Enabling Disconnected Transitive Communication in Mobile Ad Hoc Networks

Xiangchuan Chen
University of Rochester
P.O. Box 270226, CSB 734
Rochester, New York, USA 14627
chenxc@cs.rochester.edu

Amy L. Murphy
University of Rochester
P.O. Box 270226, CSB 734
Rochester, New York, USA 14627
murphy@cs.rochester.edu

ABSTRACT

Recently, mobile ad hoc computing has attracted much attention in the research community. Different from most current wireless networks, mobile ad-hoc networks have no fixed infrastructure, all hosts are capable of movement, and the network is continuously reconstructed into multiple disconnected clusters. In such environments, connection is unstable, and communication is unpredictable. Current research mainly focuses on providing the same models of communication in this environment as in fixed networks, focusing on routing protocols for message delivery within connected subsets of hosts, also referred to as clusters. Although this kind of work is crucial, it does not address the possibility of communication across clusters, taking advantage of the movement of mobile hosts which themselves are able to carry messages from one cluster to another. In this paper, we propose a new model of communication model, Disconnected Transitive Communication (DTC), which focuses on cross-cluster communication, and provide the details of a routing protocol to enable it.

Keywords

Moibile ad hoc networks, message passing communication

1. INTRODUCTION

The development of compact computing devices such as notebook computers and personal digital assistants allows people to carry computational power with them as they change their physical location in space. The number of such components is steadily increasing. One goal, referred to as ubiquitous computing, is for these devices to become seamlessly integrated into the environment until we are no longer explicitly aware of their presence, much the way the electric motor exists in the world today. Part of enabling this vision is coordinating the actions of these devices, most likely through wireless mediums such as radio or infrared.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for pro£t or commercial advantage and that copies bear this notice and the full citation on the £rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior speci£c permission and/or a fee.
POMC '01 Newport, Rhode Island USA
Copyright 2001 ACM 1-58113-397-9/01/08 ...$5.00.

Ad hoc mobility is an extreme model of a mobile environment in which no fixed infrastructure exists to support communication. In other words, the distance between hosts and their communication range determine connectivity and as components move, the network is continuously reshaped into multiple clusters, with connectivity available within each partition but not across partitions.

Freeing mobile users from a fixed infrastructure makes the ad hoc network environment ideal for many scenarios including systems of small components such as sensors with limited resources to spend on communication, disaster situations in which relief workers enter a region where the infrastructure has been destroyed, and for settings in which establishing an infrastructure is impossible as in a battlefield environment or economically impractical as in a short duration meeting or conference.

Much effort has been invested in developing protocols for point to point and multicast communication among hosts in ad hoc networks [2, 5]. In many ways, this work mimics routing algorithms in the fixed network, with the primary difference being the lack of hierarchy in the distribution of hosts. Another feature of several proposed protocols is the on-demand discovery of the path from the source to the destination, as opposed to fixed routing strategies which build routing paths even if there is no traffic in the network. This approach in ad hoc environments comes from the observation that the network of connected hosts is constantly changing and pre-computing routes which may never be used unnecessarily consumes costly wireless resources. The focus in these message delivery protocols is on allowing communication between nodes which are multiple hops apart in the ad hoc network. For example, Figure 1 shows an ad hoc network. Even though hosts D and F are not directly connected, any messages sent between them can pass through host E, with E effectively playing the role of a router in the ad hoc setting, enabling a style of transitive communication.

All of these protocols work at the packet level, and delivery is only possible if a path exists from the source to the destination for a period of time long enough to discover the route and transmit the packet. We propose a new communication model, *Disconnected Transitive Communication*, which removes the assumption of connectivity between source and destination from message delivery, allowing a message to be passed from one host to another even if the source and destination are never connected either directly or transitively. It is designed to act as a mid-

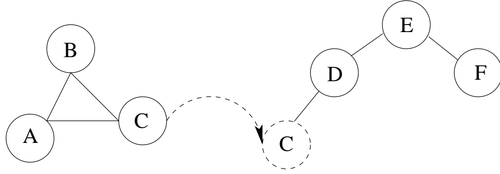

Figure 1: A sample network with six mobile components in two clusters. Dark lines indicate direct connectivity between devices. Dashed lines movement of node C from one cluster to the other.

dleware, exploiting existing wireless routing protocols while providing a useful abstraction for message delivery to applications. Our strategy is most applicable where a high degree of asynchrony in message delivery is tolerable.

