Spring 2011

A Penalty-Based Approach to Handling Cluster Sizing in Mobile Ad Hoc Networks

Ryan Florin
Old Dominion University

A PENALTY-BASED APPROACH TO HANDLING CLUSTER SIZING IN MOBILE AD HOC NETWORKS

by

Ryan Florin
B.S. May 2005, Old Dominion University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2011

Approved by:

Stephan Olariu (Director)

Hussein Abdel-Wahab (Member)

Ravi Mukkamala (Member)

ABSTRACT

A PENALTY-BASED APPROACH TO HANDLING CLUSTER SIZING IN MOBILE AD HOC NETWORKS

Ryan Florin
Old Dominion University, 2011
Director: Dr. Stephan Olariu

In Mobile Ad Hoc Networks (MANETs) nodes are allowed to move freely which causes instability in the network. To handle this, the nodes are grouped into clusters which make the topology of the network appear more stable. In proposed algorithms, the size of these clusters has been either ignored or handled insufficiently. This Thesis proposes a penalty-based approach to handle cluster sizing in a more appropriate manner. A configurable penalty function is defined which assigns penalties to each of the possible cluster sizes. The penalty is then used in conjunction with a merge qualifier to determine if a merge is allowed. Merges will be allowed if the total penalty of the two clusters decreases as a result of the merge. Additionally a split merge process has been developed to allow a number of nodes to split from a cluster and merge with a new cluster. A separate split merge qualifier is used to determine if a split merge will be allowed to happen; it will as long as the total penalty of the two clusters after the split merge is less than the total penalty before the split merge. Simulations and thorough analysis of the results show that the proposed changes are on par with the base algorithm used; however, the penalty function allows for a more complex clustering sizing strategy.

ACKNOWLEDGMENTS

I thank my professors and committee members for the classes they taught me and the guidance along the way. I thank my advisor Dr. Olariu for planting the idea of doing a Master's Thesis in the first semester of my Master's classes. I thank my family and friends for the support and encouragement throughout my time as a master's student. I thank my parents for teaching me the importance of education and taking pride in my accomplishments. Additionally, I thank my father for instilling in me a love for computers and a fascination with science. Finally and most importantly of all, I thank my wife Lauren for all of her love and support, and for putting up with the long nights of homework even though I would constantly claim I'm almost done. Thank you all so very much.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Mobile Ad-Hoc Networks (MANETs) are a type of wireless network which consists of nodes which work as hosts and routers. There is no infrastructure for the network other than that which can be made up from the nodes themselves. In other types of wireless networks, such as cell phone networks and 802.11, most the routing is done on a wired infrastructure with only the first and last hops being wireless. In MANETs, there is no wired infrastructure; all hops are wireless. This is both a benefit and a problem. Since there is no infrastructure, MANETS are easily and quickly deployed [1]; however, it also carries the huge complexity of managing the nodes. The better managed the nodes are, the easier routing messages to the nodes will be.

Since there is no infrastructure, MANETS are ideal for using in areas where infrastructure does not yet exist, or where the infrastructure is broken or unusable. Consider remote battleground locations where the infrastructure would be an ideal target of the enemy. Also, consider a natural disaster in which the infrastructure is crippled. MANETS would allow forces or disaster relief workers to communicate with one another.

The nodes in the network are free to move where they please; however, the introduction of mobility leads to constant topology changes in the network [2]. Traditional wired network algorithms are based on routing tables; suppose one of these global routing tables is created for a mobile network. In terms of network bandwidth such a routing table would be very expensive to broadcast; furthermore, by the time it is received by each of the nodes, the table may no longer be valid. Mobility also leads to failure in the link between two nodes and may cause partitions in the network [2]. The transmission range of nodes is limited by the hardware of the node; thus, when two nodes are no longer within transmission range, the link fails. Since links may fail, there is no guarantee that an intermediary node will be present; this opens up the possibility for a partition in the network. Partitioning in wired networks is extremely rare; however, in MANETS, it is common and must be handled appropriately.

In MANETS, network bandwidth is a scare resource. Care must be taken to ensure the maximum use of the available bandwidth is achieved by each cluster [1]. This is even more of a concern for the scalability of MANETs [3]. Consider a densely packed network of nodes. If several nodes are within transmission range of one another, then there is a better chance that the messages of two nodes will collide. Due to the

The journal model followed by this thesis is *Journal of Network and Systems Management.*

limitations of the hardware, a node can only receive one message or send one message at a time. If any message overlaps with another message at the source or destination, then both are lost. There needs to be a way to divide network bandwidth throughout the network fairly and manage when each node is allowed to send a message to avoid collision.

CLUSTERING

A process called clustering helps to alleviate some of the problems mentioned above. Clustering is performed by grouping nodes together that share a common characteristic. Consider a cell phone network. Clustering in cell phone networks is done by placing all cell phones within the range of a tower into a cluster. All the cell phones within this cluster use that cell phone tower as its point of communication to the rest of the network. In the case of cell phones, the cluster each phone is placed into is a result of the location of the phone. One of the common characteristic shared by most MANET clustering algorithms is to group nodes based on locality; however, further grouping depends on what is trying to be accomplished. For example, a clustering algorithm which is designed to be energy aware will surely take battery power into consideration when forming clusters.

Clusters are normally formed around a cluster head which is responsible for managing the cluster. Cluster heads will be voted in, or self nominated based on the clustering protocol. The other nodes in the cluster are known as cluster members. A cluster member will either join a cluster head of its choosing, or be enlisted by the cluster head. Nodes which have neighbors in another cluster are known as gateway nodes. In order to communicate with other clusters, messages must go through a gateway node.

The main idea behind clustering is to make the topology of the network to appear more simplistic. Essentially clustering is grouping nodes together to make them more manageable. In doing this, the nodes get a level of organization which does not exist without clustering. Also, bandwidth and communication of several nodes can be managed by a single head node. Finally, a security infrastructure can be built between the members of the cluster. Since a single node is responsible for much of these benefits, its battery power will be used more quickly. The quicker drain of battery power of cluster heads is one such disadvantage which arises from each of the listed benefits.

To help lessen the effects of mobility, clustering adds a layer of organization to the nodes in the network. This allows a large network to appear smaller and a highly mobile network to appear less mobile.

This is especially important in routing; the more stable the network topology is, the more efficient routing will be [5]. Consider a routing algorithm in which nodes in a cluster relay messages to the cluster head. The cluster head will route the message to the receiving nodes cluster head, which will then deliver the message to the receiving node. The more stable the clusters are, the more likely the message will be delivered [6].

Another important benefit of clustering is that it extends the bandwidth of the network. Suppose the same dense network from above. Now suppose that each of the nodes is split up into clusters. Each of the clusters communicates in a different frequency band [7]. Now, only the nodes in each cluster are competing for bandwidth. Furthermore, the cluster head of each cluster can dictate when each node is allowed to communicate. One such way this is handled is by the cluster head using Time Distributed Multiple Access (TDMA) to ensure each of the nodes has a specific time period in which they are allowed to send a message [7-8]. This can greatly reduce the number of message transmissions. [4]

Clustering also allows for a security infrastructure. In MANETs, each node will take part in the routing of messages for other nodes. A fraudulent node in the network can send invalid messages and can even choose not to forward messages for another node [9]. Adding security and trust to a cluster can alleviate some of these issues. In one such protocol, [10], the nodes within a cluster communicate using public and private keys. Each cluster is responsible for secrecy, confidentiality, trust, and the handling of fraudulent nodes. Keys are changed once a member is removed or enlisted, so in such a protocol, the stability of the cluster members is extremely important.

In MANETs battery power is an important factor to deal with [1]. The algorithms need to be careful to evenly utilize each node; instead of relying on a few nodes to do much of the work. The more a node is used in the network, then the quicker its battery will be used. Most algorithms use cluster heads to handle much of the routing and maintenance of the cluster. The extra work for the cluster head causes its batteries to drain quicker than the other nodes.

There are two main stages in most clustering algorithms. The first stage is the cluster formation stage; this is where the actual clusters are formed. The algorithms which handle these vary greatly, but most are based on one or some of the following, node id, number of hops, weights, mobility, and battery power. An ideal algorithm will be one in which the cluster formation is indistinguishable from the next step, cluster maintenance [13].

The second stage is the cluster maintenance stage. This stage is needed because mobility exists within the network. When nodes move, the clusters formed in the first stage start to break down. Maintenance is needed to keep the structure of the clusters intact. Cluster maintenance includes forming new clusters, and breaking clusters apart. Different protocols handle these steps differently. Some of the ways include the following: adding nodes to clusters, removing nodes from clusters, splitting clusters, merging clustering together, and cluster head election.

Clustering algorithms need to handle the changing topology of the network caused by mobile nodes. This includes allowing the nodes to be mobile during the cluster formation step [11]. Several early proposed algorithms do not include this as a requirement, most recent proposed algorithms do. Also, the algorithms should form clusters which meet an ideal size. Clusters with many nodes may experience a lot of traffic, while clusters with too few nodes may cause additional routing overhead [11]. Additionally, clusters with multiple hops, rather than single hops, between the cluster head and members normally have more nodes. Cluster heads which are responsible for large clusters may spend too much time handling members [12]. The amount of time handling members should be reduced per the amount of time sending effective messages.

