

Lecture 16 Part B: Enabling Disconnected Transitive Communication in Mobile  
Ad-Hoc Networks

Instructor: Soma Chaudhuri

Scribe: Danyu Liu

## 1 Lecture topic

This lecture is about the presentation of the paper *Enabling Disconnected Transitive Communication in Mobile Ad-Hoc Networks* written by Xiangchuan Chen and Amy L. Murphy. The presentation was done by Danyu Liu.

## 2 Introduction

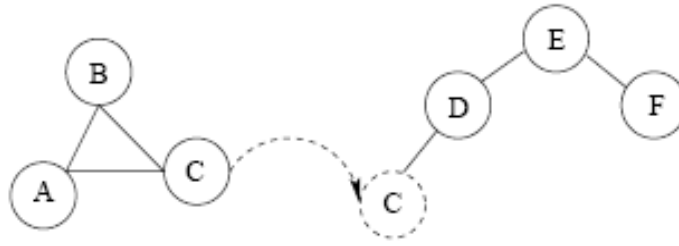
With the development of compact computing devices such as notebook computers and personal digital assistants, people can carry computational power with them as they change their physical location in space. Each computing device can be seamlessly integrated into the environment until we are no longer explicitly aware of their presence. Many research works have been invested in developing protocols for point to point and multicast communication among hosts in ad hoc networks and the proposed protocols work at the packet level, so 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. This paper proposes a new communication model, Disconnected Transitive Communication, 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.

## 3 Solution Strategy

The Dynamic Source Routing (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. The default DSR procedure sends 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, the packet is delayed for twice that timeout. This process repeats for at most thirty seconds, after which the delivery fails.

In contrast to DSR, the proposed DTC works as follows: Rather than simply fail when immediate delivery is not possible, move the entire message to another host as close to the destination as possible, where a closer host is dened to be one which is likely to be in contact with the destination earlier than the source.

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  $F$ ,  $A$



**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.**

can pass the message to  $C$  (using standard DSR), and  $C$  can pass the message to  $F$  after  $C$  migrates to the new cluster.

## 4 Disconnected Transitive Communication

The biggest challenge is determining which host, if any, is closer to the destination than the host currently holding a message.

### 4.1 Overview

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.

When a host should initiate the process of finding the next hop for a message?

- In the ad hoc environment
  - This discovery should occur each time the membership of the host's cluster changes
  - It is difficult for a host to accurately detect changes in its cluster membership
- In the DTC environment
  - Discover the next host periodically
  - 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
  - The Interval is lengthened when cluster membership changes are infrequent

Discovery protocol runs once every period and has three distinct phases:

1. *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. 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
2. *Utility collection*: 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
3. *Message redistribution*: a redistribution of the message to one or more of the cluster members, using a lower level unicast routing protocol to deliver the message

## 4.2 Utility Computation

The computation of the next hop on the path to the destination is the key to the success of the routing strategy. The overall utility of a host to serve as the next hop for a message is computed by using five parameters.

1. the list of hosts most recently noticed ( $U_{MRN}$ )
2. the list of hosts most frequently noticed ( $U_{MFN}$ )
3. the future plan of a host ( $U_{cal}$ )
4. the power level ( $U_{Power}$ )
5. the rediscovery interval ( $U_{rdi}$ )

How to calculate utility from these parameters?

- Brute force method:
  - collect all the needed information from each cluster member, then perform a local maximization to determine the next hop for the message
  - (undesirable) the amount of network band-width required to send the information
  - (undesirable) the requirement that the hosts explicitly divulge the values of each of the utility components
- Proposed method
  - each host calculates its own utility on demand
  - reply back to the source if the computed utility is above the threshold
  - (desirable) reduces the overall network traffic

### 4.2.1 Utility Components

Five components are used to compute the overall utility of a host to serve as the next hop for a message.

1. most recently noticed ( $U_{MRN}$ )

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

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.  $U_{MRN}$  assigns a higher utility for a node which has recently noticed the destination ( $D$ ).

2. most frequently noticed ( $U_{MFN}$ )

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

$U_{MFN}$  increases with the frequency of previous encounters.

3. Future plans ( $U_{CAL}$ )

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

$U_{CAL}$  is a function which decreases with the time to the next meeting.

4. Power ( $U_{Power}$ )

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

5. Rediscovery Interval ( $U_{RDI}$ )

$$U_{RDI} = \left(1 - \frac{MIN_{RDI}}{RDI}\right) * 100$$

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.

### 4.3 Source-define Utility

With the system defined utility parameters, the source of the message has only the freedom to define weights. We can introduce *utility function* to give the source of the message more control of the sender in affecting the routing.

Figure 2 provides a simple example of a source dene 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).

```

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.**

#### 4.4 DTC Routing Protocol Details

As outlined previously, the DTC 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 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. Figure 4 illustrates these operations. 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.

## 5 Conclusion

This paper identified a limitation of current communication models in mobile ad hoc networks, namely their inability to provide any communication across disconnected clusters. It also proposes a new model for disconnected transitive communication and describes 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.

```

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.

```

RECEIVEUTILITYPROBEh(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))

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

RECEIVEMESSAGEh(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.