The remainder of this paper provides a more precise definition of the model of disconnected transitive communication (Section 2), a detailed description of our application level routing protocol (Section 3), a discussion of our results (Section 4), and finally some conclusions (Section 5).

2. SOLUTION STRATEGY

The communication model, termed disconnected transitive communication described in this paper is intended to provide a new level of asynchronous communication not available in the lower-level mechanisms currently being explored. In protocols such as Dynamic Source Routing (DSR) [3], the default procedure to send a packet initially attempts to find a route to the destination. If no such path can be found within a short timeout of the request for transmission (500ms), the packet is delayed for twice that timeout. This process repeats for at most thirty seconds, after which the delivery fails.

Our idea is simple: Rather than simply fail when immediate delivery is not possible, move the entire message (not a single packet) to another host as close to the destination as possible, where a *closer* host is defined to be one which is likely to be in contact with the destination earlier than the source. Note that this definition does not strictly relate to physical proximity of the hosts. For example, in Figure 1, if host A wants to send a message to host F, a delivery scheme such as DSR will fail. If, however, it can be determined that C will soon be in contact with F, A can pass the message to C (using standard DSR), and C can pass the message to F after C migrates to the new cluster. We do not limit ourselves to a single intermediate hop as this example suggests, but rather provide a general mechanism to move a message through the network, constantly getting *closer* to its destination. Each time the message is transferred, the new host effectively becomes the sender, taking on the responsibility of propagating the message closer to the destination.

3. DISCONNECTED TRANSITIVE COMMUNICATION

This section provides the details of our disconnected transitive communication routing protocol. The biggest challenge is determining which host, if any, is closer to the destination than the host currently holding a message. For this, we turn to application-level knowledge about host movement and connectivity patterns as well as some general characteristics of hosts in ad hoc networks. The protocol outlined in this section sits at the application level (or more accurately at the middleware level, between the operating system and applications), requiring support from lower level unicast and broadcast protocols.

3.1 Overview

The routing protocol is hop by hop in nature, where a hop need not be a direct neighbor. Each host tries to find, within its current cluster, the host which is the next best candidate to carry the message closer to the destination. The case where the final destination host is within the cluster is treated as a special case where the destination is the absolute best candidate host.

The first question to answer is when a host should initiate the process of finding the next hop for a message. In the ad hoc environment, the obvious answer seems to be that this discovery should occur each time the membership of the host's cluster changes, however it is difficult for a host to accurately detect changes in its cluster membership, especially considering that many changes occur several hops away from the source host and are not directly visible by monitoring `hello` packets.[1] We also note it is possible that even if the membership in a cluster is not changing, the host with the highest utility may change over time. In other words, the choice of the best next hop for a message is parameterized by the current time. The details of this will be evident in Section 3.2. Therefore, we choose to discover the next host *periodically*, where the period is tunable by the application and is able to approximately detect changes in cluster membership without adding a great deal of network overhead. The interval is shortened when cluster membership changes are more frequent, and lengthened when changes are infrequent. In Section 4 we discuss the details of this parameter.

The discovery protocol runs once every period and has three distinct phases: utility probe, utility collection, and message redistribution. During the utility probe, the host holding a message probes its current cluster for the utility of the member hosts with respect to the message and its destination. It is possible to probe for information about the destination of multiple messages simultaneously, however for simplicity we limit our description to discovery for a single message. The transformation necessary to send multiple messages is straightforward. After the probe, the source collects the utility information from its cluster members and makes a decision about which host(s), if any, to send the message. This decision is followed immediately by a redistribution of the message to one or more of the cluster members, using a lower level unicast routing protocol to deliver the message. The details of each phase are discussed in Section 3.4.