There are claims that these large clusters may not be as efficient, and smaller clusters are not as efficient [3-4, 11]. The ideal size of the cluster is application specific, so it should be considered a parameter of the algorithm. Most algorithms that are aware of cluster size only consider a maximum cluster size instead of an ideal cluster size, the difference being that the maximum cluster size is a limit and the ideal cluster size is a recommended cluster size. Clusters sizes should be distributed near this ideal cluster size.

This Thesis focuses on expanding a clustering algorithm proposed by Wang in [13]. The algorithm deals in most part with the process of merging and splitting clusters. This process is explained in more detail in Section 3. In particular I am expanding the part of the algorithm that handles when and which clusters will merge. The merging of the clusters will be determined by a penalty function and a merge qualifier, both of which are properties of the size of the cluster. Additionally, I add a process called a split merge which allows a cluster to take some nodes from another cluster. Each of these is explained in Section 3.

Section 2 provides a survey of recent clustering algorithms proposed for MANETS. Section 3 includes an explanation of Wang's Clustering algorithm, which I am using as a base for my proposed changes. These changes include the Penalty Function and Split Merge process, each of which is explained in detail. The details of my simulation, metrics used, and the results of the simulation is included in Section 4. Finally, Section 5 includes a summary of my contributions included in this Thesis and some recommendations for future expansion.

2. STATE OF ART

I split the clustering algorithms into a few different categories. The first category is the early Simplistic Algorithms. These algorithms determine clusters based on very simple characteristics. Next I explain the Weight Based Algorithms. These are algorithms which use a combination of simple and complex metrics to compute a weight which is used to determine the clusters. The algorithms explained in many of the remaining categories may overlap into different categories. I attempted to place them appropriately, without introducing too many sections. The third category is the Mobility Based Algorithms, which includes mobility in the determination of clusters. The fifth category is the Energy Aware Algorithms. These algorithms use battery power to determine clusters. The next category is a Flooding Based Algorithm. This algorithm did not seem to fit in with any of the others in that it uses rounds of flooding messages to determine the clusters. The final category includes the algorithms which take in account Cluster Sizing. The algorithms explained in this category each attempt to limit the size of the clusters either directly or indirectly.

SIMPLISTIC ALGORITHMS

The simplistic algorithms are named specifically because the characteristic for determining the clusters is very simple to calculate. Examples of the simple characteristic include lowest node id [4, 14] and highest connected node [15]. These algorithms were made in early MANET research when the computing power of the nodes was very low; wasting time and energy on a complex metric for the use of clustering was not affordable.

In the Lowest-Id [14] and Highest Connectivity [15] clustering algorithms, the cluster heads are chosen in a similar way. The nodes in the network first learn of their neighbors. In the Lowest-Id algorithm, the node with the lowest node id of each of its neighbors will become the cluster head. In the Highest Connectivity Cluster algorithm, the node which can cover the most nodes of each of its neighbors will be selected as the cluster head. After cluster head selection, both algorithms are similar; the nodes will join the cluster if it is a neighbor of a cluster head.

Another of the early clustering algorithms is Least Cluster Change (LCC) [16]. Cluster formation is performed using Lowest-Id Clustering Algorithm or Highest Connectivity Cluster Algorithm. The rest of the steps make up the cluster maintenance phase. When a node moves out of the range of its cluster head, and into the range of a new cluster, the node will update its cluster to the new cluster. When the node moves of the range of its own cluster where there is no cluster, it will become its own cluster head and form a cluster. When a cluster head moves into the range of another cluster head, they will decide which gets to remain a cluster head using lowest-id or highest connectivity.

Lin and Gerla's clustering algorithm [4] is also based on lowest node Id. If a node has the lowest node id of all its one hop neighbors it broadcasts its cluster to each of its neighbors. Upon hearing a cluster broadcast, a node checks to see if the broadcasted cluster id is less than its own cluster. If so, then it will update to the smaller one and then broadcasts its cluster id to each of its neighbors. This ensures all the nodes know each neighbor node and its cluster. The algorithm forms clusters of diameter 2; this means there is at most two hops between any of the nodes in the cluster. If the cluster loses its diameter 2 property, the most highly connected node remains as the cluster head. Nodes who are not a neighbor of this node will be removed from the cluster.

WEIGHT BASED

The Weight-based algorithms are for most part an extension of the simplistic algorithms, except more care is taken to determine a particular weight by which the nodes will choose their cluster heads. After analysis of the above algorithms, the developers determined that by using specific calculated weights, the clusters formed can be more stable.

The Weighted Clustering Algorithm (WCA) [18] bases its clustering on a set of weighted parameters. The following four parameters are used: the difference between the nodes degree and the ideal size of the cluster, the sum of distances between the node and the rest of its neighbors, a calculation of the mobility between the node in reference to each of its neighbors, and the amount of time the node has been a cluster head. Respectively the weights are calculations of the number of nodes in the cluster, the width of the network, the mobility of the cluster, and the remaining battery power of the cluster head. Each of these parameters is assigned a weight; of which the sum of each equals 1.

Prior to the start of the clustering algorithm, each node computes its weight. It then needs to send its weight it to each of its neighbors. Neighboring nodes determine which of its nodes has the smallest weight of all its neighbors. This node becomes the cluster head, and each of its one hop neighbors joins the cluster. The algorithm continues until each of the nodes join a cluster.

New cluster heads are elected when a node is no longer in the range of a cluster head. Each of the nodes affected will then need to recalculate its weight and send it to each of its neighbors to determine the new cluster head. In the presence of high mobility, there is a higher frequency of WCA re-elections which causes inefficiency [19-20]. The proceeding two algorithms were introduced to minimize the effects of this issue.

Improved Weighted Clustering Algorithm (iWCA) is proposed in [19]. In this algorithm, nodes with low battery power cannot be elected as cluster heads. Additionally, a cluster head will resign if its battery power reaches a certain battery power threshold. When a cluster head resigns, the weights are re-calculated and sent to each of its neighbors. The one with the smallest weight is elected as the new cluster head. In this particular algorithm, the re-calculation is localized to the members of the cluster which helps to alleviate the re-calculation issues from above. In iWCA, when a node is no longer able to communicate with any cluster heads, it becomes a new cluster head.

In Weighted Clustering Algorithm with Location cluster-heads election (WCA-L) [20], nodes which are not longer able to communicate with any cluster heads will also become its own cluster head. Also, in this algorithm is a provision to keep two cluster heads from being one-hop neighbors. When two cluster heads become neighbors, the member nodes of each cluster will attempt to join any of the other clusters in the vicinity. The leftover nodes will run the WCA Cluster Selection algorithm to determine the new cluster heads. The re-calculation of the weights is localized to only those which did not join a new cluster.

In WCA, the cluster head must send out periodic hello messages which the member nodes use to infer a distance. When a member node reaches a certain threshold distance from its cluster head, it must communicate its intent to leave the cluster. In MPWCA-L, proposed in [21], the overhead of all the hello messages is reduced by the use of mobility prediction. This allows the hello messages to be sent less frequently. Additionally in this version of the WCA algorithm, a node will become its own cluster head if it is not in the range of another cluster. As with WCA-L, when two cluster heads are within one hop of each other, one of the cluster heads will resign.

MOBILITY BASED

The mobility based algorithms are those which are based on the movement of the nodes. Simplistically speaking, the position of each node is determined by the power of its signal. The movement of the node is determined by how much the power of the signal changes since its last message. Using such an approach, the node can only determine if other nodes are moving towards or away from it.

Another variation on the WCA algorithm is Entropy-based Weighted Clustering Algorithm (EWCA) [22]. This algorithm replaces the average speed of the node from the original WCA algorithm with a parameter which is a measure of the entropy of the cluster. The Entropy parameter is a metric based on the distance between each node and its neighbors and the change of this distance over time. This ensures the historical movement of the nodes is taken into consideration when selecting the cluster heads. Each node calculates its entropy locally.

MOBIC [23] is a clustering algorithm which considers the relative mobility of its neighbors when forming clusters. The algorithm used is similar to the Lowest ID algorithm from [14], except a metric based on mobility is used. For each neighbor of a node, two measurements of the neighbors power received are considered. A ratio of the first power over the second power is calculated. The variance of ratios of each node are computed and distributed to each of its neighbors as the nodes mobility metric. This variance is indicative of the mobility of the node as compared to each of its neighbors. The nodes with the lowest mobility metric, thus being the least mobile nodes of the neighbors, are chosen as the cluster heads. The basic idea is that the least mobile nodes will form an infrastructure for the more mobile nodes.

Another mobility based clustering algorithm is the Mobility-Aware Pro-Active Low Energy (MAPLE) clustering algorithm [8]. In MAPLE, the mobility is again determined by the signal strength of previous messages; however, it also uses these calculations to predict the next position of the node. Using this prediction, a node can proactively handle link failures. This saves time and energy by handling link failures before they happen. In this algorithm, cluster heads are not all chosen in a cluster formation phase. When a node is turned on it will sense the network for cluster heads. If one is within transmission range, it will attempt to join. If multiple are within range, it will choose the closest to join and will use the next closest as a backup cluster head. If no cluster heads are within range it will become a new cluster head and will make itself available for other nodes.

