CRESQ: Providing QoS and Security in Ad hoc Networks

Puneet Sethi
Ubinetics India Pvt. Ltd.,
Bangalore -560 027
e-mail: Puneet.Sethi@ubinetics.co.in

Gautam Barua
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, Guwahati
- 781039
e-mail: gb@iitg.ernet.in

Abstract—**Ad hoc networks, which have seen drastic increase in their usage scenarios and convergence of different applications' traffic lately, are getting ready to support QoS and secure traffic. Existing protocols for ad hoc networks provide little or no support for QoS and security. We present a new routing protocol 'CRESQ' for ad hoc networks, with adequate support for QoS using resource reservation. Ad hoc networks pose certain unique challenges for QoS implementation, in addition to those presented by the wireline and wireless scenarios where resource reservation, high speed of the mobile nodes, and frequent hand-offs from the base-stations are the issues. Additional challenges in ad hoc networks are attributed to mobility of intermediate nodes, absence of routing infrastructure and, low bandwidth and computational capacity of the nodes. The proposed protocol 'CRESQ' takes all these limitations into account and provides a general** *framework* **for implementation of QoS. CRESQ, which is a cluster based routing protocol, effectively uses the clustering to minimize the routing overhead and to provide QoS guarantees. CRESQ is a source routing protocol, with an ability for localized route recovery to minimize route and QoS re-establishment delay. Simulations have shown that CRESQ performs considerably better than the existing protocols, in terms of** *packet drop ratio* **and** *routing overhead.*

Keywords: **ad hoc networks, routing, QoS, clusters.**

I. Introduction

An ad hoc network is a collection of mobile hosts with wireless interfaces, which can dynamically form a multihop wireless network using peer to peer communication without the aid of any fixed or pre-existing infrastructure. Such networks have the ability to provide a quick and cheap communication link, hence they find useful applications in military surveillance, rescue operations, ubiquitous computing and disaster recovery. We can see that all such and many more applications provide a scenario quite different from the scenarios of wireless networks with fixed infrastructure, where nodes are characterized by their high speed and frequent hand-offs from the base stations. Constraints in ad hoc networks usually arise due to low computational and bandwidth capacity of nodes, mobility of intermediate nodes in an established path and absence of routing infrastructure. A routing protocol or a QoS scheme for ad hoc networks should focus on these very problems and is expected to be: 1) Distributed in nature, so that no fixed infrastructure is required, 2) Computationally inexpensive, as nodes may be poor in computational resources, 3) Efficient in reducing the route discovery and recovery time.

From a military surveillance scenario, where security of the data is of prime importance, to ubiquitous computing where real-time QoS requirements may have to be met, we can deduce that QoS and security are of great importance in ad hoc networks. However, it is non-trivial to implement a QoS framework in ad hoc networks due to lack of infrastructure. Routing protocols like AODV, DSR, DSDV, TORA and many others exist for ad hoc networks, but none of them provides a framework for implementation of QoS. It is really challenging to provide a QoS solution which will not compromise the efficiency of a routing algorithm to deal with the inherent problems of ad hoc networks. We attempt to provide a QoS framework for ad hoc networks and also present the simulation results which prove the worth of the scheme.

We present a new cluster based routing protocol CRESQ, which provides adequate support for QoS and security. We also provide a broad design for implementing QoS over CRESQ in the form of a QoS layer on top of the CRESQ layer. CRESQ incorporates clustering in its design as it helps in providing QoS guarantees, minimizing QoS re-establishment and, minimizing routing overhead and delays. The complete CRESQ design consists of a initial clustering algorithm, a cluster management algorithm, route discovery and establishment algorithm, and a route maintenance algorithm. For want of space we mention the clustering algorithms in brief and the details can be found in [14]. In the next section, we describe the CRESQ algorithm and other relevant details. In Section III we present a QoS framework that can be implemented over CRESQ. In Section IV we present the results of simulations with appropriate discussions. In Section V, we describe our plans for future work and in Section VI, a conclusion of the work is provided.

II. CRESQ: *C*luster based *R*outing for *E*nd-to-end *S*ecurity and *Q*oS *S*atisfaction.

The new proposed routing protocol **"***C*luster based *R*outing for *E*nd-to-end *S*ecurity and *Q*uality of service satisfaction - CRESQ**"** is a cluster based routing protocol which considers QoS parameters before making a connection and also makes the source aware of the intermediate nodes, in case security is desired. The routing protocol provides computationally inexpensive source routing, with support for speedy route recovery through localized node substitution.