3.2 Utility computation

The key to the success of our routing strategy is the computation of the next hop on the path to the destination. To this end, we introduce a new concept, *utility*, to describe the usefulness of a host as the next (application level) hop

[1] `hello` messages, or beacons are typical in many mobile ad hoc protocols. A `hello` message typically contains the identity of the sending host and is only able to be perceived by the hosts within range of the sender. `hello` packets are not typically forwarded, making their periodic transmission ideal for immediate neighbor discovery.

for a message. In other words, for one specific message, the utility of a host, h, reflects the possibility that h will meet the destination of the message before the message becomes invalid. A message carries a time to live value, and when this expires, the message is dropped from the system. This prevents stale, no longer useful messages from consuming system resources.

In addition to host identity, we have identified five characteristics of mobile hosts which can be exploited to increase the dependability of the utility calculation. These include the list of hosts most recently noticed, the list of hosts most frequently noticed, the future plan of a host, the power level, and the rediscovery interval. Each of these is discussed in more detail later in this section.

To calculate utility from these parameters, we have two basic options. First, a host holding a message and wishing to discover the utility of its cluster members with respect to this message can collect all the needed information from each cluster member, then perform a local maximization to determine the next hop for the message. This solution is undesirable both because of the amount of network bandwidth required to send the information and the requirement that the hosts explicitly divulge the values of each of the utility components. The second option, which we employ, is to have each host calculate its own utility on demand. During the probe phase of our protocol, the source broadcasts a request which reaches all members of the cluster, identifying the key parameters of the message for utility calculation, namely the destination of the message, the timeout of the message, a threshold value and possibly other information discussed later. The timeout specifies the lifetime of the message. The threshold is a minimum utility which must be met to consider a host as a candidate next hop and is typically set to the current host's utility for the message. When the utility probe arrives, the cluster member calculates its own utility, and if the utility is above the threshold, sends a reply back to the source. While the threshold value is not necessary, it reduces the overall network traffic if many hosts in the cluster have low utility.

3.2.1 Utility Components

The five components which we identify for calculation of the utility with respect to a message are common on many mobile hosts and can be effectively used to evaluate the possibility of future connectivity to the target host. They can roughly be categorized into history, future, and system properties. In many mobility scenarios, the historical connectivity pattern of a host provides a reasonable estimation of the future connectivity due to the propensity of users to repeat patterns of movement. The two historical properties identified here are appealing because they do not require any application intervention to collect, but instead can be extracted directly from lower level information. On the other end of the spectrum, we can exploit the future expected behavior of a host which has been explicitly entered by the user, for example in the form of a calendar (a typical application for mobile devices such as personal digital assistants) or a movement plan recorded for a traveling business. The final type of property we consider includes system parameters such as power and the rediscovery interval of the routing protocol itself.

Most recently noticed. Because history is often a good predictor for the future movement of physically mobile components, we exploit knowledge of the hosts which have recently been noticed, keeping a FIFO queue of elements of size proportional to the portable device (assuming that a laptop computer with a large disk can store a larger history than a PDA with relatively little memory. To populate this queue, a host listens to all packets on the network including `hello` messages and any data messages, snooping for message sources. If the host of a message is already in the queue, it is removed and reinserted at the end of the queue, stamped with the current time. A new host is simply inserted at the end. If this insertion causes an overflow, the host from the head of the queue (the oldest) is evicted.

It is reasonable to expect that a host which has recently noticed the target will do so again in the future, therefore this component of the utility calculation (U_{MRN}) assigns a higher utility for a node which has recently noticed the destination (D):

$$U_{MRN} = (1 - \frac{CurrentTime - TimeLastNoticed_D}{TimeOut_m}) * 100$$

where $TimeLastNoticed_D$ is the time recorded in the MRN queue for node D, $CurrentTime$ is the current wall clock time, and $TimeOut_m$ is the time after which the message m is no longer valid in the system. This time value serves to normalize the MRN utility among all host. If the destination host has never been detected, or the right hand side of the equation is negative, U_{MRN} is set to 0. We also enforce this non-negative utility policy on each of the following utility computations.

