 30. But in the last two cases we got the throughput as 90kbps and 150kbps respectively as there is no congestion.

2. Now we increased the number of flows and experimented with different sending rates. We observed that when congestion occurs, the packets were dropped according to their drop precedence.

3. In the third scenario, I have taken two flows which belong to the same QoS level and started one flow at its committed rate and and another flow at above peak rate. Then I observed the packets of the flow which is sending at its committed rate also are marked with codepoint 30 and these packets are dropped. This is because both flows belong to the same QoS class and our scheme is per class QoS provisioning.

4. In the fourth scenario I have taken more flows such that only some of them are accepted. And after some time I have stopped some of the accepted flows. Then I observed that the flows which are not admitted initially get admitted after session record timeout period of stopped flows. This is because the reserved bandwidth of the stopped flows is reclaimed to available bandwidth after the session record timeout period only.

4.1.3.2 Effect of Interference on QoS

We did another small experiment to know the effect of interference on QoS achieved.

As shown in Figure 4.1(a) we have taken six static nodes such that 1, 2 and 3 nodes are in the transmission range of each other and 4, 5 and 6 nodes as one set. None of the sets are in interference range of each other. Then we have started a flow between 1 and 3 and another flow between 4 and 6 at the rate of 64kbps. We got the throughput for each flow as 64kbps.

Now as shown in Figure 4.1(b), we have placed node 5 in the interference range of node 2. Then we observed that the throughput for the second flow is zero.

By this experiment we can conclude that, even though we are assuring the flows at their committed rate we are not achieving this because of interference effect. So for providing QoS in MANETs, we need to provide some solutions at MAC layer also.

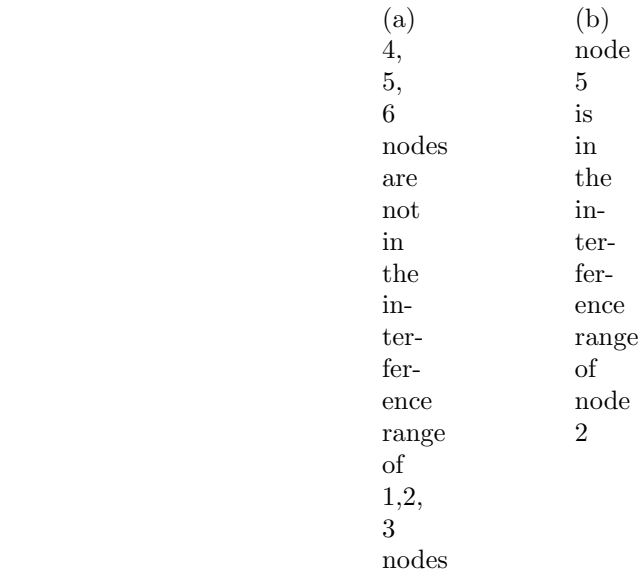

Figure 4.1: Interference Experiment

4.2 Simulation Results and Discussion

In this section, we present the results of simulations done to measure the performance of our scheme in comparison with ASAP and SWAN. Each of the following sections is a discussion of the performance with respect to the parameters identified earlier.

4.2.1 Throughput and Packet Delivery Ratio

In this simulation, we are using the QoS levels which are shown in Table 4.2. We have taken total five flows and from these flows, two flows belong to Committed Rate(CR) of 8kbps and two flows belong to CR of 16kbps and the remaining flow belongs to CR of 64kbps. These flows are sent at their minimum bandwidth requirement. The traffic of these flows is of type Constant Bit Rate(CBR) and the packet size is 125 bytes. The exact same configuration is taken for ASAP[1] and SWAN[3]. We did simulations for ASAP with and without adaptation enabled. To do simulation of ASAP without adaptation, we have taken the refresh time period as total simulation time. We have simulated each scenario about 5 times by varying the source and destination pairs for the connections. We measured the average cumulative throughput and packet delivery ratio for each QoS level. This is done to reduce variance of the results obtained.

4.2.1.1 Low Density Scenario

Figures 4.2(a) and 4.3(a) show respectively, the average cumulative throughput and packet delivery ratio for ASAP with adaptation, ASAP without adaptation, SWAN and DiffQ-DSR for the density of 50 nodes with 5 pause time respectively. From the graphs we can observe that

- DiffQ-DSR is giving more QoS guarantee than SWAN.
- When we compare with ASAP with adaptation, for low priority levels it is giving more throughput than ours but it is not at all giving minimum assurance throughput for high priority level.
- Coming to comparison with ASAP with no adaptation, for all the levels it has higher throughput than our scheme. But this is because ASAP sends its flows at their maximum bandwidth requirement, i.e. its sending rate is higher leading to a higher throughput. We can observe from Figure 4.3(a) that the packet delivery ratio of ASAP with no adaptation is less than that of DiffQ-DSR. So, we can say that the performance of DiffQ-DSR is better overall.