A. Terms

- *Cluster* - It refers to a collection of nodes, grouped for the functioning of CRESQ. A cluster may be referred to by its Master (see below).
- *Master* - Every cluster is characterized by a unique node called its *master.* It has certain extra responsibilities.
- *Bridge* - *Bridge* is a node which belongs to more than one cluster. It thus has more than one master.
- *Slave* - All cluster nodes other than bridges and master, are called *slaves.* Each *slave* has only one master (denoted by M_S for *slave* S) and hence belongs to only one cluster.
- *State* - A node's *state* describes whether the node is a slave, bridge, master or *none* (*none* means the node is uninitialized, i.e. it does not belong to any cluster).We will also refer to a node as slave, if its *state* is slave (similarly for bridge, master, none).
- *Mapping* - On detecting a link failure to B from A, A informs M_B or M_A depending upon its state. M_B (or M_A) may suggest a node C to A as a substitute for B. A will now forward all the traffic of the corresponding connection to C, which is called a *mapping*

for B.

B. CRESQ Algorithm

The overview of the CRESQ algorithm is as follows:

1. Initially all the nodes in the network are put in state none. Then the whole ad hoc network is clusterised using a clustering algorithm described in section II-C.
2. Once the initial clustering is done, then the cluster management algorithm takes over which is described in section II-D. The cluster management algorithm accomplishes the task of maintaining the clusters using periodic transmission of HELLO packets.
3. Whenever a slave S wants to establish a connection to a node D, S sends a ROUTE REQUEST packet to its master M_S specifying the destination and parameter values for the *QoS specification*[1]. M_S then broadcasts the packet to its bridges.
4. All the bridges of M_S further broadcast the packet and the packet is received by all of their masters, which further broadcast to their bridges. This process continues till a master of D hears the request, which adds the destination's address in the *Addr_List*[2] of a ROUTE REPLY packet and sends it to the node from which it heard the ROUTE REQUEST.
5. Each of the nodes (except M_S, which directly forwards ROUTE REPLY to S) which receive the ROUTE REPLY packet, sends it to the node from which it heard the ROUTE REQUEST after checking the QoS constraints and adding in the *Addr_List*, its own address (if it is a bridge*)* or a slave's address (if it is a master). (See section II-E.1)
6. When S receives the ROUTE REPLY, it simply uses the path specified in *Addr_List* to route its data packets.

Note: Cases where S is a master or bridge are appropriately handled in the actual algorithm. ROUTE REQUEST loops that may be generated, are also avoided by suitable mechanisms. All these and other details are omitted in order to maintain clarity.

C. Initial Clustering

An ad hoc network is formed, when nodes with wireless capabilities start discovering each other. The discovery may be initiated by any node, and is usually triggered by an application level program. It is at this point that initial clustering has to be done. When a node discovers the neighbouring nodes, it may send *trigger* messages to all of them. All of the receiving nodes start the discovery procedure and flood the *trigger* message.

In the process, all the nodes become aware of n-hop (n may be 1,2,3..) information of their neighbours. Now a clustering algorithm may be run at each node to initialise its *state*. However, this does not place any restriction on the addition of nodes to the network, as new nodes are handled appropriately by the cluster management algorithm, if they enter with state none. The choice of an initial clustering algorithm is really important for efficient performance of CRESQ, but CRESQ is also able to perform without any initial clustering with all the nodes initially set to none.

Some research is still required to devise performance benchmarks for efficient initial clustering, but we suggest certain performance criteria and a suitable initial clustering algorithm in [14].

D. Cluster Management Algorithm

Once the clusters are established, it is up to the cluster management algorithm to maintain them. The cluster management algorithm works by periodic transmission of HELLO packets. A broad overview of the cluster management algorithm is as follows:

1. Each node periodically transmits HELLO packets, to make the relevant neighbours aware of its presence. (HELLO packets may not be transmitted, if some other packet was sent in the recent past).
2. Nodes which receive HELLO[3] packets from their cluster partners, use them to update the information about the *state*, QoS capabilities of the sender node[4].
3. If no packet is received from a node A in the cluster for a certain duration, then A is expelled from the cluster.
4. If a slave or bridge comes into contact with a new master, then it adds the new master in its list of masters (subject to resource constraints). The new master and other masters are informed by HELLO packets.
5. If a master moves into the cluster of another master, then the incoming master is made a slave of the other master.[5]
6. If a slave or bridge loses all of its masters (or a master loses all of its slaves and bridges*)*, then it is set to state none.
7. A node in none state periodically transmits HELLO packets and hears HELLO packets from other masters (and after a certain time it also listens to HELLO packets from other slaves and bridges). Eventually, it becomes a slave to some master (or a master of some slaves or bridges).

