Distributed Fault-Tolerant Quality of Wireless Networks

Larry C. Llewellyn, Kenneth M. Hopkinson, *Member*, *IEEE*, and Scott R. Graham, *Member*, *IEEE*

Abstract—A mobile ad hoc network (MANET) consists of a group of communicating hosts that form an arbitrary network topology by means of any of several wireless communication media. MANET communications represent a diversification in communication technology necessary to solve the stringent end-to-end requirements of QoS-based communication networks. Of the many challenges in this complex distributed system, the problem of routing based on a predefined set of customer preferences, critical to guaranteeing quality-of-service, is the focus of this research. Specifically, this paper modifies a cluster-based QoS routing algorithm for mobile ad hoc networks with the aim of providing fault tolerance, which is a critical feature in providing QoS in the link failure-prone environment of mobile networks. Performance of this new fault-tolerant cluster-based QoS wireless algorithm is evaluated according to failure recovery time, dropped packets, throughput, and sustained flow bandwidth via simulations involving node failure scenarios along QoS paths.

Index Terms—Fault-tolerant distributed routing, mobile computing, wireless networks.

1 Introduction

Quality of Service (QoS) is a defined level of performance in a communications network required by a type of network traffic. Strict QoS requirements are present in many network situations, such as in critical infrastructure control and military communication. Effective mobile ad hoc networks (MANETs) require QoS capabilities that provide fault tolerance and fast recovery when links fail on an intermittent or permanent basis.

MANET topologies can change often and unpredictably. Most protocols for multihop MANET routing maintain best-effort routes. High churn or node mobility can cause QoS requirements to become unachievable. Excessive node mobility can lead to topology changes before network updates can propagate [1]. Chakrabarti and Mishra call this combinatorial stability. Only combinatorially stable networks are considered in this paper.

This paper addresses stability and recoverability, two main problems in routing QoS traffic in mobile networks.

The first issue is stability. With most ad hoc wireless networks that support QoS, each node acts as a router. In many distributed reactive routing schemes, if a node does not know the QoS parameters of its neighbors it broadcasts the route request packet and the neighboring nodes share their QoS parameters using broadcast packets. The broadcast packets used to discover the QoS parameters of nodes' neighbors and negotiate QoS paths can flood the network. A clustered approach can lower this communication overhead to more scalable levels by limiting intercluster control communication to gateway nodes.

The second issue is minimizing the QoS impact due to network failures. If a supporting node fails when traffic is routed through multiple hops then, in the worst case, the connection must be rerouted from the source. This global fault-recovery method requires that the source renegotiate a new QoS path, which is costly in computation and communication. If multiple sources were using the failed node in QoS paths then each source must negotiate a new path. Despite its cost, this failure method appears to be common. By contrast, if a protocol allows intermediate nodes to repair connections locally then the associated connections will likely only suffer minor disruption. Local repair can make the difference in meeting time-sensitive deadlines. Experiments, in this paper, show that a local fault-tolerant algorithm has significant benefits over a global alternative. The local method used is similar to Chen and Nahrstedt's [2] repair algorithm where QoS connection failures are handled at the site closest to the link breakpoint. While the two protocols share this similarity, there are major differences in the repair methods used and the fault tolerance achieved. This paper presents the key features, definitions, and assumptions of the extended fully distributed cluster-based (EFDCB) routing protocol, which is a fault-tolerant extension to FDCB [3].

- *L.C. Llewellyn is with the Systems Validation Branch, HQ AFNIC/EVSS, 203 W. Losey St., Scott AFB, IL 62225. E-mail: Larry_C.Llewellyn@us.af.mil.*
- *K.M. Hopkinson and S.R. Graham are with the Department of Electrical and Computer Engineering, Air Force Institute of Technology (AFIT)/ENG, 2950 Hobson Way, WPAFB, OH 45433-7765. E-mail: Kenneth.Hopkinson@afit.edu, Scott.Graham.5@us.af.mil.*

Manuscript received 26 June 2008; revised 26 Mar. 2009; accepted 24 Dec. 2009; published online 4 Aug. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2008-06-0249.
Digital Object Identifier no. 10.1109/TMC.2010.148.

2 Related Work

2.1 Quality-of-Service Routing

Many QoS routing algorithms have been proposed.

Local Proportional Sticky Routing (PSR) was the first localized QoS routing scheme [4]. PSR is simple yet stable and is used as an alternative to global QoS routing. PSR operates in two stages: proportional flow routing and computing flow proportions. Proportional flow routing

determines the path of traffic during a cycle. When a cycle is complete, a new flow proportion is found for each path based on *blocking probabilities*.

Credit-Based Routing (CBR) uses a credit scheme that rewards a path for flow acceptance and penalizes rejection. Path selection is based on path credits where higher credit paths are preferred. CBR also monitors flow blocking probabilities for each path to use in future paths [5].

Quality-Based Routing (QBR) determines paths based on QoS metric values [6]. QBR monitors a path and translates flow values into average path qualities. QBR rewards successful flow and punishes flow error like CBR. The difference is that CBR assigns credits based on blocking probabilities while QBR uses average path quality.

Delay-Based QoS Routing (DBR) uses the average delay on a path to make its routing decisions [7]. The average path delay is used to measure the path's quality, and, upon flow arrival, the path with the least average delay is used to reroute the incoming traffic.

Stable and Delay Constraints Routing (SDCR) [8] works in two major phases: routing discovery and maintenance. Link stability and delay constraints are considered in the two phases. When in discovery, it sends a QoS request to the destination first, and selects the most stable path. If there is no stable path, it will broadcast a route request (RREQ). When the source receives a route report (RREP), it will calculate end-to-end delays and determine the best path. When maintaining source routes, SDCR monitors the network delay changes. If it receives a route error message (RRER), it will delete that route from its cache. It will then recalculate the best route for traffic.

2.2 Multiconstrained QoS Routing

Multiconstrained QoS routing, an NP-Hard problem [9], involves finding and reserving routes that satisfy multiple independent constraints. QoS routing can be centralized, distributed, or hierarchical.

Centralized routing requires that nodes maintain global knowledge at the source. The global state has to be updated frequently to cope with network dynamics.

Distributed routing algorithms can be more scalable since path computation is divided among the nodes. Many distributed schemes make routing decisions hop-by-hop, but rely on global state for QoS routing.

Hierarchical routing shares advantages of both centralized and distributed schemes. Each node maintains partial network state. Groups of nodes are aggregated for scalability. Source routing occurs at each hierarchical level to find feasible paths, with some inaccuracy [10].

2.3 Quality-of-Service Routing Using MPLS and Multicommodity Flows

Multiprotocol Label Switching (MPLS) is a QoS support scheme where packets can be labeled. Routers use the labels to forward packets along predefined paths. MPLS is often used with a multicommodity flow optimizer, which finds routes based on flow constraints [11].

Mitra and Ramakrishnan [12] use multicommodity flows for QoS routing with MPLS. Applegate and Thorup [13] show this technique can yield large routing tables. Thorup's solution requires an algorithm like Mitra's [12] as a front end. Memory size in both can be prohibitive.

TABLE 1
Sequence of Operations for Maintaining Communications

1	Each idle node broadcasts a short beacon packet at periodic intervals containing its cluster ID announcing it is an active cluster member
2	When a non-cluster member receives the packet it learns it can contact the neighboring cluster through that node
3	The receiver sends a short beacon reply packet containing its cluster ID
4	The two nodes are then gateway nodes which provide access to each others cluster

2.4 Quality-of-Service Routing Using Heuristics

Yuan [14] proposes two heuristics for multiconstrained QoS routing based on the extended Bellman-Ford algorithm (EBFA). EBFA computes a feasible path given multiple constraints, but its runtime/memory can be exponential. The motivation of both heuristics is to limit the number of optimal QoS paths maintained in each node. The time complexity of the limited granularity heuristic is $O(|N|k|E|)$ for N nodes, E edges, and k QoS constraints. The second heuristic, limited path, limits QoS paths in each node for scalability. $O(|N|2lg(|N|))$ QoS paths are stored per node, which is optimal. The probability of finding a QoS path that satisfies constraints is high. Both heuristics employ source routing and scale poorly.

3 Quality-of-Service Routing Using FDCB

Nargunam and Sebastian's fully distributed cluster-based (FDCB) [3] algorithm addresses QoS routing in MANETs. With FDCB, scalability issues in centralized routing are circumvented. The FDCB method is similar to hierarchical routing in that each cluster node only maintains QoS information for other cluster members, a fraction of the network. Thus, an increase in nodes should not significantly increase memory or runtime. Further, since global network state is shared and maintained by all, the communication overhead is greatly reduced. In FDCB, if a flow's source and destination are not in the same cluster, the source sends a route request packet to the gateway node, which forwards it to adjacent cluster(s). As long as the intermediate gateway nodes and links can support the requested QoS constraints, this process is repeated until the destination is found. The discovered path is sent back to the source and the resource reservation made. The distributed nature of FDCB allows it to avoid unmanageable shared global state. FDCB's distributed routing adds initial latency for the route discovery. Route requests may not flood the network due to its clustered architecture, but precautions are needed to ensure route queries propagate efficiently from source to destination.

Each cluster in the FDCB algorithm has the potential to obtain gateway nodes, which maintain communication with adjacent clusters via the operations in Table 1.

With FDCB there is no need for the node aggregation used in hierarchical routing since clusters need not be represented by an aggregate data structure.

Although FDCB addresses many of the difficulties with traditional QoS routing scehemes, it employs a distributed routing method, which has significant drawbacks. The paper does not discuss how failures are handled. Support for cluster joins and leaves are provided; however, the

TABLE 2
Cases of Disruption

1	The source moves out of range of the first intermediate node in the path.
2	An intermediate node moves out of range of a preceding or successive node (preceding node is on the source's side of the intermediate node, successive is on the destination's side of the intermediate node)
3	The destination moves out of range of its preceding node (preceding node is the last intermediate node before the destination)
4	Any node in the path leaves the network

problem of mitigating the impact on QoS in the event of an unpredicted node leave/failure is untreated. This event is presumably handled by the common practice of rerouting QoS traffic from the source.