Figures 4.2(b) and 4.3(b) show the average cumulative throughput and packet delivery ratio for 50 nodes with 15 pause time. From the graphs we can observe that DiffQ-DSR has slightly less throughput and packet delivery ratio than other schemes for low priority levels. But it has more throughput and packet delivery ratio for high priority levels than other schemes. That means DiffQ-DSR is giving more preference to high priority flows, which is good from the perspective of QoS.

The average throughput and packet delivery ratio for the density of 50 nodes with 30 pause time are shown in Figures 4.2(c) and 4.3(c) respectively. From the graphs we can observe that DiffQ-DSR has more throughput and more packet delivery ratio than other schemes for medium and high priority flows.

Finally when we compare with committed rates, DiffQ-DSR is giving throughputs for high priority flow whose committed rate is 64kbps as 56, 55, 44kbps for pause times 5, 15 ad 30 respectively.

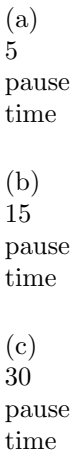

Figure 4.2: Average Throughput for each QoS level for the density of 50 nodes

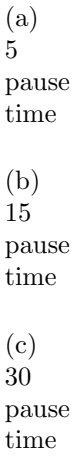

Figure 4.3: PDR for each QoS level for the density of 50 nodes

4.2.1.2 Medium Density Scenario

Figures 4.4(a), 4.4(b) and 4.4(c) show the average throughput for 75 nodes with pause times 5, 15 and 30 respectively. From the graphs we can see that DiffQ-DSR has more throughput than other schemes for high priority flows for all the pause times even though it has low throughput for low priority levels. This clearly shows that DiffQ-DSR is giving more preference to high priority flows as already stated in the previous section. But all schemes are giving less throughput than the assured QoS for the high priority level except in case of 15 pause time. This is because as the density increases, interference from the nodes increases. We have already explained the effect of interference on the throughput of flows. The experiment we did to know the interference effect is for static nodes and in that the second flow's throughput is almost zero. But here nodes are moving; so we are getting some throughput.

Figures 4.5(a), 4.5(b) and 4.5(c) show the packet delivery ratio for all schemes with pause times 5, 15 and 30 respectively. Graphs show that the packet delivery ratio for DiffQ-DSR is more than the other schemes.

4.2.1.3 High Density Scenario

Figures 4.6(a) and 4.7(a) show the average throughput and packet delivery ratio for each QoS level respectively for 100 node density with 5 pause time. In the case of 100 nodes SWAN throughput for the high priority level is slightly higher than DiffQ-DSR but it is not that much significant. The same behavior is observed across ASAP and DiffQ-DSR as in case of 50 nodes.

In case of 15 pause time, DiffQ-DSR has more throughput and more packet delivery ratio than other schemes except for low priority level. This is shown in Figures 4.6(b) and 4.7(b). Figures 4.6(c) and 4.7(c) show the throughput and packet delivery ratio for 30 pause time. From the graphs we can see that ASAP with no adaptation has more throughput than DiffQ-DSR. But at the same time it has less packet delivery ratio than DiffQ-DSR i.e. it has more packet loss than DiffQ-DSR. Actually, ASAP with no adaptation is sending data at the rate of 128kbps and it is getting 77kbps throughput for class-4. But DiffQ-DSR is sending data at the rate of 64kbps and it is getting 59kbps throughput. That means ASAP with no adaptation has 51kbps loss whereas DiffQ-DSR has 5kbps loss only.

