DTCAST: Delay and Disruption Tolerant Multicasting Protocol

Alexander Afanasyev, Keith Mayoral, Zhenkai Zhu and Soon Young Oh
Computer Science Department
University of California, Los Angeles
{afanasev,keithmay,zhenkai}@cs.ucla.edu

Abstract

Mobile ad hoc delay/disruption tolerant networks (DTNs) networks with intermittent connectivity, have recently received a lot of attention because of their applicability in various applications, including multicasting. We are proposing design of DTN multicast protocol which is called DTCAST for ad-hoc multicast in intermittent connectivity environments. DTCAST protocol can be defined as dual routing and data delivery protocol, which combines on-demand routing phase of ODMRP protocol with short-distance epidemic data dissemination and local route injection to increase robustness in highly mobile environment. One of the key features of the proposed protocol is two-class delivery guarantee for data messages using explicit and implicit acknowledgements. In the paper we present various experimentation results of our protocol implementation as well as simulation results to show the advantages of epidemic dissemination and network in highly mobile environments.

I. Introduction

Several different methods for providing multicast routing and delivery of messages in mobile ad hoc and delay/disruption tolerant networks (DTN) have been proposed over the last several years. These networks will be widely deployed in the future to provide a variety of services, for example, network extending without additional infrastructure or sensor networks, which collect environmental data and vehicular MANETs which help with collaborative driving and central traffic control. Often in these applications, we need to disseminate data to a group of receivers, e.g. we may want to reset a series of thresholds in sensing on a certain subset of sensors, or the officer want to provide traffic information to the vehicles in near blocks. Thus, multicast service in DTN is highly desired.

We are proposing design of DTN multicast protocol which is called DTCAST for ad-hoc multicast in intermittent connectivity environments, which introduce additional features that many other existing solutions cannot provide. We have reviewed different aspects of proposed protocol, as well as performed experimental and simulation evaluation of the protocol under variety DTN connectivity and mobility conditions.

Our project is focused on a dual routing and data delivery protocol with additional local route injection design and implementation and its experimental evaluation, as well as on extensive simulations of multicast implementation of epidemic routing and network coding extension to ODMRP [1] protocol coupled with local recovery phase and short-distance epidemic routing to provide informal proof of effectiveness of proposed protocol design.

Epidemic routing is a mobile ad-hoc routing algorithm most suited for partially-connected, delay-tolerant networks. It works on the premise of flooding the network with messages for intended recipients. A node which carries a message for another node will infect any node it comes in contact with, whether or not it is the intended recipient. The only exception would be if the node just encountered had previously been seen before recently; then no infection would occur. The idea behind this is simple: the more nodes you can infect with your message, the better your chances become that one of those infected nodes will eventually come into contact with the intended recipient. Your chances improve, of course, when you allow infected nodes to continue to infect others, thus causing an epidemic of your message. To control this epidemic, it therefore becomes important that you set a Time-To-Live (TTL) and Max Hop Count for each message you send. These functions ensure that your message eventually gets stopped (hopefully after its been received) and doesnt bog the network down with unnecessary message infections.

Combination of reactive on-demand ODMRP-like routing approach and opportunistic epidemic routing provides extended quality of multicasting service in delay/disruption tolerant networks, as well as decreases control overhead of protocol.

The rest of the paper is organized as follows. Section II lists related protocols in multicast in ad-hoc networks (MANETs) and points out its key features, Section III discusses our proposed protocol design features. Section IV is devoted to experimental evaluation of our implementation of DTCAST protocol. Section V provides simulation results for DTCAST protocol evaluation. And finally, Section 7 concludes the paper.

II. Background and Related Work

In last several years, there have emerged a number of research projects proposing different routing approaches in the mobile ad hoc networks (MANET). Hong et al. in [2] have classified them to flat, hierarchical and geographic position assisted routing protocols. Flat routing can be represented as proactive (table-driven) and reactive (on-demand) protocols. Delay and disruption networks imply high probability of network nodes interconnectivity changes, which eliminates proactive protocols, which induces excessive amount of overhead due to persistent maintaining routing tables.

ODMRP [1] is a very popular multicast protocol in MANETs. Rather than maintaining routes proactively, ODMRP provides pure on-demand approach to route establishment by network loop-prevention flooding from the source node who wants to send data with *Join Query* packets. Multicast group members interested in receiving data send *Join Reply* upon receiving the *Join Query*, which is used to establish actual multicast forwarding paths. The source sends out *Join Query* periodically to maintain the mesh of multicast forwarding paths, and a node is no longer considered as a group member if it does not reply the *Join Query* message during predefined timeout period. ODMRP is robust to mobility, fast fading, obstacles and jamming, but have considerable control overhead due to frequent network flooding.

Local route recovery was introduced by E-ODMRP [3] to tackle with the high mobility environment with lower control overhead. With this scheme, receiver estimates packets arriving interval and set up a corresponding timer. If timer expires, the disconnected node proactively grafts onto the forwarding group mesh instead of waiting until next route refresh.

The ADMR [4] protocol performs adaptive learning of source traffic broadcasting patterns to distinguish between sporadic connection loses and node permanent disconnections. On detecting connection loss, protocol is employing local recovery and global recovery procedures. To perform channel detecting if source temporarily stops generating data messages, protocol define keep-alive message sending at increasing inter-packet intervals. Permanent disconnection of source node is detected by significant deviation of silence period from traffic generation pattern. However, it is claimed that the protocol doesn't required network flooding to maintain required source-based routing trees, it still requires explicit network-wide flooding of join messages from receiver (*multicast solicitation*) and either network-flood or unicast type reply from source node. Moreover, the protocol requires that an additional *Receiver Join* message to be sent from receiver to source as a reply to unicast keep-alive reply on *Multicast Solicitation* message.

MZR [5] is a routing protocol which combines proactive zone routing table maintaining (i.e. hierarchical routing scheme) with reactive source initiated on-demand building multicast source routing tree based on session identification using <source_id, group_id> pair. It is proposing to use simplified distance vector protocol inside each zone to maintain Zone Routing Table using local broadcast of special *Advertisement* packet. The whole multicast tree is created in on-demand way in multi-phase process - initial source tree building inside zone and recursive source tree expansion over zone limits using source tree branch creation from each border node. However MZR protocol provides limited network flooding while route discovery, it still has significant overhead due to proactive nature of zone route maintaining.

