Improving Routing and Network Performance in Mobile Ad Hoc Networks Using Quality of Nodes

Navid Nikaein and Christian Bonnet
Institut Eurécom
Sophia Antipolis, France
`Firstname.Name@eurecom.fr`

***Abstract*—In this paper we suggest a mechanism to describe the quality of nodes over time from the network point of view, and use this quality for extracting the links connecting the pair of best nodes. Therefore, links (or nodes) can be properly selected so as to improve the routing performance. We also present a distributed algorithm to construct a forest of nodes with the high quality. The constructed forest reduces the broadcasting overhead by selecting a subset of the neighboring nodes for forwarding a packet. Furthermore, the subset of the forwarding nodes belongs to the set of nodes with the high quality, which in turn improves routing performance. We also provide two simulation results to show the efficiency of the algorithm.**

***Index Terms*—Manet, routing, broadcasting, QoS, graph-theory.**

I. INTRODUCTION

Mobile ad hoc networking is a challenging task due to the frequent changes in network topology as well as the lack of wireless resources. As a result, routing in such networks experiences link failure more often. Hence, it is essential that a routing protocol for an ad hoc network considers the reasons for link failure to improve the routing performance. Link failure stems from node mobility and lack of network resources both reside in the wireless medium and in nodes. Therefore it is essential to capture the aforesaid characteristics to identify the quality of nodes and hence the quality of links. In this paper we suggest a mechanism to describe the *quality* of nodes over time from the network point of view, and use this quality for extracting the *links* connecting the pair of best nodes. Due to the reasons of link failure in mobile ad hoc networks, this quality should not only reflect the available network resources but also the stability of such resources. Therefore, links (or nodes) can be properly selected so as to improve the routing performance. We also present a distributed algorithm to construct a forest of nodes with the high quality. The rationale behind the forest construction is to reduces the broadcasting overhead since a subset of the set of neighboring nodes is selected for forwarding a packet. Furthermore, the subset of the forwarding nodes belongs to the set of nodes with the high quality, which in turn improves routing performance. This forest is dynamic because the quality of nodes change over time, and also it is connected thanks to gateway nodes.

II. QUALITY OF NODES

Unlike fixed/wired networks, the performance of ad hoc routing strictly depends on the "quality" of each individual node. This quality should not only represent the available network resources reside both in the wireless medium and in the mobile nodes but also the stability of these resources. This is because mobile ad hoc networks potentially have less resources than wired networks. Furthermore, mobility may result in link failure which in turn may result in a broken path. Therefore, more criterion are required in order to capture the quality of the links between nodes. We define *QoS state* as the *buffer level* and *stability level* of a node to represent the "quality" of nodes, and hence the quality of network [1]. In this work for the sake of simplicity, we only consider buffer level as the available network resources in a *symmetric* environment where all nodes have similar capabilities such as transmission range and buffer capacity. However, additional QoS states such as link SINRs and power level can be used based on an asymmetric environment. Note that a QoS state is internal to a node and it is periodically evaluated by each node. The *QoS state* of a particular node reveals whether the node is *forced* to be *selfish* or not. In the *selfish* mode, a node ceases to be a router and acts only as a host. We assume that each node periodically broadcasts its presence and its QoS state in the form of a *beacon* to its neighboring nodes.

- ***Buffer Level*—** which stands for the available unallocated buffer. It represents a node's internal state, and we assume that a node is capable of determining its state. Note that if the buffer level of a particular node is low, then this implies that a large number of packets are queued up for forwarding, which in turn implies that a packet routed through this node would have to experience high queuing delays. This metric is translated into a two-bit code which indicates the QoS state of a node in terms of available buffer. This two-bit code is used to indicate *high*, *medium*, *low* and *selfish* QoS state in terms of the buffer level. A *high* QoS state indicates that the corresponding node no packets queued up for forwarding, while *selfish* QoS state shows that the available buffer is less than 25 percent of its size. Since there is a slight delay between the broadcast of this metric and its use, instantaneous buffer-level may be misleading. Hence, a node should maintain the average buffer-level such as exponentially weighted moving average (EWMA).
- ***Stability Level*—** we define the connectivity variance of a node with respect to its neighboring nodes over time as the stability of that node. This metric is used to avoid unstable nodes to relay packets. We estimate the stability of a node x as:

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

  N_{t_1} and N_{t_0} represent the nodes in the neighborhood of x at times t_1 and t_0 respectively. Note that, $t_1 - t_0$ denotes the time period in which nodes exchange beacons. A node is unstable if a large number of its neighbors change. Further, if most (or all) of the neighbors remain the same at the two times t_1 and t_0, then we call this node stable. Note that $N_{t_1} \cap N_{t_0}$ (the numerator of $stab(x)$) denotes the set of nodes that have remained in the neighborhood of x between times t_0 and t_1. The denominator of $stab(x)$ is a normalization term. A node has *high* stability if none of its neighbors change ($N_{t_1} = N_{t_0}$) , in this case we have $stab(x) = 1$. A node is *unstable* (no stability), if all its neighbors change ($N_{t_1} \cap N_{t_0} = \phi$), in this case we have $stab(x) = 0$. We say that a node has *low* stability if $0 < stab(x) \leq 0.5$ and that it has *medium* stability if $0.5 < stab(x) < 1$. A two-bit code maps the stability to four QoS states of *high*, *medium*, *low* and *no* stability. For the sake of conformity with the other metric, if a node has *no* stability, we say that it has *selfish* stability.

In order to facilitate the notion of QoS state, we need to map the the QoS state onto a single weighted metric which can be compared and whose best can be chosen. Suppose s and b denote the stability and buffer levels of a particular node. Note that $s, b \in \{0, 3\}$ since we are using a two bit code to capture these metrics. We say that the QoS state of a node V is:

$$V = f(s, b) = \alpha \cdot s + \beta \cdot b$$

The weights α and β denote the relative importance of stability and buffer level amongst themselves. Since we desire stability to be most important followed by the buffer level, we propose $\alpha = 2$ and $\beta = 1$. Hence, given two nodes, we are always in a position to select the *better* one. For example in Fig. 1, if a node f has $s = 2$, $b = 3$ then its QoS state is: 7. On the other hand a node k with $s = 3$, $b = 3$ has a QoS state value of 9. Hence, in our scheme, node k is a "better" node than node f.

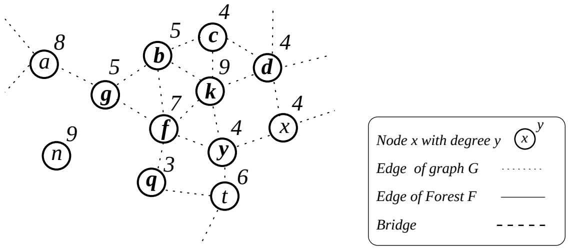
III. ALGORITHM FOR FOREST CONSTRUCTION WITH HIGH QUALITY NODES

Quality of service forest (QoS-F) is a distributed algorithm to construct a forest of high quality links (or nodes) from network point of view, and use these for routing and broadcasting. The main objectives of QoS-F are to reduce the overhead of broadcasting and to improve routing performance. To reduce the broadcasting overhead, QoS-F elects the smallest subset of the set of neighboring nodes in order to forward a packet. A packet is forwarded if the message has not been received by the node before, and the node belongs to the subset of the forwarder nodes. On the other hand, it improves routing performance since the subset of forwarding nodes belongs to the set of high quality nodes. The QoS-F algorithm consists of three cyclic time-ordered phases: preferred neighbor election, quality of service forest construction, and neighboring table construction, which are carried out based on the information provided by *beacons*. A beacon is a periodic message exchanged *only* between a node and its neighboring nodes. We assume that initially each node knows the QoS state of its neighboring nodes. Then, each node in the network topology carries out the preferred neighbor election algorithm to choose a preferred neighbor. The preferred neighbor of a node is the node that owns maximum neighborhood QoS state among neighboring nodes. Then, a forest is constructed by connecting each node to its preferred neighbor and vice versa. It has been proven that whatever is the network topology, connecting each node to its preferred neighbor always yields a forest (i.e. we have no cycle) [2]. The neighboring table contains the set of nodes with which there exist a direct link over which data may be transmitted. However, only a subset of them are selected to forward a packet, hence reducing broadcasting overhead and therefore increasing network lifetime. Furthermore, this subset provides the high quality connections between nodes, and thus improving routing performance.

