Mobility-Tolerant, Efficient Multicast in Mobile Cloud Applications*

Ju Wang[1], Hui Chen[1], Kostadin Damevski[1], and Jonathan Liu[2]

[1] Virginia State University, Petersburg, VA 23806, USA
[2] University of Florida, Gainesville, FL
jwang@vsu.edu

Abstract. Interactive mobile applications require a highly available multicast service for information dissemination and collaboration, while being able to withstand mobility-induced network connectivity problems. However, efficient and reliable wireless multicast has remained a difficult challenge. We propose a novel wireless multicast scheme that allows more efficient and mobility-proof multicast in mobile cloud environments. Our scheme uses a distributed caching and deferred acknowledgement (ACK) technique to reduce delivery ACK traffic during a multicast session. Packets with pending ACK are cached in selected network nodes to provide fast re-delivery. A distributed multicast tree construction algorithm is also utilized to provide fast topology repair under dynamic network conditions. The tree maintenance requires each node to keep track of its 2-hop neighborhood connectivity. Our scheme's ability to overcome frequent network topology changes leads to a low message exchange overhead to correct local topology errors.

Keywords: multicast, mobile cloud, reliability, mobility, fail-recovery, MAC protocol, wireless network.

1 Introduction

Cloud computing with mobile devices [10–12] enables many new exciting applications that were unavailable in the past. For example, a mobile cloud could be made of smart phones to process real-time data (e.g., video and audio feeds, GPS coordinates). Other mobile cloud applications might involve realtime control/actuation (e.g., to physically turn on a ventilating fan) in wireless powered sensor networks. Such new applications need a capable and reliable communication infrastructure (both wireless and wired portions) to move large amount of data between mobile nodes in a timely manner.

One challenge in the intersection of mobile and cloud computing is the lack of mobility-tolerant middleware services. Reliable wireless multicast service would greatly enhance many mobile cloud applications. For instance, a mobile application utilizing a cloud of street video cameras can support a wide range

* This material is based upon work supported by the National Science Foundation under Award No.1040254.

of image-based searching tasks (e.g. missing children, terrorist suspects). Such applications would depend on a reliable multicast service to distribute target pictures to all cloud nodes. Other wireless-based applications can also benefit from the service tremendously. Network-wide information dissemination, such as distributing new program/firmware across a re-programmable sensor network, could be performed more efficiently.

However, multicast in wireless mobile networks is expensive and significant network resource and energy must be committed to overcome a wide range of communication problems, from device failures, short term link lost, to packet delivery failures due and node movements. The fundamental challenge of multicast reduces to the efficient construction and maintenance of a multicast tree. The problem has been studied by many researchers [1, 3, 6] for both wired and wireless networks as an instance of the Minimum Connected Dominant Set (MCDS) problem, and recent work focuses on distributed multicast protocols [7]. Distributed solutions are attractive since localized decision making is more adaptive to a changing mobile environment.

When reliable packet delivery is added to the requirement, the problem becomes even more challenging. An intuitive solution requires all target nodes to acknowledge the receipt of each multicast packet. The multicast source is forced to schedule retransmissions until all nodes confirm a packet reception. If the multicast network contains more than one level of relay structure, the ACK traffic must be relayed back to the source node through the same network (but in reverse direction). To further complicate the matter, node movements during the multicast session will cause topology change and possibly interrupt the ACK process. The ACK from a moving node might have to be relayed to its original parent node to avoid unnecessary retransmission. All scenarios considered, significant amount of network traffic would be induced to an already congested network.

We observed that multicast traffic is sessional by nature: multicasting a video frame at the application layer will result in a stream of multicast packets from the same source node. Such traffic pattern is utilized in our design to improve reliability at a moderate message cost. Our solution consists of two key techniques:

1. We use a distributed caching and deferred ACK protocol to reduce the required ACK messages while still providing reliable tracking of data delivery. Multicast packets are cached throughout the network to provide a swift retransmission of dropped packets. The readily available of large flash memory in today's mobile devices provides significant cache benefit. Specifically, we are able to save significant ACK-related messages through the deferring scheme.
2. For route construction, we use a modified Distributed Local-Gain-Maximizing (DLGM) algorithm [17] to form a decentralized multicast tree that avoids network flooding. The scheme utilizes a multicast session concept so that multicast packets belongs to the same session can be handled with efficiency by reusing the routing path of a previous packet. Our DLGM-s algorithm, short

for DLGM-with-Session, is designed to handle multicast failure caused by node movements and reuse routing decision from previous packets. The uniqueness of our algorithm is that the multicast tree is formed during the media access stage and the algorithm is executed by all nodes to determine whether or not it should relay, delay, or ignore a new multicast packet. The algorithm is inherently distributed and adaptive to topology changes. Each node's local decision takes into account a dynamic multicast-gain calculated from the local connectivity and relaying activities at nearby nodes.