ENERGY AWARE

The energy aware clustering algorithms keep track of the battery power its current or potential cluster heads has left. Since cluster heads use more battery power, it makes sense to not allow a cluster head to remain if its battery power gets too low.

In [24], a Mobility-Sensitive Clustering Algorithm (MSCA) is proposed. This algorithm attempts to use the calculation and predication of the nodes mobility to conserve battery power. As in many other algorithms, information of the nodes is broadcast in hello beaconing messages; however, the frequency of these messages for this algorithm is based on the mobility of the nodes. When the mobility of the node is low, the hello beaconing message will not be sent as frequently; this is okay since slower moving nodes will take more time to move out of the range of the cluster head. Since fewer messages are being sent, energy is being saved. The algorithm also allows the sizes of the clusters to increase or decrease depending on this cluster mobility. Clusters with more highly mobile nodes will become smaller. For highly mobile nodes, smaller clusters are more stable. Additionally, this algorithm includes a way to rotate the cluster head of the node. By rotating the cluster head it ensures that one nodes battery power will not be drained by being nominated as the cluster head.

The Distributed Scenario-based Clustering Algorithm (DSCAM), proposed in [2], has two parameters used to define a cluster. The parameter r is the maximum distance between a node and its cluster head. The parameter k represents the minimum number of cluster heads in the within range r of the node. The parameter r is defined in number of hops and the parameter k is defined in number of nodes. Cluster head redundancy is introduced when k is larger than 1. To determine the cluster heads, a dominating set is computed for the network. Next, each node in the dominating set computes a quality metric based on its stability, degree, residual battery power, and the transmission rate. The quality metric is broadcast to each of its neighbors and the nodes will then join the clusters with the best quality metrics. A cluster head will accept the node unless a battery threshold has been met. If a node is not accepted by the cluster head, the node will attempt to join the next best quality node. During cluster maintenance, new nodes or nodes changing clusters will listen for Cluster Head Advertisement Messages. They must hear at least k of them; otherwise, they become their own cluster head. This ensures the k and r parameters are met.

Mobılıty and Energy Aware Clusterıng Algorıthm (MEACA) [25] bases ıts cluster head electıon on two attrıbutes mobılıty and energy Its goal ıs to maxımıze the stabılıty of the cluster It does thıs by choosıng cluster heads whıch have hıgh energy and low mobılıty A node wıll change ıts cluster when ıt loses contact wıth the cluster head, and the cluster head wıll change ıts role when ıt loses contact wıth ıts fınal member

Power Aware Vırtual Base Statıon (PA-VBS) [26] attempts to maxımıze the battery power of ıts nodes Sınce cluster heads do most of the work of the cluster, theır battery power wıll draın quıcker than the rest of the nodes In PA-VBS there are four thresholds whıch are based on battery power In order from the hıghest to the lowest power, the thresholds are as follows max power of the node, threshold 1, threshold 2, and zero power Between max power and threshold 1, a node wıll attempt to joın a cluster ıf there exısts another node wıth a power greater than ıts own Between threshold 1 and threshold 2, a cluster head wıll dıscard any merge requests ıt may receıve Between threshold 2 and zero power, the cluster head wıll resıgn from beıng cluster head The thresholds allow the cluster heads to be utılızed based on theır battery power

Trust-related and Energy-concerned Dıstrıbuted MANET Clusterıng (TEDMC) [27] ıs a clusterıng algorıthm based on trust and energy Wıthın thıs algorıthm, the trust for a partıcular node ıs a probabılıty based value that the node wıll act correctly as per the protocol A cluster starts out wıth a publıc and prıvate key used for sıgnıng messages Hello messages sent ınclude the publıc key, resıdual battery, and ıts one hop neıghbors' trust values The trust values are based on the neıghbors' reputatıon, whıch ıs decreased when nodes ınvalıdly sıgn messages, or attempt to cheat Cheatıng happens when a malıcıous node attempts to assıgn ıtself as cluster head, however, they wıll be dıscovered sınce cluster heads must be voted ın by other nodes Cluster heads are voted ın based on the energy levels and theır reputatıon values

FLOODING BASED

The Mın-Max-D cluster algorıthm [17] forms clusters through round of broadcast messages The d ın the name of the clusterıng algorıthm ıs a parameter alludes to the hop dıstance of the cluster For a cluster hop dıstance of d, there wıll be 2d rounds of broadcast messages The fırst round of broadcasts ıs the floodmax stage In thıs stage, each of the nodes broadcasts the largest node ın ıts WINNER array The WINNER array ıs ınıtıalızed wıth ıts own node ıd After each round the largest node ıd receıved ıs populated ınto the

WINNER array. There is one slot in the WINNER array for each round performed. This stage is performed d times. The next round of the broadcasts is the floodmin stage. The same process is performed, except now the smallest value from the WINNER array is broadcasted. Again, it is performed d times.

There are three rules to determine the cluster and cluster heads. First, if a node received its node id from another node, the node becomes a cluster head and skips the next two rules. Second, the node finds each of the node Ids which were winners in both the floodmax and the floodmin stages. The smallest node id is set as the cluster. Finally, if the node still has no cluster head, the largest node id from the floodmax round is chosen as the cluster.

CLUSTER SIZE

The following algorithms each include the size of the cluster as part of its scheme for clustering. In (α, t), the clustering sizing is an indirect property of other parameters. In the others [12, 18, 28-29], an ideal or maximum size of the cluster is set and adhered to.

In the (α, t) clustering algorithm [5], there is a probability α that a path between all nodes in the cluster will exist for a time t. When a node in the cluster determines that it cannot comply with the (α, t) principle, it will remove itself from the cluster and attempt to re-join another cluster. In the (α, t) clustering algorithm [5], the size of the cluster is defined by the α and t parameters. For a set α, if t is increased, then the size of the cluster will be smaller to accommodate the longer time span. If t is decreased, the size of the cluster will increase as a result of the smaller time span.

In the Vote-Based Clustering Algorithm [28], the weights for the cluster heads are determined by the remaining battery power of the node and the number of neighbor nodes. There are two weights used to determine the vote. The first is based on the number of neighbors of the node divided by the maximum number of neighbors allowed. The second is the nodes battery power divided by the max of its neighbors battery powers. The two weights are added together to form the vote. The vote metric is broadcast to each of its neighbors. The node with the largest vote metric of each of its neighbors is the cluster head.

In the Adaptive Multi-hop Clustering Scheme [29], there are two thresholds of cluster sizes. One is an upper bound U, and the other is a lower bound L. When clusters merge and form a cluster size larger than U, then the cluster will split. When the cluster size is less than L, then the cluster will merge with other clusters Clusters within the thresholds will not merge or split

Bandwidth-adaptive Clustering (BAC) [12] allows for H hops between the cluster head and each cluster member Since multi-hop clustering algorithms normally have more nodes than single hop clusters, an upper bound U is set on the number of nodes in the cluster In the maintenance phase of the algorithm, clusters can either merge with other clusters, or single nodes can join a cluster Neither will be allowed if the resulting cluster is larger than the upper bound U is met The values for H and U are predefined values Additionally, each node is assigned a Weight based on its capability Nodes with a larger weights are chosen as cluster heads in this algorithm, clusters are defined by H and U

Of the clustering algorithms discussed, the only which mentions an ideal size of the cluster is the WCA family of algorithms One of the metrics used to determine the cluster head is the following the difference between the nodes degree and the ideal size of the cluster This means the cluster head will be chosen partially on how close it is to its ideal cluster size The difference between an ideal size of the cluster and a maximum size of a cluster is how the thresholds are treated For a maximum size of a cluster, no cluster over this threshold will be allowed For an ideal cluster size, a cluster over this threshold may be created, but a size closer to the threshold is preferred over one further The ideal cluster size is explained more throughout this paper

3. TECHNICAL EXPLANATION

WANG'S CLUSTERING ALGORITHM

Wang's tree based clustering algorithm forms clusters of nodes which have at maximum a diameter of 2. This means the hop distance between any nodes in the cluster is at most 2. Identity information for each node is disseminated to each neighbor through the use of Hello Beaconing Messages. Clusters will split themselves when the diameter-2 principle fails. When clusters get close to one another, they will merge together, as long as the diameter-2 principle holds.

The two main processes for a clustering algorithm, Cluster Formation and Cluster Maintenance, are handled the same way in Wang's algorithm. A single node is treated as a cluster. This single node cluster will merge with another cluster in the same way a larger cluster will. From the start of the algorithm, cluster maintenance is being performed. The cluster maintenance actions performed are splits and merges. Splitting will take a cluster that is no longer diameter-2 and form clusters that are diameter-2. Merges take two clusters and put them together to form one cluster, as long as the diameter-2 principle holds.