Efficiency of the cluster management algorithm can be greatly improved with the help of lower layers like 802.11, which report link failure to the routing layer. This part of the cluster management algorithm has been clubbed with Route Maintenance in section II-E.3.

Note: Many details including contents of HELLO packets, scheduling and postponing transmission of HELLO packets, identification of the incoming master*,* certain customisations in *cluster* management for CRESQ etc. have been hidden and can be found in [14].

E. Route Discovery, Establishment and Maintenance

E.1 Route Discovery

Whenever a source S wants to transmit data to destination D and does not have a cached route to D, it initiates the route discovery process by sending a ROUTE REQUEST packet. If S is a slave, then it unicasts[6] the ROUTE REQUEST packet to its master. If S

[1] *QoS specification* captures the application's QoS requirements, by assigning values to the parameters (like bandwidth, priority etc.) in the *specification* itself.

[2] *Addr_List* is a field in a ROUTE REPLY packet, which will eventually contain the list of nodes to be traversed in order to reach D.

[3] HELLO packets also contain information like *state* of the node, masters of the node if node is a bridge, QoS capabilities of the node etc.

[4] Only masters maintain information about state, QoS capabilities of other cluster nodes. Slaves and bridges just need to maintain their list of masters.

[5] Frequent changing of masters is avoided if it deteriorates the performance of the routing protocol. For example when a slave A of a master M is handling a connection, then *state* of M is not changed until M comes to know that A is out of reach.

[6] Here, unicasting does not imply more MAC load in terms of CTS, RTS in 802.11. Since the sender is identified by its packet and each receiving node pro-

is a bridge or a master or a none, then it broadcasts the ROUTE REQUEST packet. Every node which receives the ROUTE REQUEST packet acts, based on its state, in the following manner:

1. Every slave or none node discards the packet.
2. Every bridge *checks*[7] if the packet is from its master and it has not recently heard that request[8] and *TTL*[9] field has not expired and it can provide the required QoS:
 (a) Store the *previous hop* of the ROUTE REQUEST and QoS *specification* and broadcast the ROUTE REQUEST packet (after decreasing *TTL* field).
3. Every master *checks* if the packet is from its own slave or bridge[10] and it has not recently heard that request and *TTL* has not expired:
 (a) If D is in its cluster,
 i. then, it sends a ROUTE REPLY to the *previous hop* of ROUTE REQUEST, after adding D's address in the *Addr_List* of the ROUTE REPLY packet.
 (b) Else, it broadcasts the packet after decreasing *TTL* field and storing the *previous hop* of ROUTE REQUEST.

After transmitting the request, S waits for REPLY_WAIT_TIME seconds. If it does not receive the reply within that period, then it may chose to re-transmit ROUTE REQUEST or to inform the upper QoS layer about the network's inability to provide the required QoS, which may then suggest lesser QoS levels.

E.2 Route Establishment

A route is said to be established whenever the source S becomes aware of the path that its data packets need to take in order to reach the destination D. This will be accomplished once the master of D i.e. M_D hears the ROUTE REQUEST packet in the route discovery process. On hearing the ROUTE REQUEST, M_D unicasts a ROUTE REPLY packet to the *previous hop* of ROUTE REQUEST packet after adding D's address in the *Addr_List* of ROUTE REPLY packet. Now every node which receives the ROUTE REPLY[11] packet acts based on its state, in the following manner:

1. Every none node discards the packet.
2. Every slave *checks* if the ROUTE REPLY is from its master and it is 'S' (i.e. source of connection):
 (a) then, it caches the *Addr_List* of the ROUTE REPLY packet. Then it dequeues all the packets destined for D and places the route in their packet header and puts them on air. (All the data packets get routed by source routing at the intermediate nodes).
3. Every bridge *checks* if the ROUTE REPLY is from its master:
 (a) If it is not 'S',
 i. then, it forwards the ROUTE REPLY packet to the *previous hop* of the ROUTE REQUEST after adding its own address in the *Addr_List*, provided that it satisfies the QoS requirements.

cesses packets only from certain cluster nodes, so only the appropriate node i.e. master will process the packet, even if it is broadcast.

[7] If this *check* fails, then the node drops the packet.

[8] If a similar ROUTE REQUEST arrives before REPLY_WAIT_TIME seconds have expired after the receipt of an earlier one, then it is rejected as a duplicate.

[9] *TTL* (Time to live) field is initially set to MAX_REQ_DEPTH, and is decremented at each hop. Packet is dropped, if *TTL* becomes zero.