The rest of this paper is organized as follows: Section 2 provides a summary of related work; Section 3 shows the architecture of the proposed scheme and its packet delivery performance; Section 4 provides a qualitative analysis of reliability against device mobility for the relevant methods; Section 5 concludes this paper.

2 Related Works

Cloud computing [8] originates from service-oriented computing where hardware and software resources are provided as services and applications are regarded as consumers. Recently, the cloud computing model is extended to utilize mobile devices (such as smart phones) as hardware farms [13] at the cloud side to augment its computing capabilities. Mobile cloud applications can leverage all of the advantages from the conventional cloud concept, such as computation offloading to data centers [11]. In [14], an Android based smart phone and the Elastic Compute Cloud service of Amazon Web Services (http://aws.amazon.com/ec2/) is used to create a mobile cloud traffic light detector application. The smart phones are used mainly as video capture devices, and Amazon's cloud receives and processes the video frames.

Reliable multicast in a traditional network requires two all-to-one transmissions per packet: the Confirm-To-Send (CTS) stage and the ACK stage. Both consume considerable network resources. Several solutions were proposed in the past to use certain delay strategies to avoid CTS/ACK collision, or use selective CTS/ACK replying to reduce the problem to a manageable scale. However, most of this work is performed within a single hop network and there is no consideration to support multicast session. We observe that many practical multicast tasks would consists of many packets from the same source. We thus believe that, instead of replying ACK immediately, a deferred ACK scheme at the session level could allow a node to reduce ACK traffic without compromising the overall network trackability.

On the routing side, many existing multicast protocols construct a multicast tree through network flooding. However, simple flooding is known for its low efficiency due to many overlapped transmissions. Traditional multicast protocols are mostly best-effort type of services without delivery confirmation to save bandwidth.

The construction of the multicast tree can be formulated as an instance of the Connected Dominant Set (CDS) problem. CDS is known as an NP-complete problem and many heuristic algorithms have been discussed [5], some of which are distributed solutions for ad-hoc networks [4, 6, 7]. The method by Wieselthier et al.[15] shows that discovering the minimum-size multi-point relay set (MSMRS) is NP-hard. Qayyum et al. proposed MRP (Multipoint Relay Protocol)[3] where each relay node must select some of its 1-hop neighbors to further relay the multicast packet. A source-initiated CDS is thus formed as the broadcast packet is relayed throughout the multipoint relays.

The algorithm discussed in [5] is based on a generalization of Chvatal's greedy algorithm for the set cover problem. The algorithm obtains a dominant set S first, then it grows the dominant set by searching S's neighbor nodes and including vertexes of the highest degree. The MCDS algorithm requires the global knowledge of the network topology, which is expensive to implement in a distributed environment. Wan [4]shows that at least $O(nlogn)$ messages are required.

Wu and et.al. [1] [2] use a two-stage algorithm that takes an opposite approach. Their algorithm initially obtains a relative redundant CDS by choosing nodes whose neighbor sets are not completely overlapped. This is followed by a prune process to eliminate as many locally redundant nodes as possible while maintaining the required connectivity. The mark process and the original two prune rules require N^2 neighbor knowledge and is a purely discrete algorithm. Wang et.al, [17] use a concurrent CTS method to reduce the CTS overhead and DLGM algorithm to construct a hidden routing tree. The work presented here could be considered as an extension of this work in multicast session scenarios.

3 Overview

We assume a hybrid ad-hoc wireless network architecture that consists of three types of nodes: (1) A central node (C-Node) for monitoring and controlling purpose; (2) a set of stationary relay nodes (R-node) whose sole purpose is to facilitate routing and packet forwarding; (3) and a dynamic set of mobile nodes (M-node) who provide application specific computing, sensing, and actuating resources needed in mobile cloud applications. The concept of R-node is commonly used in the Wireless Sensor Network (WSN) community to alleviate the packet routing problem, similar to wireless access point in WLAN or base station in cellular networks. In addition to the fixed R-nodes, any M-node could claim itself as an R-node based on its position in the topology. Compared to other work, one distinct difference of our network architecture is that M-node assume a critical role in packet routing and delivery confirmation. As will be demonstrated later, such design greatly enhances the network's ability to operate even in high node mobility situations. Figure 1 shows a mobile cloud application at our institute where the proposed network architecture will be used to create a smart high-tunnel greenhouse. The demonstration network consists of wireless sensor devices, both static and mobile, to collect crop growth information and control irrigation/ventilation/pesticide delivery.