Hello Beaconing Messages

In Wang's algorithm, Hello Beaconing Messages are used to notify neighbor nodes of identity information. The information included in each hello message is as follows: node id, cluster id, cluster size, and its list of neighbors. The list of neighbors includes the node id, cluster id and signal strength of each one hop neighbor. By receiving Hello Beaconing Messages from each of its one hop neighbors, a node will know each of its two hop neighbors. Using this information each node keeps track of each neighbor in its cluster within a Breadth First Search (BFS) Tree. Since clusters are diameter-2, any node in the cluster will know the identity information for all the other nodes in the cluster.

Splitting

The Splitting process starts when a link is broken within the cluster which causes it to no longer be diameter-2. This can occur in two different instances: a single-link failure, or a single node failure. The first happens when two nodes in the cluster move out of transmission range of one another. The second happens when a node is no longer alive. Single-link failures and single-node failures do not always result in a diameter-2 violation.

When the signal strength of a link reaches a particular threshold, the cluster computes its diameter-2 BFS tree. The cluster will compare the cluster size to the size of the BFS tree. In the case of a mismatch, it sends a VIOLATION message to each of its one hop neighbors. Included in the VIOLATION message is the node id of the offending link. Each node receiving the VIOLATION message will compute its diameter-2 BFS tree. If there is a mismatch, the VIOLATION will be forwarded on. The maintenance leader is determined when the size of the BFS tree matches the cluster size; this indicates the node can reach all the other nodes in the cluster in two hops. Wang proves that the maintenance leader will be found once the VIOLATION is forwarded at most once.

Each maintenance leader will compute its diameter-2 BFS tree. It then follows a series of steps to split the nodes of the cluster into separate diameter-2 clusters. In the first step, the maintenance leader finds a minimum set of level-1 nodes which have links to all the level-2 nodes. The nodes in this minimum set are referred to as the critical set. Other level-1 nodes are referred to as the redundant set. Each node in the critical set will become a new cluster. Each level-2 node is assigned to one of these new clusters one at a time based on the best signal strength. The node will scan the signal strength of the links to each of the critical set nodes, and whichever has the best signal strength will be the cluster the nodes join. In the second step, the nodes in the redundant set are assigned to the new clusters in the same manner, as per the best signal strength. In the third step, if there are any nodes left over in the redundant set, the maintenance leader will create a new cluster containing itself and these left over nodes; otherwise, it will assign itself to one of the critical set clusters based on the best signal strength.

Now the maintenance leader will package the list of all the new clusters and send it to each of its 1-hop neighbors in a MAINTENANCE RESULT message. Note that some or all the one hop neighbors are the critical set nodes which will start new clusters. Upon receiving a MAINTENANCE RESULT, a node will determine if it is one of the critical set nodes, and thus send a MEMBER ENLIST to notify each node of its new cluster. Once the new clusters are set, the single link failure splitting process is complete.

There are two different types of single node failures in Wang's algorithm. The first occurs when a node in the cluster will lose connectivity to another node. The second occurs when this is not the case.
In the first case, the failed node's BFS tree is only a depth-1 tree. This means it is a central node and can reach all the other nodes in one hop. If a tree determines its BFS tree is a depth-1 tree, it will run a

Minimum Dominating Set (MDS) algorithm to determine the candidate maintenance leaders in case the node fails. If the node does fail, the each candidate maintenance leaders will send out a MEMBER ENLIST message with the results of the MDS algorithm

In the second case, Wang proves that there will still be a node with a diameter-2 BFS Tree. This node will become the maintenance leader. A node can determine a single node failure when the broken link suddenly fails. When this happens, the node is removed from its neighbor list and it checks the cluster size against the size of the diameter-2 BFS tree. In case of a mismatch, a VIOLATION message is sent and the single node failure algorithm continues as in the single link failure algorithm.

Merging

In order to make manage the size of the clusters, Wang's algorithm includes a desirable cluster size k. Also included are two tolerances: α and β. When the size of a cluster is less than $k - \alpha$, the cluster is considered to be a deficient cluster. It is deficient in that it needs more nodes to be within the boundary set by $k - \alpha$ and $k + \beta$. When merging clusters, Wang's algorithm focuses on creating a cluster within these bounds. If the cluster's size is within these bounds, it will no longer merge.

More generically, for a Cluster A and Cluster B, the clusters will merge as long as the following conditions are true:

1. Cluster A is smaller than Cluster B
2. The resulting size of Cluster A and Cluster B is less than $k + \beta$
3. Both merging clusters are smaller than $k - \alpha$.
4. If multiple clusters are merging, the resulting cluster which is closest to k will be the winner.

Suppose two deficient clusters, A and B, move near one another. Cluster A is not currently merging with another cluster, so each node in Cluster A that is one hop from Cluster B sends a MERGE REQUEST to Cluster B to indicate it would like to merge together. If Cluster B is not currently merging it replies with a MERGE ACK.

Now each node in Cluster A checks if the combined Cluster A and Cluster B results in a cluster which is diameter-2. It does this by checking the size of the BFS tree with sum of the cluster sizes $|A|+|B|$. If the cluster sizes match up, Cluster A sends a MERGE CONFIRMATION to Cluster B; otherwise, Cluster A sends a VIOLATION to Cluster B. If Cluster B receives a VIOLATION from Cluster A, it will then forward the VIOLATION to each node in the cluster to inform them that the merge has been aborted. If a MERGE CONFIRMATION is received, the merge operation is complete.

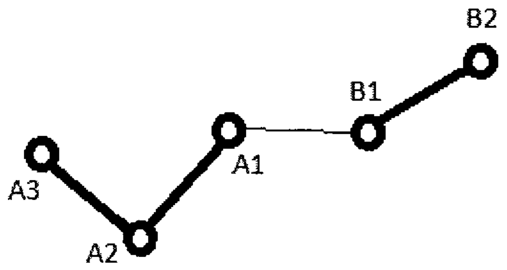
My Implementation

This section explains my implementation of Wang's algorithms. In most cases I tried to remain true to Wang's explanation, but in some cases I was unable to do so. The parts of my implementation that deviate from Wang's are explained below.

Wang's description of the single link failure is very good; with one exception. He does not explain how he handled multiple maintenance leaders. For ease of computation, I use the MEMBER ENLIST results from the lowest maintenance leader node id. I did not implement Wang's single node failure since my simulations do not include nodes turning off.

When implementing this part of Wang's algorithm, certain details of the merge operation are not clear. How does each node in Cluster A communicate internally the intent to merge with another cluster? What happens if two nodes in the same cluster attempt to merge with different clusters? What happens if Cluster A and Cluster B both attempt to merge with one another at the same time?

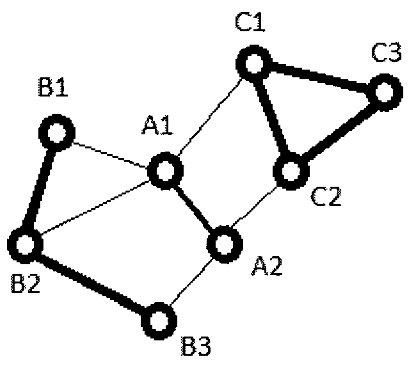
In my implementation, each node is free to make a decision on behalf of the cluster. Unless a merge or split is known to be happening, the node will perform the action. If this results in different actions being performed within the same cluster, one of the actions will be allowed to continue, while the other action will be aborted.



1. Cluster Merge Example 1

When a node in Cluster A determines to merge with Cluster B, it sends the nodes in cluster B a MERGE REQUEST. Each node in Cluster A also receives this message. When a node in Cluster A hears a MERGE REQUEST from another node in Cluster A, it will then know that Cluster A is merging with Cluster B. Suppose that there are 3 nodes in Cluster A, A1, A2 and A3, see Figure 1. The node A1 is the one who sent the MERGE REQUEST to Cluster A. This node is linked to A2, which is linked to A3. Assume there is no link from A1 to A3. Given my merge implementation, A2 overhears the MERGE REQUEST to Cluster B, but A3 will not. Additionally, unless Cluster B is also linked to other nodes in Cluster A, it will not pass the diameter-2 principle which is used to determine a cluster. For these reasons, I place a restriction on which nodes in the cluster can initiate merges. Only nodes in the cluster which are one hop from all other nodes in the cluster may initiate merges. This ensures if a MERGE REQUEST is sent from a node in Cluster A, all the other nodes in Cluster A are aware of the merge.

Now, suppose two nodes in Cluster A send MERGE REQUEST messages to differing clusters; A1 sends one to Cluster B and A2 sends one to Cluster C; see Figure 2. Note, in order for this to happen, Cluster A and Cluster B must have the same size, since a larger cluster will not attempt to merge with a smaller cluster. Upon overhearing one another's MERGE REQUEST, a decision needs to be made as to which merge should continue. In my implementation, the Cluster A will continue to merge with the cluster with the lowest node id; however, the choice can alternatively be made using the stability measurement of the cluster.



2. Cluster Merge Example 2

The final question is the case in which a node in Cluster A attempts to merge with Cluster B, and a node in Cluster B attempts to merge with Cluster B. In this case, the MERGE REQUEST coming from the larger cluster id is ignored. The one from the smaller cluster id is continued without delay. In this case, it does not matter which request is ignored, both clusters will merge together anyways.