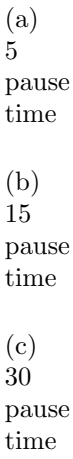

Figure 4.4: Average Throughput for each QoS level for the density of 75 nodes

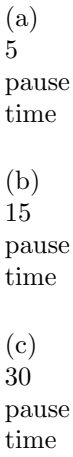

Figure 4.5: PDR for each QoS level for the density of 75 nodes

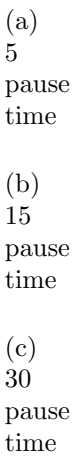

Figure 4.6: Average Throughput for each QoS level for the density of 100 nodes

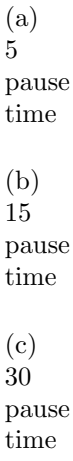

Figure 4.7: PDR for each QoS level for the density of 100 nodes

4.2.2 Comparison of guaranteed QoS

We did an experiment to understand how much QoS guarantee is given to the flows by the protocols when there is congestion, we did an experiment. As shown in Figure 4.1(a), we have taken three static nodes 1, 2 and 3 which are in the transmission range of each other. We have started two flows between 1 and 3 in which one flow(f1) with CR of 16kbps and another flow(f2) with CR of 64kbps. We are sending these flows at the rate of 300kbps and 64kbps respectively. Now we calculated the throughput of each flow in each protocol and the result is as follows:
In ASAP : Both flows are getting throughput at their maximum bandwidth requirement. ASAP is not allowing more than the flow's maximum bandwidth(32kbps and 128kbps) even though it has available bandwidth.
In SWAN : Because of misbehavior of one flow, SWAN degrades the throughput of the other flow also. This is because it does not reserve any resources for any flow. So it is not giving any QoS guarantee when congestion occurs.
In DiffQ-DSR : The flow f1 which is misbehaving only gets affected and the other flow is getting its QoS assurance. But in this scheme also, if both the flows belong to the same class and in that one is misbehaving, then both the flows get affected. This is because we are providing per class QoS provisioning and is similar in effect to Diffserv.

4.2.3 Call Acceptance Ratio(CAR)

We calculate call acceptance ratio in our scheme as the number of flows accepted. In ASAP, all flows are accepted, but as Best-Effort(BE) if QoS can not be satisfied. So we calculate CAR as

$$\frac{numberofflows - numberofflowsacceptedasBE}{numberofflows} \tag{4.1}$$

We are unable to find a way of determining call acceptance ratio for SWAN. So we are not presenting call acceptance ratio for SWAN. We have done simulations by varying number of flows for each density. We have taken all these flows with CR of 64kbps. These results are presented in Figures 4.8(a) and 4.8(b)

From the figures we can observe that call acceptance ratio for DiffQ-DSR is higher than ASAP. This is because of the following reason: ASAP reserves the resources up to the flow's maximum bandwidth requirement if it has sufficient bandwidth whereas

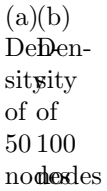

Figure 4.8: Call acceptance ratio by varying number of the flows

we are reserving only minimum bandwidth requirement of the flow. However DiffQ-DSR transmits packets at peak rate also as along as bandwidth is available, but marks them with higher drop precedence.

As density increases DiffQ-DSR's call acceptance ratio decreases. This is because as density increases the QRREP packets may be dropped more because of collisions. So less number of flows get accepted. But even in this case, the call acceptance ratio of DiffQ-DSR is more than the call acceptance ratio of ASAP without adaptation.

4.2.4 Latency to Start Data Plane Operations

Figures 4.9(a), 4.9(b) and 4.9(c) show the average latency to start data plane operations of all the five connections for 50, 75 and 100 nodes respectively for pause times 5, 15 and 30 sec respectively. Here we are not presenting latency of SWAN because we are unable to find that parameter from the implementation that I have downloaded for SWAN.

Actually the latency for DiffQ-DSR in the graph represents the time taken to find the route and to reserve the resources whereas, the latency for ASAP in the graph represents the time taken to find the route only. This is because in the implementation of ASAP that I have downloaded, data packets are sent even before reserving the resources on the path. In some scenarios DiffQ-DSR latency is higher than ASAP latency. But if we consider the time taken for reserving resources in ASAP, it should be higher than that of DiffQ-DSR.

4.2.5 Scalability

In this experiment we have done simulation for the density of 50 nodes with 15 connections for DiffQ-DSR and ASAP. The QoS levels we have taken for this simulation are shown in Table 4.3. We changed the application rates so that more

(a) Density of 50 nodes with 5 connections

(b) Density of 75 nodes with 5 connections

(c) Density of 100 nodes with 5 connections

Figure 4.9: Latency to start data plane operations

connections can be started without saturating the network. This will allow us to determine the effect of scalability properly since the number of queues maintained by DiffQ-DSR is one fourth that of ASAP. Among 15 connections, we have taken 5 connections of class-2 type, 5 connections of class-3 type and remaining 5 connections of class-4 type. The cumulative throughput for each QoS level is shown in Figure 4.10