Nargunam and Sebastian [3] illustrate the problems with conventional clustering where each cluster has exactly one node, the "cluster-head," responsible for organizing the cluster. Traditional cluster construction requires a cluster-head election each time one fails or leaves. If the cluster-head fails or leaves, all of its information and responsibilities become orphaned until a replacement is elected. To avoid this problem, the authors propose a distributed architecture in which each cluster member maintains a QoS parameter table for each of its cluster members and a table containing all cluster gateway nodes. FDCB has no effective way to handle connection failures. The distributed routing design presented is also ill-suited for MANET QoS applications. FDCB provide a path to developing a scalable QoS routing solution, but it could be improved in challenged environments.

4 Fault Tolerance in QoS Ad Hoc Networks

Each cluster in the FDCB routing algorithm has the potential to obtain gateway nodes, which maintain communication with adjacent clusters via the operations in Table 1.

Chen and Nahrstedt [15] propose fault tolerance techniques to reduce the impact of QoS disruptions due to link failures or network dynamics. The authors only consider applications which do not require hard guarantees [15]. Further, the authors state that many multimedia applications accept soft QoS and use adaptation techniques to reduce the level of QoS disruption [16], [17], [18]. One technique is to repair the broken path at the failed node by shifting traffic to a neighboring node and then routing around the breaking point. This method avoids the costly process of rerouting the traffic from the source. The second technique uses multilevel path redundancy, which establishes multiple paths for the same connection. First-Level Redundancy sends all data along all paths independently, which is used for "critical" QoS. Second-Level Redundancy sends data along the primary path and only uses secondary paths if the primary path is lost. It is used for connections which can tolerate a degree of QoS failure. Third-Level Redundancy is like second level except the secondary paths are not reserved; only calculated. On failure, an attempt is made to reserve the secondary path.

The repair algorithm responds to the Table 2 cases. When case 2 occurs, the preceding node broadcasts a repair-request message to its neighbors asking if any of them can take over for the defunct intermediate node. The neighbors that have links to the successive node reply with their resource availabilities to the preceding node. If, based on the replies, the preceding node finds node i has sufficient resources for that role, it adds the link from itself to node i to the routing path and then sends i a path-repair message. When i receives the path-repair message, it reserves the required resources and adds the link from itself to the successive node to the routing path. Once the path has been repaired, a path-validation message is sent to insure that the repaired path does not violate its end-to-end constraints. A path-validation message is sent to the destination which sends the message to the source. The source checks to see if the end-to-end requirements have been violated. If they have, the source reroutes the traffic or QoS negotiation with the user application takes place. The QoS ratio is the performance metric used during simulation of the repair algorithm, defined as

$$QoS\ ratio\ = \frac{total\ QoS\ time}{total\ QoS\ time\ +\ best-effort\ time}.$$

Best-effort time is the amount of time spent repairing the broken path. The x-axis is the mobility ratio, defined as

$$mobility\ ratio = \frac{total\ moving\ time}{total\ stationary\ time\ +\ total\ moving\ time}.$$

For a mobility ratio of less than 10 percent, the QoS ratio is above 95 percent. For a mobility ratio above 35 percent, the ratio is below 80 percent, which is a bad fit for highly mobile networks.

5 Problem Definition

5.1 Introduction

This paper considers the fault tolerance problem in MANETs designed to support QoS requirements. The previous section discussed two techniques that have been offered [2]; a path redundancy technique and a local repair algorithm. The protocol presented here is similar to the local repair algorithm by Chen and Nahrstedt [2] where QoS connection failures are handled at the site closest to the link breakpoint. While the two protocols share this similarity, there are significant differences in the repair methods used and the level of fault tolerance achieved. This section presents the motivation, definitions, and assumptions for the EFDCB routing protocol, which is a fault-tolerant modification to FDCB routing [3].

5.2 Goals and Hypothesis

Nargunam and Sebastian [3] propose a fully distributed algorithm, FDCB, in which clustering provides scalability by lowering the amount of information maintained at each node. FDCB addresses MANET scalability, but fails to effectively maintain QoS connections when nodes move, leave the network, or fail. Since the authors provide no details, it is assumed that when a QoS path suffers a link breakage, the source reroutes the traffic via a completely new path. FDCB also uses a distributed reactive routing technique which causes undesired packet transmission

latency for QoS routing applications. FDCB does not provide a feasible routing scheme or local fault tolerance, but serves as groundwork to that end.

EFDCB extends FDCB to provide the scalability, efficiency, and fault tolerance critical to maintain QoS connections in a mobile environment. The goal is to determine if EFDCB provides efficient QoS route recovery by testing it against FDCB. EFDCB algorithm only has to consider a fraction of the total number of network links when finding a new feasible path through local recovery in the cluster. Hence, the burden of negotiating newly calculated QoS paths, as is done in rerouting by FDCB, is significantly reduced. For this reason, the new local method is expected to have a considerable runtime advantage resulting in improved QoS route recovery time. Faster QoS recovery time equates to lower QoS disruption time, fewer dropped packets, and improved throughput.

5.3 Approach

To achieve efficient fault tolerance, FDCB is augmented so that the cluster-head has complete "cluster-state" knowledge. The cluster-head has connectivity awareness for all cluster nodes. This awareness includes knowledge of all QoS connections currently supported by each cluster member, each member's resource availability, and the cluster topology. With this scheme, when cluster node i leaves the cluster, due to mobility or failure, and the QoS paths supported by i are broken, the cluster-head has all information required to begin a renegotiating to reestablish the connection with minimal delay if possible. The cluster-head collects this knowledge via two processes: communication with the other clusters via clustered FSR and local clustered information exchange. These processes ensure, with high probability and low overhead, that knowledge of the systems' state is maintained both to repair existing paths and to initiate new ones.

5.4 System

5.4.1 Services

EFDCB is targeted toward routing QoS packets in challenging MANET environments where links can break often and without warning. In these environments, a routing algorithm needs a contingency plan for link breakages. EFDCB provides QoS disruption mitigation. When EFDCB is successful, packets are delivered such that the applications dependent upon the network are fully functional. Conversely, if the protocol fails then the dependent applications could suffer lengthy QoS disruptions since the source will have to resort to rerouting.

5.4.2 Design

Clustering. Numerous schemes exist for clustering nodes in MANETs. FDCB constructs nonoverlapping clusters based on bandwidth and delay factors for each link [3]. The Virtual Grid Architecture (VGA) [19] uses location information from the Global Positioning System to cluster nodes into a fixed rectilinear virtual topology to make routing and network management as efficient as possible.

The clustering algorithm adopted here is a modified version of the Generalized Distributed and Mobility Adaptive Clustering (GDMAC) [20]. GDMAC performs well in the mobility models (*random waypoint*, *random walk*, and *Manhattan*) used in many simulations [20]. The protocol is also straightforward, allowing easy modification.

GDMAC has been modified to support fault tolerance in QoS supporting MANETs. It is notable that the original clustering algorithm was not applied to QoS. This means an underlying QoS routing protocol and supporting procedures (i.e., for path negotiation, resource reservation, and resource deallocation) must be added. The QoS support procedures have been built into GDMAC in EFDCB.

The optimum cluster size is dependent on several MANET characteristics. Gupta and Kumar [21] show that the per node capacity of a random ad hoc network, where each of n nodes can transmit W bits per second, is $\Theta(W/\sqrt{n \log n})$ using a geometric analysis. The cluster size used leaves sufficient bandwidth for the required control packets (assuming each node has a 54 Mbps transmission rate). It is assumed clusters are situated so that the only intercluster nodes able to communicate are gateways.

Larger cluster sizes typically yield high communication overhead and lack diversity from other networks with respect to changing channel conditions. Gupta et al. showed that larger cluster sizes can cause exponentially higher overhead [22]. Kim and Wang created a way to reduce cluster overhead. Their Bandwidth Adaptive Clustering (BAC) makes cluster members forward their overhead messages probabilistically [23]. The more bandwidth a member has, the higher the likelihood it will send its overhead communications. In tests, BAC has shown better performance on the construction and maintenance of mobile clusters, adaptivity to network conditions, and effectiveness in reducing overhead.

QoS routing. FDCB uses an on-demand reactive routing scheme, but EFDCB adopts a more proactive approach. The QoS routing scheme used by EFDCB is Clustered Fisheye State Routing (CFSR) [24]. CFSR proposes a clustering framework to reduce redundant broadcast routing control messages. For FSR, the frequency at which node i sends its link state information to node j depends upon the distance from i to j (namely, the *scope* j falls in). The greater the distance, the less frequent the link state update.

In CFSR, cluster-heads and gateways execute the original FSR protocol to send link state updates about the cluster, while ordinary nodes only send link state about themselves. This limits the messages from much of the network (ordinary nodes). The result is lower overhead. Assuming combinatorial stability, each node becomes aware of the complete network state with lower bandwidth. The disadvantage is that routing control messages traverse the network at a lower rate since a smaller fraction of network nodes broadcast full control messages.

In CFSR's protocol, redundancy is not minimized; however, it is reduced considerably. In order to minimize redundancy, each cluster-head must not receive link state information about the same cluster from more than one gateway node. To accomplish this, the entire clustered network is partitioned into as many disjoint sets as the cluster has gateway nodes. These partitions are determined by finding the distance from each external network node to each local cluster gateway node using the topology graph stored in the routing table. The local cluster gateway node that has the shortest distance to the external network node

TABLE 3
CFSR Definitions

Symbol	Definition
i	generic node executing the update procedure.
A_i	set of nodes adjacent to i (i's neighbor list).
N	set of external network nodes local gateway node i is closest to (i will exchange updates with this set of nodes as described above).
$TT_i.LS(j)$	denotes the link state information reported by node j. **Link state info contains j's k weights w_{jk} for the k QoS constraints for each link j reports on.**
$TT_i.SEQ(j)$	denotes the time stamp indicating when node j generated this link state information.
TT_i	i's topology table. Each destination j has an entry in table TT_i for each QoS constraint **(e.g., if three constrained routing is used j would have $TT_i.LS(w_{j1})$, $TT_i.LS(w_{j2})$, $TT_i.LS(w_{j3})$).** Also, for each link state entry in TTi j has $TT_i.SEQ(j)$.
$NEXT_i$	i's next hop table. $NEXT_i(j)$ denotes the next hop to forward packets destined for j on the path with the required constraints.
Scope	defined as the set of nodes that can be reached within a given number of hops.
D_i	$D_i(j)$ is the shortest path distance from i to j.