[10] If the packet is from a none node, then the master adds the node as its *slave* and informs it using HELLO packets.

[11] ROUTE REPLY is usually unicast, which also helps in maintaining the consistency of the *cluster* information maintained by different nodes.

 (b) Else, it caches the *Addr_List* and dequeues all the packets destined for D and puts them on air with the route specified in their header.
4. Every master *checks* if the ROUTE REPLY is from its bridge:
 (a) If it is not 'S',
 i. then, it forwards the ROUTE REPLY packet to the *previous hop* of the ROUTE REQUEST after adding the address of its slave[12], which satisfies the *QoS requirements* in the *Addr_List* (provided that it finds such a slave, otherwise it drops the packet).
 (b) Else, it caches the *Addr_List* and dequeues all the packets destined for D and puts them on air.

Note: Whenever a master adds its slave to the *Addr_List*, it updates the data structures belonging to that slave to reflect the available QoS capabilities (like bandwidth etc.). So, a slave need not be informed about the reservations made on it, because its master will make sure that no slave is handling more than its capabilities.

Certain clustering information is also piggybacked in the routing packets like state of the node, list of masters (for bridges) etc. This helps in cluster maintenance and in reducing wastage of bandwidth.

E.3 Route Maintenance

The route maintenance process helps in correcting a failed route through some local modification in the route. Usually, a route failure in an ad hoc network is caused by the movement of an intermediate node. Hence a route is broken only due to a change in a small fraction of the whole route. Unlike other protocols like AODV([5]) and DSR([4]) in which the source is informed of the failure and it may have to redo the route discovery, CRESQ tries to minimize QoS re-establishment delay by a local perturbation. The process is initiated whenever the lower layers like 802.11 report a link failure to CRESQ. On detecting a link failure on a packet 'p', the node acts based on its state, in the following manner:

1. Every none node *checks* if 'p' is a data packet:
 (a) Send a ROUTE ERROR packet to the source of 'p'.
2. Every slave executes the following sequence:
 (a) If the link to its master has failed,
 i. then mark self as none.
 (b) Else, if 'p' is a data packet
 i. then enqueue 'p' and send a FAILURE packet to the master.
3. Every bridge executes the following sequence:
 (a) If the link to its master has failed,
 i. then, remove the master from the stored list.
 (b) Else, if 'p' is a data packet,
 i. then enqueue 'p' and send a FAILURE packet to the master of the next hop in the route. (The master of the next hop in the route can be known from the route specified in the data packet. While forwarding the ROUTE REPLY, a master in addition to specifying the slave address also mentions its own address in the *Addr_List*).
4. Every master executes the following sequence:
 (a) If the link to its slave or bridge has failed,
 i. then, remove the node from the list.
 (b) If 'p' is a data packet,

[12] If it is not able to find a *slave*, then it might add its own address in the *Addr_List* if it can provide the QoS. The addition of a *slave*'s address can be avoided, if the master deems it right. For example, if there is more than one common master between the node of the previous hop and the sender of the ROUTE REPLY packet.

i. then, if successful in finding a substitute B for the node A whose link has failed, then store B as a *mapping* for A and forward all data packets to B, for this connection.

ii. Else, send a ROUTE ERROR packet back to the source.

On receipt of a FAILURE packet, master M tries to find out a substitute for the node whose link has failed and informs the node N which sent the FAILURE packet, using a FAILURE REPLY packet. N stores this new *mapping*, and forwards all packets directed to the earlier node to this *mapping*. If M is unable to find a substitute, it advises N to send a ROUTE ERROR packet back to the source.

Note: Details of how an alternate slave or bridge chosen, action to be taken in case of a link failure on sending a FAILURE Packet, etc. have been omitted here and can be found in [14].

III. QoS and Security framework

We present the QoS framework in terms of two layers, upper QoS layer and lower CRESQ layer. Figure 1 shows a typical interaction between the QoS layer and CRESQ during the initiation of a connection.