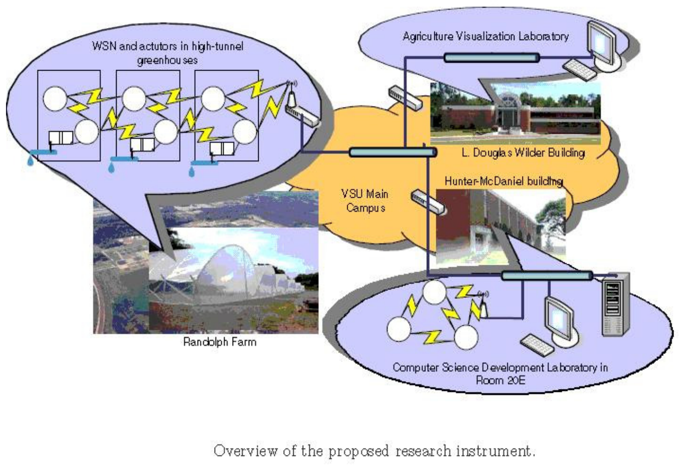
WSN and actutors in high-tunnel greenhouses
Agriculture Visualization Laboratory
L. Douglas Wilder Building
Hunter-McDaniel building
VSU Main Campus
Randolph Farm
Computer Science Development Laboratory in Room 20E
Overview of the proposed research instrument.

Fig. 1. Smart High-tunnel Mobile Cloud Demonstration Network

3.1 Problem Description

Many events can interrupt packet delivery during a multicast session: a node might be temporary out of reach, a packet might be dropped due to interference, or a node might experience hardware failure. To simplify our discussion, we exclude situations when packet delivery is physically impossible, which translates to the assumption that all nodes are connected and there is no permanent node loss. It is further assumed that all nodes are aware of the existence of some other nodes in the network and their ID, though the global network topology is unknown.

To achieve reliable multicast, the network must have a proper packet tracking ability and retransmission mechanism. Traditionally multicast packets are not cached other than in the source node, thus all packet delivery confirmations must be forwarded to the source node. If some packets need to be retransmitted, the process will start from the source node and possibly go through the entire network again even if many nodes might have already received it. In our scheme, the problem is solved by using dedicated multicast cache (M-cache) at all nodes for packet caching. Given that today's mobile device have access to gigabyte-level inexpensive flash memory for secondary storage this caching mechanism is practically achievable. The wide availability of cached copies of multicast packets is what makes the proposed scheme reliable and efficient.

3.2 Node Behavior: Deferred ACK

The rationals behind a deferred ACK scheme is the assumption that the multicast session might consist of many packets. Since a previous packet could be acknowledged through a piggy-back ACK from a later multicast packet (of the same session), it is unnecessary for all nodes to ACK for every new packet.

With carefully designed ACK and caching policy, it is possible to save much of ACK related network traffic. In the following discussion, we will use a dot notation when a specific field of the signalling packet is mentioned, e.g., RTS.seq represents the *seq* field of the RTS message.

The deferred ACK protocol is based on a modified RTS-CTS MAC layer function to regulate transmission activities in a given 2-hop cell. To relay a multicast packet, an R-node r needs to first transmit an RTS message to start acquiring the wireless channel. In a conventional multicast protocol, all nodes neighboring with r should reply with an CTS message to complete the channel-reservation process. If the R-node does not receive all CTSs, a data packet will not be transmitted to avoid collision. In our new protocol, the RTS packet will contain an *ctslist* field to specify m representative nodes for reply. If an M-node is selected, its CTS packet should be transmitted at a time slot indicated in the the corresponding RTS message. The CTS message will also function as a piggy-back ACK for its expected data packet. We hence use the term CTSACK for the modified CTS message.

After the RTS stage, the R-node will wait for $m + 2$ time slots to collect CTSACKs where the first m slots are reserved for the polled M-nodes, and the last two slots are open to any new nodes that just joined the current cell. Each CTSACK message also contains a source ID field so the R-node can update its local topology as well as the delivery status for cached packets. The R-node also interprets $CTSACK.seq + 1$ as the next expected packet of the respective M-node. If the $CTSACK.seq + 1$ is less than the sequence number of the current pending packet, an implicit retransmission requests would be entered for the corresponding packet into transmitting queue. Finally the R-node decides whether the current DATA packet should proceed, or a the RTS must be repeated.

