A Cluster-based Approach for Routing in Dynamic Networks*

P. Krishna
High Performance Networking
Digital Equipment Corporation
Littleton, MA 01460
pkrishna@hpn.lkg.dec.com

N. H. Vaidya, M. Chatterjee, D. K. Pradhan
Department of Computer Science
Texas A&M University
College Station, TX 77843
{vaidya,mitrajit,pradhan}@cs.tamu.edu

Abstract

The design and analysis of routing protocols is an important issue in dynamic networks such as packet radio and ad-hoc wireless networks. Most conventional protocols exhibit their least desirable behavior for highly dynamic interconnection topologies. We propose a new methodology for routing and topology information maintenance in dynamic networks. The basic idea behind the protocol is to divide the graph into a number of overlapping clusters. A change in the network topology corresponds to a change in cluster membership. We present algorithms for creation of clusters, as well as algorithms to maintain them in the presence of various network events. Compared to existing and conventional routing protocols, the proposed cluster-based approach incurs lower overhead during topology updates and also has quicker reconvergence. The effectiveness of this approach also lies in the fact that existing routing protocols can be directly applied to the network – replacing the nodes by clusters.

1 Introduction

Dynamic networks consist of mobile hosts which can communicate with each other over the wireless links (direct or indirect) without any static network interaction. In such networks the mobile host has the capability to communicate directly with another mobile host in its vicinity. The mobile hosts also have the capability to forward (relay) packets. Examples of such networks are *ad-hoc* wireless local area networks [4, 15, 19, 25] and packet radio networks [3, 16, 18, 24]. The term *ad-hoc* network is in conformance with current usage within the IEEE 802.11 subcommittee [4].

Example applications of such networks range from conference rooms to battlefields. To communicate with each other, each mobile user needs to connect to a static network (wide area network, satellite network). However, there might be situations where connecting each mobile user to a static network may not be possible due to lack of facilities, or it may be expensive. In such situations, it would be preferable for the mobile users to set up communication links between themselves without any static network interaction [15].

*Research reported is supported in part by AFOSR under grant F49620-94-1-0276, Texas Advanced Technology Program under grants 009741-052-C and 999903-029.

An important issue in dynamic networks is the design and analysis of routing schemes. This paper investigates the consequence of mobility and disconnections of mobile hosts on the design and performance of routing protocol in a dynamic network. Due to the limited range of the wireless transreceivers, a mobile host can communicate with another host only within a limited geographical region around it. Thus, it may be necessary for a mobile host to require the aid of other mobile hosts in forwarding data packets to their destination. The routing information will thus be maintained at the mobile hosts to assist in forwarding packets to other hosts. The problem here is the complexity of updating the routing information in such a dynamic network.

1.1 Previous Work

Numerous routing protocols have been proposed in recent years. One of the most popular techniques for routing in communication networks is via distributed algorithms for finding shortest paths in weighted graphs [5, 7, 14, 22, 23, 21, 27]. These distributed algorithms differ in the way the routing tables at each host are constructed, maintained and updated. The primary attributes for any routing protocol are :

- Simplicity : Simple protocols are preferred for implementation in operational networks [25].
- Loop-free : At any moment, the paths implied from the routing tables of all hosts taken together should not have loops. Looping of data packets results in considerable overhead.
- Convergence characteristics : Time required to converge to new routes after a topology change should not be high. Quick convergence is possible by requiring the nodes to frequently broadcast the updates in the routing tables.
- Storage overhead : Memory overhead incurred due to the storage of the routing information should be low.
- Computational and transmission overhead : It is particularly important to limit these two in mobile wireless networks because the bandwidth of a wireless link is limited, and because mobile devices are typically low-power in order to be portable, and hence do not have the resources for many transmissions and lengthy computations.

Conventional routing protocols can be broadly classified as *distance vector* and *link state* protocols. *Distance vector* routing uses the classical distributed Bellman-Ford algorithm [1, 12, 16, 21]. Each host maintains for each destination a set of distances through each of its neighbors. In order to maintain up-to-date information, each host periodically broadcasts to each of its neighbors, its current estimate of the shortest path to every other host in the network. For each destination, the host determines a neighbor to be the next hop for that destination if the neighbor has the shortest path to the destination.

Link state routing generally requires each host to have knowledge of the entire network topology [23]. However, there are link state algorithms (e.g., Nimrod [2]) in which nodes maintain only partial information about the network topology. To maintain consistent information, each host monitors the cost of each communication link to each of its neighbors, and periodically broadcasts an update of this information to all other hosts in the network. Based on this information of the cost of each link in the network, each host computes the shortest path to each possible destination host. The processing overhead and the network bandwidth overhead of *link state* protocols are generally more than *distance vector* protocols.

The problems in using conventional routing protocols in a dynamic network have been discussed in great detail in [15, 25]. For completeness sake, we briefly list the problems in the following.

- Existing protocols could place a heavy computational burden on mobile computers in terms of battery power, and the wireless networks in terms of network bandwidth.
- Convergence characteristics of these protocols are not good enough to suit the needs of dynamic networks.

The protocol described in [25] addresses some of the above stated problems by modifying the Bellman-Ford routing algorithm. They use sequence numbers to prevent routing table loops, and, settling-time data for damping out fluctuations in route table updates. The convergence on the average is rapid, however, the worst case convergence is large. Moreover, their protocol requires frequent broadcasts of the routing table by the mobile hosts. The overhead of the frequent broadcasts goes up as the population of mobile hosts increases. Another scheme based on distance vector path-finding algorithm was proposed in [24]. Although loops are avoided completely, all the nodes end up sending an update message to their neighbors during a topology update operation. In dynamic networks, where topology updates are frequent, the update overhead may be very high.

Johnson proposed a new routing method for adhoc networks based on separate route discovery and route maintenance protocols [15]. The concept of Address Resolution Protocol (ARP) is extended to discover routes. However, if proper measures are not taken, the network performance can degrade due to the propagation of redundant route discovery requests. Route maintenance is achieved by using hop-by-hop acknowledgement. However, due to such relaxed maintenance measures, the hosts can be using poor (long) routes when better (shorter) routes are available. This will degrade the network performance.

A loop-free routing protocol for dynamic networks is proposed in [5]. Routing optimality is of secondary importance. Rather, their goal is to maintain connectivity between the hosts in a fast changing topology. A distributed routing protocol for mobile packet radio networks is proposed by Corson et al. [3]. Similar to [5], routing optimality is of secondary importance. Instead of maintaining distances from all sources to a destination, the protocol guarantees route maintenance only for those sources that actually desire routes. This property helps in reducing the topology update overhead. However, because of the query-based synchronization approach to achieve loop-free paths, the communication complexity could be high.

1.2 Proposed Approach

This paper presents a new methodology for routing and topology information maintenance in dynamic networks [19]. Our approach is motivated by our study of existence of clusters (size greater than 2) in random graphs. The basic idea behind the approach is to divide the graph into number of overlapping clusters. A change in the network topology corresponds to a change in the cluster membership. The performance of the proposed routing approach (reconvergence time, and topology update overhead) will then be determined by the average cluster size in the network.

For future reference, let us formally define *clusters*.

Definition 1 *A* k-cluster *is defined by a subset of nodes which are mutually 'reachable' by a path of length at most* k *for some fixed* k*. A* k-cluster *with* $k = 1$ *is a clique.* □

This work deals with clusters of $k = 1$, i.e., *1-clusters.* (Hereafter, we refer to a *1-cluster* simply as cluster.) Each cluster is identified by its members.