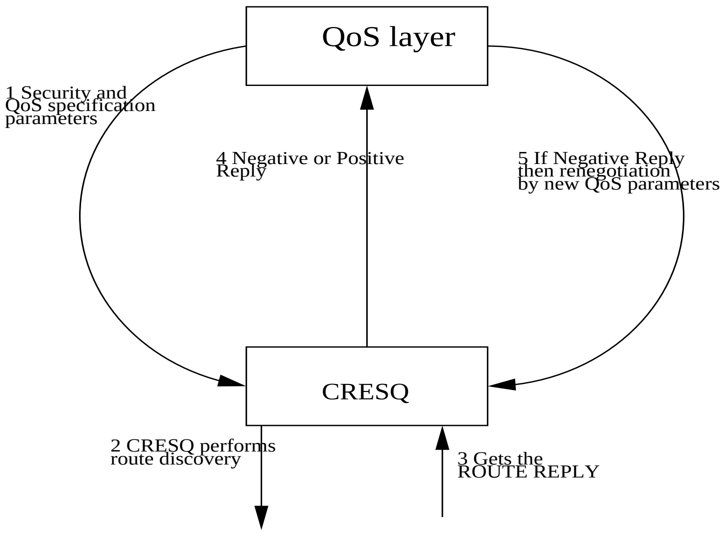

QoS layer
CRESQ
1 Security and QoS specification parameters
2 CRESQ performs route discovery
3 Gets the ROUTE REPLY
4 Negative or Positive Reply
5 If Negative Reply then renegotiation by new QoS parameters

Fig. 1. Typical interaction during route discovery

During connection initiation, the QoS layer in node 'S' (source) gives the values of parameters in the *QoS specification* to the CRESQ layer. CRESQ layer then performs the route discovery, by including these parameter values in a ROUTE REQUEST's header. Necessary reservations take place during the forwarding of ROUTE REPLY packets. A bridge forwards the ROUTE REPLY, only if it can satisfy the QoS and makes reservations on itself using a *QoS Mapping*[13]. When a master forwards a ROUTE REPLY, it chooses a slave which will be able to satisfy the required QoS levels and updates its data structures to reflect the final available QoS resources on the slave, after making reservations on that slave (Note that only masters and bridges can forward ROUTE REPLY). The CRESQ layer in node 'S' waits for REPLY_WAIT_TIME seconds before re-transmitting a ROUTE REQUEST. It may transmit a ROUTE REQUEST for MAX_REQ_TRIES times, after which it can reply in the negative to the QoS layer in 'S'. If the CRESQ layer in S is able to discover the path, then it may start sending the data, and may inform the QoS layer positively. The QoS layer may wish to negotiate by specifying lower performance levels on getting a negative answer.

For implementation of security, the source node 'S' may specify levels of security in the *QoS specification* which can be suitably interpreted to carry out authentication, encryption etc. We can also have stricter security provisions, as CRESQ provides a way for S to specify certain nodes that should or should not lie on the path, along with the *QoS specification.* This can be implemented by the masters, who will receive ROUTE REPLY packets from chosen bridges, and will select slaves on the basis of directions given in the QoS specifications. The security concerns, while forwarding a ROUTE REQUEST to the masters can be taken care of, in the following manner:

- If the master M_S of the source S is not reliable, then S marks itself as none and looks for a reliable master.
- 'S' forwards a ROUTE REQUEST to only a reliable master. All reliable nodes which receive the ROUTE REQUEST, forward it to only reliable nodes and will receive ROUTE REPLY only from reliable nodes. This is in lieu of broadcasting of ROUTE REQUEST packets.

Note: The layered structure of the framework proposed here is only for the ease of presentation. In fact the QoS layer can be an application layer or it can even be merged with CRESQ layer. Also, the given QoS framework presents a scenario of source initiated reservation and a similar destination initiated scheme can be devised.

IV. Simulations and Results

We have implemented the CRESQ routing module in the ns-2([9]) simulator. The routing module so implemented handles routing and cluster management as per the design of CRESQ, but the initial clustering module is not a part of the routing agent. The CRESQ implementation tries to read the initial cluster configuration from a file generated by an initial clustering module and this segregation also helps us in analysing the effects of different initial clustering algorithms on the performance of CRESQ. The implementation also makes no assumptions of the *QoS specification*, and a master just selects a slave handling the least number of connections from its cluster while forwarding the ROUTE REPLY. Simulations were run to compare CRESQ with existing routing protocols like AODV, DSR, TORA and DSDV. The performance metrics chosen for the simulations were *packet drop ratio*[14] and *routing overhead*[15].

In all our simulations IEEE 802.11([12]) was used as the MAC layer and the radio model used characteristics similar to a commercial radio interface, Lucent's WaveLAN([11]).