Message Ferrying [6] utilizes a special group of nodes called *message ferries* to provide communication services in sparse MANETs. Ferries move around the area along pre-determined routes, providing regular but intermittent communication services to those non-ferry nodes. Non-ferry nodes can choose to move freely and meet ferries by chance, which is highly possible assuming the number of the ferries and the routes they move are reasonable; or they can adjust their movements to meet the ferries proactively. Ferries collect packets through the contact with non-ferry nodes, and then deliver the packets to their receivers if these nodes happen to move around the ferries in the future.

Message Ferrying was originally proposed as a unicast protocol, but [7] extends it with multicast functionality. The source can simply send multicast message to the ferries, and they will handle everything else. When ferries move around, they advertise a list of active groups to the nodes they meet, and these nodes choose to join a multicast group by sending a *JOIN* message to ferry. Upon becoming a group member, the node can receive the multicast packets of that particular group.

There has been several papers addressing epidemic routing to be more well suited for mobile ad-hoc and delay-tolerant environment. Vahdad et al. [8] describes the basic protocol design of Epidemic Routing in its original form, meant for unicast traffic. Chen [7] briefly mentions how to create a multicast extension to Epidemic Routing, which became a base of our simulation evaluations. Zhao et al.[9] performs evaluations of this multicast extension of epidemic routing protocol.

Above mentioned proposals present only routing phase for data delivery process and rely on upper layer protocol to maintain data integrity. On contrary, Bose et al. [10] have presented FACE-1 and FACE-2 routing algorithms in the MANET which, besides routing phase, guarantee delivery without requirement of persistent memory on the network nodes. Also it is claimed, that algorithms do not need of packet duplication based on finding connected planar node interconnection subgraph and perform routing on this planar graph. Unfortunately, it only works in the case of static and connected network during message transmission. Moreover, experimental work shows, that algorithms proposed in [10] have limited efficiency.

III. DTCAST Protocol Design

While designing our multicasting protocol over ad-hoc overlay we have taken in account that it should satisfy several requirements:

- should perform guaranteed data delivery
- be tolerant to short-term node disconnections
- be tolerant to long-term node disconnections
- be tolerant to node mobility (connection path changes)
- should have low control data and retransmission overhead
- be able to work over any link-layer protocols

Requirements to be tolerant to short-term disconnectivity, long-term disconnectivity and node mobility, due to different type effects on the node interconnections, lead to different mechanisms incorporated into the protocol. To be able to tolerate all above mentioned interconnection changes patterns, our protocol has several phases: *route discovery*, *data transfer*, *data catch up* and *local route injection*. *The Route Discovery* phase consists of ODMRP-like [1] on-demand route discovery procedure. *Data transfer* phase includes data delivery with implicit and explicit delivery confirmations as a solution to long-term node disconnection, i.e. data will not be deleted from local message queues until some kind of confirmation is received. *Data catch up* phase is triggered when node on the path to destination detects, that next-hop node is not available. This phase is aimed to be solution to short-term disconnections and node mobility and employs short-range epidemic data dissemination [8]. *Local route injection* phase destination node performs injection of forward path routing to all its neighbors upon detecting that it stops receiving data. This is very similar to local recovery phase of E-ODMRP protocol [3], but doesn't require explicit data stop messages and instead it uses detection heuristics based on timeout after last data message received. This phase is aimed to improve data delivery in the case high node mobility.

We have implemented our protocol in Click Router [11] environment. Due to the experimental nature of the implementation we have not considered any security issues, however some insider and outsider attack prevention schemes should be implemented in the underlaying link or network layer protocols in the case of the real deployment. Our implementation borrows from ODMRP

[1] idea of on-demand building route to the destinations, but it, as it will be described in detail in Section III-A, extends forward path building. In the ODMRP protocol, each node on receiving Route Reply message decides whether to create or not forwarding rule for the multicast stream based on the ”Next” field in the message. DTCAST extends Route Reply messages and Forwarding rules to contain destinations ID list, which allows to establish delivery responsibility for source and intermediate nodes. However, this introduces additional overhead to the routing messages and forwarding tables, it can be considered as minimal due to the fact, that Route Reply messages in our implementation have time granularity 5 seconds and each destination ID is 32 bits.

On the other hand, DTCAST protocol can be considered as dual protocol: routing and actual data delivery protocol with two class servicing. All data to be send to number of destinations is encapsulated in special DTCAST data messages which are identified by unique ID number and multicast ID number. Moreover, each data message contains a special data field, which indicates the period of data message actuality, which has to be set by source node and can be in the range from several seconds to several days. Each data message is also stored internally at each node associated with the list of the destinations to which this data message should be delivered. This association is generated based on source node knowledge of the list of destinations (first-class services) and on the list of destinations from Reply messages (secondary service). First-class service allows delayed data delivery even if node is not available at the moment of the first delivery try, e.g. source will keep trying to disseminate all actual data messages to all nodes from internal list. Second-class services allows data delivery if node is interested in multicasting stream and is active at the time of delivery tries to the first-class serviced nodes.

The current implementation is available online through public GIT [12] repository http://github.com/cawka/click-dtcast/ and consists of three basic Click Router [11] elements: Source, Receiver and Forwarder. The *source* element implements local queueing and data message encapsulating; the *receiver* generates route replies and performs message decapsulating. The *forwarder* element performs interaction between other DTCAST stations in the network.

A. Routing Phase

As was mentioned earlier, each source node a-priori knows list of first-class serviced destinations ID list as well as multicast ID. In our implementation we have defined source, destination and multicast IDs be 32-bit fields. While source node SRC has any data messages intended for at least one destination it periodically (*Route Request Time* = 5s) sends *Route Request* messages. Each *Route Request* message contains *Route ID* in the form of tuple (Src ID, Stream ID), which will be used to create tree from source node for particular multicast stream.

Each *Route Request* can be uniquely identified by source node ID, multicast stream ID and sequence number. To limit network flooding and prevent loops in route discovery process, each node should process each unique *Route Request* only once. This can be done by checking special designated *Cache Table* using tuple (Src ID, Stream ID, Seq#) from the *Route Request*. In the case of duplicate *Route Request* detection, node should discard such packet without performing any further actions.