Definition 2 *The* size*, $S(C)$ of a cluster C is the number of nodes in C.*□

Definition 3 *Edges of a cluster are the edges between nodes that are members of the cluster.*

Definition 4 *A graph is* cluster-connected *if it satisfies the following two conditions :*
1) The union of the clusters covers the whole graph.
2) For a connected graph, there is a path from each node to every other node through the edges of the clusters in the graph. □

This paper is a preliminary investigation into cluster-based approaches for dynamic networks. The protocol presented in this paper does not solve every possible routing problem. A number of different issues remain to be studied:

- Extensions of the protocols to support concurrent events. This paper deals with discrete events.
- Algorithms to create and maintain clusters such that

there is **always** more than one boundary node between any two overlapping clusters. This will add robustness during a boundary node failure.
• Generalization of 1-clusters to k-clusters (where $k > 1$). This will require design of more complex algorithms for cluster creation and maintenance. The interesting issue will be to determine if there is any performance improvement using k-clusters (where $k > 1$) instead of 1-clusters.
• In this paper we restrict to single-level clustering, which will be more suitable for moderately sized networks. For larger networks, a hierarchy of clusters needs to be looked into.

Section 2 presents the problem of routing in dynamic networks. Protocols to create and maintain the clusters are presented in Section 3. Section 4 presents the routing protocol based on clusters. Section 5 presents the performance evaluation of the cluster-based approach. Section 6 presents an overview of the other clustering approaches in literature. Concusions are presented in Section 7.

2 Preliminaries

The problem addressed in this paper can be defined as follows:
Given: A dynamic network configuration.
Problem: Find a 'good' loop-free routing between each pair of mobile hosts in the network, where the topological connectivity is subject to frequent unpredictable changes.

The problem requires a loop-free distributed routing protocol which determines an acyclic route between each pair of hosts whenever a change in the topology is detected. The protocol is intended for use in networks where the rate of topological change is not so fast as to make "flooding"[1] the only viable routing method, but not so slow as to make any static topology routing applicable. In a loop-free[2] route, the path from one host to another does not traverse the same node twice. Loop-free routing is desirable to minimize the consumption of resources during routing.

Our algorithm determines 'good' routes from one host to another which are not necessarily the shortest paths. In an environment of frequent topological change, a 'good' route's length is comparable to the shortest route. Each host maintains a data-structure describing the network topology and some routing information. The routing protocol adapts in a distributed fashion to arbitrary changes in topology in the absence of global topological knowledge. Let an undirected graph, $G = (V,E)$, represent a network of mobile hosts. Each node u in the graph denotes a mobile host H_u. Due to the limited range of wireless transceivers, a mobile host can communicate with another host only within a limited geographical region around it. This region is called the host coverage area – d being the radius. The geographical area covered by a host coverage area is a function of the medium used for wireless communication. A host H_u is in the vicinity of H_v if the distance between nodes u and v is less than or equal to d. An edge (u,v) connects node u and node v if the corresponding hosts are in the vicinity and have a communication link established between them. A host may sometime be isolated where there is no other mobile hosts in its vicinity. Such a host will be represented in the graph by a disconnected node. A host H_{v1} is connected to another host H_{v2} if there exists at least one path from node $v1$ to $v2$.

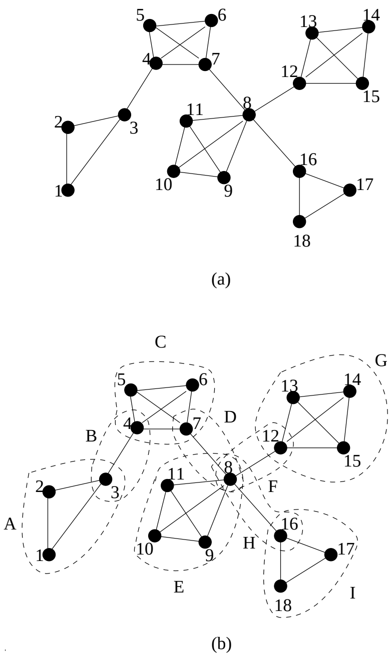

Figure 1: An Example of Clusters

Similar to [3, 24], an underlying link-level protocol is assumed which assures the following:

- A node is aware of all its neighbors at all times.
- All packets transmitted over a link are received correctly and in proper sequence within a finite time.
- All control messages are processed one at a time at the nodes in the order in which they occur.

Example 2.1: The graph in Figure 1(a) is formed based on geographical locations of 18 mobile hosts. In this example, the graph is connected as each node is *reachable* from every other node. It can be observed that based on the positions, some nodes form clusters. The graph can be divided into nine clusters as shown in Figure 1(b). The clusters and their respective members are as follows : A (1,2,3), B (3,4), C (4,5,6,7), D (7,8), E (8,9,10,11), F (8,12), G (12,13,14,15), H (8,16) and

[1] Flooding is an algorithm whereby a node broadcasts a message packet to its neighbors, who in turn broadcast the packet to all their neighbors, except the neighbor from which it was received. This process goes on till the message packet reaches the intended destination. This happens provided the destination is connected to the node which originated the flood [3].

[2] Loop-free routing requires prevention of loops in the routing tables. We assume that there is negligible amount of user traffic affected by the temporary loops.

I (16,17,18). If the routing information is based on clusters, routing from node 1 to node 16 will be done through the edges of the clusters A, B, C, D and F. The graph in Figure 1(b) is *cluster-connected* because, (i) the union of the clusters covers the whole graph, and (ii) there is a path from each node to every other node using the cluster edges. □

A topological change in the mobile host network corresponds to a change in the graph structure $G(V,E)$ to $G'(V',E')$. We outline four events that can cause changes in the graph (in the following H_A and H_B are mobile hosts) :

A) H_A switching ON: A host H_A switching ON will include itself in the graph and make connection with all the hosts in its 'vicinity'. Hence, $V' = V \cup \{A\}$ and $E' = E \cup \{(u, A)$, s.t. H_u is connected to $H_A\}$.

B) H_A switching OFF: A host H_A switching OFF will exclude itself from the graph and delete all its edges. Hence, $V' = V - \{A\}$ and $E' = E - \{(u, A)$, s.t. $(u, A) \in E\}$.

C) H_A gets connected to H_B: Here, an edge between A and B will be added to the graph. Hence, $V' = V$ and $E' = E \cup \{(A, B)\}$.

D) H_A gets disconnected from H_B: Here, the edge between A and B will be removed from the graph. Hence, $V' = V$ and $E' = E - \{(A, B)\}$.

A routing protocol will change its routing information based on the above four types of changes in the graph. We now present some definitions and properties which will assist in describing the proposed routing protocol.

Definition 5 *The cluster set S_n of a node n is defined as the set of all clusters in which n is a member.* □

Definition 6 *If cluster-connectivity between any pair of nodes (n, n') is not affected due to removal of a cluster C, then cluster C is redundant.* □

In other words, if two nodes initially cluster-connected are no longer cluster-connected after the removal of a cluster C, then cluster C is not redundant (i.e., non-redundant). For example, in Figure 1(b), there are no redundant clusters.

Definition 7 *A node is a* boundary *node if it is a member of more than one cluster.* □

In Figure 1(b), node 3 is a boundary node as it belongs to two clusters, (1,2,3) and (3,4). However, node 1 is not a boundary node as it only belongs to (1,2,3).

Property 1 *Addition of each new node to the graph adds at least one new non-redundant cluster. However, when the new cluster is added to the graph, the new cluster may cause one or more clusters to become redundant.* □

At least one new cluster should be added to include the new node. Otherwise, the graph will not remain cluster-connected after addition of the new node.

3 Cluster Formation

Our proposed routing protocol is based on the formation of clusters. Hence, efficient cluster formation will be the crux of a routing protocol of this nature. Clusters should be formed in such a way that the resulting graph is *cluster-connected* (See Definition 4). Routing from one node to another will consist of routing inside a cluster and routing from cluster to cluster. A change in the dynamic network may or may not result in a change in the cluster compositions. Here, we have assumed clusters with $k = 1$ (See Definition 1). As mentioned in Section 2, we have identified four different possible types of changes in the dynamic network graph in the occurence of a single event. We assume that each cluster has an unique identifier, *id*. Each node maintains a list of its neighbors, a list of clusters (*Clus_List*) in the network, and a list of boundary nodes (*Bound_List*) in the network. There can be multiple boundary nodes between overlapping clusters. If there are multiple boundary nodes between clusters, the one[3] can also be used with the biggest cluster set is chosen to be the boundary node and is maintained in the *Bound_List*. Note that a node can be a boundary node for more than two overlapping clusters.