A. Movement and communication model

Our simulations were run for N (N=50, 100) nodes in a L$\times$ L (L=600, 800) area, and the movement model used was the ran-

[13]*QoS Mapping* gives the required reservations to be made in terms of the quantum of resources (like router queues, bandwidth capacity etc.), for the given *QoS specification.*

[14]*packet drop ratio = # of packets dropped/ #of packets sent*

[15]*routing overhead refers to the total number of routing packets sent in the whole session.*

dom way-point model. In this model, nodes are initially placed at random positions and then they start moving towards a random destination with a speed ranging from 0 to a maximum value. On reaching their destination, they wait for a *pause time* and again select a new destination. The mobility of the ad hoc network varies inversely with the change in the value of *pause time.*

As the goal of the simulations was to compare the performance of routing protocols and to calculate the optimum value of certain parameters, we chose the traffic to be Constant Bit Rate (CBR). The CBR sources and destinations (10 CBR sources/50 nodes) were spread randomly over the network. Packet size was set to 512 bytes with a rate of 4 packets per second. We did not chose TCP sources, because TCP varies the data load based on its perception of the network capacity. This hampers an independent analysis of the routing protocol.

B. Parameter Estimations

Since we did not know beforehand optimum values of certain parameters involved in the CRESQ algorithm, we ran simulations to predict optimum values for the same. The values of these parameters are very critical to the performance of the CRESQ routing protocol. For example a low value of HELLO_INTERVAL will increase the clustering overhead and will worsen the packet drop ratio. A large value for the same parameter, leads to inefficient clustering due to infrequent updates.

B.1 Estimation of MAX_MASTERDEG and MAX_MASTERS

MAX_MASTERDEG refers to the maximum total number of bridges and slaves that a master can have. A very low value (say 1 or 2) for this parameter, undermines the benefits of clustering and makes localized correction difficult. A high value overloads the master and makes it a bottleneck. For a moderate mobility 50 node scenario, an optimum value was found to be 10.

MAX_MASTERS refers to the maximum number of masters that a bridge can have. A high value increases the span of the route discovery but it overloads the bridges. Simulations (for 50 nodes) have shown that a value of 9 is optimum for the parameter in a moderate mobility scenario.

B.2 Estimation of MAX_REQ_DEPTH and REPLY_WAIT_TIME

MAX_REQ_DEPTH is the maximum number of hops that a ROUTE REQUEST can travel before being discarded. Values of 5 and 8 were found to be optimum for 50 and 100 nodes respectively. It was observed from different sets of simulations that the optimum value was very sensitive to network configuration. Based on these results, we can modify the CRESQ to dynamically determine the MAX_REQ_DEPTH. The route discovery process is started with a lower value, and then it is incremented if no ROUTE REPLY is received for REPLY_WAIT_TIME seconds. When, a ROUTE REPLY is received for some value of MAX_REQ_DEPTH, then it is stabilized at that value for certain time for that destination.

REPLY_WAIT_TIME is the time for which a source will wait before re-transmitting the ROUTE REQUEST packet. Its value is dependent on the MAX_REQ_DEPTH. An optimum value was found to be 1.0 seconds for the 50 node moderate mobility network. However, a technique similar to the one suggested for MAX_REQ_DEPTH determination should be adopted for REPLY_WAIT_TIME.

B.3 Estimation of HELLO Parameters

HELLO_INTERVAL is the average time interval between transmission of two consecutive HELLO packets by a node. An optimum value of HELLO_INTERVAL was found to be 1.0 seconds.

ALLOWED_HELLO_LOSS is the number of HELLO packets that can be assumed to be lost, before expelling a node from the cluster. A value of 3 was found to be optimum.

A possible enhancement to the current scheme will be to have a different parameter for masters. Since masters transmit more packets in the network, we can have a separate parameter ALLOWED_HELLO_LOSS_FOR_MASTER, which will determine the expiry time of masters.

C. Comparison of CRESQ, AODV, DSR, DSDV and TORA

Figure 2 shows the comparison between CRESQ, AODV[5], DSR[4], DSDV[2] and TORA[3] in terms of *packet drop ratio* vs *pause time* and *routing overhead* vs *pause time.* Considering the *packet drop ratio,* it can be seen that all the protocols drop lesser packets when the mobility is lower. However, AODV, DSR and CRESQ outperform all others in low mobility scenarios. In moderate and low mobility scenarios CRESQ performs better than DSR (even though both are source routing algorithms) because of its ability to locally correct routes. Performance of DSR also worsens due to the problem of stale caching[6]. However, it can be seen that performance of CRESQ will be greatly improved once the implementation also includes specific parameters for QoS *specification* and information for QoS capabilities of nodes is made available.

Considering the *routing overhead,* CRESQ outperforms all others because of effective use of its clustering. DSR is found to have a lower overhead than AODV and TORA, because of its aggressive caching.(*Routing overhead of* DSDV which is constant and greater than AODV is not shown).