Figure 4.10: Average Throughput for each QoS level for the density of 50 nodes with 15 connections

The number of meters, policers and queues used by DiffQ-DSR is restricted to number of QoS levels i.e here it is 4, whereas in ASAP as there are 15 connections, it uses that many number of meters, policers and queues at nodes. That means DiffQ-DSR drastically reduces the processing and storage overhead at every node as number of connections increases. This is more important in power constrained MANETs. From the Figure 4.10 we can observe that throughput for DiffQ-DSR is less than the throughput of ASAP with no adaptation. This is because as we have already explained in the previous sections, ASAP with no adaptation sends date at the rate of flow's

Table 4.3: QoS class levels

Classes	*Range(kbps*
Class-1	*best-effort*
Class-2	*2-4*
Class-3	*4-8*
Class-4	*16-32*

maximum bandwidth requirement(32kbps). In the implementation, we do not know how to control sending rate of ASAP with no adaptation. So, we just increased the sending rate of DiffQ-DSR to flow's maximum bandwidth requirement and calculated the throughput. Then we observed that it is greater than the throughput of ASAP with no adaptation. This is shown in Figure 4.10 for CR-16k. In DiffQ-DSR the average cumulative throughput for class-4 is 10kbps whereas its committed rate is 16kbps. This is because among the 5 flows of class-4 some flows are affected more by interference and their throughput is around 3-4kbps. So because of these flows we are getting average throughput less than the committed rate.

4.2.6 Average End-to-End Delay

End-to-end delay is calculated as the difference between the time of reception at the destination and time of transmission at source. So, all packets that are lost are not considered as part of end-to-end delay. Figures 4.11(a), 4.11(b) and 4.11(c) show the average end-to-end delay for five flows for 50, 75 and 100 nodes respectively with different pause times. Among 5 flows, two flows belong to CR of 16kbps and two flows belong to CR-32kbps and and the remaining flow belongs to CR of 64kbps. All these flows are of type CBR traffic. This simulation is done for only one sample. From the graphs we can observe that

1. The end-to-end delay for DiffQ-DSR is less than that of ASAP in all the cases. This is because in DiffQ-DSR the data plane operations at intermediate nodes are very light weight whereas in ASAP they are not. ASAP is a per flow mechanism and it needs to classify each packet of every flow to enqueue into the corresponding queue at intermediate nodes also. ASAP inserts a new option into the IP header to carry the flow ID with every packet and uses this to identify the queue for that flow. As flows increase, the number of queues maintained

per node also increases and the overhead to process all these queues increases. In DiffQ-DSR, intermediate nodes just check the DSCP of packets and enqueue in to the corresponding queue. The overhead to process the queues is also less because the number of queues maintained per node are restricted to number of QoS levels.

2. The end-to-end delay of SWAN is worse than DiffQ-DSR except for 100 nodes with 5 and 15 pause times. SWAN estimates congestion in the network by maintaining a running average of latency to transmit a packet at the MAC layer. This leads to overhead at nodes. SWAN treats all CBR flows as real-time traffic and enqueue all packets into the single queue. Because of this, queue length increases and packets in the queue are delayed for transmission.

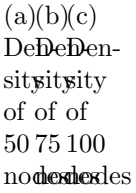

Figure 4.11: Average end-to-end delay for 5 connections

4.2.7 More Simulation Results

We have done simulations 50, 75 and 100 nodes with 5, 10 and 15 connections. For each scenario we have varied the pause time to be 5, 15, 30 sec to know the effect of mobility. We have done each simulation ten times by varying source-destination pairs of the flows. We observed that the throughput achieved saturates at around 100-120kbps. Hence we found that with 10 or 15 connections, the throughput achieved by the individual flows drops below the committed rate for all protocols. So we present the results obtained with 5 connections only. For all these simulations, the QoS classes we have taken are shown in Table 4.4. In the five connections, we have taken two connections of class-2, two connections of class-3 and and one connection of class-4 level. We are sending all these flows at their minimum bandwidth requirement. For all these scenarios we calculated the overall throughput and overall packet delivery ratio for all the connections over the entire simulation time. We have calculated the average throughput for an interval of 15 sec for the entire duration of simulation. I have not includes the overall throughput and packet delivery ratio graphs as they are not of that much significance from the perspective of QoS.