In a connected network, *Clus_List* is the same in all the nodes. It is not true in a partitioned network. This is because nodes in a partitioned network may not be aware of all the clusters in the network. Unless otherwise mentioned, the following discussions of the protocols consider a connected graph. Thus, unless otherwise mentioned, all the nodes in the network have the same *Clus_List*. We now present the protocols for cluster updates with each type of topological change. The proof of correctness for these protocols is presented in Appendix A.

3.1 Host H_A switches ON

The new graph structure $G'(V',E')$ is formed with the added node. The new node A will result in at least one new cluster so that with the cluster, node A can route to the rest of the graph. However, if A connects two disjoint subgraphs, it may result in more than one added cluster. These new clusters are denoted by *essential* clusters and are determined by A itself. The addition of new clusters may result in zero or one or more clusters being redundant. The two tasks performed during the topological change are (i) addition of new clusters, and (ii) removal of redundant clusters. The goal is to have a small number of clusters such that the network remains *cluster-connected*. Note that although the algorithm tries to minimize the number of clusters, it does not guarantee a minimum number of clusters. The protocol initiated by new node A is shown in Table 1.

The new node A broadcasts a message to its neighbors indicating its arrival. Upon receipt of the arrival message, the neighbors send a list of their neighbors, and *Clus_List* to A. Based on the neighbor information received from its neighbors, A determines the possi-

[3] For simplicity, we retrict to single boundary node. Using multiple boundary nodes may be more robust and also distribute the packet forwarding load. We believe our algorithms can be extended to use multiple boundary nodes with minor changes.

Table 1: Switch ON Procedure

	Procedure Switch ON(A);
	Begin;
1.	A sends messages to its neighbors about its arrival;
2.	Each neighbor sends list of its neighbors and *Clus_List* to A;
3.	A determines those clusters that are included in the cluster set of its neighbors and stores them in *Local_List*;
4.	A uses the neighbor information and creates new clusters using **Create Clusters** (A) and stores them in *All_List* ;
5.	A executes **Find Essential**(A,*All_List*);
6.	A assigns new *ids* to the *Essential Clusters*;
7.	A appends the *Essential Clusters* to *Local_List*;
8.	A executes **Find Redundant** (*Local_List*) ;
9.	A appends *Local_List* returned by **Find Redundant** to *Clus_List*;
10.	A determines new boundary nodes from the updated *Clus_List* ;
11.	A broadcasts the updated boundary node list (*Bound_List*) and cluster list (*Clus_List*) to its neighbors;
12.	Updated boundary node list and cluster list is then propagated to rest of the network by only the boundary nodes;
	End;

Table 2: Create Clusters Function

Function Create Clusters(A);

Data Structures:
C_i = i-th Cluster;
$Neighbor(n)$ = List of neighbors of node n that are **also** neighbors of A;
DONE(n) = Indicator of whether clusters including node n have been already created or not.

Initialization:
$1 \leq n \leq |V|$, DONE(n) = FALSE;
$All\_List = \{\emptyset\}$
i = 1;

Begin;
For each node x in $Neighbor(A)$ do
    $C_i = \{x, A\}$;
    For each node y in $Neighbor(x)$ do
        if DONE(y) = FALSE
            if $C_i \subseteq Neighbor(y)$
                $C_i = C_i \cup \{y\}$;
            else
                $All\_List = All\_List \cup C_i$; i = i + 1;
                $C_i = \{x, y, A\}$;
                $All\_List = All\_List \cup C_i$; i = i + 1;
    DONE(x) = TRUE ;
End;

ble clusters using the **Create Clusters** function shown in Table 2, and stores them in *All_List*. The clusters that the **Create Clusters** function determines depends heavily on the order in which each node is added to the network. This function will not return the maximum clique for all the orders. This function uses a 'first-fit' strategy to generate clusters, which does not necessarily produce maximum sized clusters. The time complexity of the **Create Clusters** function is $O(\mathcal{D}^3)$, where $\mathcal{D}$ is the maximum nodal degree[4].

Property 2 *The clusters returned by the* **Cluster Create** *function are characteristic of the order in which each node is added to the network.* □

Figure 2 illustrates an example where different node numbering (i.e., order in which a node is added to the network) leads to two different sets of clusters being created at the new node (Node 6) by the **Create Clusters** function. Note that a cluster of the largest size may or may not be determined by the algorithm[5] presented in **Create Clusters** function. However, as has been shown later, the algorithm ensures the connectivity of the new node with its neighbors through the clusters.

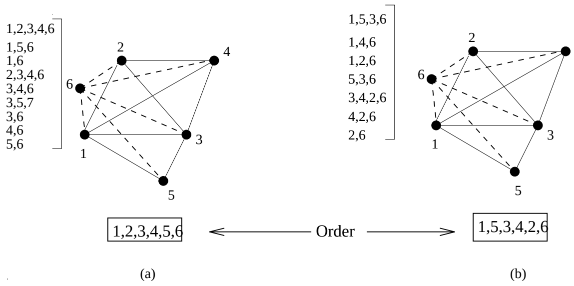

Figure 2: Different Clusters Created at New Node for Different Orders of Node Addition

Once, the clusters are created using **Create Clusters**, the new node A then executes the **Find Essential** function shown in Table 3. The **Find Essential** function sorts the clusters in *All_List* in a non-descending order of their sizes. Initially all the clusters are marked *essential*. Each *essential* cluster C is then examined to find if a node (other than the new node A) in C is a member of any other *essential* clusters. If so, it marks the cluster C as *non-essential*. This will ensure that a node (other than the new node A) is a member of no more than one *essential* cluster. Moreover, since the clusters are sorted in a non-descending order of their sizes, the **Find Essential** function returns the largest clusters possible. The *essential* clusters determined by **Find Essential** function are stored in *Essential Clusters*.

The new node A assigns new cluster *ids*[6] to the *essential clusters*. It then appends the *essential* clusters to the list of local clusters (*Local_List*). The list of local clusters (*Local_List*) is obtained from *Clus_List* (Step 3 of **Switch ON**). Local clusters are those clusters in *Clus_List* which are also included in the cluster set of A's neighbors.

Addition of the *essential* clusters may make one or more existing clusters *redundant*. The new node A then executes the **Find Redundant** function shown in Table 4. Node A first sorts the clusters in *Local_List* in ascending order of size. Clusters of same size are sorted in the order of ascending *ids*. The **Find Redundant** function then determines *redundant* clusters based on Definition 6. The new cluster list is then obtained by appending the clusters remaining in *Local_List* after removing the *redundant* clusters, to *Clus_List*. Node A then determines the list of boundary nodes (*Bound_List*) from the updated *Clus_List*. If there are multiple boundary nodes between overlapping clusters, the one with the biggest cluster set is chosen to be the boundary node. Node A then broadcasts the updated boundary node list (*Bound_List*) and cluster list *Clus_List* to its neighbors. The neighbors then replace their cluster list and boundary node list with the ones obtained from A. The updated boundary node list (*Bound_List*) and cluster list *Clus_List* is then propagated to the rest of the network only by the boundary nodes.

If $\mathcal{B}$ is the upper bound on the number of boundary nodes, and $\mathcal{D}$ the maximum nodal degree, the message complexity of **Switch ON** is $O(\mathcal{B}+\mathcal{D})$. The number of boundary nodes, $\mathcal{B}$, is upper bounded by the number of nodes in the network, N.