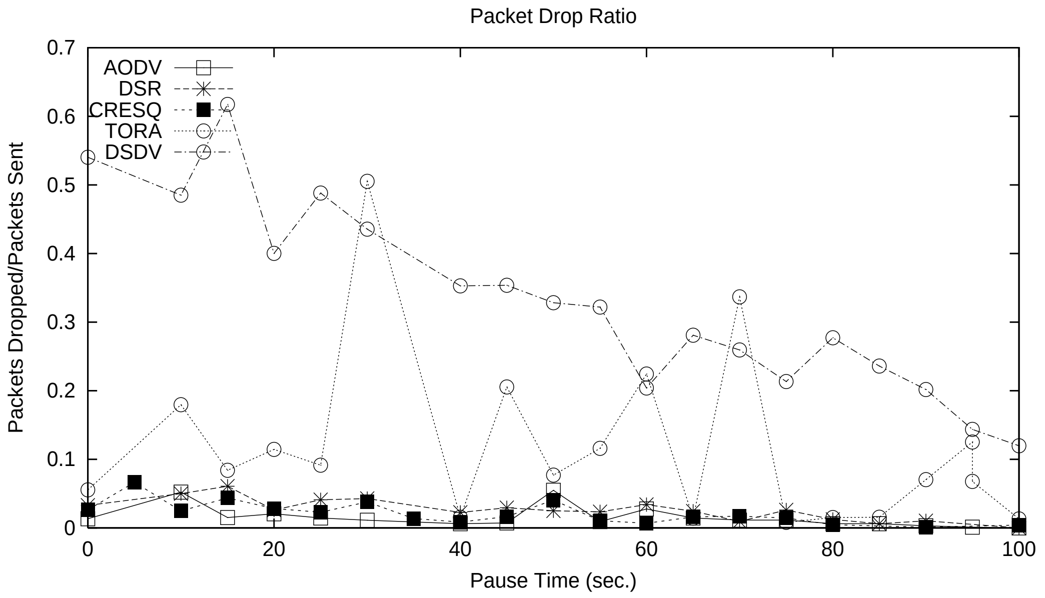
V. Current Status and Future Work

A. Multicasting support

One of the possible enhancements to CRESQ can be developing support for multicasting. In a source initiated scenario (like ODMRP[13]), a source can send JOIN_QUERY to its master, which can further deliver it to other clusters. In a destination initiated scenario, a node can send JOIN_REQUEST to its master, which can forward it to other masters if it is not in the multicast tree.

B. Co-Existence with AODV

CRESQ can be made to co-exist with AODV in an ad hoc network. Since AODV only uses HELLO packets for its neighbour detection, it can be incorporated in the CRESQ network without any extra overhead. Bandwidth and router queues can be shared between the two routing agents in every node.



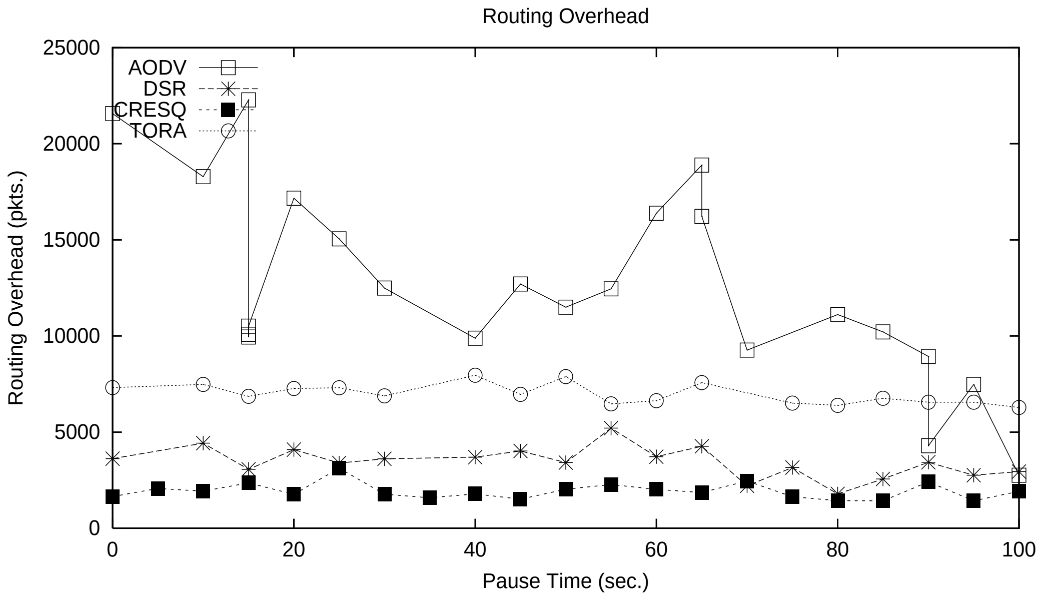


Fig. 2. Comparison of CRESQ, AODV, DSR, DSDV and TORA.