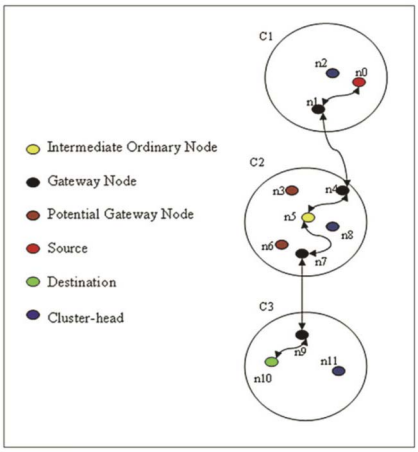

Fig. 1. Clustered ad hoc network.

TABLE 4
Cases Causing QoS Path Breakage

1	Ordinary I node moves out of range of a succ I node in the cluster
2	Ordinary I node moves out of range of a prec I node in the cluster
3	I *GWN* moves out of range of a succ I node in the cluster
4	I *GWN* moves out of range of a prec I node in the cluster
5	I *GWN* moves out of range of a succ I *GWN* in the cluster
6	I *GWN* moves out of range of a prec I *GWN* in the cluster
7	I *GWN* moves out of range of a succ *GWN* not in the cluster
8	I *GWN* moves out of range of a prec *GWN* not in the cluster
9	I *CH* moves out of range of a succ or prec I node in the cluster

includes that external node in its control message. Due to the FSR mechanics, the gateway node closest to the external network node will be the first gateway to receive the external node's link state update. It is responsible for providing this information to the cluster.

CFSR is QoS ready with the addition of bandwidth and channel quality information to the link state entry. EFDCB uses CFSR with a few modifications. CFSR is initiated once the clustering converges. Before presenting CFSR pseudocode, some definitions are provided. The definitions are summarized in Table 3. Bolded text represents modifications to the original definitions.

Each node begins with an empty neighbor list A_i and an empty topology table TT_i. After node i initializes in the *NodeInit()* procedure, it learns about its neighbors by examining the sender ID of each received packet. i then calls the *Pkt_process* procedure on the received packet which contains the link state information from its neighbors. *Pkt_process* ensures the most up-to-date link state information is used by comparing the local sequence number with the embedded sequence number *pkt.SEQ(j)*. If any entry in the incoming message has a newer sequence number for destination j, $TT_i.LS(j)$ is replaced with *pkt.LS(j)* and $TT_i.SEQ(j)$ is replaced by *pkt.SEQ(j)*.

FindSP(i) generates the shortest path tree rooted at i. The shortest path algorithm needs to generate a next hop table for each path created. This shortest path tree is used by i to send route updates to the set of nodes in N.

RoutingUpdate(i) scans the topology table and if $D_i(x)$ is within range of the fish-eye scope level l, $TT_i.LS(x)$ is included in the update message. The $UpdateInterval_l$ attribute is used to adjust the link state update frequency for the various fish-eye scopes.

By adopting this new scheme over the original FDCB method of distributed routing, more memory is required. Also, although routing control packets are sent periodically, CFSR significantly reduces the broadcast control packets over pure proactive protocols. This is because only a small fraction of the network population is allowed to broadcast full control messages. These changes allow each node to be aware of the complete network state at a lower bandwidth cost. Given that link state information is now available, using any efficient QoS routing algorithm (such as the limited path heuristic [14]), EFDCB is able to routing packets based on QoS constraints.

Fault tolerance. Definitions are presented here to enable a detailed description of the EFDCB fault-tolerant approach, which are summarized in Table 5. An *intermediate (I) node* is a node that supports a QoS connection. A *defunct (D) node* is a cluster node that previously was an I node; but has either moved out range or failed. A *gateway node (GWN)* is a cluster node used to communicate with an adjacent cluster. A *potential GWN (P-GWN)* is a node with the ability to communicate with the same adjacent cluster as the current *GWN*; it is only used if the current *GWN* becomes a D node. For example, in Fig. 1, *n3* is a *P-GWN* since it can communicate with *n1* and the current *GWN* is *n4* since it is communicating with *n1*. A *cluster-head (CH)* is a cluster node responsible for monitoring and updating a cluster table that records all QoS connections the cluster supports. *CH* is also responsible for initiating QoS connection repairs. A *CH* can also be a *GWN*. An ordinary node is neither a *CH* nor a *GWN*.

In Fig. 1, let *n0* be the source, *n10* the destination, and P the QoS path. Each node i in P has a successive (succ) node except *n10*. Further, each node i in P has a preceding (prec) node except *n0*. In a clustered ad hoc network, P can be broken if any of the cases in Table 4 occur.

When D node is an ordinary I node (cases 1 and 2), the cluster-head aggregates all available cluster resources and

TABLE 5
EFDCB Definitions

Symbol	Definition	Symbol	Definition
v	The generic node executing the algorithm (assume v includes the node's ID and its weight) [20]	**Gateway Node**	**Cluster node that is used to communicate with an adjacent cluster**
Cluster-head	this variable in which every node records the ID of the cluster-head that it joins (note *Cluster-head*$_v$ denotes node v's cluster-head) [20]	***Ch(–)***	Boolean variables. Node v sets $Ch(u)$, $_{u \in \{v\} \cup \Gamma(v)}$, to true when either it sends a CH(v) message ($v = u$) or it receives a CH(u) message from $_{u(u \neq v, u \in \Gamma(v))}$ [20].
Cluster-head	**cluster node which has the responsibility of monitoring and updating the cluster QoS table as well as handling connection failures (i.e., aggregating cluster resources, supported QoS connection constraints, finding feasible paths through the cluster, and notifying cluster nodes of the new paths)**	*NT*	**node QoS table containing the current existing connections of the associated node (i.e., *v.NT* refers to node v's own QoS table) – sent by node v to the cluster-head when either 1) node v's cluster-head changes 2) node v's QoS table changes or 3) update to the cluster-head is required**
GatewayNode(–)	**Boolean variable. *GatewayNode(v)* is set to true when cluster node v is adjacent to, and can communicate with, at least one other node in an adjacent cluster**	*AT*	**table containing the current available resources of the associated node (i.e., v.AT refers to node v's own available resources table)**
w_v	Weight of v, an integer > 0 which indicates how good v is for serving as a cluster-head. The weight might be computed based on the nodes bandwidth, available energy, or its mobility [20]	*CT*	**cluster QoS table containing the address, weight, *NT* and *AT* of all cluster nodes – knowledge shared by all cluster nodes via periodic cluster-head broadcast**
Ordinary Node	**cluster node which is neither a gateway node nor a Cluster-head**	***w*(*p*)**	is the vector sum of all weights for all constraints of all edges in path p
GT	**table of gateway nodes for the cluster, maintained and broadcast by the cluster-head**	***Cluster*(*v*)**	the set of nodes in v's cluster, initialized to Ø [20]
$\Gamma(v)$	the set of all nodes one hop away from v in the same cluster [20]	$\Pi(v)$	the set of all nodes one hop away from v and in another cluster. Initialized to null and only updated if v becomes a Gateway node
H	The H parameter implements Ghosh and Basagni's idea that cluster re-organization is only needed when the new cluster-head is better than the current one by some specified value. That is, a clustered node switches to a newly arrived cluster-head only when the weight of the new cluster-head exceeds the weight of its current cluster-head by a quantity H. By manipulating the value of H, the likelihood a node will switch to a new neighboring cluster-head can be controlled [20].	K	Ghosh and Basagni's K parameter controls the spatial density of the cluster-heads. Up to $K \geq 0$ cluster-heads are allowed to be neighbors. By setting $K > 0$, the probability of cluster re-organization is lowered since a cluster-head is not forced to give up its position when up to K-1 cluster-heads with bigger weights become its neighbors [20]. To simplify EFDCB, values of $H = 0$ and $K = 0$ are used. By setting $H = 0$, $K = 0$, cluster-heads are not allowed to be neighbors.
***Connex*(–)**	**Boolean variable. Cluster-head sets *Connex*(*v*) to true when v is supporting a connection. *Connex*(*v*) is false otherwise**	***ConnexParams*(–)**	**table of variables into which the Cluster-head records the QoS connection requirements of a particular path (e.g., *ConnexParams*(*p*) would contain a table of the QoS parameters for path *p*)**
$PATH_N$	**newly calculated set of feasible paths through the cluster for the supported connections**	*PATH*(*dst*)	**the set of QoS constrained paths to the destination dst**
$PATH_F$	**set of paths through the cluster that have failed due to the failed node**	$PATH_O$	**the original set of feasible paths through the cluster for the supported connections**

cluster supported QoS routes and recalculates the feasible QoS paths for the routes that traverse the cluster. If the required resources for the QoS constraints of all paths exist; these new routes will be the optimum routes for the current cluster. The route resource negotiations are all handled within the cluster and the route is restored. When the *D* node is a *GWN* (3-8), the *CH* first ensures the *P-GWN* can handle the QoS constraints previously supported by the now *D GWN*. Once it is determined that the *P-GWN* can handle this traffic, the necessary resources are allocated. After a specified time not receiving a beacon message from the *D GWN*, the preceding node to the *D GWN* attempts to route traffic through the *P-GWN* and restores the route. Note: Case 9 (the *CH* fails while supporting a QoS connection) is addressed in Section 5.4.2.4.

EFDCB reduces the impact of a connection failure since the cluster-head has complete cluster connectivity awareness. The result is efficient fault-tolerant QoS route maintenance for MANETs.

EFDCB protocol. EFDCB is primarily message driven, as is FDCB [20]. The procedure executed by a node depends on the message it receives. Several message types are exchanged between nodes. Key definitions are also given. To aid in distinguishing parts of the protocol, assumptions and definitions that are originally in FDCB are in plain text. Modifications or additions are in bold. A summary is in Table 6.