PROPOSED ADDITIONS TO WANG'S ALGORITHM

Wang's algorithm deals for the most part in how clusters split and merge. In the changes I propose, I first show a new way in which nodes can merge together, I call this a split merge. In the split merge, a smaller cluster will take some nodes from a larger cluster in order to form evenly sized clusters. My second contribution is a change in when and which clusters merge. I base which clusters merge on a penalty function. The basic idea of the penalty function is that clusters will merge together if the total penalty between the merging clusters drops.

Split Merge

In the split merge process, a smaller node will take some of the nodes from a larger node in order to make similar sized clusters. This process is performed in order to make the size of the clusters more even.

Assume Cluster A moves into the vicinity of Cluster B. Through hello beaconing messages, Cluster A learns it is within the range of two of the Cluster B's nodes. The process of determining when a split merge is appropriate is discussed later in the penalty function section. For now we assume that Cluster A would like to take these two nodes and add them to its own cluster in order to get closer to the ideal cluster size. Similarly, Cluster B would like to get rid of these two nodes in order to get to closer to the ideal node size. Cluster A sends a MERGE REQUEST to Cluster B, naming the nodes S it would like to merge into its cluster. If Cluster B is not already merging with another cluster, it will reply with a MERGE ACK.

Once the merge has been acknowledged, the nodes in S will calculate the BFS tree of the new Cluster A and match it against the current size of Cluster A plus the size of S. If this size matches, the new Cluster A will be a diameter 2 cluster.

The existing nodes in Cluster B will also need to calculate their BFS tree to ensure their cluster remains diameter-2. Suppose one of the nodes in S is the only existing link for a branch of the cluster to the rest of the cluster. Removing this node will split up the cluster. To protect against this, the remaining nodes in

Cluster B calculate their BFS tree, excluding the nodes in S, and they match it up against the expected cluster size. If the two do not match, these nodes will send a VIOLATION message to the nodes in S. These nodes will then send a MERGE ABORT message to Cluster A. Neither of the clusters will continue on with the split merge.

If there is no VIOLATION sent, then the nodes in S will send a MERGE CONFIRMATION to Cluster A to signify that the split merge will go through. The nodes in S then set their cluster to A, and the nodes in Cluster A and B update their neighbor lists.

Penalty Function

Wang uses a desired Node size, k, along with two tolerances α and β to manage the size each cluster. In this algorithm clusters are allowed to have nodes around k within the tolerances α and β. This is not ideal since once a cluster reaches a size between $k- \alpha$ and $k + \beta$, the number of nodes in the cluster will not change. We would ideally like the size of the cluster to eventually reach k.

I propose to replace Wang's sizing model with a penalty function which is more robust. The idea of the penalty function is to penalize the cluster for having a cluster size n which does not match the ideal cluster size k. The basic idea is that the further away n is from k, the larger the penalty is. Based on the penalty function P(x), the qualifier to determine if a merge can happen is $P(A) + P(B) > P(A + B)$. In other words, the penalty of the size of Cluster A plus the penalty of the size of Cluster B needs to be greater than the penalty or the sum of Cluster A and Cluster B in order to merge.

Before merging, a cluster will determine which merge will result in the best overall penalty of the system. If the system overall penalty increases due to the merge, the merge will not take place. Suppose Cluster A can merge with Cluster B or Cluster C. Cluster A will first determine if merging with Cluster B will benefit the system. It will calculate the merge qualifier and determine if the total penalty before the merge is greater than the penalty after the merge; if so, the merge will benefit the system. Next it will calculate the penalties to determine if merging with Cluster C will be beneficial. If both are beneficial, the one with a lowest post-merge penalty will be chosen for the merge. If neither benefits the system, then neither will merge.

There is an additional qualifier to determine if a split merge should take place. Suppose the same example above with Clusters A, B, and C. Cluster A will first determine if split merging with Cluster B is beneficial to the system. It will calculate $P(A) + P(B) > P(A + a) + P(B - a)$. If the total penalty before the split merge is greater than the total penalty after the split merge, then the split merge will be beneficial. Next Cluster A will calculate as if split merging with Cluster C. The one with the lowest post-merge penalty will be chosen as the winner. In case a merge and a split merge are available for Cluster A, the one with the lowest post-operation penalty will be chosen.

Properties of the Penalty Function

In order to create a penalty function, we first need to know how the two qualifiers will react in some general case scenarios. We will use these general cases as building blocks to create more complex penalty functions. The only building blocks needed are lines and exponential curves. To make things simpler, I will only consider an exponential curve of degree 2.

The penalty function based on a line is $P(x) = mx + b$. If we plug this into the cluster merge qualifier, we get $2b > b$. This is always true as long as b is greater than 0. If it is, the cluster merge will always happen within the bounds of the line. If we plug this into the cluster split merge qualifier, we get $2b > 2b$. This is never true, thus the split merge will not happen within the bounds of the line.

The penalty function based on an exponential curve of degree 2 is $P(x) = (x-b)^2$. If we plug this into the cluster merge qualifier, we get $b^2 > 2AB$, or $AB < \frac{1}{2}b^2$. Within the bounds of the curve, the merge will occur as long as the product of A and B is less than half of the square of b. If we plug this into the cluster split merge qualifier, we get $0 > a + A - B$, where a is positive. This means that as long as B is greater than the sum of A and a, then the split merge will occur.

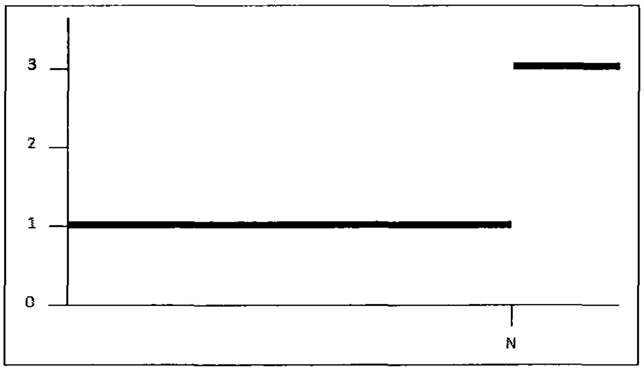
The above discussion of the qualifiers is only true when within the bounds of the line or curve. We will now consider a piecewise function as the penalty function. When all the cluster sizes being considered in the qualifier are within the same boundary in the piecewise function, the above will hold. If parts of the qualifiers span different boundaries of the piecewise function, their properties will need to be determined separately. Due to the complexity of these situations, I cannot supply a generic answer. I will instead discuss the boundary conditions which occur for each example below.

The final building block needed for the penalty functions is a way to signify a maximum cluster size. I do this by adding a horizontal line with height of infinity starting at N+1, where N is the maximum cluster size. For a merge between Cluster A and Cluster B, where the $|A+B| > N$, the merge cannot take place unless P(A) + P(B) > infinity. To ensure this does not happen, we must limit the penalties before N to real numbers. Cluster split merges are not affected by the max size N, since the resulting clusters A' and B' are both smaller than the larger originating Cluster B. We have already seen from the cluster merge that a cluster cannot form larger than N.

Example Penalty Functions

Now that we have some building blocks for the penalty functions, we can look at some examples. In the following section I simulate an algorithm with a max cluster size N by using the penalty functions. Next I simulate Wang's Algorithm. I then show my proposed penalty function followed by a theoretical one to show the robustness of the model.

I generalize the clustering algorithms which have a max cluster size as max cluster size algorithms. These can be simulated by setting the penalty function as in Figure 3. To get the behavior of max size N, I set the penalty function at N to 3. This works just the same as infinite since no two clusters can have a combined penalty greater than 2. The discussion of lines in the previous section holds as long as all pieces of the qualifiers being considered are less than N. The max cluster size discussion from the previous section also holds.