Otherwise, each node inserts or updates data in the *Source Routing Table*, which can be represented as a list of tuples (Src, NextNode, LastUpdate), where Src, $NextNode$ – IDs of the source and next node on the path to the source, $LastUpdate$ – last time of the information update to purging unused paths in the network. After updating *Source Routing Table* node should decrease RTT value and rebroadcasts *Route Request* packet. *Source Routing Table* should be periodically checked to delete old records.

The destination, on receiving *Route Request* for the expected $Stream\ ID$, forms and broadcasts a *Route Reply*, where it includes itself ID. Each node on receiving *Route Reply* checks if its ID equals $Next\ ID$ in the *Route Reply*. If it is true, then node search record in *Source Routing Table*. If there are no records - path to the source node is unknown, than *Route Reply* is discarded, otherwise it should fill record in *Forwarding Table* for the particular $Route\ ID$ with tuple (Dst, Fw flag, Timestamp), where $Fw\ flag$ is set to true. List of destinations in the *Forwarding Table* is used to establish delivery responsibility and because of dynamic character of the table it contains all downstream active destinations of the multicast tree.

While there are records in the *Forwarding Table* and correspondent records in *Source Routing Table*, node periodically form and broadcast *Route Reply* packet for each $RouteID$ designated to $NextID$ node, containing associated list of destinations (Figure 1). If node detects, that source path is no longer available, than it should delete the corresponding records in *Forwarding Table*.

The example of the route discovery presented in Figure 2. There is source node 1 and two destination nodes 6 and 8. Source node broadcasts *Route Request*, containing stream ID X (RR<X>). Each node on the path constructs their own *Source Routing Table*, e.g. remembers source of the route request packet, and further rebroadcasts *Route Request*. Nodes 6 and 8 on receiving *Route Request* form and broadcast their *Route Reply* for the stream ID X, containing correspondent destination IDs – RT<X>[6] and RT<X>[8] accordingly. On receiving *Route Reply* node 4 updates its *Forwarding Table* to include node 6 in the downlink destination node list, or in other words, establish delivery responsibility for node 6. Similarly node 5 establish delivery responsibility for node 8. Node 3 would receive *Route Reply* from both node 4 and node 5 and will update *Forwarding Table* to contain records for both nodes 6 and 8 as downlink destination list for stream ID X. Finally, node 3 will generate cumulative *Route Reply* packet containing list of all downlink nodes (node 6 and 8) and will broadcast it to the source node 1.

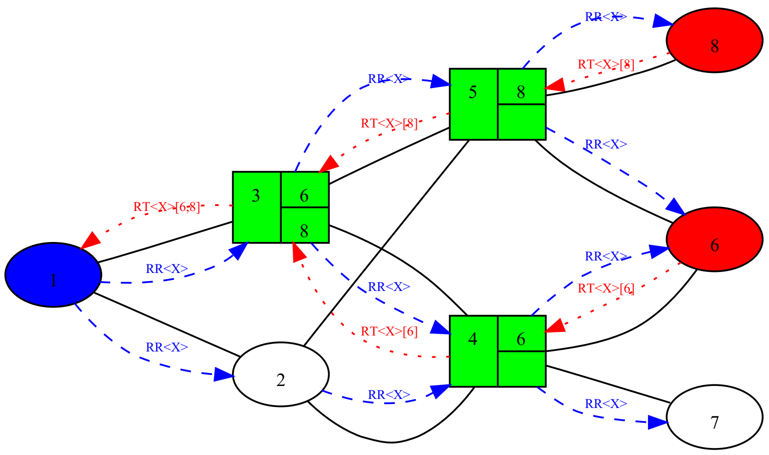

Fig. 1: Route Discovery

B. Data Transfer Phase

Each data message in DTCAST protocol is uniquely identified by stream ID and sequence number. On the other hand, it is associated to particular route using tuple ($Stream\ ID$, $Src\ ID$). This means that each source should disseminate data using distinct stream ID and, in the same time, data message from one source can be disseminated using different multicast routing trees. Such approach is one of the key elements of the guaranteed delivery service and especially useful in the case of high node mobility. Additionally, data message contains *Time of Actuality* field, which is used to decide time moment of undelivered data discarding from local queues. *Time of actuality* is set by the source in can be in range from several seconds to several days.

Delivery guarantee in delay and disruption tolerant networks implies the possibility to store each data message for long-time periods. Actual data storage requirements depends on source data rate, overall data length and node connectivity. To be able to recover from short-term disruptions, protocol should have short data retransmission period, but there is tradeoff between this period and network overhead. Our implementation includes heuristics to have small retransmission period (5 seconds) and in the same time have small network overhead. This heuristics can be described as pseudocode in Figure 4. As it can be seen, message will be broadcasted twice only in the case of non-empty intersection of destination list of the message and list of downlink destinations. Destination list of the message decreased each time node receives explicit or implicit data acknowledgement.

```
OnRetransmitTimer( DataMessage ){
        serviced_dsts = DataMessage.getDsts();
        downlink_dsts = ForwardTable.getDsts( DataMessage.getRouteID() );

        if( serviced_dsts ∩ downlink_dsts ≠ ∅ )
                broadcast( DataMessage );
}
```

Fig. 2: Heuristics to trigger data transmission

Because our protocol allows retransmission nodes should allow the reception duplicate data packets, but at the same time should prevent excessive data rebroadcasting, e.g. upstream nodes should not rebroadcast data message, which is broadcasted by downstream node. Additionally, upon receiving data message node should notify upstream node of successful data message receptions. We have defined two types of notifications: *implicit* (passive) and *explicit* (active). In the first case data rebroadcasting plays role of implicit notifications and in the second case special acknowledgement packet is send to the upstream node (see Figure 3).