At cluster set up, or when a node is added to the network, its variables are initialized as follows:

Cluster-head	**NULL**
***Connex*(–), *Ch*(–), *GatewayNode*(–)**	**false**
***PATH*$_O$, *PATH*$_F$, *PATH*$_N$, Π (*v*), *Cluster*(–)**	**Ø**
***CT, GT, NT, AT, ConnexParams*(–)**	**NULL**
H, K	**0**

Assumptions

- All nodes have a unique identifier.
- Two nodes can be members of the same cluster if their euclidean distance is ≤ 30 m (as in 802.11g).

TABLE 6
Messages

CH(v)	Used by a node v to communicate to its neighbors that it intends to be a cluster-head [20].	CLSTR_PRMS_UPDT(v, *CT*)	Broadcast at regular intervals by cluster-head v to update each cluster nodes' cluster QoS table.
HELLO(u, *Cluster-head, Init*)	Creates gateways via adjacent nodes in other clusters. Node u periodically sends a HELLO(u, *Cluster-head, Init*) message to try to receive a HELLO(v, *Cluster-head, Reply*) reply.	REPAIR(*ConnexParams*(p), v, *Cluster-head*)	Sent by the cluster-head to notify node v to restore a connection using the information in the *ConnexParams*(p) table.
RESIGN(w)	Used to require the resignation from the role of cluster-head of any receiving cluster-heads whose weight is $\leq w$ [20].	REPAIR_FAILURE (v, *Cluster-head*)	Sent when v's attempts to repair a connection fails.
QOS_VALID(u, v)	Message sent from source u to destination v after a failed link has been repaired. The QoS validation message is initialized by source u after receiving LINK_REPAIRED(*failed_node*, v, u) from the new node to the path – v.	**PATH(v.*rsrcs, dst*)**	Used by source v to request resources from each node u along the path of a new potential QoS connection to destination *dst*.
JOIN(v, u)	Sent by a node v to communicate to its neighbors that it will be part of the cluster whose cluster-head is node u [20].	FAILED_CONNEX(*failed_node*, p, v)	Sent to source v of QoS path p in the event that the failed connection supporting p could not be repaired.
CTS(u)	Sent by destination v back to source u along the initialized (intermediately allocated) path to finalize the resource allocations	**PARAMS(v.*NT, v.AT, Cluster-head*)**	Used to send information about the supported connections (v.*NT*), as well as the available resources (v.*AT*), of node v to the cluster-head.

- Nodes signal their presence via a periodic beacon message and the drifting in of a new node is realized when its new neighbors hear its beacons.
- When a node does not hear from a known neighbor within a set timespan, it assumes the neighbor is either "dead" or out of range due to mobility
- Determining a node has failed or moved out of range will prompt the corresponding procedure.
- All procedures are atomic **except Route_traffic(u) and the procedures executed to respond to the PATH(v.rsrcs, dst) and CTS(u) messages.**
- When discussing the relevant features of EFDCB, it is assumed that the CFSR has converged. That is, all gateway nodes in the network have path routing table entries for all network destinations. Also, it is assumed that applications using this QoS network have soft QoS constraints and use adaptive techniques to minimize QoS disruptions. Combinatorial stability is also assumed. Further, nodes have the ability to send and receive best-effort traffic along with QoS traffic. Finally, resources allocated for a QoS connection are deallocated after a specified period of inactivity.

Messages and Procedures

EFDCB is an inherently message-driven protocol. A summary of the key messages is shown in Table 6. An illustration of two of the most important procedures in AFDCB, Node_failure and Init, are illustrated in Fig. 2.

EFDCB's protocols are presented next. A few of the basic clustering procedures are largely the same as in FDCB, but are included for completeness. Most procedures are new. As stated earlier, pseudocode appears in bold when it represents a modification or addition FDCB.

Init. *Init* remains predominantly as described in FDCB [20]. When the cluster is initialized, or when a node v is added to the network, v executes the *Init* procedure to determine its role. If among its neighbors, there is a cluster-head with bigger weight, then v will join it and send a PARAMS message providing the cluster-head with v's *NT* and *AT* which the cluster-head will then use to update the *CT*. If no node exists which has a weight bigger than v, v will be a cluster-head. In this case, the new cluster-head v checks the number of its neighbors that are already cluster-heads. If they exceed $K = 0$, then a RESIGN message is also transmitted, carrying the weight of the first cluster-head (the node with the lowest weight) that violates the K-neighborhood condition (the weight is determined by operator $\min_k$). Since $K = 0$, the node with the largest weight will replace all cluster-heads in range who have a lower weight than v. A visual representation of this procedure is shown in the lower part of Fig. 2.

Node_failure. When node v is made aware of the failure of node u, v checks if its own role is cluster-head and if u was in its cluster. If so, v removes u from *Cluster*(v). In this case, if u was an intermediate node supporting one or more connections, cluster-head v aggregates all cluster resources and all supported QoS traffic and determines new feasible QoS paths. Node v then advises all relevant cluster nodes to support the QoS connection via the REPAIR message. If all QoS connections cannot be supported by the available cluster resources (i.e., an infeasibility exists), the cluster-head sends a FAILED_CONNEX message to all sources that were using resources on the failed node and no path changes are implemented in the cluster. If v is an ordinary

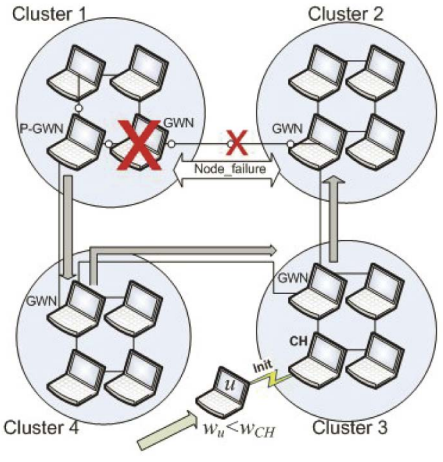

Fig. 2. Overview diagram of Node_failure and Init Procedures of EFDCB.

node, u was its own cluster-head supporting a QoS connection, and v is the node with the next weight w_v in descending order after w_v (i.e., $\exists v\, \forall z \in \{\Gamma(v) - \{u\}\}$: $w_v > w_z$), node v then becomes the new cluster-head and attempts to fix u's failed connection. In the case where v is an ordinary node which is not the next weight w_v in descending order after w_u, and u was its own cluster-head (whether or not supporting a QoS connection), v waits a specified period of time to receive the CH(y) message from the cluster node with the next highest weight to u, node y. If v receives the CH(y) message in the allotted time, v joins y's cluster. If v does not receive the CH(y) message in the required time, v must find a new role for itself. In this case, v determines if there exists a cluster-head $z \in \Gamma(v) : w_z > w_v$. Node v joins the cluster-head with the bigger weight and sends its *NT* and *AT* to its new cluster-head, otherwise it becomes a cluster-head. In any case, where u was the cluster-head and was also supporting a connection, the new cluster-head will attempt to repair u's failed connection only if the cluster remains relatively preserved (i.e., the node next in weight after w_u becomes the new cluster-head). The reclustering process is invoked for each node if node v, where $\exists v\, \forall z \in \{\Gamma(v) - \{u\}\} : w_v > w_z$, does not become the new cluster-head. The time required for re-clustering will add significant time to the connection recovery process; therefore, rerouting is employed in this situation. It is likely the cluster will change very little when a cluster-head fails since combinatorial stability is assumed. Further, since all cluster nodes are already aware of the next potential cluster-head (i.e., this information is broadcast by the cluster-head via the *CT* at periodic intervals), once a cluster-head fails cluster members wait a short amount of time (propagation delay + processing delay + error) to receive the CH(v) message from the expected new cluster-head. If the CH(v) message is not received in the allotted time, all cluster nodes must determine their new roles. The *Node_failure* procedure is shown in Fig. 2.

New_link. *New_link* is the same as in FDCB except for the addition of the PARAMS message. Once cluster node v discovers a new node u, it checks to see if u is a cluster-head. If u is a cluster-head and weight w_u is greater than the weight of v's current cluster-head, u becomes v's new cluster-head and sends u a PARAMS message. Conversely, if v is a cluster-head and the number of neighboring cluster-heads is greater than 0, the weight of the cluster-head x that violates the $K = 0$ condition is determined. If $w_v > w_x$, node x will be sent the RESIGN message. If there is no cluster-head x such that $w_v > w_x$, v will no longer be a cluster-head and will join the cluster-head with the biggest weight. Node v will also send its new cluster-head the PARAMS message.

Route_traffic(u). Source node v is made aware of the need to route new traffic by the associated application. Node v checks its cluster members to see if u is in this set of nodes. If u is in the current cluster, u's available resources are obtained from the *CT* table. If the required resources are available, they are reserved and the traffic is sent. If the destination is not in the current cluster, v forwards the PATH(*v.rsrcs, dst*) to the cluster gateway nodes.

The following procedures are initiated when the corresponding message is received:

On receiving PARAMS(*u.NT, u.AT, Cluster-head*): performed by the Cluster-head (in this case, node v): On receiving the message PARAMS(*u.NT, u.AT, v*), Cluster-head v updates the *CT* with this new information. v then checks the time since the broadcast of the last *CT*. If sufficient time has passed, v broadcasts the *CT* to all cluster nodes. During construction of a new cluster, the cluster-head may receive many *CT* updates. The *CT* is never sent at intervals less than some time T. This allows the cluster to reach a level of stability before broadcasting cluster table updates. A CLSTR_PRMS_UPDT(CT, v) message is broadcast at regular intervals (similar to the beacon message) throughout the life of the cluster-head.

On receiving PARAMS(*CT, Cluster-head*): On receiving the message PARAMS(*CT, Cluster-head*), v first ensures that the cluster-head which sent the message is v's cluster-head. Node v then checks to see that the *CT* has an accurate account of v's *NT* and *AT*. If these two conditions hold, v records the received cluster QoS table. Otherwise, if v received from the correct cluster-head but *CT* is incorrect, v sends its *NT* and *AT* to the cluster-head.