The M-node behavior after receiving a RTS is fairly straightforward:

On Receiving an RTS

1. If the sender is not my uplink node, skip,
2. If the sender is not in my neighbor list, prepare a CTSACK message with subtype TU (topology update) and content for one of the two free CTS slots.
3. Otherwise, if the local node ID is listed in the ACK responder field of the received RTS, prepare CTSACK and transmit it at the designated slot. The current expected packet sequence number is included in the CTSACK.

For both R-node and M-node, a newly arrived multicast packet will be added to the local M-cache. For each entry in the M-cache, a node v maintains two lists: a ACK list and a CTS list to keep track of the status of its M-nodes. The ACK list contains all M-nodes that are associated to v. The CTS list will be changed if a node change its association or is not interested to a particular multicast packet. The ACK list is initialized to CTS list and updated accordingly when an CTS packet is observed. The pseudo-code in Figure 2 described the detailed protocol of the R-node to manage the multicast cache and determine its retransmission activity.

```
R-Node Protocol
  $st \in \{READY, mRTS, rRTS, DATA, IDLE$
  $T_0$ : $timer$; $recv_{count}$ : $integer$; $WSIZE$ : $integer$

      (st==READY) and (have new packet in M-cache)
         send RTS
         set $T_0 = m + 2$ $CTS$ $slot$ and start $T_0$
         $\dashrightarrow$ $st := mRTS$
      (st == mRTS) and ($T_0$ not expired) and (received a CTSACK packet)
         seq = CTSACK.seq
         if cache[seq+1] does not exist
               insert rtxUP event
         else
               insert rtxlocal event
         src = CTSACK.src
         cache[seq]->ack[src] = 1
         cache[seq]->ackcount ++
         if (cache[seq]->ackcount > 1/2 * N) mark replaceable flag
         CTScount++
         if (CTScount > 1/2 * m)
               $\dashrightarrow$ $st := DATA$
       (st==DATA)
         send data packet
         st:=IDLE
       ($T_0$ expired)
         if (CTScount < 1/2 m)
            $\dashrightarrow$ $st = rRTS$
            TUTrigger -=1
         else reset TUTrigger
      (st == rRTS)
            determine the new list of ACK[wait] list
            append the code ids in
            the MRTS packet.
            reset $T_0$ and re-send MRTS packet
            st:= mRTS
    st==IDLE & rtxlocal event
         select the oldest packet from
           cache that is marked for retranmission
         remove its retx flag
         add the packet into tranmission buffer
          st := READY
    st=IDLE & rtxUP event
         compile the missing packet ids
                 and send request to upstream.
    st=IDLE TUtrigger==0
         broadcast topology update 'hello' message
```

Fig. 2. R-node behavior. N is the size of local cell. m is the length of ACK list field. The protocol requires at least 50% of polled nodes to ACK.

3.3 Routing Behavior

Defining cell $N(v)$ as node v's neighbor set, the multicast routing problem is formulated to a MCDS problem [4, 6, 7]. Denoting graph $G(V, E)$ for the underlying network and $s \in V$ the multicast source node, the problem of seeking the optimum multicast tree is to find the smallest tree T, such that

$$\bigcup_{v \in T} N(v) = V$$

and

$$s \in T$$

We now discuss how a distributed multicast scheme would be constructed by maximizing the local multicast gain.

Since a distributed routing solution can't assume the knowledge about the global topology $G(V, E)$, we seek to maximize the number of newly covered nodes per transmission during a multicast session. The basic routing method is the DLGM (Distributed Local Gain Maximum) algorithm in [17]. The main modification here is the mechanism to reuse past routing decisions. For clarity, we summarize some of the key features of DLGM.

Unlike other routing methods where each node must maintain a list of neighbor node for local relay, DLGM rely on each nodes making individual decision based on a locally calculated gain factor. To maximize multicast coverage, the self-nominated relaying nodes should cover as many new nodes as possible. Meanwhile, the relaying nodes should be sufficiently separated each other to allow parallel relaying actions. Such design has obvious advantageous in a highly dynamic environment where the bulk of effort for routing tree update could be spared.