Example 3.1: For an easier understanding, Figure 3 gives an example involving a network with 4 nodes. Figure 3(a) has 4 nodes and two clusters, namely, (1,2,3) and (2,3,4). When node 5 is switched ON, it sends messages to nodes 1, 3, and 4 (Figure 3(b)). On receiving information back from the nodes 1, 3 and 4, node 5 forms clusters (1,3,5), (3,4,5) and (4,5) as seen in Figure 3(c). It chooses (3,4,5) as the *essential cluster* and then determines redundant clusters from the cluster list of {(1,2,3), (2,3,4), (3,4,5)}. In the redundant removal phase, the new node 5 detects the cluster (2,3,4) to be redundant. The final clusters are (1,2,3) and (3,4,5) as in Figure 3(d).□

3.2 Host H_A switches OFF

When host H_A turns OFF, its disappearance will only be detected by its neighbors. The clusters in the cluster-set of node A shrinks in size. The neighbors of node A who are cluster-mates of the shrunken cluster will 'expand' the cluster. By expanding a cluster, we mean that the neighbor will determine new nodes to become members of that cluster. Neighbors of node A that are not cluster-mates of A will not initiate any update procedures.

There could be more than one node detecting the removal of a node. The **Switch OFF** procedure is similar to the **Switch ON** procedure in the sense that there are new clusters formed and redundant clusters removed. Concurrent independent executions of the

[4] The maximum nodal degree could either be a configured number such that any node will admit at most $\mathcal{D}$ neighbors, or the maximum node degree observed in the network graph.

[5] Finding the largest size cluster is NP-Hard [9].

[6] Unique *ids* can be guaranteed if each node maintains a local counter which increments whenever a new cluster *id* is assigned by it. The *id* assigned to the new cluster is a combination of the counter value and the node identifier.

Table 3: Find Essential Function

	Function Find Essential(A, *All_List*);
	Begin;
1.	Sort the clusters in *All_List* in a non-descending order of their sizes;
2.	For each cluster $C \in$ *All_List* do
3.	$Mark(C)$:= *essential* ;
4.	For each cluster $(C \in list) \wedge (Mark(C) = essential)$ do
5.	For each node $(n \in C) \wedge (n \neq A)$ do
6.	For each cluster $(C' \in All\_List) \wedge (C' \neq C) \wedge (Mark(C') = essential)$ do
7.	if $(n \in C')$
8.	$Mark(C)$:= *non-essential*;
9.	break;
10.	if $(Mark(C) = essential)$
11.	*Essential Clusters* := *Essential Clusters* $\cup$ C;
	End;

Table 4: Find Redundant Function

	Function Find Redundant(*Local_List*);
	Initialization; Set of nodes: $S = \{\emptyset\}$; $T = \{\emptyset\}$;
	Begin;
1.	Sort the clusters in *Local_List* in non-descending order of their size. Clusters of same size are sorted in non-descending order of their *id*;
2.	For each cluster $C \in$ *Local_List* do
3.	$S = S \cup C$; /* Nodes in C are appended to S */
4.	For each cluster $C \in$ *Local_List* do
5.	$T = \{\emptyset\}$;
6.	$\forall C'$ s.t., $C' \in$ *Local_List*, $Mark(C') = FALSE$;
7.	For each cluster $C' \in$ *Local_List* $\wedge (C' \neq C) \wedge (Mark(C') = FALSE)$
8.	if$(T = \{\emptyset\})$
9.	$T = T \cup C'$; /* Nodes in C' get appended to T */
10.	$Mark(C') = TRUE$;
11.	else for each node $(i \in T)$
12.	For each cluster $C'' \in$ *Local_List* $\wedge (C'' \neq C) \wedge$ $(Mark(C'') = FALSE)$
13.	if$(i \in C'')$
14.	$T = T \cup C''$; /* Nodes in C'' get appended to T */
15.	$Mark(C'') = TRUE$;
16.	if$(T = S)$ /* Cluster-connectivity maintained */
17.	*Local_List* := *Local_List* $- C$;
	End;

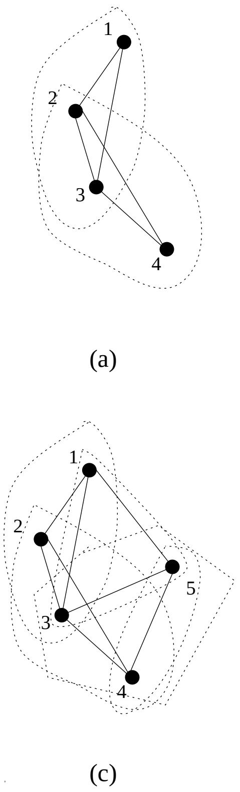

Figure 3: An Example of a Node Addition

Switch OFF procedure could lead to violation of the *cluster-connectivity* condition. We use an arbitration procedure to avoid concurrent independent executions. We require the node (neighbor of A, say, B) that is a cluster-mate of A in the most number of clusters, to initiate the **Switch OFF** procedure[7]. The execution of the **Switch OFF** procedure will expand those clusters in the cluster-set of A that node B is a member of. However, there still remain clusters in the cluster-set of A which do not contain B. In those remaining clusters, we determine the node (say, C) that is a member of the most number of clusters. This process continues till all the clusters in the cluster-set of A are covered. Unlike node B, node C will not execute the **Switch OFF** procedure. However, node C will just try to expand the shrunken clusters of which it is a part of, and not remove any redundant clusters. The new boundary list (*Bound_List*) and the new cluster list (*Clus_List*) is determined by C and broadcast to its neighbors. The lists are then further propagated to the rest of the network only by the boundary nodes.

The procedure initiated by node B is shown in Table 5. Let us illustrate it with an example.

Example 3.2: Figure 4 shows the cluster formations when a node is turned OFF in a network. Figure 4(a) has six nodes with three clusters, namely, (1,2,3), (2,3,4) and (4,5,6). When node 6 is turned OFF, the cluster (4,5,6) shrinks to (4,5) (Figure 4(b)). Node 4 and 5 detect node 6 switching OFF. Since, node 5 has the higher identifier, it initiates the **Switch OFF** procedure. Node 5 gets neighbor information and the cluster list from 3 and 4. It then expands the cluster (4,5) to (3,4,5) (Figure 4(c)). Node 5 now has {(1,2,3), (2,3,4), (3,4,5)} in the cluster list. In the redundant removal phase, node 5 detects the cluster (2,3,4) to be redundant. The final clusters are (1,2,3) and (3,4,5) as in Figure 4(d). □

[7] If there are multiple such nodes, we use a tie-breaking test; e.g., node with the larger *identifier*.

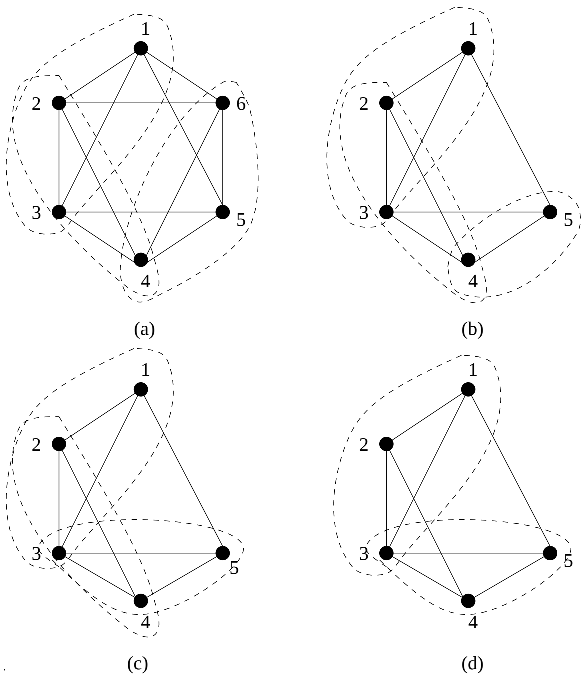

Figure 4: An Example of a Node Removal

The message complexity of **Switch OFF** is also $O(\mathcal{B}+\mathcal{D})$, where, $\mathcal{B}$ is the upper bound on the number of boundary nodes, and $\mathcal{D}$ the maximum nodal degree. As stated earlier, the number of boundary nodes, $\mathcal{B}$, is upper bounded by the number of nodes in the network, N.