Most frequently noticed. By adding minimal overhead to the MRN queue we can also record information relevant to the frequency of noticing a host, namely the number of encounters and the time of the first encounter. Using this information we can calculate U_{MFN}, a utility value which increases with the frequency of previous encounters:

$$U_{MFN} = (1 - \frac{CurrentTime - FirstTimeNoticed_D}{NumTimesNoticed_D * TimeOut_m}) * 100$$

Future plans. The previous two values are nice because they require no user intervention to collect or to exploit for the utility computation. The only point where the users are aware of this information is the storage requirement, which, as mentioned earlier, can be adjusted to reflect the storage available on the device. In other situations, the user can have a more direct impact on a utility predictor by registering a plan of future interaction. This can be as simple as a list of the hosts which are expected to be met in a given time frame, or as complex as a calendar where meeting names and types can be associated to individual hosts. An immediate issue which arises here is privacy of the calendar information. This can be addressed in two specific ways. First, we expect that a calendar application will provide a limited interface which extracts only the information necessary for the utility computation, namely the time to the next meeting with a host D. If the host does not wish to reveal this information, the return value is indistinguishable from the case where no meeting is scheduled, and no private information is leaked. Second, the computation of the utility value is wholly contained within the host that owns the calendar. We discuss this in more detail in Section 3.2.2, but the idea is that by containing the calculation, the host has a greater degree of control over the information released.

To calculate the utility based on this type of future information, U_{CAL}, we define a function which decreases with the time to the next meeting:

$$U_{CAL} = (1 - \frac{NextMeetingTime_D - CurrentTime}{TimeOut_m}) * 100$$

As before, if no meeting with the given destination is scheduled, U_{CAL} is set to 0.

Power. We now turn to basic system properties which do not depend on the user, but rather are intrinsic to the host itself such as the remaining power in the battery. The driving idea behind using power for utility computation is that the longer a host will remain alive, the higher the probability that it will meet the destination. Similarly, a host with low power is likely to shut down, removing itself from the system and making it an undesirable next hop:

$$U_{Power} = (1 - \frac{TimeOut_m}{EstimatedRemainingPower}) * 100$$

Rediscovery Interval. The final parameter we exploit for utility computation is an artifact of our protocol, namely the frequency which a host initiates the probe phase searching for the next appropriate hop for the messages it is holding. The rediscovery interval of a host is allowed to vary over time to reflect the environment of the host, increasing when the members of the cluster are changing rapidly and decreasing when the cluster members are stable. Section 3.4 provides some insight into a mechanism to change this value, but the intuition for the utility computation is to send a message to a more active host, or a host with a lower rediscovery interval (RDI).

$$U_{RDI} = \frac{MIN_{RDI}}{RDI} * 100$$

where MIN_{RDI} is the minimum rediscovery interval for a host, the value of which is determined based on the environmental characteristics where the DTC communication protocol is being employed. More details of the RDI are discussed in Section 3.4. It is interesting to note that U_{RDI} is the only parameter which is not affected by any properties of the message.

3.2.2 Calculating Utilities

The overall utility of a host to serve as the next hop for a message is computed using the above five values. Because different values will have more meaning depending on the environment of the application, we give the programmer the opportunity to specify different weights for each parameter. For example, some applications may consider the frequency of past meetings more important than the future potential of meeting. In this case, the weight applied to U_{MFN} will be higher than the weight of the calendar. The only restriction is that the sum of the weights must equal 1. The default is to assign each utility equal weight. The specific weights are distributed with the utility probe message.