The DLGM algorithm requires that each nodes keep track of multicast status in its neighbor area. For each node $v \in V$, we denote its direct neighbor set by $N(v)$. At each node v, we maintain (1) $N(v)$, and (2) $N(u)$ for each $u \in N(v)$. That is, each node knows the network topology of its 2-hop neighborhood. The 2-hop neighbor set surrounding v is denoted by $N^2(v) = N(v) \bigcup_{i \in N(v)} N(i)$.

The modified DLGM algorithm will be executed by all nodes when receiving a data packet to decide whether the received packet will be relayed, delayed or discarded (the protocol behavior is based on an arbitrary node v). The algorithm utilizes previous relaying experience to accelerate decision making. For a new packet from node u, node v will examine weather the N^2 neighbor of u has been changed since last relay. If nothing has been changed, v will delay the same $d_{backoff}$ slot before MRTS. The assumption is that other nodes in $N(u)$ will make a similar decision to use their old delay slot. Since there is no conflict among the delay selection in the previous packet, the probability of no collision is high with the local topology unchanged.

Distributed Local Gain Maximizing with Session (DLGM-S)

- For each node $i \in N(v)$, define a multicast gain function $g(i)$ as the number of nodes in $N(i)$ that are not marked. Initially all nodes are not marked, thus $g(i) = |N(i)|$.

- When receiving an MRTS packet from a node u, mark all nodes in $N(u)$ as received.
- For each node $i \in (N(v) \bigcap N(u))/v$, update their $g(.)$ function accordingly. Particularly, $g(u)$ will become zero.
- If there exists a node $i \in (N(v) \bigcap N(u))/v$ such that $g(i) >= g(v)$, node v will mark a delay flag. and wait.
- wait for the data transmission from node u to complete.
- **If $N^2(u)$ topology is intact, and the past packet from u is relayed without collision**
  - *wait $d_{backoff}$ slot as determined in the previous multicast packet.*
  - *enter MRTS stage for the pending packet.*
  - *If MRTS collision, redraw $d_{backoff}$ and mark collision flag.*
- *if $N^2(u)$ changed since last packet, or there were MRTS collision in the last relay attempt*
  - *backoff a random period from $d_{backoff}$ = [1 $g(u)$] slots;*
  - *otherwise send an MRTS to relay the multicast packet.*

3.4 Protocol Analysis

In this section, we demonstrate some properties of the DLGM algorithm by proving two facts: (1) the deferred ACK algorithm is correct, in the sense that all nodes will receive the multicast packet as long as the network is connected, and (2) the DLGM-S algorithm provides the local greedy propagation.

Proposition 1: Let $G(V, E)$ be the graph representation of a wireless network under consideration, S be the multicast source node and M be the data packet; then for $\forall v \in V$, v will receive M in a finite period of time.

Proof: The proof for a stationary network is given in [17], hence we focus on the scenario where node v missed a packet after its movement or other topology change. Assume that v is associated to a new upstream node u, and u has completed broadcasting of p before v join. According to the deferred ACK protocol in Fig 2, node u employ a round robin selection for its *ctslist*, which imply that each of its neighbor, including v, will be selected to CTSACK at some time. Since node v's $CTSACK.seq$ field serve as the last received packet at node v, this gives node u a chance to lookup its local cache and schedule a future retransmission (see the *rtxlocal* and *rtxUp* state in Fig 2. If the local cache in u does not contain such packet (due to replacement), the retransmission request will be passed up to node u's upstream until the request can be served. ■

Proposition 2: Let node S be the multicast source node and M the data packet. Let $N(S)_1$ be S's neighbor and $N(S)_2$ be S's two-hop neighbor set. After S sends out M, the next relaying node r must have the largest degree in $N(S)_2 - N(S)_1$.

Proof: Since the extension in DLGM-S is concentrated on the determination of waiting slot. It inherit most of the features in DLGM, including the logic to determine local relay behavior. In particular, each node still must track the delivery status of its neighbors, thus the proof in [17] still apply. A new cases need to be addressed where $N(u)$ is changed at the time of relay decision at node

v. This could be argued in a similar fashion as in [17] under the assumption that the join of a new node needs to be broadcasted from node u, thus all exisiting neighbor of u will learn the topology change at the same time. ■

3.5 Performance Evaluation

The performance of the proposed protocol stack is evaluated by simulations. The goal is to observe and compare the performance of the proposed scheme under different network configurations such as network size, density, and mobility.

As in [17]. our simulations consider a 1000*1000 square meters area. To exclude noise of extreme topology, the testing network is generated such that a specified network size and node density requirement are satisfied. To determine the connectivity of a generated network, a uniform radio transmission range $R \in [50100]$ *meters* is assumed. Any pair of nodes is connected if their distance does not exceed R.