On receiving a data packet each node performs heuristic decision procedure whether to rebroadcast packet (”send” implicit notification to the uplink), or not to broadcast and to send explicit delivery notifications. Decision procedure can be described using pseudocode in Figure 5. In the case of reception data packet and cached list of serviced destinations is equal to current list of serviced destination for the *Route ID* obtained from *Forwarding Table*. This is indication that message is from the upstream node, which haven't detected implicit notification and node send explicit notification. If the list of currently serviced destinations contains at least one node, which is not present in cached list, then node performs message rebroadcasting.

```
OnReceiving( DataMessage ){
        serviced_dsts = Cache.previously_serviced_dsts( DataMessage );
        downlink_dsts = ForwardTable.getDsts( DataMessage.getRouteID() );

        if( serviced_dsts = downlink_dsts )
                broadcastActiveAck();    // Probabably source haven't
                                         // received implicit notification
        else if( downlink_dsts − serviced_dsts ≠ ∅)
                broadcast( DataMessage );
}
```

Fig. 3: Data broadcasting decision procedure

On receiving implicit or explicit notification, node finds corresponding record in *Message Queue* for (Route ID, Seq#) and removes from $Dst\ list$ of the found record all $Dst\ ID$'s containing the notification. If after removing records $Dst\ list$ contains no records, the node removes whole record from *Message Queue*.

Example of the DTCAST data transfer with active acknowledgements is presented in Figure 6. Source node 1 broadcasts message X using previously established route. After receiving acknowledgements from node 3, node 1 discards message X from *Message Queue*. Node 3 broadcasts message X and after receiving acknowledgements from nodes 4 and 5 discards message X from *Message Queue*. Similarly, message X will be delivered to destination nodes 6 and 8 and discarder from local queues on nodes 4 and 5.

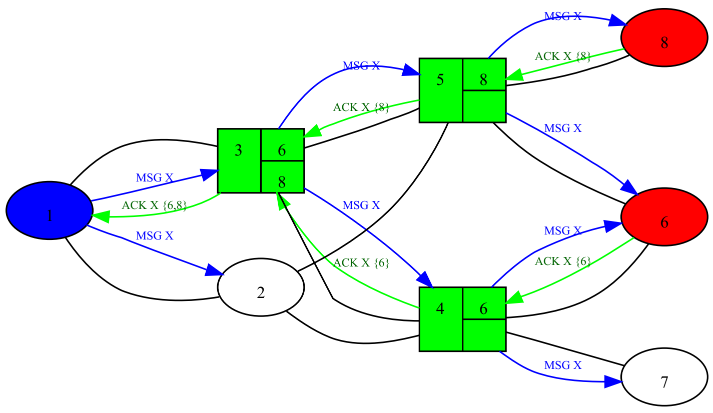

Fig. 4: Data transfer (full connectivity)

C. *Data Catch Up Phase*

Data catch up phase is triggered if some node detects downlink path disconnection. Detection is based on observation downlink path disappearance ($\texttt{cached\_serviced\_dsts} - \texttt{downlink\_dsts} \neq \varnothing$). Detection is performed only on intermediate nodes on the path from source to destinations.

Upon triggering, node updates change original source ID of appropriate records in *Message Queue* to itself ID. This step allows to build new multicast tree from the intermediate node and rebroadcast data using new route. Additionally, node sets "Epidemic" flag to same records for 20 seconds and performs epidemic data dissemination [8] in the range of 2 hops, which ensures data delivery in the case of high mobility and fast route changes.

On receiving data with EPIDEMIC flag set each node broadcasts acknowledgement with EPIDEMIC flag set and TTL=4 and performs local delivery if data cache doesn't contain record for data, decrement TTL field and rebroadcasts data if TTL permits.

On receiving acknowledgement, additionally to data transfer phase actions, node checks if acknowledgement has "Epidemic" flag. If it has, node decreases TTL field and rebroadcasts packet if TTL permits.

Example of phase action is presented in Figure 7. Path between node 4 and 6 accidentally breaks (e.g. due to mobility of the nodes), node 4 after detecting disconnection will send data packet containing message with epidemic routing flag set, TTL value equals to 2. Assuming, that nodes move relatively slow and there is high connectivity between nodes, there is high probability, that data packet will reach node 6, which in the same epidemic manner will acknowledge all nodes in the range 2xTTL.

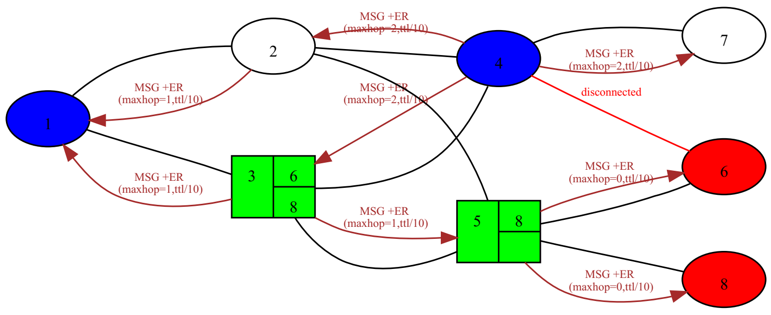

Fig. 5: Data transfer (route loss)

D. *Local Route Injection Phase*

Local route injection phase is triggered if a destination node hasn't received new data messages during a pre-defined timeout value (in our implementation we have defined this timeout be 5 seconds). For 5 seconds after having been triggered, the node will generate a special form *Route Reply* packets, which have TTL=1 and $NextNode$ field equals to special *All Nodes* value. This phase can help to recover from sporadical connection loses be injecting forwarding path to all neighboring nodes.

IV. DTCAST Performance Evaluation

A. Experimental Testbed

For the experimental evaluation, our DTCAST implementation was deployed to 7 WiFi enabled laptops. Interconnections as illustrated in Figure 8 were modeled using internal firewall on each computer. Node 1 is a source, node 6, 7 are receivers with first-class delivery service, other nodes are second-class receivers. This model allows to evaluate impact of critical and non-critical path nodes (nodes 4 and 3 consequently) disconnections.

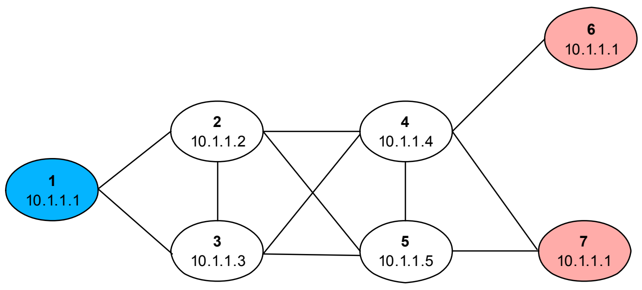

Fig. 6: Experiment interconnection scheme

Experimental evaluation was performed using source data stream configuration as follows:

- data messages size 1024bytes
- data speed 1 packets/second
- continuos data transfer for 40 seconds

The purpose of evaluation is to get multi-hop data propagation delay characteristics for all test-bed nodes depending on different network states. Delay was calculated as difference in test data message timestamp and reception time on each receiver. System clocks of all testbed machines was previously synchronized using NTP protocol (http://www.ntp.org/).