Another consideration is that a host may not want to reveal some piece of information about itself through the utility function calculation. In this case, one of the utility values will appear to be zero, where in fact the actual value is positive. While this does affect the performance of the routing algorithm, it is something we cannot expect to control in the anarchic mobile ad hoc environment. This, however, does mean that because the utility calculation is contained completely within a host, that host has complete control over the amount of information shared, and on whether they will even participate in the DTC routing protocol. A host that never replies to utility probes will never receive any packets (including those with it as the destination).

```
int utilityComp() {
    identifier hopList[]= {C, E, F};
    /* retrieve this host's id */
    identifier myID = getLocalHostID();

    if (myID==HopList[0]) return 30;
    if (myID==HopList[1]) return 50;
    if (myID==HopList[2]) return 100;
    else return 0;
}
```

Figure 2: Specifying a hop list as part of a user defined utility calculation.

3.3 Source-de£ned utility

With the system defined utility parameters of the previous section the source of the message has only the freedom to define weights, however this still limits the control of the sender in affecting the routing. For example consider the case where the source has explicit information about how to route a message, such as a sequence of hosts which is likely to result in delivery of the message. The utility parameters cannot effective encode this information. While it would be possible to simply add another utility parameter to handle this case, we opt for an overall increase in flexibility, allowing the source to provide the code of a specifically tailored *utility function* (utilityComp). This function is distributed during the utility probe phase of the protocol, is evaluated at each host in the cluster and the value is returned during the utility collection phase.

To avoid many of the security concerns associated with general mobile code, we assume that this source-defined function only has access to a very restricted set of system parameters which are made explicitly available by the host. These parameters are either known in advance or extracted on the fly using a reflection mechanism. In this paper we do not propose a specific solution, but instead rely on the existing and growing body of work on mobile code. Figure 2 provides a simple example of a source define function which specifies a *hopList*, or a sequence of hosts which, if traversed, will most likely reach the destination. The only system information used by this function is the identity of the host on which the function is executing (seen in the getLocalHost() function call);

In the example, A specifically lists the sequence $\{C, E, F\}$, meaning that F is the destination, and E is closer to F than C. As the message moves through the system, if a node which is closer to the destination in the sequence is in the cluster, the message should be transferred to that node. Hosts in the sequence can be skipped. In the example of Figure 1, the message will not be explicitly transferred to node E because the destination, F, is immediately accessible when C migrates (the message will pass through E as a consequence of the underlying point to point ad hoc routing protocol, but at no point will E be responsible for finding the next hop at the level of our protocol).

Another possible use of this source defined function is to

```
findNextHop (message m)
    utilityResponses = {};
    myUtility = LOCALUTILITY(M);
    threshold = myUtility;

    while (true) do
        DELAY(RDI);
        utilityResponses = {};

        /* phase 1, utility probe */
        BROADCAST(UTILITYPROBE(m.dest,
                                m.timeOut,
                                m.weights,
                                m.utilityComp),
                  threshold);

        /* phase 2, utility collection*/
        DELAY(ResponseTimeOut);

        /* phase 3, message redistribution*/
        myUtility = LOCALUTILITY(m);
        if ( utilityResponses != {}
            ∧(x is host in utilityResponses
                with maximum utility))
            SEND(x.host, m);
            break;
    enddo
```

Figure 3: Three phase protocol for routing a single message m to its next destination. Functions in all capital letters are system functions. The parameters of send are the unicast destination and the message being sent.

```
RECEIVEUTILITYPROBE_h(x)
    /* probe message x from host h */
    if (x.utilityComp ! = null)
        /* use user defined utility computation */
        utility = EXEC(x.utilityComp);
    else
        /* use default or provided weights with
           usual utility parameters */
        utility = x.weights
                * LOCALUTILITY(x.dest, x.timeOut);
    if (utility > x.threshold)
        SEND(h, utilityResponse(utility))

RECEIVEUTILITYRESPONSE_h(x)
    /* response message x from h */
    utilityResponses += (h, x);

RECEIVEMESSAGE_h(x)
    /* receive a data message from host h */
    if (x.destination == myID)
        pass x to application
    else
        /* initiate DTC for message x */
        findNextHop(x);
```

Figure 4: Atomic operations at each host which fire in response to the arrival of either a control message (*utilityProbe, utilityResponse*) or an actual data message.

exploit the location awareness of nodes in certain situations. Consider a set of robots with integrated global positioning systems. Interesting utility functions can be devised to send a message *toward* a home base with hosts moving in that direction, or always propagate message *to the north*. The flexibility of the application programmer to take advantage of specialized hardware and keep the same communication model makes this model of a user-defined utility function attractive.