C. *Performance over Bluetooth*

Since CRESQ is a cluster based routing protocol, so it is expected to perform better with a MAC layer protocol like Bluetooth[8] which is inherently cluster(piconnet) based and provides some support to manage clusters. However, research has to be done in order to confirm the improvement in performance.

VI. Conclusion

CRESQ, which is a cluster based routing protocol provides an excellent platform for implementation of QoS and Security in ad hoc networks. In CRESQ, a route is established with the involvement of intermediate clusters (instead of involvement of just the nodes as in AODV, DSR etc.), so QoS can be ensured. Since in the CRESQ routing algorithm interactions are at the cluster level, things like substitution for a node which has moved, providing QoS guarantees on the basis of slaves' resources etc. are possible. It is non-trivial to implement QoS in AODV or DSR where all the interactions are at the mobile nodes' level. Also, the clustering doesn't add any significant overhead to the CRESQ's routing load. Clustering in the worst case, uses periodic transmission of HELLO packets which is also present in AODV (see section V-B). Simulations show that even without the availability of QoS information, CRESQ performs comparable to the existing protocols. Existing ad hoc routing protocols have been observed to suffer from problems like large route recovery delays, considerable routing overhead and, overloading on computational and memory resources of the nodes. Certain key advantages of CRESQ over existing protocols, can be summarized as:

- Lesser load on nodes, as they are not expected to maintain any tables or to perform any complex routing logic.
- Support for QoS and security, by making effective use of clus-

tering.

- Considerably lesser routing overhead than existing protocols, as it avoids flooding of ROUTE REQUEST packets by using clustering.
- Ability to minimize route recovery delay, by localized route correction.

REFERENCES

[1] Talukdar A.K., Badrinath B.R., Acharya A., “MRSVP: A Resource Reservation protocol for an Integrated Services Network with mobile hosts”, *In Wireless Networks Journal, February 2001.*

[2] Charles E. Perkins, Pravin Bhagwat, “Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers”, *In the Proceedings of SIGCOMM’94 Conference on Communications Architectures, Protocols and Applications, August 1994.*

[3] Vincent D. Park, M. Scott Corson, “Temporally-Ordered Routing Algorithm(TORA) version 1: Functional Specification”, INTERNET-DRAFT, *draft-ietf-manet-tora-spec-00.txt, November 1997.*

[4] David B. Johnson, David A. Maltz, John Broch, “DSR: The Dynamic Source Routing for Multihop Wireless Ad hoc Networks”, *Adhoc Networking 2001.*

[5] Charles E. Perkins, Elizabeth M. Royer, “Ad-hoc On-Demand Distance Vector Routing”, *In the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA’99).*

[6] Samir R. Das, Charles E. Perkins, Elizabeth M. Royer, “Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks”, *In the Proceedings of IEEE Infocom 2000 Conference on Computer Communications, March 2000.*

[7] L. Ramachandran, M. Kapoor, A. Sarkar, A. Aggarwal, “Clustering algorithms for wireless ad hoc networks”, DIAL C 2000.

[8] http://www.bluetooth.com/

[9] Kevin Fall and Kannan Varadhan, editors. ns notes and documentation. The VINT project, UC Berkeley, LBL, USC/ISI and Xerox Parc November 1997. http://www.isi.edu/nsnam.

[10] Barua G., Chakraborty I., “Adaptive Routing for Ad hoc Wireless Networks Providing QoS Guarantees”, *Accepted for ICPWC 2002, New Delhi, December 2002.* http://www.iitg.ernet.in/engfac/gb/acrpn.pdf

[11] Bruce Tuch. Development of WaveLAN, as ISM band wireless LAN. AT&T Technical Journal, 72(4):27-33, July/Aug 1993.

[12] IEEE Standards Department. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, IEEE standard 802.11-1997, 1997.

[13] Ching-Chuan Chiang, Mario Gerla and Lixia Zhang, “Shared Tree Wireless Network Multicast”, *In the Proceedings of IEEE Globecom’98, 1998.*

[14] Puneet Sethi, “CRESQ: Providing QoS and Security in ad hoc networks”, *Undergraduate thesis, Computer Science & Engineering Deptt., Indian Institute of Technology Guwahati, 2002.* http://www.iitg.ernet.in/engface/gb/cresq.pdf