A. *Preferred Neighbor Election*

Let x and y be any nodes of the graph $G = (V, E)$. We assume that initialy each node x knows the ID numbers and the QoS state of its neighboring nodes. However, they are periodically provided in a beacon. Based on these two information, node x can determine its PN. For this purpose, node x computes a set of nodes whose QoS state are equal to maximum neighborhood QoS state. This set is denoted by $PN_x = \{y | QoS\_State(y) = \max(QoS\_State(N_x))\} = \{y | V(y) = \max(V(N_x))\}$, where N_x is the neighboring nodes of node x. We distinguish three cases:

- ***No PN***— if the set is empty, then node x has no PN which means it has no neighbors. In Fig. 1, node n has no neighbor and consequently no PN;
- ***Single PN***— if the PN_x has only one member, then this member is the elected PN. For example, in Fig. 1, node f has five neighbors: k, y, q, b, g, but the set of PN_f has only one member which is the node k;
- ***Multiple PN***— the set of PN_x can have more than one member which is the case for node x, since $PN_x = \{y, d\}$. This means that there are more than one neighbor with the maximum neighborhood QoS state. In this case, we assume that node x elects a node with the greatest ID number. So, node x elects node y since its ID number is greater than node d (regarding to the alphabetical order).



(a) An arbitrary graph G (b) Legend

Fig. 1. Each node in the graph is characterized by its QoS state and a letter which represents its ID number, and we assume that each node knows the QoS state and the ID number of its neighboring nodes.

Consequently the main idea of the algorithm is to select, for each node n in the network topology, a neighbor that has the maximum QoS state in the neighborhood. For nodes that evaluate two identical QoS state values, we break ties by setting the convention that nodes with higher IDs are preferred. We say that node y is the *preferred neighbor* of node x, if y is in the neighborhood of x and has the maximum QoS state among its neighbors. In this manner, each node in the network selects a preferred neighbor, and we can obtain a forest. Note that a node that has any one of these metrics as *selfish* is not considered for preferred neighbor election (see section II).

B. *Quality of Service Forest Construction*

A forest is built by connecting each node to its PN. In [2], it is proven that, whatever is the network topology, this approach always yields a forest (i.e. we have no cycle). This is because the way in which a node is elected follows a *monotonic increasing function* depending on its QoS state and on its ID number. Fig. 2 shows the forest of high quality links (or nodes).

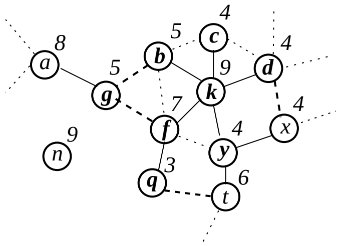


Fig. 2. Constructed forest with high quality links.

C. Beaconing and Neighboring Table Construction

Basically, neighboring table is the table through which node x detects changes to its neighborhood. This table consists of three information: a neighboring ID (NID), its QoS state (QoS_STATE), and whether node x is a forwarder node of this neighbor (FORWARDER_STATE). The value of FORWARDER_STATE is null at the beginning, which represents that node x is a forwarder of its entire neighborhood. Such information is considered valid for a limited period of time, and must be refreshed periodically to remain valid. Expired information is purged from the table. Table I shows the neighboring table of node f in Fig. 2. One of our goal is to reduce the number of neighbors whose forwarder node is x, hence reducing broadcasting overhead.