3.4 DTC routing protocol details

As outlined previously, our disconnected transitive communication protocol proceeds in three phases: utility probe, utility collection, and message redistribution. Figure 3 outlines the process taken to send a single message m to its next hop. The host holding the message iterates through a loop containing the three phases and when the message has been passed to the next hop, the loop terminates. While we show this for only a single message, adding multiple messages is trivial and can be done in one of two ways. Either a separate loop can exist for each message, effectively running multiple instances of the same algorithm in parallel, or the loop can be expanded to account for multiple messages and sending a single utilityProbe message representing all of the pending messages. This latter idea is more appealing because it more efficiently uses the network bandwidth, an important concern in a wireless network.

While the critical parts of the protocol appear in Figure 3, several actions to handle the arrival of system messages occur in parallel, supporting the routing protocol. These operations are outlined in Figure 4, and are distinguished by the type of message which arrives. A host can expect to receive a UTILITYPROBE message at any time, in which case it will simply compute the utility using either the user utility function or the system parameters and weights. If the utility is above the threshold sent with the message (x.threshold), a reply is generated to the host which initiated the UTILITYPROBE. These UTILITYRESPONSE messages will be received during phase two of the protocol, and will simply be stored, to be examined during phase three. The only other messages in transit are the actual data messages which are distributed in the final phase. When one of these messages arrives, if the receiving host is the destination, it passes the message to the application, otherwise it begins the rediscovery process for this message, calling the function defined in Figure 3.

Rediscovery Interval. The rediscovery interval (RDI) is critical to the routing protocol because it indicates the frequency which the host holding the message searches for a new host to carry the message. A small rediscovery interval means a high load placed on the network to transmit the utility probe and response packets. Alternately, if the rediscovery interval is too large, the host runs the risk of not taking advantage of a transient connection with a host with a very high utility. Therefore, we must dynamically balance between the network load and the chance of successful delivery.

Our current solution sets a default RDI and doubles it each time a utility probe is completed, effectively slowing down the frequency of rediscovery exponentially. When a host detects new hosts in its cluster, the value is immediately reset to the initial value, forcing a quick rediscovery. For a host to detect that new host are present, every host listens to the periodic HELLO messages from its neighbors,

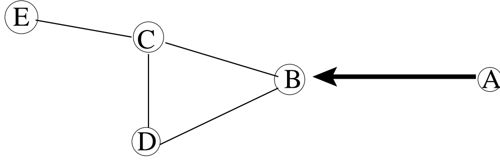

Figure 5: An example of RDI resetting: An isolated host A moves toward a cluster.

recording whether there are new neighbors since the last utility probe. This only captures the immediate neighbors of a host as HELLO packets are not propagated to the whole cluster. In order to detect information beyond the immediate neighbors, we augment every HELLO message with a boolean variable which indicates whether the RDI of the HELLO sender has been reset since the last HELLO message was transmitted. By adding this single bit of information, we can effectively detect the dynamicity of the network. The result is that the response to a new host is delayed by time proportional to the number of intermediate hops between hosts, as this is the time that it takes to propagate the extra bit of information through the network.