3.3 Host H_A gets connected to Host H_B

The new connection between hosts H_A and H_B could be detected simultaneously by both the nodes. We require that only the node with the larger[8] *identifier* execute the procedure to determine new clusters due to the new connection. This is possible because each node periodically sends a beacon which includes the node identifier (Section 4.3). Let the node with the larger identifier be A, and the node with a smaller identifier be B. Node A then initiates the **Switch ON** procedure. Node B becomes one of the neighbors taking part in the **Switch ON** procedure by sending the neighbor list and the cluster list to node A. The new cluster list and the boundary node list is determined and propagated to the rest of the network as explained earlier in Section 3.1.

[8] Any tie-breaking test will suffice.

Table 5: Switch OFF Procedure

	Procedure Switch OFF(A,B);
	Begin;
1.	B requests the list of neighbors and *Clus_List* from the cluster mates of the shrunken cluster(s);
2.	The cluster mates send the list of their neighbors and *Clus_List* to B;
3.	B determines those clusters that are included in the cluster set of its cluster mates and stores them in *Local_List*;
4.	Using the neighbor information, B expands the cluster(s). The *id*s of the cluster(s) do not change;
5.	B appends the expanded cluster(s) to *Local_List*;
6.	B executes **Find Redundant** (*Local_List*) ;
7.	B appends *Local_List* returned by **Find Redundant** to *Clus_List*;
8.	B determines new boundary nodes from the updated *Clus_List* ;
9.	B broadcasts the updated boundary node list (*Bound_List*) and cluster list (*Clus_List*) to its neighbors;
10.	Updated boundary node list and cluster list are then propagated to rest of the network by only the boundary nodes;
	End;

3.4 Host H_A disconnects from Host H_B

Here, we identify two cases as follows.

1. Node A was not a cluster-mate of node B: The topological change will result in no change in any clusters in the network.

2. Nodes A and B belong to same clusters: Here, the topological change will result in the shrinking of the involved clusters. Both A and B will detect that the link between them has broken. They will both initiate the **Switch OFF** protocol. The **Switch OFF** protocol comprises adding new clusters and removing redundant clusters. Concurrent independent executions of the **Switch OFF** protocol at two different nodes could lead to violation of the *cluster-connectivity* condition. We avoid independent executions of the **Switch OFF** protocols at two different nodes by requiring only the node with the larger *id* (say, A) to execute the **Switch OFF** protocol. The other node with smaller *id* (say, B) provides new *id*s to the shrunken clusters, updates *Clust_List*, determines a new boundary node list (*Bound_List*) and broadcasts both these lists to its cluster mates. The lists are then further propagated to the rest of the network only by the boundary nodes. Thus, node B (i.e., the node with smaller *id*) does not remove any redundant clusters. Redundant cluster determination and removal is done only by A during its execution of the **Switch OFF** protocol. The new cluster list and the boundary node list is determined and propagated to the rest of the network as explained earlier in Section 3.2.

4 Routing Protocol

We first discuss the necessary data structures to be maintained at each node for the routing protocol. We then explain the route construction and maintenance procedures in the network.

Table 6: *Clus_List* at Each Node

ClusterId	*Nodes*
A	1,2,3
B	3,4
C	4,5,6,7
D	7,8
E	8,9,10,11
F	8,12
G	12,13,14,15
H	8,16
I	16,17,18

Table 7: *Bound_List* at Each Node

ClusterIds	*Node*
A,B	3
B,C	4
C,D	7
D,E,F,H	8
F,G	12
H,I	16

4.1 Data Structures

As stated earlier, the following lists are maintained at each node :

- *Clus_List*: This list provides the mapping between the clusters and their members.

- *Bound_List*: This list maintains the 'designated' boundary nodes between overlapping clusters. As stated earlier, there may be more than one boundary node between overlapping clusters. Only one among them is chosen to be the designated boundary node (Section 3).

Using the information in *Clus_List* and *Bound_List*, each node then generates the routing table *RouteTable* used for routing packets. Each entry in the routing table contains the destination identifier, the next hop node and the number of hops it takes to reach the destination via that next hop node. The *Clus_List* and the *Bound_List* for the network in Figure 1 are shown in Tables 6 and 7. The *RouteTable* for node 6 is shown in Table 8.

Table 8: *RouteTable* at Node 6, Cluster C

DestNode	*NextHop*	*Hops*
1	4	3
2	4	3
3	4	2
4	-	1
5	-	1
6	-	0
7	-	1
8	7	2
9	7	3
10	7	3
11	7	3
12	7	3
13	7	4
14	7	4
15	7	4
16	7	3
17	7	4
18	7	4

4.2 Protocol

A routing protocol can be divided into two phases, namely, *route construction* and *route maintenance*. During the *route construction* phase, routes are constructed between all pairs of nodes. The *route maintenance* phase takes care of maintaining loop-free routes in the face of unpredictable topological changes.

4.2.1 Route Construction Phase

The protocols to maintain clusters in the face of various network events have been explained earlier. Upon receipt of new cluster information, a *boundary* node stores the new cluster list in its *Clus_List*, the new boundary list in its *Bound_List*, and then rebroadcasts the information. A boundary node has to forward the new information only once[9]. Nodes other than the boundary nodes listen to this information and just update their tables. In this manner, the information about each network event is distributed to all the nodes. Each node now has the topology information of the whole network. For a connected network, the boundary nodes also form a connected network. In this network of boundary nodes, two boundary nodes will have a link between them if they have common clusters; e.g., boundary nodes 3 and 4 have a link between them. A shortest-path algorithm (e.g., Dijkstra's algorithm [1]) is run on this connected network of boundary nodes. If a cluster has multiple boundary nodes, the nodes in that cluster will choose the boundary node with the shortest path for a destination as the next hop node for the destination. The next hop node and the number of hops for each destination is maintained in the *RouteTable*.

Each message packet contains the identifier of the destination node in its header. When a node receives a message packet, it looks up the *RouteTable* to determine the next hop node for the packet's destination. The node then forwards the message packet to the next hop node. This process of forwarding continues till the packet reaches its destination.

4.2.2 Route Maintenance Phase

This phase begins when there is a change in the network topology (host connection/disconnection, link failure/recovery). The route maintenance in our approach basically boils down to cluster maintenance. The protocols for cluster maintenance have been explained previously. After a change in topology, all the nodes have the complete topology information in the form of cluster list (*Clus_List*) and boundary node list (*Bound_List*). If all the nodes have a consistent view of the topology, routing loops are not formed. However, due to long propagation delays, network partitions, etc., some nodes may have inconsistent topology information. This might lead to formation of routing loops. However, these loops are short-term, because they disappear within bounded time (required to traverse the diameter of the network) [23].

The new cluster information will be propagated throughout the network. It should be noted that only the boundary nodes are responsible for broadcasting and re-broadcasting any new information. This helps in quick dissemination of information across the network. Thus, the reconvergence of the cluster-based protocols is very quick. Let us illustrate it with an example. Let node 2 in Figure 1 disconnect. This event will be detected by nodes 1 and 3. Since node 1 is not a boundary node, it will just update its tables to indicate the change. Node 3 being the boundary node broadcasts the new cluster information. Node 4, a boundary node, upon receipt of the new cluster information from node 3, re-broadcasts it. This broadcast will be received by nodes 3, 5, 6 and 7. Since node 3 has already broadcasted this cluster information, it neglects this information. Nodes 5 and 6 being non-boundary nodes just update their tables. However, node 7 being a boundary node, updates its tables and rebroadcasts the new cluster information. Similarly, other boundary nodes 8, 12 and 16 upon receipt of the new cluster information re-

[9] In the presence of noisy links, retransmissions will be necesary. However, as stated in Section 2, we assume a link-level protocol that guarantees that packets transmitted over a link are received correctly.

broadcast it so that every node in the network has the new cluster information. The non-boundary nodes just listen and update their tables and do not re-broadcast.