B. Experimental Results

Experimental results show message delay changes for all testbed nodes relatively to source node in different interconnectivity dynamic environments. Figure 9 presents behavior of the DTCAST in the static lossy environment (static WiFi network). Due to chosen retransmission timeout granularity, first data message, which triggered on-demand route discovery, has been delayed up to 12 seconds. Almost all other data messages experienced delay under 1 second. It is clearly visible, that node 6, which has at least 3 hop distance from source, had the most message delay. Message delay in this case can be described as queuing effect on one of the intermediate nodes (delay is almost equal message queue check granularity). Delay increase for the message number 5 and 12 could occur due to packet loss. It can be deducted from the graph, that loss for message 5 occur between 1 hop and 2 hop nodes, because we can observe proportional delay increase for nodes 4, 5, 6 and 7. Loss for data message 12 occur in the path between node 7 and its uplink, because this loss have affected only delivery to node 7.

Figures 10 and 11 illustrate protocol behavior consequently in the case of short-term and long-term disconnection of node 3 which is not on critical path to any other node. Former case was produced by restarting network interface on the target laptop on 15th second of the experiment. Latter case was performed by connecting node 3 to the network on 9th second of ongoing experiment. Negative delay values mean that particular message was not delivered by the receiver. On both

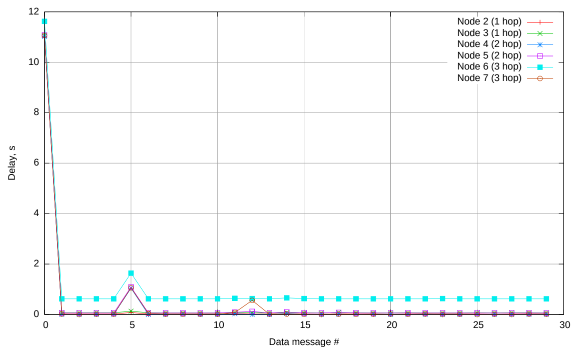

Fig. 7: Full connectivity, data transfer 1 KB/s

diagrams we can observe prolonged period of stable route establishment and this can be described first Route Request packet loss. After its retransmission (retransmission period is 5 seconds), delay for messages on one-hop receivers became almost zero. Due to queueing delays on each distance level, we can clearly distinguish one-hop, two-hop and three-hop distance receivers in the DTCAST stable phase. Short-term disconnection (around 1 second) affected only data message number 15 by increasing delay to 1.2 seconds (retransmission granularity is 1 second). Data packet loss on 31st second is also have impact only on this message delivery delay for node 3.

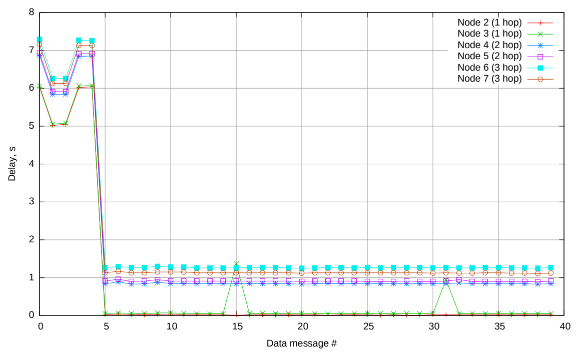

Fig. 8: Short-term disconnection of node 3 on 15th second (non-critical path)

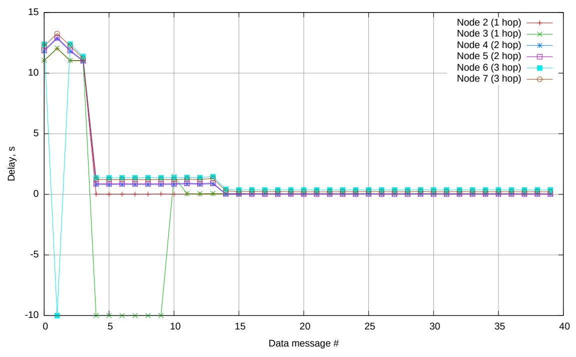

Fig. 9: Node 3 was disconnected from network until 9th second (non-critical path)

As it was expected, in the case of delayed by 9 seconds connection node 3 to the network, there are some undelivered data messages due to second-class guaranteed delivery service (see Secion III). Due to prolonged initial route discovery, node 3 is able to receive first four data messages, and because source node doesn't provide first-class guarantee delivery service to node 3, data messages 4 through 9 haven't been delivered. Delay dynamics between 10 and 11 seconds of the experiment shows that initially node 3 have received data message 10 from one of 2-hop nodes (delay almost equal to 3-hop nodes) and than have happened route rebuilding process and it became stable 1-hop receiver. Dynamics between 13 and 14 seconds of the experiment is one more example of route rebuilding. There was observed unexpected loss of second data message for the receiver 7. Such unexpected spontaneous loses were observed also in other experiments and partly can be described by heuristic nature of detection upstream/downstream nodes (see Section III-B) – in the case of wrong decision, upstream node can send acknowledgement to the transmitting downlink node (i.e. node 5 has transmitted data message, node 7 due to loss hasn't received it, but node 3 due to wrong decision can transmit acknowledgement, which cause local queue purge and impossibility of future retransmissions).

Figures 12 and 13 illustrate protocol behavior consequently in the case of short-term and long-term disconnection of node 4 which is on critical path node 7. Experiment scheme was analogous to the previous case with the difference of disconnection and connection time moments. For the short-term disconnection experiment interface of node 4 was restarted on 7th second of the experiment and for long-term disconnection experiment node 4 was connected to the network on 23rd second of experiment.