On receiving REPAIR(*ConnexParams*(p), v, *Cluster-head*): On receiving the REPAIR(*ConnexParams*(p), v, *Cluster-head*), v first ensures that the cluster-head that sent the message is v's cluster-head. Node v then checks to see that it has the resources to support the new connection in the *ConnexParams*(p) message. Once resources are verified, v uses the information it has about the connection (from the *ConnexParams*(p) table) to attempt to restore the link. If the link is restored, LINK_REPAIRED(*failed_node*, v, u) is sent from v to source u of the QoS traffic. If v finds it cannot communicate with the nodes to make the connection, v sets the boolean ERROR to true, and $v.NT$ and $v.AT$ are sent to tell the cluster-head whether the connection is reconnected or not. This procedure is executed for the reconnection of links that may or may not have failed since the cluster-head finds a new set of feasible paths for all connections through the cluster when a failure occurs.

On receiving LINK_REPAIRED(*failed_node, u, x*): This concept is borrowed from Chen and Nahrstedt [15]. On receiving the message LINK_REPAIRED(*failed_node, u, x*), source node v sends the QOS_VALID(*y, v*) to any destination node y which received QoS traffic that passed through failed_node. Once y receives this message, it sends the QOS_VALID(*v, y*) message enabling v to determine if end-to-end delay constraint has been violated.

On receiving REPAIR_FAILURE(*u, Cluster-head*): performed by the *Cluster-head*: On receiving the message REPAIR_FAILURE(*u, Cluster-head*) cluster-head v immediately sends a FAILED_CONNEX(*failed_node, v, x*) back to any source x which was using resources on *failed_node*.

On receiving FAILED_CONNEX(*failed_node, u, v*): node v attempts to reroute QoS traffic via Route_traffic(t) for each route r which traversed the failed_node.

On receiving QOS_VALID(v, u): On receiving the message **QOS_VALID(*v, u*)**, node v immediately sends a **QOS_VALID(*u, v*)** back to the source.

On receiving HELLO(*u, Cluster-head, Init*): On receiving the message HELLO(*u, Cluster-head, Init*), node v checks the value of *Cluster-head* to determine whether the sender, u, is a

member of the same cluster or of an adjacent cluster. If u is a member of the same cluster the message is discarded. If the sender is a member of an adjacent cluster v then checks to see if it has received a HELLO message from u earlier. If v has not received a HELLO message previously from u, v notes that it can contact the adjacent cluster through node u. Node v then notifies the cluster-head of this, and transmits a HELLO(v, *Cluster-head*, *Reply*) message to the sender. u receives the HELLO(v, *Cluster-head*, *Reply*) message, notes it can contact the adjacent cluster through v, and notifies its cluster-head. Node v is now a gateway node from its cluster to u's cluster and u is a gateway node from its cluster to v's cluster.

On receiving MYNGHBRS((Πu) *Cluster-head*): performed by the *Cluster-head*: On receiving the message MYNGHBRS(Π, *Cluster-head*), *Cluster-head* amalgamates the received $\Pi(u)$ into the cluster gateway table (*GT*). *Cluster-head* then broadcasts the *GT* to all cluster nodes. MYNGHBRS messages are likely to be rare so no limitation is imposed on the frequency with which CLSTR_GN_UPDT(*GT*, v) messages can be sent.

On receiving PATH(*u.rsrcs, dst*): On receiving the message PATH(*u.rsrcs, dst*), v checks to see if it has the required resources using its availability table ($v.AT$). If not, v drops the PATH packet. If v has the required resources, it does an intermediary allocation of the resources (adjusting the *AT* to reflect this potential additional connection). If v is the destination, it allocates the necessary resources and responds with a CTS(u) message which traverses back to source u. Node v waits a predetermined amount of time to receive the data packet. If the data packet is not received in time, v deallocates the resources. If v is not the destination, once the intermediary resource allocation is done, a count-down timer is initiated. If the associated CTS(u) message is not received before the counter expires, the resources are deallocated. Note that this procedure is not atomic since intermediary nodes must be looking for the receipt of the CTS(u) message or handling other received messages or events while decrementing its counter. By using the count down timer, resources are not held for extended periods when the path is never used (e.g., if a node farther down the path cannot support the QoS request). If v is not the destination, v checks to see if the destination is in its routing table and if the path in the routing table entry meets the required constraints. If so, v routes the PATH(*u.rsrcs, dst*) message on to the destination. If no path exists to the destination which can support the required constraints, v discards the packet.

On receiving CH(u): The procedure executed is almost the same as in FDCB. The exception is the addition of the two send PARAMS lines which are executed after a node accepts a new cluster-head. When v's neighbor u becomes a cluster-head and v receives the CH message from node u, v checks to see if w_u is larger than the weight of v's current cluster-head (plus parameter H, which equals 0 in this work as mentioned in the definitions). If it is, v joins u's cluster. If v is a cluster-head with more than K neighbors which are clusters (K also equals 0 as mentioned in the definitions), the cluster-head with the smallest weight is found so that it may give up its cluster-head position.

On receiving CTS(u): On receiving CTS(u), v checks to see if it is source u. If it is, it begins transmitting QoS traffic. If not, and v has earmarked resources for u's connection, the previous intermediate resource allocation is finalized and the node table ($v.NT$) is updated to reflect the connection. v transmits the update to the cluster-head.

On receiving JOIN(u, z): The procedure executed upon receipt of JOIN(u, z) is similar to the FDCB JOIN(u, v). The only exception is the addition of a send PARAMS line executed after a node accepts a new cluster-head. After receiving the JOIN(u, z) message, the behavior of node v depends on whether it is a cluster-head. If v is a cluster-head, checks for one of two cases. Node v checks if u is joining its cluster (i.e., $z = v$) or if u belonged to its cluster and is now joining another cluster (i.e., $z \neq v$). In the first case, u is added to *Cluster*(v) and sends its *NT* and *AT* to the cluster-head. In the second, u is removed from *Cluster*(v). If v is not a cluster-head and u was its cluster-head v has to decide its role. It will join a new cluster-head x such that $w_x > w_v$ if one exists. Otherwise it will become a cluster-head and ensure the K-neighborhood is respected.

On receiving RESIGN(w): The procedure executed on receipt of the RESIGN(w) message is essentially the same as in FDCB. The exception is the addition of the send PARAMS line which is executed after a node joins a new cluster. After receiving the RESIGN(w) message, node v checks if $w_v \leq w$. If so, v gives up its cluster-head status and joins the nearest cluster-head with the largest weight. Once v has received the RESIGN message and confirmed the need to resign, it sends its *NT* (supported connections list and available resources) to the new cluster-head.

5.5 Implosion Avoidance Techniques

Our system does not employ feedback implosion avoidance, but there are several techniques that can be used. Tracked Sender List (TSL) [25] is initially an empty message queue that tracks the most recent network feedback. If Negative Acknowledgments (NAKs) are sent throughout the network, TSL ensures that these NAK messages are not flooding the network by testing the TSL queue and the NAK message that the host is forwarding. If the NAK matches a message in the queue, it suppresses its NAK and forwards the NAK received. Another technique to mitigate feedback implosion is Hierarchical ARQ (H-ARQ) [26]. H-ARQ consists of three schemes: terminal side, Local Recovery Router (LRR), and Data Broadcast Server (DBS). On the terminal side, instantaneous NAKs will not be sent. If a faulty packet is received or packets are not received at all then a node will stop data reception and will display an error message. If after receiving one or several faulty packets it receives a good packet, a node will send a NAK for the faulty packet(s). Next, the LRR scheme uses a cache to store NAK packets. If this cache reaches a threshold, all other standing NAKs will be forwarded directly to the DBS system. After the DBS collects all NAKs from the LRRs, it will calculate whether it can recover from the NAKs on its own. If so, it sends the NAKs back to the LRRs to be recovered. If they cannot be dealt with locally, the NAKs in the DBS are broadcasted to the network for assistance. These techniques are just two of many that can be incorporated into our system.

5.6 Summary

This section presents key design features of the EFDCB routing protocol. EFDCB unifies modified GDMAC and FDCB protocols and uses CFSR for QoS routing. EFDCB uses a cluster-head model to mitigate connection failures. This model employs cluster state knowledge sharing with to make the cluster-head aware of supported QoS connections in the cluster. Cluster state knowledge is shared with all cluster members for use in the event of a cluster-head failure. CFSR allows each node in the EFDCB network to be aware of the complete network state with low bandwidth impact. EFDCB is different from FDCB since link failures are handled locally instead of rerouting traffic from the source and QoS traffic is routed with lower packet transmission delays. The EFDCB fault-tolerant method differs from Chen and Nahrstedt's [15] work in that a clustered approach is used to help avoid their repair method's limitations. EFDCB does not require that the predecessor of a failed node be capable of reaching the failed node's successor. EFDCB presents an innovative solution to fault-tolerant QoS supporting MANETs.

6 Analysis and Results

6.1 System Boundaries

This research does not focus on EFDCB's ability to lower resource requirements [3]; it focuses on EFDCB's ability to reduce recovery time for QoS traffic. The most vital component of the system is the QoS routing protocol.

Given the focus of this research, the intent is not to implement all of EFDCB, but only the features required to determine if the EFDCB QoS routing protocol provides efficient route recovery. Hence, the clustering portion of EFDCB is "bootstrapped," meaning the MANET is already clustered when the system initializes. Since clustering is hardcoded, node i moving out of communication range of node j is simulated by forcing i to fail. CFSR is also bootstrapped in this simulated system by using a (centralized) QoS routing algorithm which employs source routing based on bandwidth. The algorithm models traffic requests as multicommodity flows to determine if traffic bandwidth demands can be satisfied. With this routing model, all nodes have complete network state knowledge as with CFSR; however, traffic flows can be split. Simulations were run on ns2 using custom middleware application agents and a custom routing module to emulate EFDCB. The MANET is simulated by using a method in which lower bandwidth is used for routing control and beacon packets (i.e., 54 Mbps throughout cluster) and higher bandwidth (i.e., 100-200 Mbps links using channels/power to go between adjacent nodes) for data traffic. Mobility was emulated by causing links to fail or recover. This allowed greater consistency between trials and easier comparisons between EFDCB and FDCB. The standard nodal protocol stack was used in ns2. Failures were simulated by causing nodes to go offline.