Figure 5 shows a sample of setting the RDI. Initially, hosts B, C, D and E are in one cluser, and A, a previously isolated host moves within range of B. During the first HELLO period hosts A and B will reset their RDIs. In the second period, C and D will reset their RDIs, and finally in the third period, host E will reset its value. To avoid a host constantly resetting its RDI, we impose a two cycle delay between any resets. In the example, this means that although C and D will see HELLO messages from one another in the third round with the RDI bit indicating a reset, they will only reset their RDI one time (in response to the message from B in round two, ignoring the fact that the reset bit is also sent in message received in round three).

4. DISCUSSION AND RELATED WORK

One of the features of the disconnected transitive communication protocol we present is the flexibility available to the application designer to either accept the default parameters of specify their own weights or even a function for utility calculation. Because the choice of utility function and weights affect the behavior of the algorithm, we believe it is critical to put this control in the hands of the application programmer. Another way to increase the likelihood of successful delivery is to increase the number of hosts the message is propagated to on each hop. Instead of simply choosing a single host to carry the message to the next hop (as shown in Figure 3), the algorithm can be tailored to select the top n hosts in the cluster with the highest utilities, where n is specific either to the application or unique to each message. This reduces the chance that a message will get stuck in a local maximum, unable to get closer to the destination. However, this benefit must be weighed against the increase in network traffic and the overall storage required throughout the system during message delivery.

Another possibility is to assign the rediscovery interval on a per message basis rather than per host. A message with high priority can be given a small RDI while one with low priority or a distant timeout can be assigned a low RDI. This allows the sender to have even greater control over the forward progress made by a message, but at the same time may place unreasonable demands on the intermediate hosts both in terms of computation as well as network overhead if it becomes the next hop for several high priority messages.

We must also consider the overhead on the wireless network. Because the utility probe is distributed using broadcast, we need to pay attention to the effects of broadcast storms and insist on the use of existing techniques to reduce to the overhead required to broadcast [6]. Second, when there are large numbers of messages in transit through the system, we can consider caching information about the general utilities of the members of the cluster in much the same manner which ad hoc routing algorithms such as AODV [5] store information about the current path to various hosts. This style of caching must be carefully conceived as the cached values will need to be more generic than the utility calculations presented in the previous section as they can only take into consideration the message destination, and no information specific to the message (such as the timeout or a user-defined utilityComp() function).

Using our algorithm in conjunction with DSR [3] also offers many opportunities for optimization. At the highest level, the DSR algorithm first enters a discovery phase to find a path to the next host, waits for a response, then sends a message to the discovered destination. These phases closely match the phases of our algorithm, and combining our utility probe with the route discovery of DSR seems a natural place for reducing network overhead. Of course, we must consider the tradeoff between remaining independent of the underlying routing algorithm and the potential increase in efficiency.

In either case, the return of utility values to the source is an interesting place to consider for optimization. We currently assume that as soon as the utility computation is complete, the UTILITYRESPONSE message is sent via unicast directly to the host that sent the original UTILITYPROBE. While this is functionally correct, the burden on the system to establish these individual routes to the source may be unreasonable, especially if an on-demand routing mechanism such as DSR is used. It may be possible during the broadcast phase to build a short-lived spanning tree rooted at the source, along which the reply message can be sent without the additional routing overhead of unicast. Alternately, or in addition, multiple UTILITYRESPONSE message can be collected and sent in a single message to the source.

Another approach to disconnected communication, developed at Duke university leverages off of the idea of epidemic algorithms to propagate messages in an ad hoc network [7]. When two hosts come into communication range, the host with the smaller identifier initiates an *anti-entropy session* with the host with the larger identifier. During this session, two hosts exchange packets that have not been seen by the counterpart. Given sufficient buffer space and time, these anti-entropy sessions, with reasonable probability, will eventually deliver the message. Unlike our approach, they do not leverage off of any application level information. Thus, the number of packet exchanges may be numerous. There is also a high possibility that every message will flood the entire set of hosts, regardless of the probability of hosts meeting. By exploiting application knowledge and the ability of hosts to remember their own history, our approach also achieves a high probability of success but with many fewer messages.