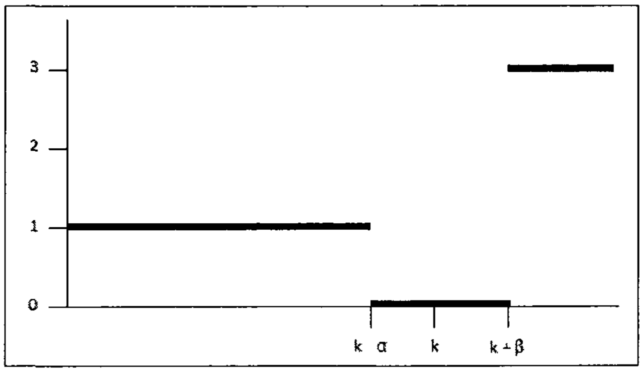


3. Max Cluster Size Penalty Function

The penalty function used to simulate Wang's sizing model is shown in Figure 4. This is not exactly the same, because in Wang's algorithm, a cluster of size 1 cannot merge with a cluster of size k – α. There is not a way to accomplish this using the building blocks explained above, it will have to be done with additional rules. Each of the horizontal lines works as explained previously. The effect of the max cluster size N is also the same as before, except N is labeled as k + β in this case The only piece that needs discussion is the boundary at k – α

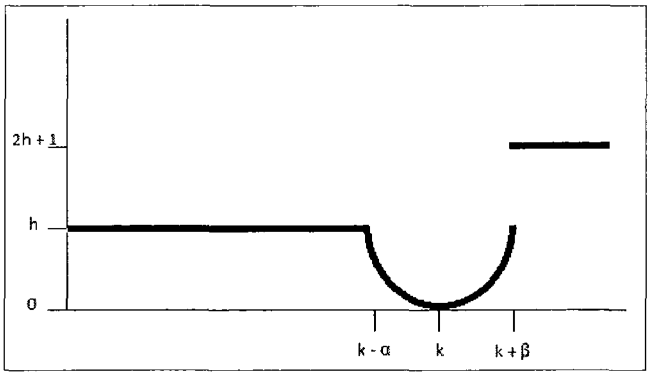
For merges, consider a Cluster A and a Cluster B They will merge together to form a cluster B' of size |A+B| If |A| and |B| are both less than k – α, and |A + B| is greater or equal to k – α, then the penalty before is 2, and the penalty after is 0 Since the total penalty drops, the merge will occur If |A| is less than k – α and |B| is greater than k – α, then the penalty before is 1, and the penalty after is 0, the merge will occur

For a split merge, consider the same clusters A and B and a set of nodes a which will be moved from Cluster B to Cluster A Consider |A| is less than k – α, |B| is greater than k – α, the resulting Cluster A' is less than k – α, and the resulting Cluster B' is greater than k – α, this will not occur since the penalty before is 1 and the penalty after stays the same If in the same situation, Cluster B starts out greater than k – α and B' is less than k – α, the merge will not occur since the penalty will increase The only way this will work is if Cluster A starts out less than k – α and finishes greater than k – α



4 Wang's Algorithm Penalty Function

The first penalty function I propose is shown in Figure 5 This is similar to Wang's simulated penalty function, except for exponential curve between k - α and k + β Within the boundaries of each section of the function, the properties are the same as has been explained previously All merges occurring from clusters less than k – α will occur since the penalty will always decrease, however, resulting clusters with sizes close to k will be preferred Split merges occurring over the boundary will occur only if the total penalty is decreasing This will happen only when the resulting Cluster A' is greater than k – α



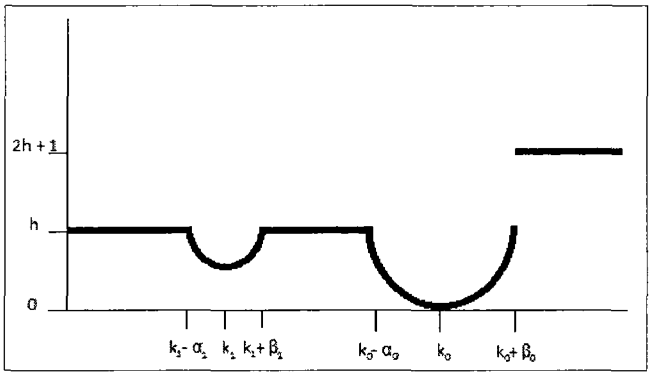
5. Proposed k-centered Penalty Function

The final penalty function I propose, Figure 6, is similar to the k-centered penalty function; however, it includes a secondary minimum. The ideal node size and thresholds have been subscripted with a 0, and the secondary set is subscripted with a 1. This penalty function a theoretical example given to show the robustness of the model.

The secondary minimum in the penalty function will be placed at a point in the function where the application has a secondary point of efficiency. Suppose this is some point k_1 which is less than $k_0 - \alpha_0$. Thresholds $k_1 - \alpha_1$ and $k_1 + \beta_1$ can be assigned to limit the minimum. The real benefit of this minimum happens then the network is too sparse for a cluster of size k_0 to exist, instead, the clusters can form into a cluster within bounds of $k_1 - \alpha_1$ and $k_1 + \beta_1$.

Even if it is found that a secondary point of efficiency does not exist, the secondary minimum can be used to help build clusters more efficiently. To do this, we can set k_1 to be $k_0 / 2$. When building clusters, those within the range of $k_1 - \alpha_1$ and $k_1 + \beta_1$ will be created first. Then these clusters will merge together to form clusters within the range of $k_0 - \alpha_0$ and $k_0 + \beta_0$.

Merging will occur at all boundaries; however, clusters that merge with resulting sizes in $k_i - \alpha_i$ and $k_i + \beta_i$ will have a preference to merge closer to k_i. Cluster split merging at the boundaries will only occur if one of the clusters receives a lower penalty.



6. k-centered with secondary minimum Penalty Function

Summary

In adding the penalty function and qualifiers to the Wang's algorithm, I make the cluster sizing model of the algorithm more robust. The addition of the penalty function allows there to be specific weights associated with each cluster size, and the qualifiers specify if the merge is allowed to happen. Also, I added a split merging process to the algorithm. This addition of the algorithm allows clusters which are already formed to move some of its nodes to a new cluster. This allows for clusters which are already formed to more evenly distribute its nodes.

4 SIMULATION

SIMULATION PARAMETERS

In each simulation run, I set the parameters to match that of Wang's dissertation [13] I am doing this to superficially show correctness for my algorithm by matching his results In his simulations, he had 100 nodes running for 300 seconds Wang did not say how many times a second his nodes would run, in my implementation I ran the algorithm five times a second The boundary is a 500m x 500m square environment The transmission ranges start at 30m and increment by 30m until a max of 210m Each node has a constant velocity of 5 m/s The mobility model used is the random way point model which is discussed in [30]

METRICS

The metrics I used in my simulations are the same as those which Wang used These include Average Time of a Cluster, Average Number of Clusters, Total Number of Clusters, and Total Number of Node Cluster Changes

The Average Time of a Cluster is a measurement of what length of time each cluster is alive At each clock cycle, the cluster id of each node is recorded A cluster is alive as long as one node in a clock cycle considers itself a member of the cluster The length of time for each cluster is calculated and the average is taken

The Average Number of Clusters is a measurement of the number of clusters existing at a point in time At each clock cycle, the number of distinct clusters is calculated The average over all clock cycles is then computed

The Total Number of Clusters is a count of all the clusters generated by the algorithm This is calculated by counting all the clusters for each clock cycle and summing them all together If a cluster with the same name is created twice over the lifetime of the simulation, it will be added twice to the Total Number of Clusters

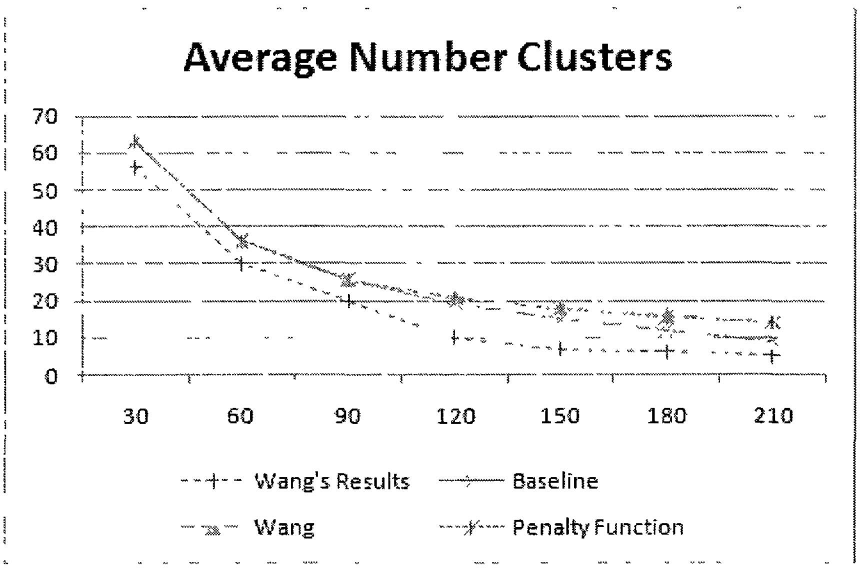
The final metric is the Total Number of Node Cluster Changes This metric is a measurement of the number of times a node changes from one cluster to another cluster This is calculated by looking at each node at a particular clock cycle If in the next clock cycle it has a new cluster id, then the node is considered to have changed clusters. All of these are counted and put into the Total Number of Node Cluster Changes.