6.2 Workload

The system workload is the rate at which nodes in the network fail. A different node failure rate value is used for each group of simulations. The particular node that fails is chosen randomly. The number and type of QoS connection requests made by each source node is kept constant (as are the number of source-destination pairs and length of connections) during the simulations; however, these values are changed between experimental phases. The intracluster bandwidth is also altered between experimental phases. Since the EFDCB algorithm is tested against the original FDCB protocol, the same workloads are used with both algorithms.

6.3 Performance Metrics

The system performance metrics in this analysis are connection recovery time, number of dropped packets, throughput, and amount of sustainable flow bandwidth. Connection recovery time is the time to reestablish a failed connection from the moment data traffic stops. This measures how efficient the routing algorithms are at repairing broken links, so that conclusions can be made about how well the offered traffic load is serviced. Traffic is expected to be serviced at a higher rate in EFDCB than FDCB since connection interrupts should be minimized.

Testing will illustrate EFDCB's ability to maintain flow demands given network failures over FDCB's global rerouting alternative. If EFDCB is only allowed to perform reconnections locally within the cluster, then in cases where the cluster cannot support the failed flow due to available resource limitations EFDCB will have to determine which flows to support and which to drop. By looking at EFDCB's ability to maintain flow demands while removing its ability to reroute from the source, insight about EFDCB's efficiency is obtained. Data collected on the number of dropped packets and throughput is critical in determining the protocol's ability to provide QoS.

6.4 Experimental Factors

6.4.1 System

Due to EFDCB's distributed nature, it is resilient to many factors which affect FDCB. EFDCB has the resources to repair a failed connection or not. For this reason, bandwidth is a key factor when finding feasible paths. Cluster topology is a factor since this set of edges must be considered when finding feasible paths. When considering FDCB, many more factors affect it since its failure handling is centralized. One is the distance from source to destination. The message notifying the source of a failed connection must make its way from the cluster-head to the source. During routing, negotiation messages must travel from the source to relevant clusters. The communication overhead impacts FDCB's performance.

6.4.2 Workload

By manipulating node failure rate it is possible to illustrate FDCB's inability to efficiently operate through failures. The primary intent is to alter the network by removing nodes supporting QoS connections. Manipulating these support nodes has the effect of increasing network dynamics. The increase in failed connections will demonstrate whether FDCB provides efficient protection in such challenging situations. The primary interests here are connection recovery time, number of dropped packets, throughput, and the sustainable flow bandwidth.

TABLE 7
Experimental Design for the First Phase of Experiments

Routing Algorithm	Experiment	Failure Frequency (ms)	Failures/Sec	Number of Flows	Average Cluster Bandwidth (Mbps)	Average Bandwidth Demand/Flow (Mbps)	Number of Runs
FDCB (or EFDCB)	1	40	25	6	400	100	10
	2	50	20	6	400	100	10
	3	60	16.70	6	400	100	10
	4	70	14.29	6	400	100	10
	5	80	12.5	6	400	100	10
	6	90	11.11	6	400	100	10
	7	100	10	6	400	100	10
	8	150	6.67	6	400	100	10
	9	200	5	6	400	100	10
	10	250	4	6	400	100	10
	11	300	3.33	6	400	100	10

TABLE 8
Experimental Design for the Second Phase of Experiments

Routing Algorithm	Experiment	Failure Frequency (ms)	Failures/Sec	Number of Flows	Average Cluster Bandwidth(Mbps)	Average Bandwidth Demand/Flow (Mbps)	Number of Runs
FDCB (or EFDCB)	12	40	25	12	200	125	10
	13	50	20	12	200	125	10
	14	60	16.70	12	200	125	10
	15	70	14.29	12	200	125	10
	16	80	12.5	12	200	125	10
	17	90	11.11	12	200	125	10
	18	100	10	12	200	125	10
	19	150	6.67	12	200	125	10
	20	200	5	12	200	125	10
	21	250	4	12	200	125	10
	22	300	3.33	12	-200	125	10

6.5 Failure Rate

Consider a large QoS supporting MANET in which $3/4$ of the nodes are filling a supporting role—that is $3/4$ of the nodes have no data to send but provide connectivity between other source-destination pairs. If the supporting nodes are gradually removed from the network, the number of possible connections decreases. Since each intermediate node has a particular set of QoS capabilities, removing one node could prevent a source from transmitting its data. In this experiment, nodes are randomly removed and then, after 200 milliseconds, returned to the network. With this method, there is no concern for running out of network resources as long as the rate at which nodes return is greater than or equal to the rate nodes are removed. The node failure rate is studied at 11 levels. The exact levels for this factor are shown in Table 7.

Table 8 illustrates the experiments for sustained flow bandwidth response. For this set of experiments, the network is initialized so that the available network bandwidth is extremely small. EFDCB is not allowed to reroute from the source. The desire is to saturate the network so that EFDCB will quickly run out of resources and be forced to drop flows. All cluster links are reduced to half the original bandwidth. The average bandwidth per flow is increased by increasing the number of source destination pairs as well as the demand for the additional traffic. The average cluster bandwidth is calculated as the average bandwidth available from the incoming gateway node to the outgoing gateway node. For comparison, FDCB is run under the same load and topology characteristics just described.

6.6 Evaluation Technique

Using ns2 [27], the system is configured as shown in Fig. 3. Each node has a simulated best-effort omnidirectional interface (for cluster maintenance) as well as a QoS supporting directional interface. At simulation start, all QoS links have the ability to support any one of the requested QoS connections, but once a QoS connection has been established the associated intermediate nodes may or may not have the bandwidth available to support additional QoS requests. The arrows indicate gateway node and potential gateway node connections. Clusters are defined at simulation initialization. The goal is to balance the gateway interconnections, the number of cluster, and the overall size and complexity in the scenario tested.

Once a source-destination connection has been established, and the source begins to transmit the data, an intermediate node is randomly removed. This forces the routing algorithm to either reroute the traffic from the

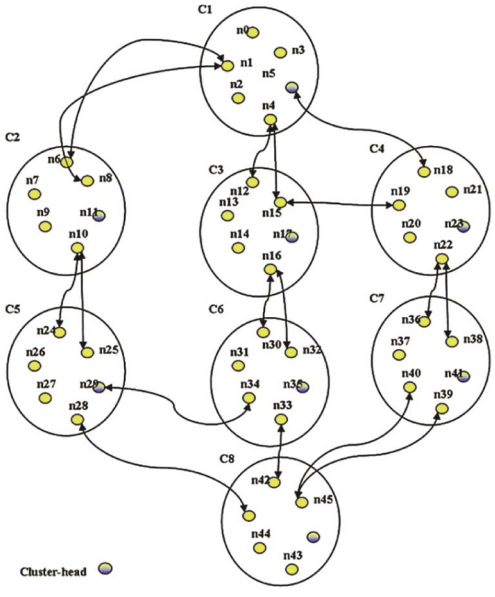

Fig. 3. Experimental network architecture.

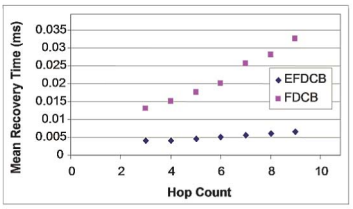

Fig. 4. Recovery time versus number of hops.

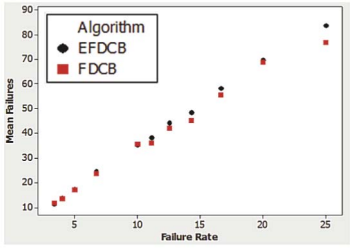

Fig. 5. Mean failures versus failure rate.

source (FDCB) or attempt to reestablish the connection (EFDCB) in the cluster associated with the removed node. In the case where the source and destination are separated only by a single node, it is likely that the local connection reestablishment option will be just as costly (in terms of recovery time) as having the source recalculate a route. As more nodes and clusters are added between the source and destination, the local algorithm will prevail in terms of time necessary to reestablish the path.

6.7 Experimental Design

The key is to demonstrate that the EFDCB algorithm is quicker at connection reestablishment. By repeatedly adding/removing random nodes from the set supporting the source-destination connections, the connection failure handling of the protocol under test is exercised. All experiments are performed on both FDCB and EFDCB.

6.8 Preliminary Testing

Initial testing on the effects of distance between source-destination pairs for a constant failure rate shows it has a major impact on performance for the FDCB algorithm.

These initial tests show that the EFDCB system is minimally affected by this parameter, as shown in Fig. 4. FDCB displays significant growth in mean recovery time per failure response as hop counts increase. Experiments investigating effects of this parameter on EFDCB and FDCB were not explored further; however, these initial experiments help to confirm the intuition that distance has a significant effect on the global nature of FDCB.

Additional testing was performed to evaluate the failure rate at which the competing algorithms begin to be unsuccessful, assuming instantaneous failure discovery for both algorithms. The threshold failure rate at which FDCB begins to break down is roughly 25 failures per second. EFDCB continues to function properly up to 100 failures per second. The limiting factor is the maximum recovery time. If the length of time to repair a broken link is longer than the interval between new broken links, EFDCB does not perform updates fast enough to have an accurate view of the network state. This relates to combinatorial stability. For FDCB, the maximum recovery time depends upon the network size. For EFDCB, the maximum recovery time depends upon the cluster size, a fraction of the total network. In both cases, maximum recovery time is a consequence of network topology.

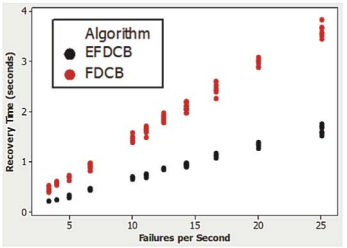

Fig. 6. Raw data plot of recovery time versus failure rate.