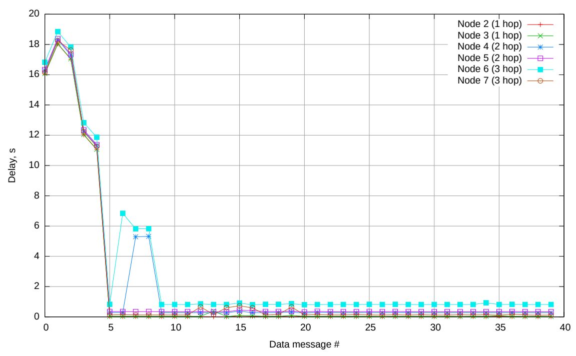

Fig. 10: Short-term disconnection of node 4 on 7th second (critical path to node 7)

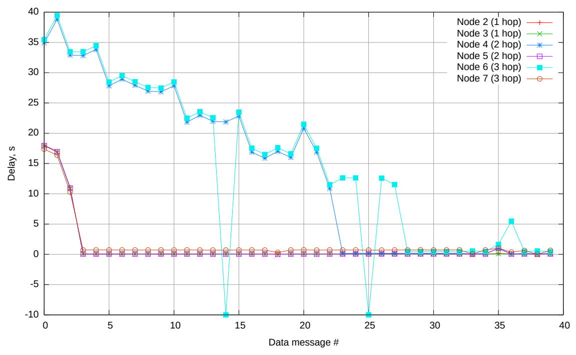

Fig. 11: Node 4 was disconnected from network until 23rd second (critical path to node 7)

During short-disconnection node 4 and node 7 weren't receiving any data messages; after a coarse timeout, a lost message was successfully retransmitted from one of the intermediate nodes. It should be stated, that delivery delay to the node 7 is constantly higher that delay to node 4, because of connection path to node 7 dependence on node 4. Very interesting behavior is experiencing data delivery in the case of delayed node 4 connection to the network on 23rd second. Due to broadcasting nature of data delivery process, node 4 start instantly receiving data messages starting from number 23 and simultaneously node 4 starts route discovery procedure, which generally repeats starting behavior of all experiments, but only for single node 7. Unsuccessful delivery of data message 14 and 25 as in previous experiments, can be partly described by sporadic decision heuristics misses. All data prior node 4 connection to the network due to first-class service for node 7 was buffered at the source node and after path discovery data buffer was almost gradually delivered and source queue purged. As in short-term experiment, delivery delay to the node 7 is more than delay to the node 4.

Experiments have confirmed workability of designed protocol in variety of dynamic network environments. But also help to discover several weaknesses. First of all, there is small percentage of unsuccessful delivery for the first-class serviced nodes, which is probably caused by data loop-prevention heuristics mismatches in some cases and there is need additional research to close all cases in this heuristic procedure. Also, experiments show, that there is different equilibrium points for data delivery delays – on Figure 11 we can see two different stable states between seconds 5–13 and 14–40. Additionally, it was quite unexpected, that initial route discovery process can have variety duration (1 to 5 seconds) and have different impact on delivery delays (5 to 14 seconds). All these observations are to be resolved in the ongoing research.

C. Simulation Environment

The simulation field size is 2000×2000m. There are 50 nodes moving within the area and, in most cases, the multicast group has 11 members, including 1 source. Nodes are distributed in the area randomly at the beginning of the simulation and then are moving according to random waypoint model with speed randomly chosen from 1 to 35 m/s and a pause time of 3 seconds. All the nodes use *802.11b* as the physical layer protocol. And ODMRP *Join Query* interval [1], [3] is set to 3 seconds, while the *Packet Recovery* interval is set to 1 second. The source generates and sends 200 packets in the beginning of the 500-second simulation time, with interval of 0.25 second.

To better evaluate the protocol in a large area with more nodes, the protocol implemented in the simulation is a little different from the one implemented in field experiments. Here, instead of guaranteed service, the nodes in the forward group do not keep state of dissemination process to down stream nodes. The source compares the received Join-Replies to the member-list it keeps and determines if some group members cannot be reached; if it is the case, the source set on the Epidemic Routing Flag in the header of the packets. The hop limits of *Epidemic Routing Packet* and *Recovery Request Packet* are both set to 1.

In the future work we are planning to implement another modified version of DTCAST, which additionally integrates network coding.

D. *Simulation Results*

In all above simulations, the achieved packet delivery ratio, PDR $\approx$ 100%, e.g. almost all data messages were successfully delivered to all multicast group receivers. Thus we will not mention this metric in the later in this section.

Figure 14 shows the time every node needed to collect certain amount of data packets. Note that these received packets need not to be consecutive. As we can see, all the receivers receive 10 data packets within 20 seconds, and it is relatively easy to collect data packets when the source is sending data, since when the nodes happen to be reachable in ODMRP multicast, they will receive several packets. But after the source stopping sending data, for those packets that are missing, a node need to move close to another node that happens to have these packets, and gets a part of missing packets in local recovery process or epidemic routing process. Thus, it becomes more difficult to get the missing packets; as we can see from the Figure 14, the time needed to collect another 10 packets grows quickly after 50 simulation seconds. As the result, to get higher PDR, the dissemination process need more time to finish, e.g. it takes about 180 seconds to ensure that every group member receives more than 95% data packets, while it will take about 245 seconds to increase PDR to 99.5%. Fortunately, the information expires in a relatively long period where the DTCAST protocol is used.