SIMULATION RESULTS

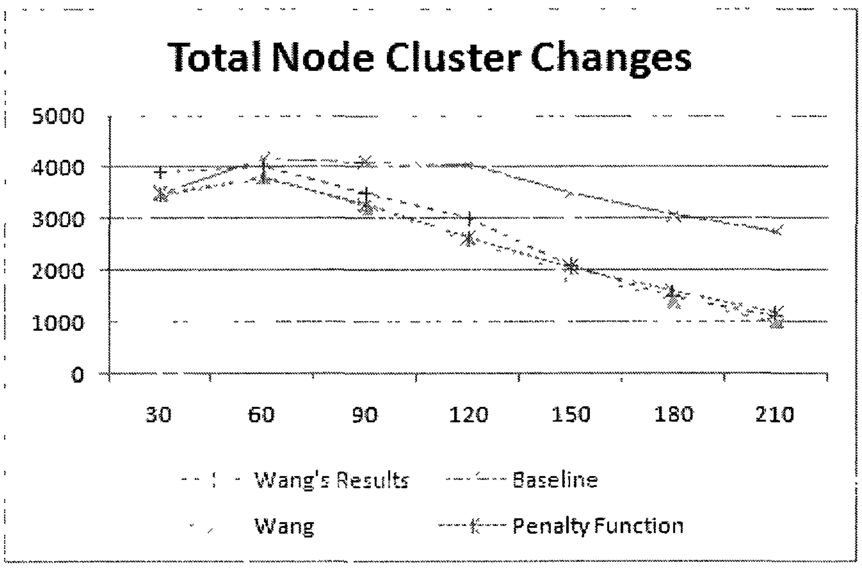
In my simulation runs, I ran three versions of Wang's algorithm. The first is a baseline algorithm as which is explained in [13]. In the baseline algorithm, there is no maximum cluster size, and the stability of the clusters is not considered when splitting a cluster. The second is Wang's algorithm as I have implemented it. This does not include the penalty function or the split merge processes. The third is Wang's algorithm with the cluster sizing replaced by the penalty function.

In Figure 7, Figure 8., Figure 9., and Figure 10. , the data shows results for Wang's Results, Baseline, Wang, and Penalty Function. Wang's Results are the results which are reported in [13]. The rest are the simulation results from my simulations for the Baseline, Wang, and Penalty Function as explained above. For each of these figures, I will explain the differences between the Baseline, Wang, and Penalty Function results. A discussion of the deviations from Wang's Results will proceed.

Figure 7 shows the Average Number of Clusters for each simulation. The y-axis show the number of nodes and the x-axis shows the transmission range of the nodes. As with each of the figures, the Baseline deviates from the other two because it allows larger clusters than the other two simulations. Since the Wang and Penalty Function simulations include a maximum cluster size and the Baseline does not, the Baseline starts to deviate starting at 120m. The curve for the Wang and Penalty Function simulations are nearly identical; however, at 150m and beyond, the Penalty Function simulations prove to be slightly better. This difference is between .25 and .5 nodes better and is related to the Penalty Function allowing merging between the threshold sizes of $k - \alpha$ and $k + \beta$.



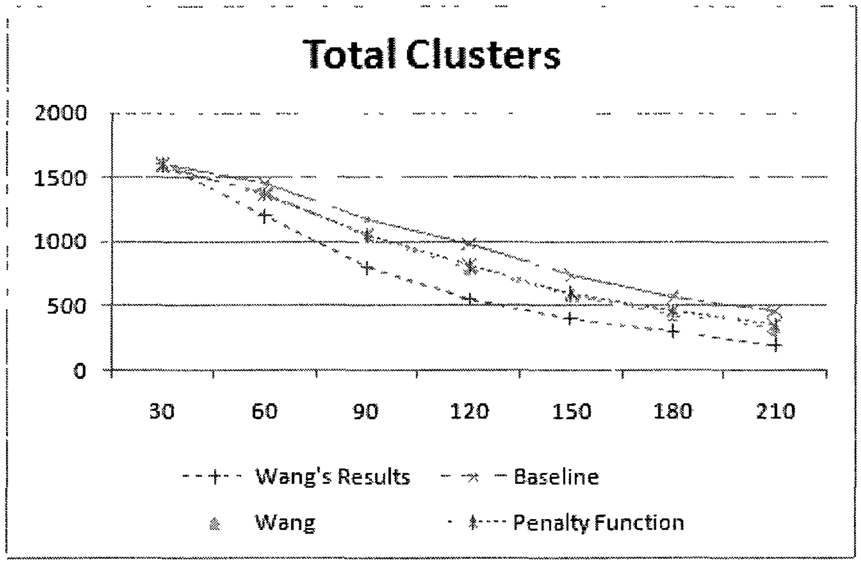
7. Average Number of Clusters



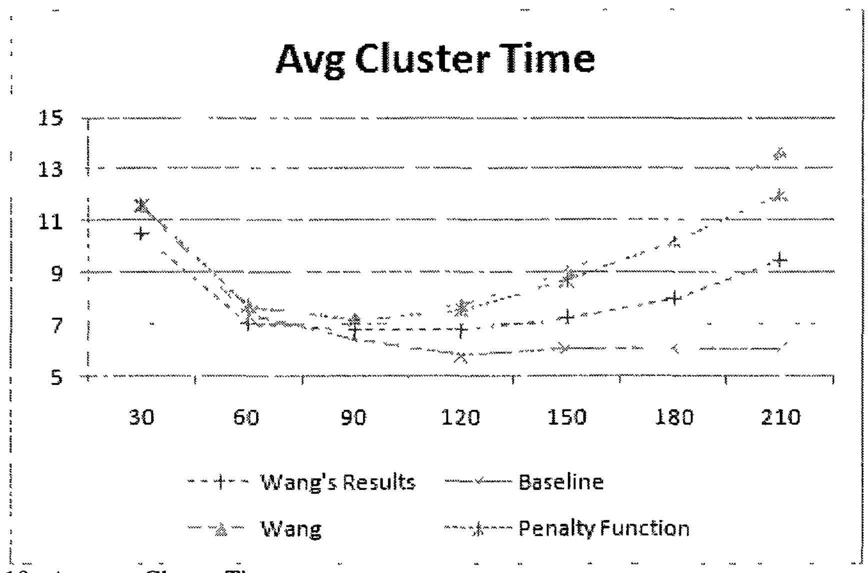
8. Total Node Cluster Changes

Figure 8. shows the Total Node Cluster Changes. The y-axis is the count of the number of times a node changes clusters. The x-axis is the transmission range of the nodes. The curve of the Baseline simulation follows that of Wang and Penalty Function up to 60m, after which it remains much higher than the others. As which each of these figures, the Baseline is off due to there being no limit to the cluster size. As the clusters get larger they are more susceptible to nodes moving in opposite directions. Such mobility causes the diameter-2 property of the cluster to be broken and thus cause a split to happen. Smaller clusters that fit within an ideal node size will not be as susceptible to this issue.

Figure 9 shows the total number of clusters for each simulation. The y-axis shows the number of clusters and the x-axis shows the transmission range of the nodes. The Baseline simulations have around 100 additional clusters for each transmission range over 30. This is attributed again to it not having an ideal or maximum cluster size. The results for the Wang simulation is slightly better than that of the Penalty Function simulation because the Wang simulation merging is based solely on stability, while the Penalty Function first considers cluster size, then uses stability in case there are multiple possible merges with the same penalty.



9. Total Clusters



10. Average Cluster Time

Figure 10. shows the Average Cluster Time of each simulation. The y-axis shows the time in seconds, and the x-axis show transmission range of the nodes. The Baseline starts to deviate from the Wang and Penalty Function simulations at 60m. At 90m the Baseline evens out near 6s for its Average Cluster Time. This shows that even as the clusters get larger, they do not appear to be any more or less stable for the Baseline. All three simulations drop significantly at 60m, and even out. At 120m the Wang and Penalty Function simulations start to increase again. This first drop shows that the nodes with the 30m transmission range are more stable; however, when compared to the Average Number of clusters in Figure 7, they are also very small. The Wang and Penalty Function simulations Average Cluster Time start to increase at 120m. This means the clusters are getting more and more stable. When compared to the Average Number of Clusters in Figure 7, the clusters are also staying the same size. The increasing transmission range is the cause of the increasing stability shown. The Wang simulation increases faster for this simulation. This is again a result to the Wang algorithm using stability to merge its clusters.

Each of the results is identical at a range of 30m. This is because the ideal cluster size or the max cluster sizes are not being met; therefore, there is not much of a difference between the algorithms for such a small transmission range.