TABLE 9
Initial Experimentation Based on Source-Destination Hop Distance

Experiment	Number of Hops	Number of Runs
1	3	10
2	4	10
3	5	10
4	6	10
5	7	10
6	8	10
7	9	10

6.9 Results of Simulations

Fig. 5 illustrates that 9 out of 11 times the EFDCB algorithm encountered more average failures than FDCB. Therefore, the FDCB algorithm has a slight advantage in terms of offered load. That is, the EFDCB algorithm on average must handle more network failures than its predecessor during this experimentation. This is completely a product of the randomness of the failures.

Fig. 6 shows a raw data plot of recovery time versus failure rate for the failure rate values shown in Table 9. Recovery time is calculated as the sum of the individual recovery times of each reestablished connection for a given single experiment. The graph shows that both algorithms appear to have linear responses to linear increases in the failure rate. The data further demonstrate a spreading trend for FDCB as the rate of failures increase. This suggests a linear positive correlation where the variation of recovery time depends on the rate of failures. In general, much more variation in recovery time is recorded for FDCB than for EFDCB. This makes sense since more hops means more packets must be sent for route negotiation with intermediate clusters, and more transmitted packets means more processing and propagation time.

The scatterplot of mean recovery time versus failure rate (Fig. 7) demonstrates an obvious linear relationship between the two variables. Average recovery time is calculated as the average time spent handling failures per experiment (i.e., for the one node failure per 40 ms

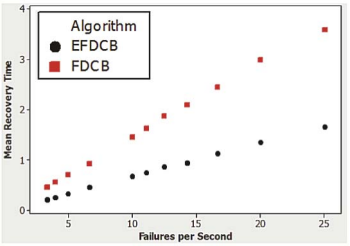

Fig. 7. Mean recovery time versus failure rate.

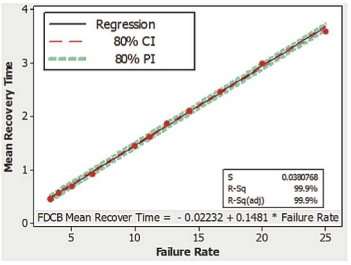

Fig. 8. Fitted line plot for FDCB mean recovery time versus failure rate.

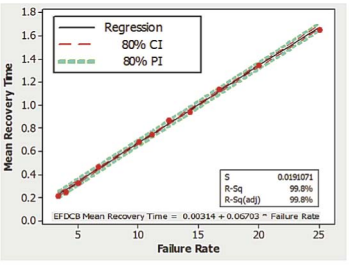

Fig. 9. Fitted line plot for FDCB mean recovery time versus failure rate.

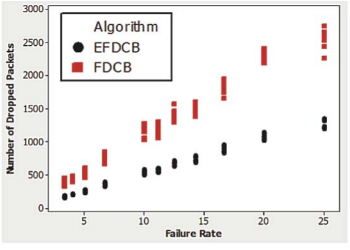

Fig. 10. Raw data plot of dropped packets versus failure rate.

experiment, run 10 times, the average recovery time is calculated, for the one node failure per 50 ms experiment, run 10 times, the average recovery time is calculated, and so on). For every failure rate tested, EFDCB has a faster recovery time than FDCB by more than a factor of two.

The mean recovery time fitted line plots for FDCB and EFDCB are shown in Figs. 8 and 9. The 80 percent prediction intervals capture the mean values indicating that these mean recovery time models fit the data well. The prediction interval provides a range within which one can expect the predicted response for a single sample to fall. Note that FDCB has more than twice the slope of EFDCB, hence more than twice the rate of increase for mean recovery time as the failure rate increases.

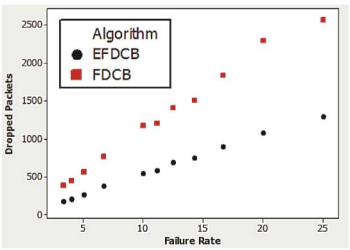

Fig. 11. Mean dropped packets versus failure rate.

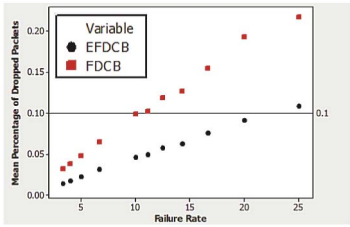

Fig. 12. Mean percentage of dropped packets versus failure rate.

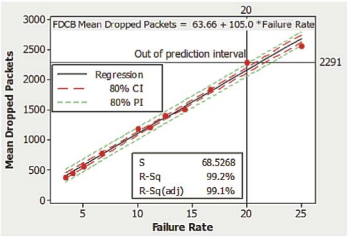

Fig. 13. Fitted line plot for FDCB mean dropped packets versus failure rate.

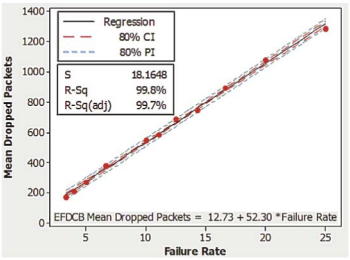

Fig. 14. Fitted line plot for EFDCB mean dropped packets versus failure rate.

The raw data illustrating the effects of failure rate on number of dropped packets are shown in Fig. 10. The trend is similar to that noted for recovery time; however, the relationship between variables appears to be less linear.

The scatterplot of mean dropped packets versus failure rate (Fig. 11) and mean percentage (Fig. 12) as well as the fitted line plot (Fig. 13) and mean fitted line plot (Fig. 14) provide additional evidence to support the notion of inconsistency in true linearity for the dropped packet response to failure rate input for FDCB. The fitted line shows that a wider prediction interval is necessary to capture the range that one can expect the predicted response for a single sample to fall for FDCB compared to EFDCB with 80 percent confidence. The difference in

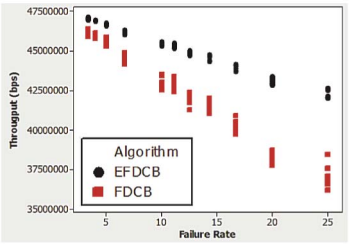

Fig. 15. Raw data plot of throughput versus failure.

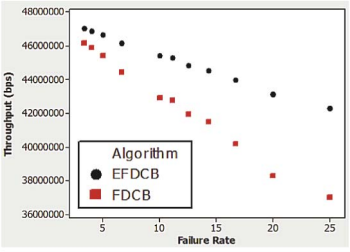

Fig. 16. Mean throughput versus failure rate.

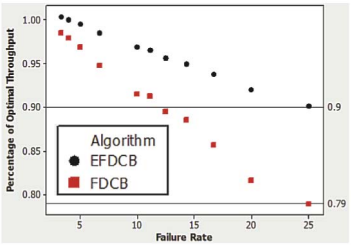

Fig. 17. Percentage of optimal throughput versus failure rate.

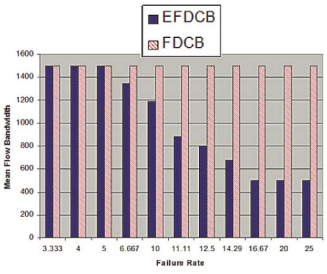

Fig. 18. Sustained flow bandwidth versus failure rate.

prediction interval ranges is even more noticeable if one notes that the maximum value for the y-axis of the FDCB fitted line plot is more than twice that of the EFDCB fitted line plot. Fig. 13 shows that the prediction interval fails to capture one particular mean dropped packet for FDCB. This deviation from consistent linearity for FDCB is likely due to interaction caused by variations in the distance between the cluster-head where the failed node is located and any source directly impacted by the failed node. Thus, the greater the distance (in hops) between these two nodes, the more likely QoS packets will be dropped. This is because as distance increases the time required for the source to realize the node in its QoS path has failed also increases. EFDCB, by contrast, does give the impression of a strong linear dropped packet response to linear increases in failure rate. This makes sense since EFDCB is somewhat resistant to variations in distance between the failed node's cluster-head and the sources using resources on that failed node. Similar to the mean recovery time results, for every failure rate tested, EFDCB has less mean dropped packets than FDCB by more than a factor of two.

The throughput versus failure rate results (Figs. 15 and 16) show that both algorithms have a general linear response. The deviations from concise linearity can be attributed to the effects of distance interacting with the failure rate. Fig. 17 shows the percentage of optimal throughput versus failure rate. Percentage of optimal throughput is *(realized throughput)/(throughput without failures).*

EFDCB's mean throughput never falls below the 90 percent threshold. FDCB dips to 79 percent of the optimal throughput.

The bar chart of sustained flow bandwidth versus failure rate (Fig. 18) shows that with a saturated network the "pure" local protocol (EFDCB with rerouting functionality removed) is able to compete effectively with the global rerouting algorithm for failure rates up to 6.7 failures per second. Beyond 6.7 failures per second, the global rerouting algorithm (FDCB) is significantly better at finding new routes when link failures occur. The EFDCB protocol is designed to invoke global recovery in cases where the flow is no longer sustainable by local resources. The bottom line is that EFDCB obtains the advantages of both the "pure" global and the "pure" local recovery methods.

Sustained flow bandwidth remains constant at 16.7 node failures per second and above due to the criteria used to pick the random node for failure. The algorithm first checks to see if the node failure will render the flow irreparable. If removal of a cluster node makes connectivity through the cluster impossible, no cluster node will be removed; thus, the upper bound on the worst-case mean sustained flow bandwidth is less than or equal to the sum of available bandwidth through the clusters without breaking connectivity. The choice between the set of flows to support and the set to drop involves picking the set that optimizes the sustained flow bandwidth.

Figs. 19a and 19b confidence interval graphs compare the number of dropped and received packets for EFDCB and FDCB. EFDCB drops fewer packets and recovers more packets than FDCB. Figs. 19c and 19d compare the number of failures and recovery time for EFDCB and FDCB. EFDCB allows more failures and has lower recovery times.

7 Conclusion

This paper presents a distributed fault-tolerant routing protocol for QoS support in mobile ad hoc networks, which mitigates disruption time under network failures.

The work demonstrates that the traditional method of rerouting QoS traffic from the source given a link failure yields serious negative QoS disruption consequences; an efficient local fault-tolerant algorithm can significantly mitigate the time required to reestablish a connection. Lowering the reconnection time reduces QoS disruptions.