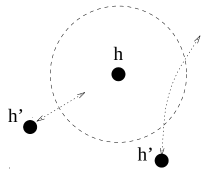

Figure 5: Movements That Cause Unnecessary Link Creations/Deletions

4.3 Implementation Details

Each host periodically broadcasts a beacon which includes its identifier. If a host h receives a beacon from another host h' which is not in its current neighbor set, it means that there is a prospective new link to be created. However, the **Switch ON** procedure is not immediately initiated. Only after a certain number of successive beacons are received from the same host is the **Switch ON** procedure initiated. This is to avoid unnecessary oscillations due to the host h' moving in and out of host h's vicinity. Figure 5 shows the scenarios where the movement of h' could cause a sequence of unnecessary link creations/deletions.

If a host h does not receive a certain number of consecutive beacons from its neighbor h', it will assume that either h' has moved out of its vicinity or that h' is disconnected. Host h will then follow the procedure for host disappearance as explained in Section 3.2.

5 Performance Evaluation

5.1 Complexity

This section compares the cluster-based approach's worst-case performance with the performance of *Distributed Bellman-Ford* (DBF) [1], *Ideal Link State* (ILS) [8], *Diffusing Update Algorithm* (DUAL) [8], NP [3] and flooding. The ILS protocol [8] requires that each topology change be transmitted to every node. The DUAL protocol [8] is a distance-vector loop-free algorithm based on internodal coordination spanning multiple hops. DUAL is known to be the lowest complexity distance-vector algorithm. NP protocol [3] is a source-initiated routing protocol that provides loop-free routing only to desired destinations in a dynamic network. Flooding does not have any control overhead due to topology updates/maintenance. Everytime a node wants to send a packet to a destination, the node broadcasts the packet to its neighbors, who in turn broadcast the packet to all their neighbors, except the neighbor from which it was received. This process goes on till the message packet reaches the intended destination.

The performance metrics are the time complexity (TC) and the communication complexity (CC). Time complexity is defined as the number of steps required for the network to reconverge after a topology change. The number of messages required to accomplish the reconvergence is called the communication complexity. The assumptions made while making the comparisons are same as in [8]. They are as follows:

- The routing algorithm behaves synchronously, so that every host in the network executes a step of the algorithm simultaneously at fixed points in time.
- At each step, the host receives and processes all the inputs originated during the preceding step and, if required, sends update messages at the same step.

We borrow the complexity computations of DBF, ILS, and DUAL from [8]. Table 9 lists the protocols with the complexities. The per packet cost of sending a packet is listed in Table 10. The complexity parameters are as follows:

- N: Number of nodes in the network.
- E: Number of links in the network.
- d: Diameter of the network. The diameter of a network is defined as the length of the longest shortest path in hops between any two nodes.
- $\mathcal{D}$: Maximum degree of a node.
- $\mathcal{B}$: Upper bound on the number of *unique* boundary nodes in the network. Overlapping clusters may have more than one boundary node between them. However, only one of them will be considered as the boundary node and will be used to pass messages between clusters. The other boundary nodes are considered as non-boundary nodes. The procedure to select a boundary node has been described in Section 3.
- x: Number of nodes affected by the topological change.
- l: Diameter of the affected network segment.

Table 9: Complexity Comparison

Protocol	TC	CC
DBF [1]	O(N)	O(N^2)
ILS [8]	O(d)	O(E)
DUAL [8]	O(x)	O($\mathcal{D}$x)
NP [3]	O(l)	O(x)
Cluster	O(d)	O($\mathcal{B}$ +$\mathcal{D}$)
Flooding	0	0

Since flooding does not have any topology update overhead, the time complexity and communication complexity of flooding is zero. However, the CC of flooding, for sending a packet is O(E) compared to O(d) for other approaches. The complexities of DUAL and NP will be high if $x \approx N$ (This is true in the situations when a node fails or switches off.), i.e., when most of the nodes in the network are affected by the topological change. In such cases, the diameter of the affected segment, $l \approx d$. The performance of the cluster-based approach depends on the number of boundary nodes and the maximum degree of a node. We resort to simulations to determine

Table 10: Per Packet Cost of Sending a Data Packet

Protocol	**TC**	**CC**
DBF [1]	O(d)	O(d)
ILS [8]	O(d)	O(d)
DUAL [8]	O(d)	O(d)
NP [3]	O(d)	O(d)
Cluster	O(d)	O(d)
Flooding	O(d)	O(E)

the variation in the number of boundary nodes and the cluster size, with the degree of the network. We will show through simulations that even for low nodal degrees, the number of boundary nodes in a network is much less than the total number of nodes in the network. We also determine the *routing overhead* of the cluster-based approach.

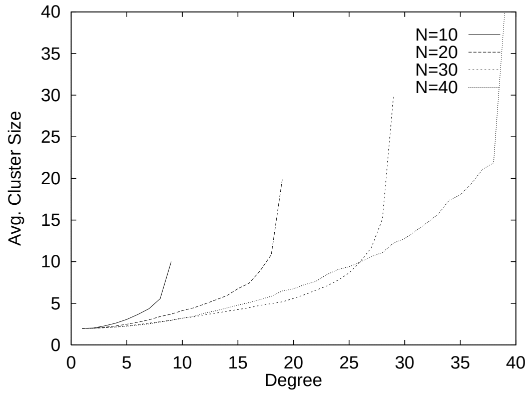

Figure 6: Variation of Average Cluster Size with Degree

5.2 Simulations

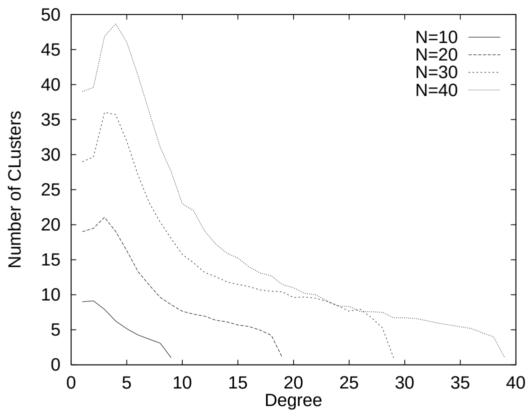

Figure 7: Variation of Number of Clusters with Degree

Simulations are performed to determine average cluster size, and number of boundary nodes for random graphs. The routing overhead of the cluster-based approach is also determined. *Routing overhead* is the ratio of the path length between a source and a destination as determined by the cluster-based approach and the actual shortest path length between them. Random graphs are generated using the *random graph generator* function presented in Appendix B. The clusters are determined using the **Switch ON** procedure described in Section 3.1. Input to the simulations are (i) N (number of nodes), and (ii) D (average degree in the network).

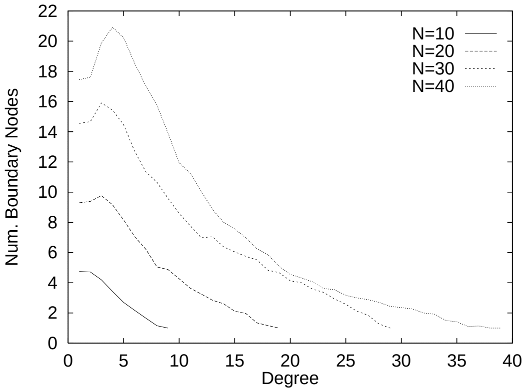

Figure 8: Variation of Number of Boundary Nodes with Degree

As shown in Figure 6, the average cluster size increases as N increases. It also increases when D increases. Figure 6 shows that there is a large region of values of N and D where the average cluster size is greater than 2. In these scenarios, clustering will benefit.

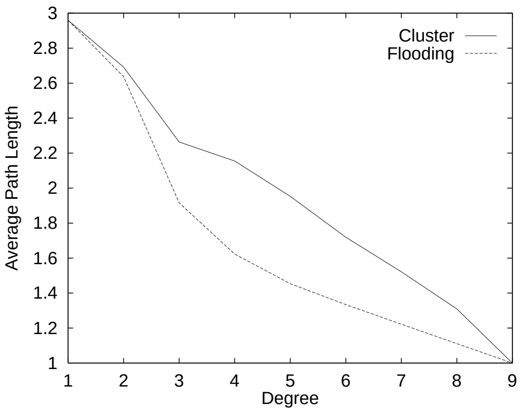

Figure 9: Variation of Average Path Length with Degree: N=10

Figure 7 and Figure 8 illustrate the variation in the number of clusters and the number of boundary nodes with node degree, respectively. Note that the number of clusters and boundary nodes in the network decreases as the node degree increases. Also note that they increase as the number of nodes in a network increases.