Table 4.4: QoS class levels

Classes	***Range(kbps***
Class-1	***best-effort***
Class-2	***16-32***
Class-3	***32-64***
Class-4	***64-128***

Figures 4.12(a), 4.12(b) and 4.12(c) show the duration-wise throughput 50, 75 and 100 nodes with 5 connections and 30 pause time. Similar results can be observed for 5 and 15 pause times also. Hence, they are not included here. From the graphs we can observe that

1. DiffQ-DSR is providing stable QoS guarantee throughout the simulation time whereas all other schemes are not. Sometimes they are giving high throughput and at some times they are giving almost zero throughput. More analysis is needed to understand why this is happening.

2. ASAP with adaptation has less average throughput than all other schemes. This is because, with adaptation the flows reduce their sending rate according to the feedback information they got from the adaptation messages. So some times the flows do not even send data at their minimum bandwidth rate. Until they got another adaptation message, they will send at that rate only. Channel utilization is very less in ASAP with adaptation.

3. ASAP with no adaptation has more average throughput than DiffQ-DSR. This is because, while admitting flows if it has sufficient bandwidth, it sends the flows at their maximum bandwidth rate throughout the simulation time and it is not adapting during the simulation.

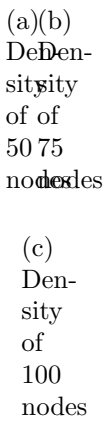

Figure 4.12: Average Throughput for each time interval(15sec) with 5 connections and 30 pause time

4.3 Summary

From the results and analysis presented in the previous section, we can conclude that

- Under most conditions tested, DiffQ-DSR gives the best throughput and packet delivery ratio for the CR-64kbps flows. Thus, it performs best for high priority QoS flows compared to ASAP and SWAN. It is only under high density and mobility conditions, it performs marginally worse than the other two schemes. For the other flows also, in most conditions, it has better throughput and packet delivery ratio than SWAN and ASAP with adaptation. It has worse throughput than ASAP without adaptation but where ASAP transmits at peak rate, in DiffQ-DSR, sources transmit at committed rate.

- DiffQ-DSR provides stable QoS guarantee throughout the time whereas other schemes are not.

- When congestion occurs because of one flow, all flows are degraded in SWAN as it is not reserving any resources and is not maintaining any state information. But in DiffQ-DSR the flows which belong to the congested flow only get affected.

- DiffQ-DSR has high call acceptance ratio than ASAP with no adaptation. It is almost twice that of ASAP. But in both the schemes, the nodes do not take into consideration of the neighborhood information while calculating the available bandwidth. If we take the neighborhood information into consideration, the

call acceptance ratio may reduce but we can give QoS guarantee to the flows which were admitted.

- In DiffQ-DSR the number of meters, policers and queues maintained per node are restricted to number of QoS levels defined whereas in ASAP, as number of connections increased these are increasing proportionally. So in this way DiffQ-DSR achieves scalability.
- DiffQ-DSR has average end-to-end delay less than both ASAP and SWAN.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Providing QoS in MANETs is a challenging task. Many solutions have been proposed but the solutions which are using resources efficiently are not scalable and the solutions which are scalable are not using resources efficiently. So we proposed a solution to achieve both the things. The proposed solution is a cross-layer QoS-aware routing protocol that supports multiple classes of service. It allocates resources to each class dynamically. It maintains per flow state information but provides per QoS level granularity. So fewer number of meters, policers and queues are required. In our scheme the source node acts as an edge router- metering, marking and conditioning of data packets. Other nodes act as core routers. We are implemented the scheme in *ns-2.29* by extending Dynamic Source Routing(DSR) protocol. We compared our scheme with ASAP and SWAN in terms of call acceptance ratio, packet delivery ratio, throughput, latency to start data plane operations and average end-to-end delay.

Simulation results show that our scheme has achieved throughput close to ASAP while using fewer meters, policers and queues. Results also show that call acceptance ratio of our scheme is higher than ASAP. From the results we can also observe that our scheme gives QoS guarantee to the flows when congestion occurs whereas SWAN does not. Latency to start data plane operations is also acceptable in our scheme. Average end-to-end delay for our scheme is less than that of both ASAP and SWAN. Overall, we are performing better than ASAP and SWAN.

5.2 Future Work

We can do some optimizations and extensions to our scheme. Those are

- *Local path repairing* : Actually in our scheme whenever route break occurs, again source only will find the new route. So if intermediate nodes find the new route instead of the source, we can reduce the packet loss.
- Efficient measurement of available bandwidth by considering neighborhood information.
- Our scheme only concentrated on bandwidth provisioning. We can also extend it to delay constraint. From the results we can observe that the average end-to-end delay for our scheme is less. So if we consider delay constraint during route discovery process, we may achieve that also as already our scheme has less end-to-end delay.