An essential question to answer is whether the incorpora-

tion of application-level knowledge makes sense in a routing protocol. To answer this, we consider the typical analysis of traditional ad hoc routing protocols using the CMU extensions to the Berkeley NS2 simulator [1]. The input to this simulator includes application-level information such as the number of nodes in the system, their speed, the size of the environment, and the model of communication among hosts. In other words, the very analysis of the algorithms depends on these parameters, but the behavior of the algorithms does not! In fact, recent literature comparing various ad hoc routing protocols suggests that the choice of routing protocol depends greatly on the application environment [4]. From this, we conclude that adding this kind of application knowledge to the routing protocol is a natural step, especially considering the potential gains in the ability to send efficiently messages across disconnected clusters.

We are currently in the process of simulating our DTC routing protocol using the Berkeley NS2 simulator [1]. After considering the four ad hoc routing protocols which are available in NS (DSDV, DSR, TORA, and AODV), we chose to build on top of DSR due to the similarities between it and our protocol. The core modification is that upon the failure of DSR to find a route, the DTC routing protocol takes over, broadcasting a utility probe message and collecting responses. The MRN and MFN queues are filled by monitoring all messages that flow through the DSR agent, extracting and recording the source. Because DSR is an on-demand protocol, it has no need of the proactive HELLO mechanism, therefore we have included the HELLO protocol from TORA in our extensions and use it to update the rediscovery interval. We also expect to exploit the power-awareness mode of NS and by assigning different initial power levels to the hosts, analyze the protocol's effect. Another important point to consider is the set of scenarios suitable for testing our protocol. The standard NS distribution contains two basic scenarios, however they do not match well the environments where our protocol will be most useful. In general, these scenarios have a high density of hosts, movement is slow, and the overall connectivity among all of the hosts in the system is high. We are currently working on scenarios where the density of nodes is much lower, leading to multiple, independent clusters with nodes moving among clusters fairly frequently.

Once the initial simulation is complete, we expect to explore several additional scenarios, including introducing *stationary* hosts which can serve as stable points in the disconnected routing infrastructures. Although these hosts will have the same communication capabilities as the mobile hosts, their lack of movement lends them different characteristics which we expect to exploit in utility calculation. We expect that addition of these extremely predictable nodes will, in general, increase the probability of successful delivery of a host.

5. CONCLUSION

In this paper, we identified a limitation of current communication models in mobile ad hoc networks, namely their inability to provide any communication across disconnected clusters. We proposed a new of model for disconnected transitive communication and also described the details necessary to implement our model in an application level routing protocol based on the calculation of host utilities, continuously moving messages to hosts which are likely to meet the destination. Future research will include the development of a more effective assignment of the rediscovery interval, testing the routing protocol under various mobility patterns, and the introduction of security mechanisms to ensure that messages transit without corruption. Although this work is at its very beginning stages, our initial findings indicate that the integration of application level knowledge into the disconnected transitive communication model has a great potential in maximizing the communication ability of ad hoc networks.

6. REFERENCES

[1] CMU Monarch Project Extensions to ns2. `http://www.monarch.cs.cmu.edu/cmu-ns.html`, 2001.

[2] S. Das, C. Perkins, and E. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proc. of the IEEE Conf. on Computer Communications (INFOCOM)*, pages 3–12, Tel Aviv, Israel, March 2000.

[3] D. Johnson, D. Maltz, Y. Hu, and J. Jetcheva. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks . Internet Draft, March 2001. IETF Mobile Ad Hoc Networking Working Group.

[4] A. B. McDonald and T. Znati. A dual-hybrid adaptive routing strategy for routing in wireless ad-hoc networks. In *Proc. of IEEE Wireless Communications and Networking Conference 2000 (WCNC '00)*, Chicago, IL, USA, September 2000.

[5] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Wkshp. on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, USA, February 1999.

[6] Y.-C. Tseng, S.-Y. Ni, and E.-Y. Shih. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad hoc network. In *Proc. of the 21st IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, Mesa, AZ, USA, April 2001.

[7] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-2000-06, Duke University, 2000.