The maximum number of boundary nodes for a given N occurs when D is low. However, the maximum number of boundary nodes is much less than N. For example, in Figure 8, for N=10, the maximum number of boundary nodes is 5 (with $D = 2$). In other words, in such a network, if our cluster-based approach is used, the number of nodes taking part in the topology update protocol will be less than 50% of the total number of nodes in the network. Figure 9 and Figure 10 illustrate the variation of

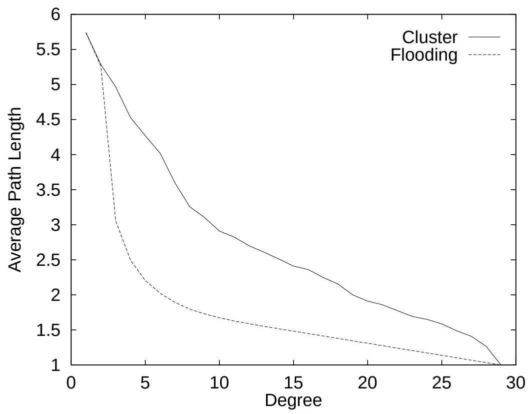

Figure 10: Variation of Average Path Length with Degree: N=30

average path length of the cluster-based approach and flooding with degree for N=10 and N=30 respectively. The average path length is computed as the average of the path lengths between each source and destination in the network. Flooding always determines the shortest path between two nodes. Note that the average path length determined by the cluster-based approach is higher than the average path length determined by flooding. The routing overhead determined as the ratio of the path lengths determined by clustering and flooding is observed to be less than 2 for both the cases considered (N=10 and N=30). Compared to savings in network load due to updates, the routing overhead of the cluster-based approach is not high.

6 Other Clustering Approaches

The problem of clustering in networks has been discussed earlier in the literature [6, 10, 13, 17, 20, 26, 28]. Our work differs from the earlier works in the following respect:

- Clustering was proposed as a hierarchical approach in earlier literature to limit the amount of routing information stored at individual hosts, and distributed and processed in the network. The entire network is thought of as a tree of hierarchies, in which each node at a higher level is made up of one or more nodes from lower levels. Each host has to take part in two updating procedures: one local within its cluster, and the other global with other distant nodes. In this paper, clustering is restricted to a single level. The main advantage behind using our cluster-based approaches is the way we maintain the clusters, which limits the number of nodes taking part in the topology-update operation, thereby, reducing the network load during topology updates.

- The cluster creation and maintenance algorithms have not been discussed in most of the literature where if it is discussed, it either is specifically for regular graph structures [17, 28], or employs a cluster controller (or leader) [26]. In this work we create and maintain a small number of clusters (cliques) in an arbitrary graph. The cluster graph is created using a sequence of **Switch ON** procedures (one procedure call for each node being added). The cluster is maintained in the face of different network events by calling the appropriate algorithms as explained in this work.

- Cluster overlapping in some approaches requires each node to be included in more than one cluster [26, 28]. However, in this work we do not require all the nodes to be included in more than one cluster.

- Unlike the previous approaches, we require our clustering algorithms to create and maintain clusters such that they satisfy the cluster-connectivity criterion (Definition 4). Since, we require the network to be cluster-connected, we can apply any routing protocol directly by just replacing the nodes by clusters. Thus, we can enjoy the advantages of a chosen routing protocol (loop-free routes, etc.), and also the cluster-based approach (low topology update overhead, etc.).

7 Conclusion

Proposed in this paper is a new methodology for routing in mobile wireless networks. Simple distributed algorithms are proposed for cluster creation and maintenance. This paper shows that routing protocols based on clusters could obtain performance improvements over previous approaches. Cluster-based protocols allow the network to enjoy the liberty of maintaining routes between all pairs of nodes at all times, without causing much network overhead. Thus, a compromise on routing optimality as suggested in [3] to avoid network congestion might not be required.

Quick reconvergence in some protocols like DSDV[25] is obtained by quick re-broadcast by each and every recipient of the broadcast, causing degradation of the availability of the wireless medium. However, in our approach, re-broadcast is done **only** by the *boundary* nodes. Nodes other than *boundary* nodes just listen and update their tables.

Similar to [3, 5] the cluster-based approach does not guarantee shortest path. This is due to the fact that the clusters are created using the first-fit approach, which does not produce the maximum clusters in the graph. However, it has been shown that the routing overhead of the cluster-based approach is not high.

Acknowledgments

We would like to thank the anonymous referee whose exceptionally detailed review has benefitted this paper a lot.

References

[1] D. Bertsekas, R. Gallager, *Data Networks,* Second Edition, Prentice Hall Inc., New Jersey, 1992.

[2] I. Castineyra, N. Chiappa, M. Steenstrup, "The Nimrod Routing Architecture," RFC 1992, August, 1996.

[3] M. Scott Corson, A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks," *ACM Journal on Wireless Networks,* Vol.1, No.1, pp. 61-81, 1995.

[4] W. Diepstraten, G. Ennis, P. Bellanger, "DFWMAC - Distributed Foundation Wireless Medium Access Control," *IEEE Document P802.11-93/190,* Nov., 1993.

[5] E. Gafni, D. P. Bertsekas, "Distributed Algorithms for Generating Loop-free Routes with Frequently Changing Topology," *IEEE Trans. on Communications,* Vol. COM-29, No. 1, pp. 11-18, January, 1981.

[6] J. J. Garcia-Luna-Aceves, "Analysis of Routing Strategies for Packet Radio Networks," *Proc. IEEE INFOCOM,* pp. 292-302, May, 1985.

[7] J. J. Garcia-Luna-Aceves, "A Unified Approach to Loop-free Routing Algorithm Using Distance Vectors or Link States," *Proc. ACM SIGCOMM Symposium on Communication, Architectures and Protocols,* pp. 212-223, September, 1989.

[8] J. J. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations," *IEEE Trans. on Networking,* Vol. 1, No. 1, pp. 130-141, February, 1993.

[9] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, New York, New York, 1979.

[10] M. Gerla and J.T.-C. Tsai, "Multicluster, Mobile, Multimedia Radio Networks," *Wireless Networks,* Vol. 1, pp. 255-256, 1995.

[11] Alan Gibbons, *Algorithmic Graph Theory,* Cambridge University Press, Cambridge, Massachusetts, 1985.

[12] C. Hedrick, Routing Information Protocol, RFC 1058, June, 1988. (Available from nic.ddn.mil)

[13] K. Ishida, "Space-Time Tradeoff in Hierarchical Routing Schemes," *Responsive Computer Systems,* edited by H. Kopetze and Y. Kakuda, Springer Verlag, NY, pp. 147-163, 1993.

[14] J. M. Jaffe, F. M. Moss, "A Responsive Routing Algorithm for Computer Networks," *IEEE Trans. on Communications,* pp. 1758-1762, July, 1982.

[15] David B. Johnson, "Routing in Ad Hoc Networks of Mobile Hosts," *Proc. of Workshop on Mobile Computing Systems and Applications,* pp. December, 1994.

[16] J. Jubin, J. D. Tornow, "The DARPA Packet Radio Network Protocols," *Proceedings of the IEEE,* Vol.75, No. 1, pp. 21-32, January, 1987.

[17] F. Kamoun, "Design Considerations for Large Computer Communication Networks," UCLA-ENG-7642, Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles, March, 1976.

[18] P. R. Karn, H. E. Price, R. J. Diersing, "Packet Radio in the Amateur Service," *IEEE Journal on Selected Areas in Communications,* Vol. 3, No. 3, pp. 431-439, May, 1985.