To provide first two information in the neighboring table, each node periodically broadcasts its presence and its QoS state in the form of a *beacon*. Upon receiving a beacon, a node can gather information describing its neighborhood, as well as detect the quality of every neighbors to act as a router. Based on the neighboring QoS state and ID, node x performs the preferred neighbor election algorithm to choose its preferred neighbor, say y. As soon as node x determines its PN y, it sets node y as a forwarder (a direct forest member) in its neighboring table, and then it must notify its neighboring nodes, especially y, of its decision. Therefore, node x sets its beacon to $B_x = (x, QoS\_STATE, y)$. Upon receiving x's beacon, each node updates its information regarding x and verifies whether they have been chosen as the PN of x. Among the neighboring nodes of x, the PN y sets node x as a forwarder too. Other neighboring nodes of x set y as a non-forwarder node in their neighboring tables if node x has already been set as a forwarder node. In this way, we say that y is learned to *only* be the forwarder of node x. On the other hand, if node x has not been set as a forwarder in the neighboring tables, then node y becomes a forwarder because it is considered as a gateway node connecting two different trees of the same forest. It has to be mentioned, nodes that are not in the same tree but they are in the direct transmission range of each other are called gateway nodes. Consequently, the set of forwarder nodes is reduced. For example in Fig. 2, node f elects node k as its PN, and sends a beacon $B_f = (f, QoS\_State, k)$. Thus node k becomes a forwarder node in f's table. Similarly, node f becomes a forwarder for node k. Furthermore, nodes f can be learned by the nodes b, y as a non-forwarder nodes because k is the elected PN of both b and y. Likewise, nodes b, y can be learned by node f as a non-forwarder nodes since node k is set as a forwarder of node f. Node g, on the other hand, becomes a forwarder node for node f because it is not a PN of any neighbor of f including f itself. Hence, the set of forwarder nodes of node f is reduced from 5 to 3. Another advantage is that the leaf nodes becomes non-forwarder nodes, e.g. node c. Table I shows the neighboring table of node f.

TABLE I

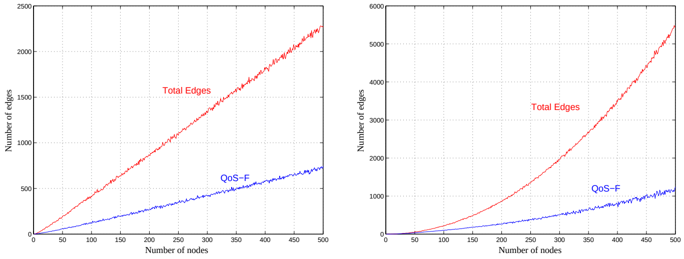
NEIGHBORING TABLE OF NODE f

NID	QoS_STATE	FORWARDER_STATE
k	9	yes
y	4	no
q	3	yes
b	5	no
g	5	yes (gateway)

However, formation of some fake-gateway nodes and thus forwarder nodes seems inevitable unless we provide some knowledge of the tree member for each node [2]. We believe that in a highly mobile network, providing such information is a waste of wireless scarce resources. Moreover, it increases the complexity of the algorithm. Therefore, we compromise some fake-forwarder nodes against network resources and the complexity of the algorithm. In Fig. 2, node q is a fake-forwarder node of t and vice versa.

D. Results

The following results were obtained by implementing the static QoS-F algorithm in C++ and measuring the metrics after the population of mobile nodes was distributed uniformly on a grid of 2000m × 2000m with each node having a transmission range of 250m. One key aspect of this measurement is how QoS-F behave with an increasing number of nodes in the network. The graph in Fig. 3 shows the number of edges in the topology and the number of edges in the QoS-forest versus the number of mobile nodes. We consider two cases: variable density where the network is sparse at the beginning and becomes highly dense, and constant density where the area covered by the ad hoc network increases as the number of nodes increases. There is a theoretical fact that the number of edges in a tree is of the order of the number of nodes. For a tree with n nodes, the number of edges must be $n - 1$. Further, since a forest has strictly lesser edges than a tree spanning all the nodes in QoS-F, the number of edges in the forest as reflected by Fig. 3 is $O(n)$.



(a) Constant Density (b) Variable Density

Fig. 3. Total number of the edges in the network topology vs. the total number edges generated by the forest.

IV. CONCLUSION

We have addressed a mechanism to describe the *quality* of nodes over time from the network point of view and use this quality for extracting the links connecting the pair of best nodes. This quality is introduced in response to the reasons of link failure in mobile ad hoc networks. Therefore, links can be properly selected so as to improve routing performance. We have also proposed a distributed algorithm to construct a quality of service forest in order to reduce the overhead of broadcasting. Forest is used to reduce the broadcasting overhead since a subset of the set of neighboring nodes is selected for forwarding a packet. Furthermore, the subset of the forwarding nodes belongs to the set of nodes with the high quality, which in turn improves routing performance.

REFERENCES

[1] Navid Nikaein and Christian Bonnet, "Layered quality of service model for routing in mobile ad hoc networks," in *WMAN*, 2003.
[2] Navid Nikaein, H. Labiod, and C. Bonnet, "DDR-distributed dynamic routing algorithm for mobile ad hoc networks," in *MobiHOC*. IEEE, 2000.