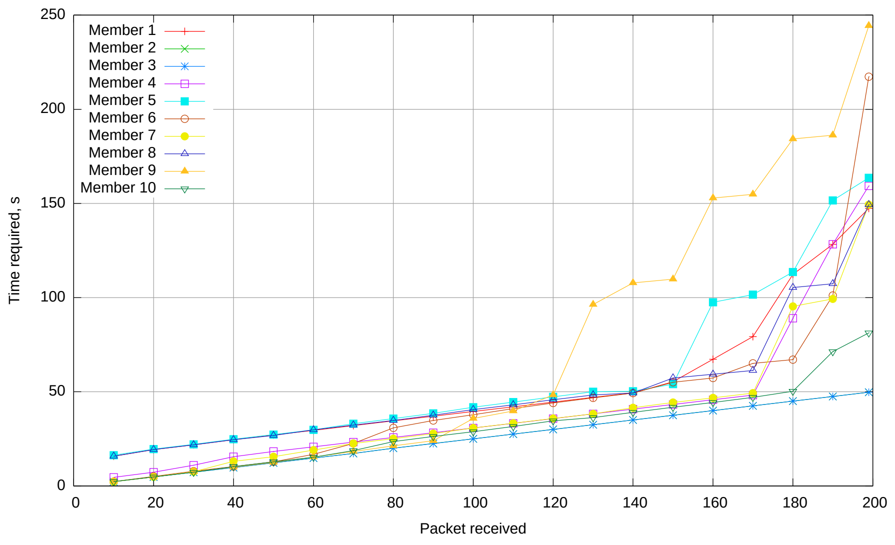

Fig. 12: Dynamic View of DTCAST Process

To better evaluate the influence of Nodal Speed, we did experiments under two simulation environments. In the first environment, ODMRP Join Query interval is set to 15 seconds, and Packet Recovery interval is set to 3 seconds, while in the second environment, these two parameters keep unchanged. In both environments, we did 6 experiments, with nodal speed chosen from 6 speed range: (1 m/s to 5 m/s), (5 m/s to 10 m/s), (10 m/s to 15 m/s), (15 m/s to 20 m/s), (20 m/s to 25 m/s), (25 m/s to 30 m/s).

The experiments results are shown in Figure 15 and Figure 16. In situation 1, both *Average Delay* and *Maximum Delay* is smallest when the nodal speed is within 10 to 20 m/s. When speed is lower, the further node has to move for longer time in order to get in touch with other group member; when speed is higher, the node moves faster and it becomes more difficult to receiving packets because the long interval between *Join Queries* and between *Packet Recovery* messages (e.g. even when one node passes by another node that has the packets it needs, they may not be able to communicate due to the longer intervals between control packets). In situation 2, both average delay and maximum delay decrease when the nodal speed increases (thought there is exception). This is because the shorter refresh time in this case: it makes the source as well as the other nodes to know the topology change quickly and take necessary actions early. Thus, when the speed is higher, the mobility of the nodes in contrast helps to promote the performance of DTCAST in terms of delays, though at the cost of higher control overhead, as we will see later.

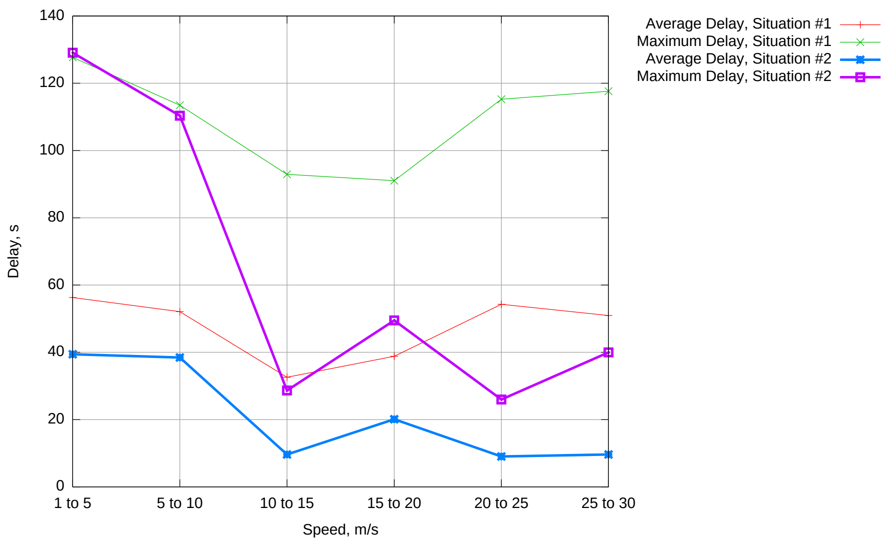

Fig. 13: Delays in two situations when nodal speed changes

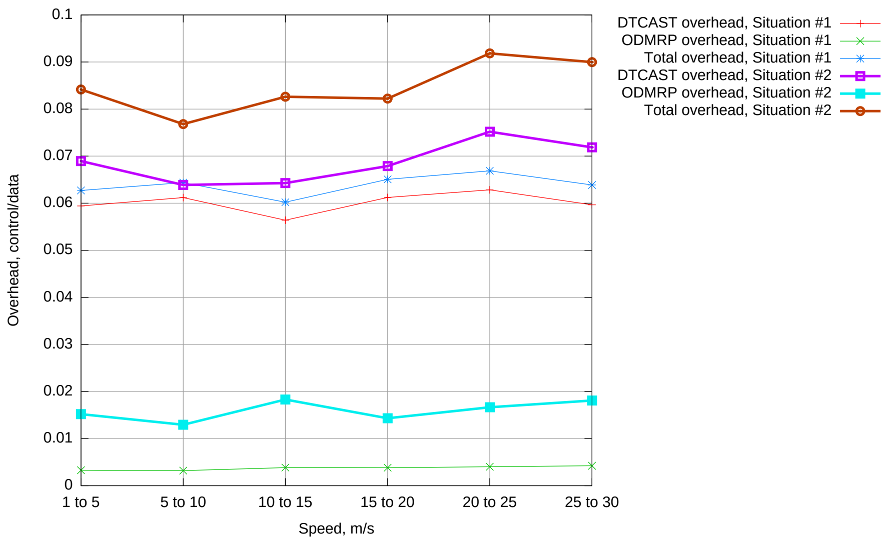

Fig. 14: Overhead in two situations when nodal speed changes

In Situation 1, control overhead is almost the same for most speed ranges, and has the smallest value of 6% when the nodal speed are chosen from 10 to 15 m/s. In situation 2, overhead introduced by DTCAST is large when nodal speed is small or large, and when the nodal speed is 5 to 15 m/s, the overhead is smallest. ODMRP overhead, in this case, is almost the same except the speed range of 10 to 15 m/s. Thus it makes the overall overhead become the smallest when the nodal speed is within 5 to 10 m/s. In both cases, the ideal speed is around 10 m/s, which is usually the speed of vehicles that are moving in a city. Due to the higher frequency of control actions, the overhead in situation 2 is 30% to 50% higher than that in situation 1.