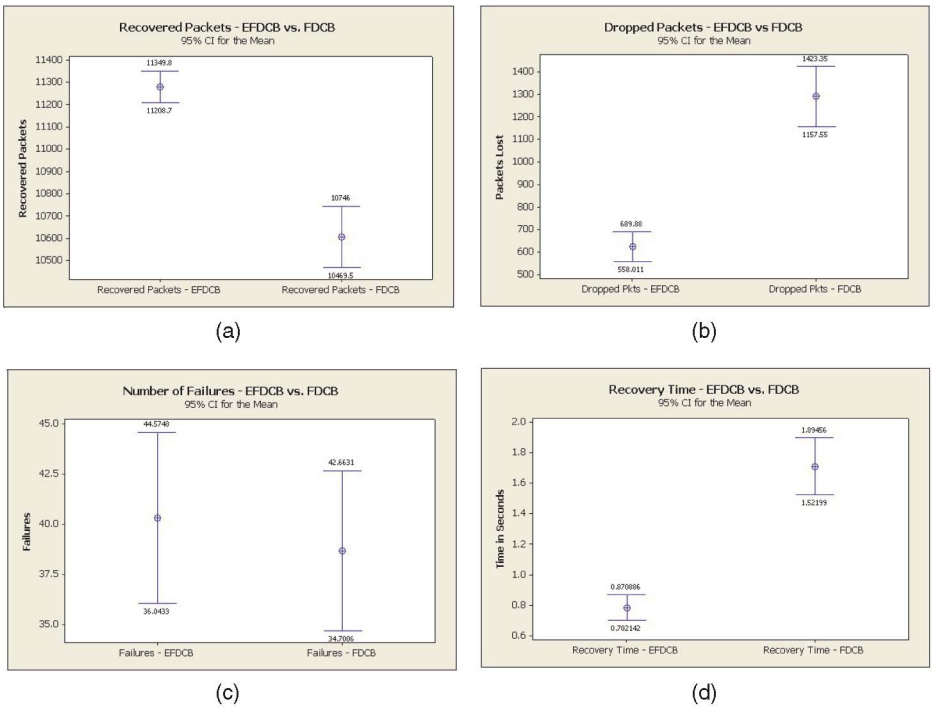

Fig. 19. Confidence interval graphs of: (a) dropped packets for EFDCB versus FDCB, (b) recovered packets for EFDCB versus FDCB, (c) number of failures for EFDCB versus FDCB, (d) recovery time for EFDCB versus FDCB.

Experiments show that a pure rerouting algorithm exhibits significant growth in recovery time as the source to destination distance increases. This is logical when one considers the message exchanges that must occur for a source to reroute its traffic. Specifically, once the cluster-head associated with the failed node realizes a node has failed it sends a "failed node" message to any source using resources on that node. The elapsed time between the moment the node fails to the time the source is notified is equal to the time required for the cluster-head to notice the failure plus the time for the failed node message to propagate to the source. Next, the source must determine a new feasible route. Upon determining a new route, the source must negotiate its demands with the cluster-heads of all clusters that have resources the source desires to use and then wait for responses from these cluster-heads. An increase in the number of clusters between the source and destination has a significant impact on the negotiation and recovery time. Recovery time results showed that EFDCB is more than two times faster than the global rerouting alternative at all failure rates tested. The dropped packet and throughput results reflected similar outcomes.

REFERENCES

[1] S. Chakrabarti and A. Mishra, "QoS Issues in Ad Hoc Wireless Networks," *IEEE Comm. Magazine,* vol. 39, no. 2, pp. 142-148, Feb. 2001.

[2] S. Chen and K. Nahrstedt, "On Finding Multi-Constrained Paths," *Proc. Record 1998 IEEE Int'l Record on Comm. (ICC '98),* pp. 874-879, 1998.

[3] A.S. Nargunam and M.P. Sebastian, "Fully Distributed Cluster Based Routing Architecture for Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Wireless and Mobile Computing, Networking, and Comm.,* pp. 383-389, 2005.

[4] S. Nelakuditi, Z.L. Zhang, R.P. Tsang, and D.H.C. Du, "Adaptive Proportional Routing: A Localized QoS Routing Approach," *IEEE/ACM Trans. Networking,* vol. 10, no. 6, pp. 790-804, Dec. 2002.

[5] S.H. Alabbad and M.E. Woodward, "Localised Credit Based QoS Routing," *IEE Proc.—Comm.,* vol. 153, no. 6, pp. 787-796, Dec. 2006.

[6] A.H. Mohammad and M.E. Woodward, "Localized Quality Based QoS Routing," *Proc. Performance Evaluation of Computer and Telecomm. Systems (SPECTS),* pp. 209-216, 2008.

[7] A.S. Alzahrani and M.E. Woodward, "End-to-End Delay in Localized QoS Routing," *Proc. IEEE Int'l Conf. Comm. Systems (ICCS),* pp. 1700-1706, 2008.

[8] P. Yang and B. Huang, "QoS Routing Protocol Based on Link Stability with Dynamic Delay Prediction in MANET," *Proc. Pacific-Asia Workshop Computational Intelligence and Industrial Applications (PACIIA),* pp. 515-518, 2008.

[9] A. Puri and S. Tripakis, "Algorithms for Routing with Multiple Constraints," Report Number UCB/ERL M01/7, Electrical Eng. and Computer Science Dept., Univ. of California, 2001.

[10] S. Chen, "Routing Support for Providing Guaranteed End-to-End Quality-of-Service," PhD dissertation, Univ. of Illinois, 1999.

[11] A. Mellouk, "Quality of Service Dynamic Routing Schemes for Real Time Systems in IP Network," *Proc. Networking Int'l Conf. Systems and Int'l Conf. Mobile Comm. and Learning Technologies ICN/ICONS/MCL,* p. 93, 2006.

[12] D. Mitra and K.G. Ramakrishnan, "A Case Study of Multiservice, Multipriority Traffic Engineering Design for Data Networks," *Proc. IEEE Int'l Global Telecomm. Conf. (GLOBECOM '99),* pp. 1077-1083, 1999.

[13] D. Applegate and M. Thorup, "Load Optimal MPLS Routing with $N + M$ Labels," *Proc. IEEE INFOCOM,* 2003.

[14] X. Yuan, "Heuristic Algorithms for Multiconstrained Quality-of-Service Routing," *IEEE/ACM Trans. Networking,* vol. 10, no. 2, pp. 244-256, Apr. 2002.

[15] S. Chen and K. Nahrstedt, "Distributed Quality-of-Service Routing in Ad-Hoc Networks," *IEEE Selected Areas in Comm.,* vol. 17, no. 8, pp. 1488-1505, Aug. 1999.

[16] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole, "A Distributed Real-Time MPEG Video Audio Player," *Proc. Fifth Int'l Workshop Network and Operating System Support of Digital Audio and Video (NOSSDAV '95),* pp. 151-162, 1995.

[17] F. Goktas, F.M. Smith, and R. Bajcsy, "Telerobotics over Communication Networks," *Proc. 36th IEEE Conf. Decision Control,* pp. 2399-2404, 1997.
[18] N. Tran and K. Nahrstedt, "Active Arbitration by Program Delegation in Video on Demand," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* pp. 96-105, 1998.
[19] J.N. Al-Karaki, A.E. Kamal, and R. Ul-Mustafa, "On the Optimal Clustering in Mobile Ad Hoc Networks," *Proc. IEEE Consumer Comm. and Networking Conf.,* pp. 71-76, 2004.
[20] R. Ghosh and S. Basagni, "Mitigating the Impact of Node Mobility on Ad Hoc Clustering," *Wireless Comm. and Mobile Computing,* vol. 8, no. 3, pp. 295-308, 2008.
[21] P. Gupta and P.R. Kumar, "The Capacity of Wireless Networks," *IEEE Trans. Information Theory,* vol. 46, no. 2, pp. 388-404, Mar. 2000.
[22] I. Gupta, K.P. Birman, and R. Van Renesse, "Fighting Fire with Fire: Using Randomized Gossip to Combat Stochastic Scalability Limits," *Int'l J. Quality and Reliability Eng.,* vol. 18, no. 3, pp. 165-184, May/June 2002.
[23] Y. Wang and M.S. Kim, "Bandwidth-Adaptive Clustering for Mobile Ad Hoc Networks," *Proc. Int'l Conf. Computer Comm. and Networks,* pp. 103-108, 2007.
[24] G. Dimitriadis and F.N. Pavlidou, "Clustered Fisheye State Routing for Ad Hoc Wireless Networks," *Proc. IEEE Fourth Int'l Workshop Mobile and Wireless Comm. Network,* pp. 207-211, 2002.
[25] A.H. Thamrin, H. Kusumoto, and J. Murai, "Scaling Multicast Communications by Tracking Feedback Sensors," *Proc. Int'l Conf. Advanced Information Networking and Applications (AINA),* pp. 1-6, 2006.
[26] L. Gu, Z. Niu, J. Lv, and H. Yoshiuchi, "An NAK-Based Hierarchical ARQ Scheme for Reliable Data Multicast in Integrated Communication and Broadcast Networks," *Proc. Asia-Pacific Conf. Comm. (APCC),* pp. 1-5, 2008.
[27] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidermann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in Network Simulation," *Computer,* vol. 33, no. 5, pp. 59-67, May 2000.

Larry C. Llewellyn received the MS degree in electrical engineering from the Air Force Institute of Technology (AFIT) in March 2007. Currently, he is the System Validation Branch chief at the Air Force Network Integration Center. His research interests lie in the area of fault-tolerant and reliable networks.

Kenneth M. Hopkinson received the PhD degree in computer science from Cornell University in 2004. He is an associate professor of computer science at the Air Force Institute of Technology. His research interests are in robust distributed systems. He is a member of Upsilon Pi Epsilon, Eta Kappa Nu, the IEEE, and the IEEE Computer Society.

Scott R. Graham received the PhD degree in electrical engineering from the University of Illinois at Urbana-Champaign in 2004. He is an adjunct professor of computer engineering at the Air Force Institute of Technology. His research interests include directional networks and networked control systems. He is a member of Tau Beta Pi, Eta Kappa Nu, and the IEEE.