[19] P. Krishna, M. Chatterjee, N. H. Vaidya, D. K. Pradhan, "A Cluster-based Approach for Routing in Ad-Hoc Networks," *Proc. USENIX Symposium on Location Independent and Mobile Computing,* pp. 1-8, April, 1995.

[20] G. Lauer, "Hierarchial Routing Design for SURAN," *Proceedings of ICC,* pp. 93-101, 1986.

[21] J. M. McQuillan, D. C. Walden, "The ARPA Network Design Decisions," *Computer Networks,* Vol.1, No.5, pp. 243-289, August, 1977.

[22] J. McQuillan, "Adaptive Routing Algorithm for Distributed Computer Networks," BBN Report 2331, BBN, Cambridge, MA, 1974.

[23] J. M. McQuillan, I. Richer, E. C. Rosen, "The New Routing Algorithm for ARPANET," *IEEE Trans. on Communications,* Vol. 28, No. 5, pp. 711-719, May, 1980.

[24] Shree Murthy, J.J. Garcia-Luna-Aveces, "A Routing Protocol for Packet Radio Networks," *Proc. ACM International Conference on Mobile Computing and Networking,* pp. 86-95, November, 1995.

[25] C. Perkins, P. Bhagwat, "Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM Symposium on Communication , Architectures and Protocols,* pp. 234-244, September, 1994.

[26] C. V. Ramamoorthy, A. Bhide, J. Srivastava, "Reliable Clustering Techniques for Large, Mobile Packet Radio Networks," *Proc. IEEE INFOCOM,* pp. 218-226, May, 1987.

[27] M. Schwartz, T. E. Stern, "Routing Techniques used in Communication Networks," *IEEE Trans. on Communications,* pp. 539-552, April, 1980.

[28] Nachum Shacham, "Organization of Dynamic Radio Network by Overlapping Clusters: Architecture, Considerations, and Optimization," *Performance,* December, 1984.

Appendix A: Proof of Correctness

This appendix presents an outline of the proof of correctness for the algorithms presented in this paper.

For the sake of convenience, let us introduce two terms, namely, *root* node and *affected* node. A *root* node is a node that initiates the cluster update algorithm, whereas *affected* node is a node whose clusters may be affected by the algorithm initiated by the *root* node. For the various types of *events*, let us determine the *root* node(s) and the *affected* node(s).

- **Switch ON**: The new node is the *root* node. The neighbors of the new node are the *affected* nodes.
- **Switch OFF**: The node n that is determined using the arbitration procedure explained in Section 3.2, is the *root* node. The neighbors of the node n are the *affected* nodes.
- **Connection between nodes A and B**: The node (A or B) with the larger *id* is the *root* node. The common neighbors of A and B are the *affected* nodes.
- **Disconnection between nodes A and B**: The node (say, A) with the larger *id* is the *root* node. The neighbors of the *root* node are the *affected* nodes. The node (say, B) other than the *root* node adds new clusters and does not remove any clusters.

Each algorithm comprises the following basic steps:

- The *root* node gets from each *affected* node, the *affected* node's neighbor information and its cluster set.
- The *root* node determines the possible clusters using the **Create Clusters** function.
- From these clusters, the *root* node determines the *essential* clusters using the **Find Essential** function.
- The *root* node adds the *essential* clusters to list of clusters it has obtained from the *affected* nodes. The *root* node then determines and removes the *redundant* clusters using the **Find Redundant** function.
- The new cluster information is then broadcast by the *root* node to the *affected* nodes.

Lemma 1 *The* root *node has connectivity to each* affected *node through at least one of the clusters returned by the* **Create Clusters** *function.*

Proof: Step 2 of the **Create Clusters** (Table 2) function ensures that at least one cluster is created with the *root* node and an *affected* node as its members. Thus, the *root* node will have connectivity to each *affected* node through at least one of the clusters. □

As shown in Figure 11, the *root* node along with the *affected nodes* form a star graph with the *root* node at the center and the *affected* nodes at the fringes. Some of the *affected* nodes may be connected to each other, and they form a *connected* segment. On the other hand, some of the *connected* segments may not be connected with other *connected* segments, and they form *disconnected* segments (e.g., A, B and C in Figure 11(a)), R being the *root* node. In the worst case, a *connected* segment is a node (e.g., C in Figure 11(a)). In such a case, all *affected* nodes are disconnected from each other. When the *root* node R switches ON or moves into the vicinity of the nodes in A, B and C, the *root* node provides connectivity between the *disconnected* segments through it. It should thus be ensured that the *affected* nodes in the *disconnected* segments become cluster-connected after the execution of the **Find Essential** function at the *root* node (as in Figure 11(c)). This is because, even if there is a path between these *disconnected* segments through the *root* node, there may not be any path between them using clusters. This will violate the cluster-connectivity criteria.

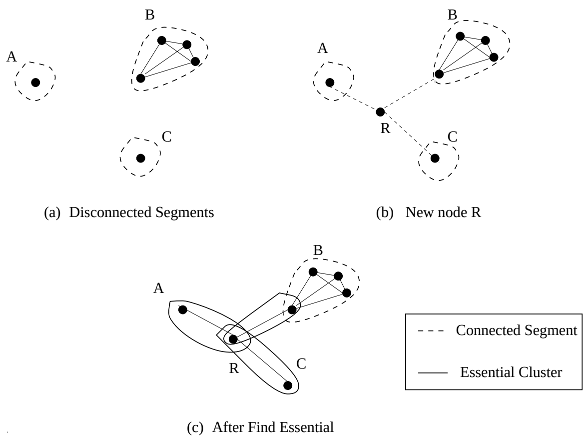

Figure 11: Clusters formed by **Find-Essential**

Lemma 2 *The* affected *nodes in the different* disconnected *segments become cluster-connected after the execution of* **Find Essential** *function at the* root *node.*

Proof: Steps 4-9 of the **Find Essential** function ensure that there is an *essential* cluster between at least one node in each *connected* segment and the *root* node. Thus, after the execution of the **Find Essential**, there is cluster-connectivity between the *affected* nodes in different *disconnected* segments. □

Lemma 3 *Nodes that were cluster-connected before the network event occurred will remain cluster-connected after the removal of redundant clusters by the* root *node.*

Proof: The *root* node executes the **Find Redundant** function to determine *redundant* clusters. This function determines *redundant* clusters based on Definition 6, which ensures that if a cluster is *redundant*, removal of the cluster does not affect the cluster-connectivity of the graph. □

Theorem 1 *Given a cluster-connected graph, the graph remains cluster-connected after any network event.*

Proof: The proof follows from Lemma 1, Lemma 2 and Lemma 3. □

Table 11: Random Graph Generator Procedure

	Random_Graph_Generator(N,D);
	Begin;
1.	Node list I = [1...N]; Edge list $T = \emptyset$;
2.	Generate a sequence S of $(N-2)$ random labels in the range [1,N];
3.	while $(\|S\| > 0)$
4.	Look for the smallest label i_1 in I that is not in S;
5.	$T = T \cup (i_1, s_1)$;
6.	Remove i_1 from I and s_1 from S ;
7.	$T = T \cup (i_1, i_2)$;
8.	$remaining = \frac{ND}{2} - (N-1)$;
9.	while $(remaining > 0)$
10.	Randomly generate 2 labels (i,j) s.t., $(i,j) \notin T$;
11.	$T = T \cup (i,j)$;
12.	$remaining = remaining - 1$;
	End;

Appendix B: Random Graph Generator

This appendix presents the algorithm used to generate random graphs. The random graph generator is based on the 'labeling' algorithm presented in [11]. The inputs to this graph generator are N and D, where N is the number of nodes in the network, and D is the average degree of the network. We use a 'labeling' algorithm to generate random spanning trees with N nodes. Then we randomly add $(\frac{ND}{2} - (N-1))$ links, so that the average degree in the final network is D. The algorithm is presented in Table 11.