As expected, the Baseline simulations prove to be worse for each metric discussed, except for the Average Cluster Size. These results are due to the cluster sizes getting too large. When the clusters sizes get too large, it includes more nodes Since there are more nodes, there is a greater change of nodes moving in opposite directions, therefore, the clusters split more often than if the clusters were smaller The larger clusters results in the Average number of clusters being smaller, but the other three metrics being worse

The Wang and Penalty Function simulations were very close to one another The Penalty Function simulations have a slightly better Average Number of Clusters, but the Wang simulations have slightly less number of nodes changing cluster and less total clusters Changes Its Average Cluster Time is also better These can be attributed to the Wang algorithm merging based solely on the stability of the nodes

Now that each of the different simulations is discussed, the differences between the Baseline and Wang's Results need to be discussed The numbers for Wang's results are estimated from the graphs supplied in [13] This may attribute to some small deviations in the graphs, but the differences shown in Figure 7, Figure 8 , Figure 9 , and Figure 10 are quite significant The obvious blame for these issues is in the deviations I took when implementing Wang's algorithm There were certain questions I could not answer from the explanation given In my implementation when there are multiple maintenance leaders, I am choosing the one with the smallest node id Also, when multiple operations are happening with the cluster at the same time, I am again choosing the one with the smallest node id as the winner Again, none of these seem to be sweeping changes which would cause the large deviations seen, however, they do contribute to some of the small differences

There is an additional reason the numbers are different When computing the metrics, certain accounting tricks can be done to make the results appear more favorable In my results, I calculated the results once for each clock cycle, or five times a second In Wang's results, he actually calculated his results once every second Consider a node in a cluster splitting off because it is out of range of any other node in the cluster This cluster will immediately attempt to find a new cluster If the node merges with a new cluster within one second, it will not be counted in Wang's results Each of the metrics will be skewed because of this Another accounting trick which may be employed is considering a merging cluster as one cluster In my results, I am considering these as two clusters until the merge is confirmed If Wang's results do this, then cluster sizes will be smaller, and the cluster's lifetime will be increased If I employed these accounting tricks, my baseline numbers would be closer to what Wang is showing in his results

5. SUMMARY AND FUTURE WORK

SUMMARY

In this thesis, I started with Wang's clustering algorithm as a base algorithm. I first extended this algorithm by adding the split merge process. Next I added a generic penalty function and qualifiers to change the cluster sizing model for the algorithm. I proposed a k-centered penalty function and the more theoretical k-centered penalty function with secondary minimum. Thorough examination of the simulations results show that my implementation of Wang's algorithm and my penalty function approach were almost identical; however, the cluster sizing model of Wang's algorithm is much more limited than the penalty function approach I proposed.

FUTURE WORK

The algorithms and ideas proposed in this Thesis have plenty room for further extension. The penalty function is not being used when splitting clusters. In my implementation of the penalty function, the maintenance leader with the lowest node id is used; perhaps the maintenance leader with the best resulting penalty should be used. Additionally, the split merge process could be made more generic. In my proposed process for the split merge, it will consider the split merge as if all neighbor nodes of the cluster will be absorbed in the new cluster. In some cases it may be more appropriate to only merge in some of these nodes.

There is also quite a bit on analysis which can be done. There is no analysis of MANET applications to determine an ideal cluster size. Also, there is no analysis of any secondary efficiency points. Also, there is no included comprehensive exploration of the penalty function. The capabilities explained in this Thesis are limited.

Finally, I believe the most powerful use of the penalty function would be if the network itself was able to determine it on its own. In this Thesis, the penalty function is pre-determined and it cannot change during the execution of the algorithm.

REFERENCES

1. Chinara, S., and Rath, S. K.: A Survey on One-Hop Clustering Algorithms in Mobile Ad Hoc Networks. J. of Netw. and Syst. Management. 17, 183–207 (2009)
2. Anitha, V. S., and Sebastian, M. P.: SCAM: Scenario-based Clustering Algorithm for Mobile Ad hoc Networks. Proc. First International Conference on Commun. Syst. and Netw. 97-104 (2009)
3. Venkataraman, G., Emmanuel, S. and Thambipillai, S.: Size-restricted cluster formation and cluster maintenance technique for mobile ad hoc networks. International J. of Netw. Management. 17, 171–194 (2007)
4. Lin, C.R., Gerla, M.: Adaptive clustering for mobile wireless networks. IEEE J. Sel. Area. Commun. 15(7), 1265–1275 (1997)
5. Mc Donald and Znati, T. F.: Mobility Based framework for Adaptive Clustering in Wireless Ad Hoc Networks. IEEE Journal on Selected Areas in Commun. 17, 1466-1486 (1999)
6. Al-kahtani, M. S., Mouftah, H. T.: Enhancements for clustering stability in mobile ad hoc networks. Proc. of the 1st ACM international workshop on Quality of service & security in wirel. and mob. netw., 112-121 (2005)
7. Zhang, L., Soong, B., Xiao, W.: Cluster-adaptive two-phase coding multi-channel MAC protocol (CA-TPCMMP) for MANETs. IEEE International Conference on Commun. 2, 1118-22 (2005).
8. Palit, R., Hossain, E., Thulasiraman, P.: Mobility-aware pro-active low energy (MAPLE) clustering in ad hoc mobile wireless networks. Global Telecomm.un Conference. 6, 3426-30 (2004)
9. Zhou, L., Haas, Z.J.: Securing ad hoc networks. IEEE Netw. 13, 24-30 (1999)
10. Gomathi, K., Parvathavarthini, B.: An efficient cluster based key management scheme for MANET with authentication. Trends in Information Sciences & Comput., 202-5 (2010)
11. Francis, S. J., and Rajsingh, E. B.: Performance Analysis of Clustering Protocols in Mobile Ad hoc Networks. J. of Comput. Science. 3, 192-204 (2008)
12. Wang, Y., Kim, M. S.: Bandwidth-adaptive Clustering for Mobile Ad Hoc Networks. Proceedings of 16th International Comput. Commun. and Netw., 103-8 (2007)
13. Wang, L.: Clustering and hybrid routing in mobile ad hoc networks. Old Dominion University, Norfolk (2005)

14. Ephremides, A., Wieseltheir, J. E., and Baker, D. J.: A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signaling. Proc. of the IEEE. 75(1), 56-73 (1987)

15. Parekh, A.: Selecting routers in ad hoc wireless networks. Proc. of IEEE international Telecommun. Symposium. (1994)

16. Chiang, C. and Gerla, M.: Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel. in Proc. IEEE SICON. 1-15 (1997)

17. Amis, S., Prakash, R., Vuong, T. H. P., and Huynh, D.: Max-Min-d-Cluster formation in wireless ad hoc Networks. IEEE INFOCOM, 1293-1302 (2000)

18. Chatterjee, M., Das, S. K., and Turgut, D.: WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks. Cluster Comput. 5, 193-204 (2002)

19. An, J., Li, C., and Li, B.: A improved weight based clustering algorithm in mobile ad hoc networks Inf., Comput. and Telecommun. (2009)

20. Bricard-Vieu, V., Nasser, N., and Mikou, N.: A Weighted Clustering Algorithm Using Local Cluster-heads Election for QoS in MANETs. IEEE GLOBECOM (2006)

21. Bricard-Vieu, V., Nasser, N., Mikou, N.: A Mobility Prediction-based Weighted Clustering Algorithm Using Local Cluster-heads Election for QoS in MANETs. IEEE International Conference on Wirel. and Mob. Comput., Netw. and Commun., 24-30 (2006)

22. Wang, Y, and Bao, F.S.: An Entropy-based Weighted Clustering Algorithm and Its Optimization for Ad hoc Network. Third IEEE International Conference on Wirel. and Mob. Comput., Netw. and Commun., WiMob. (2007)

23. Basu, P., Khan, N., and Little, T. D. C.: A Mobility based Metric for Clustering in Mobile Ad Hoc Networks. IEEE ICDCW. 413-18 (2001)

24. Wei, D., Chan, H.A., Chuwa, E.L., Majugo, B.L.: Mobility-Sensitive Clustering Algorithm to Balance Power Consumption for Mobile Ad Hoc Networks. International Conference on Wirel. Commun., Netw. and Mobil. Comput., 1645-8 (2007)

25. Wu. Y., and Wang. W.: MEACA: Mobility and Energy Aware Clustering Algorithm for Constructing Stable MANETs. Military Commun. Conference. (2006)

26. Safwat, A., Hassanein, H., and Mouftah, H.: Power-aware fair infrastructure formation for wireless mobile ad hoc communications. GLOBECOM '01. 5. 2832-36 (2001)

27. Qiang, Z., Ying, Z., Zheng-hu, G.: A Trust-related and Energy-concerned Distributed MANET clustering design. 3rd International Conference on Intelligent System and Knowledge Engineering, 1, 146-51, (2008)

28. Li, F., Zhang, S., Wang, X., Xue, X. and, Shen, H.: Vote- Based Clustering Algorithm in Mobile Ad Hoc Networks. Proc. of International Conference on Netw. Technol. 1-10 (2004)

29. Ohta, T., Inoue, S., and Kakuda, Y.: An Adaptive Multihop Clustering Scheme for Highly Mobile Ad Hoc Networks. Proc. of 6th ISADS'03. (2003)

30. Camp, T., Boleng, J., and Davies, V.: A Survey on Mobility Models for Ad Hoc Networks Research. Wirel. Commun. & Mobile Comput.: Special Issue on Mobile Ad Hoc Netw.: Res., Trends and Appl. 2 (5), 483-502 (2002)

VITA

Ryan Florin

Department of Computer Science

Old Dominion University

Norfolk, VA 23529

Ryan Florin received his Bachelor's degree in Computer Science from Old Dominion University in 2005. Starting in 2005, he worked for six months as an MIS Intern at the Chesapeake Redevelopment and Housing Authority in Chesapeake, Virginia. After his internship, he worked at reQuire, LLC in Virginia Beach, Virginia from 2005 until 2008 as a System Administrator and a Computer Programmer. Since 2008 he has been working as a Web Applications Developer at Advanced Health Media in Chesapeake, Virginia. He has been working toward his Master's Degree in Computer Science at Old Dominion University since Fall 2007 and expects to graduate in May 2011 with the completion of this thesis. His interests include Wireless Communication and Science in general.