Bibliography

[1] G. A. Jianbo Xue, Patrick Stuedi, "ASAP: An Adaptive QoS Protocol for Mobile Ad Hoc Networks," in *14th IEEE Proceedings on Personal, Indoor and Mobile Radio Communications, 2003. (PIMRC 2003)*, vol. 3, pp. 2616–2620, Sept 2003.

[2] Q. Xue and A. Ganz, "Ad Hoc QoS On-demand Routing (AQOR) in Mobile Ad Hoc Networks," *Journal of Parallel and Distributed Computing*, vol. 63, no. 2, pp. 192–207, 2003.

[3] Gahng-Seop Ahn, Andrew T. Campbell, Andras Veres, and Li Hsiang Sun, "Supporting Service Differentiation for Real-Time and Best-Effort Traffic in Stateless Wireless Ad Hoc Networks (SWAN)," *IEEE Transactions on Mobile Computing*, vol. 1, July-September 2002.

[4] Venus S. Y. To, Brahim Bensaou and Sammy M. K. Chau, "Quality of Service Framework in MANETS using Differentiated Services," *IEEE 58th Vehicular Technology Conference Proceedings*, vol. 3, pp. 3463–3467, October 2003.

[5] H. Badis, "Complete and Efficient Quality of service Model for MANETs," March 2007. IETF internet draft,draft-badis-manet-ceqmm-02.txt.

[6] Jayalakshmi Y, "QoS aware Routing Protocol using Diffserv Principles," June 2007. Master Thesis, DCIS, University of Hyderabad.

[7] N. Kettaf, H. Abouaissa, T. Vu Duong, P. Lorenz, "Admission Control enabled On demand Routing (ACOR)," 2007. IETF Internet-Draft, draft-kettaf-manet-acor-02.txt.

[8] C. Perkins and E. Royer, "Quality of Service in Ad hoc On-Demand Distance Vector Routing," Nov. 2001. IETF Internet Draft, draftperkins-manet-aodvqos-00.txt.

[9] Seoung-Bum Lee and Gahng-Seop Ahn and Xiaowei Zhang and Andrew T. Campbell, "INSIGNIA: An IP-Based Quality of Service Framework for Mobile Ad Hoc Networks," *Journal of Parallel and Distributed Computing*, vol. 60, no. 4, pp. 374–406, 2000.

[10] Bosheng Zhou, Marshall A, Tsung-Han Lee, "A Cross-layer Architecture for DiffServ in Mobile Ad-hoc Networks," in *International Conference on Wireless Networks, Communications and Mobile Computing, 2005*, vol. 2, pp. 833–838.

[11] H. Xiao, W. K. G. Seah, A. Lo, and K. C. Chua, "A Flexible Quality of Service Model for Mobile Ad Hoc Networks," *Vehicular Technology Conference Proceedings, 2000. VTC 2000-Spring Tokyo. 2000 IEEE 51st*, vol. 1, pp. 445–449 vol.1, 2000.

[12] D. Johnson, D. Maltz, and J. Broch, *Ad Hoc Networking*, ch. DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pp. 139–172. Addison-Wesley, 2001.

[13] C. Perkins and S. Daas and E. Belding-Royer, "*RFC 3561*: Ad Hoc On-Demand Distance Vector Routing," Jul 2003.

[14] Jaya Lakshmi Y, Anupama Potluri and Atul Negi, "QoS-aware Routing Scheme for MANETs using Diffserv Principles," in *proceedings of ICACC, Madurai*, pp. 99–111, Feb 2007.

[15] Mohammad Haq, Masakatsu Kosuga, Jacir Bordim, Shinsuke Tanaka, Mitsuji Matsumoto, "Admission Control and Service Differentiation Based QoS Provisioning for Mobile Ad hoc Network." Med-Hoc-Net 2004, The Third Annual Mediterranean Ad Hoc Networking Workshop, 2004. JAPAN.

[16] "Network Simulator, ns2 version 2.29," Oct. 2005. University of Southern California/LBNL/VINT, http://www.isi.edu/nsnam/ns/index.html.

[17] Dorgham Sisalem and Srikanth Krishnamurthy and Son Dao, "DiffRes: A reservation protocol for the differentiated service model," tech. rep., HRL Laboratories, CA, USA, December 1999.

[18] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services." RFC 2475, IETF DiffServ Working Group.