The next simulation setup is the same as the original one described in Section IV-C except the group size. We increased the group size from 5 to 25 and analyzed the change in control overhead. As shown in Figure 17. RR stands for recovery request overhead; ER stands for epidemic routing overhead; ODMRP stands for ODMPR overhead; Overall stands for DTCAST overall overhead.

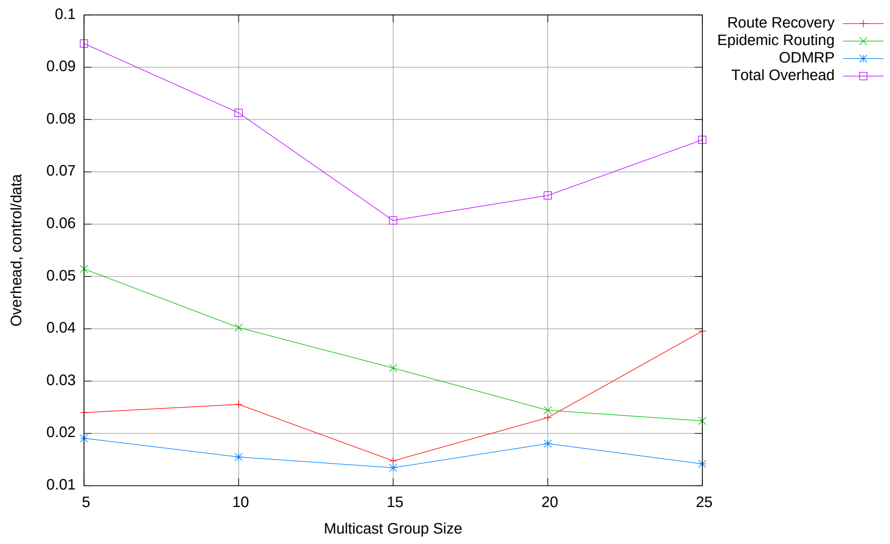

Fig. 15: Group Size and Control Overhead

When the group size grows, ER drops almost linearly, because when the node broadcast the one hop epidemic routing packets, the possibility of being received by another receiver who needs these packets grows with the group size, thus makes ER packets more efficient. RR is smallest when there are 15 members in the group. With fewer group members, one node may need to send more recovery requests to get answer from the node that has the missing packets. With more group members, more forwarding links will break and thus more nodes start recovery request. ODMRP overhead stays the same for all group size. Thus, the overall control overhead of DTCAST becomes smallest when group size is 15 in this simulation environment. And we can infer from the experiment that there is an optimal group size for a certain situation, and usually the group size is not large.

V. Conclusions

In this paper we have studied different approaches to perform multicast routing in delay and disruption tolerant ad hoc networks. We have proposed new multicasting DTCAST protocol, which combines ODMRP-like on-demand route discovery process, as well as incorporates local route injection and epidemic routing phases to increase routing and data transmission performance. Additionally, proposed protocol provide two-class data delivery service - guaranteed and best-effort data messages delivery to multicast stream subscribers.

We have implemented our protocol in various forms and performed experimental and simulation-based evaluations, which confirmed workability of designed protocol in variety of dynamic network environments. But also help to discover several weaknesses. First of all, there is small percentage of unsuccessful delivery for the first-class serviced nodes, which is probably caused by data loop-prevention heuristics mismatches in some cases and there is need additional research to close all cases in this heuristic procedure. Also, experiments show, that there is different equilibrium points for data delivery delays. All these observations are to be resolved in the ongoing research.

Additionally we have performed simulation-based evaluation of pure epidemic multicasting routing protocol. While implemented version may not be suitable for real life use, it shows that multicast traffic can be handled in delay and disruption tolerant networks (DTNs) as well as shows promise for a very robust solution if the protocol were explored further. It also shows that, in order to compete with DTCAST and other protocols, it has to be willing to be very inefficient in terms of duplicate deliveries and flooding of network resources. If you can deal with the issues that are inherent to an epidemic approach, then it would be more than possible to have a multicast solution for use in DTNs.

References

[1] Sung-Ju Lee, Mario Gerla, and Ching-Chuan Chiang, "On-demand multicast routing protocol", in *Proceedings of WCNC '99*, 1999, pp. 1298–1302.

[2] Xiaoyan Hong, Kaixin Xu, and Mario Gerla, "Scalable routing protocols for mobile ad hoc networks", *Network, IEEE*, vol. 16, no. 4, pp. 11–21, July/August 2002.

[3] Soon Y. Oh, Joon-Sang Park, and Mario Gerla, "E-ODMRP: enhanced ODMRP with motion adaptive refresh", *Wireless Communication Systems, 2005*, pp. 130–134, Sept. 2005.

[4] Jorjeta G. Jetcheva and David B. Johnson, "Adaptive demand-driven multicast routing in multi-hop wireless ad hoc networks", in *MobiHOC 2001*, Long Beach, CA, USA, 2001, pp. 33–44.

[5] V. Devarapalli and D. Sidhu, "MZR: a multicast protocol for mobile ad hoc networks", *IEEE International Conference on Communications*, vol. 3, pp. 886–891, 2001.

[6] Wenrui Zhao and Mostafa H. Ammar, "Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks", *Future Trends of Distributed Computing Systems, IEEE International Workshop*, 2003.

[7] Yang Chen, Jeonghwa Wanh, Wenrui Zhao, Mostafa Ammar, and Ellen Zegura, "Multicasting in sparse MANETs using message ferrying", in *Proc. WCNC 2006*, 2006.

[8] Amin Vahdat and David Becker, "Epidemic routing for partially-connected ad hoc networks", Tech. Rep. CS-2000-06, Duke University, Durham, NC 27708, 2000.

[9] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura, "Multicasting in delay tolerant networks: semantic models and routing algorithms", in *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, New York, NY, USA, 2005, pp. 268–275, ACM.

[10] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks", *Wireless Networks*, vol. 7, pp. 609–616, 2001.

[11] Eddie Kohler, *The Click Modular Router*, PhD thesis, Massachusetts Institute of Technology, February 2001.

[12] "Git - fast version control system", http://git.or.cz/.

[13] "The network simulator – ns-2", http://www.isi.edu/nsnam/ns/.

[14] "Original ns2 epidemic routing implementation", http://issg.cs.duke.edu/epidemic/.