3.6 Multicast Cost

We are first interested in the multicast transmissions attempts during a multicast session in a static setting. Note the transmission number T_x does not include the RTS/CTSACK signaling packets.

We increase the number of network nodes m from 10 to 150 in the simulated area. Randomly generated topologies are used to test relevant multicast protocols including MPR and the DLGM/CTSACK methods.

Figure 3.(a) shows the overall multicast transmit time for networks of different size. The average connectivity is fixed as $D = 4$. The size of *ctslist* is denoted by the variable *beta*. We set p to 0.75 and 0.4 to represent two relaying behaviors: with $beta = 0.75$ representing a relatively conservative protocol where the R-node demand 75% of its neighbors to reply. $beta = 0.4$ being a moderate aggressive protocol where the R-node only require 40% of its neighbors to transmit data packet. Our DLGM with $p = 1$ achieves another 5% reduction in the total delay time. The size of *ctslist* offers an effective control of protocol behavior over large dynamic range. A small *beta* value allow a R-node to quickly enter to data transmission stage, but might suffer increased probability of collision since other R-nodes might also have collected its needed CTS votes. With $beta = 1$ the most stringent media reservation policy is enforced, and the collision is minimized.

3.7 Transmission Energy Expenditure

To study the protocol behavior over different network topologies, we take a close look at the transmission number T_x, which has strong correlation to the total energy expenditure in a multicast session and how fast multicast can be done. Figure 4 shows T_x of the proposed scheme for two network topologies of different node density. One topology has a node density of $D = 4$ and contains $m = 30$ nodes, and the second topology has $D = 4$ and $m = 60$. The result set for

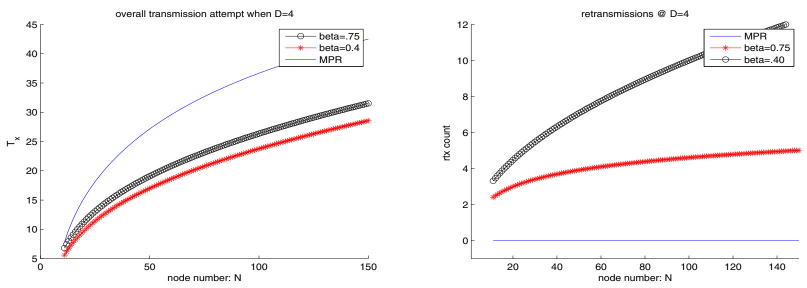

Fig. 3. transmission cost (a)tx attempt for D=4 ,(b) rtx for D=4

each case represents 500 randomly generated network topologies of the specified average network density.

Figure 4.(a) shows T_x-vs-topology for the first result set. It is observed that the performance of the multicast protocol varies significantly from topology to topology even when the total node number and degree are the same. The lowest transmission cost is observed at topology #27 with T_x as three time slots. This best-case topology is isomorphic to a degree 4 complete tree. It is worth noticing that $T_x = 3$ is the lowest transmission cost achievable for a network with $d = 4, m = 30$ (since $2 < log_4(30) < 3$). The longest transmission delay observed in this set of simulations is 20 time slots. Topologies with long delay usually have a long single chain and a cluster of nodes forming a clique. The existence of clique increases the average node degree, while the long single chain causes the long multicast time.

Figure 4.(b) shows the distribution of T_x. The observed distribution approximates a well-defined Gaussian function with a mean multicast time $\overline{T_x}$ of 12. Figure 4.(c) and (d) show the results for networks with 60 nodes ($N = 60$). The corresponding best scenario result is 7 time slots, and the worst case result is 36 time slots. A similar Gaussian distribution is observed.

4 Fault Recovery and Handoff Processing

Node failures/mobility might cause packet delivery problems to an extended network portion for ongoing and consequent multicast sessions. As shown in [17], packet loss could be caused by the poor handling of node movement even the network is physically connected. Ramani et.al, [16] reported that the break-and-reconnect period of 802.11 networks in infrastructure mode could be several seconds. Large portion of the delay is due to DHCP exchanges between the moving node and the new R-node. During the handoff period, a node will be unable to receive any packets (the decoding of overheard packet require the knowledge of the encryption key).

A more complicated scenario is that an R-node itself goes through a handoff process. As a relay vehicle moves away from its own gateway, the uplink channel quality will degrade and eventually the node needs to re-establish a connection

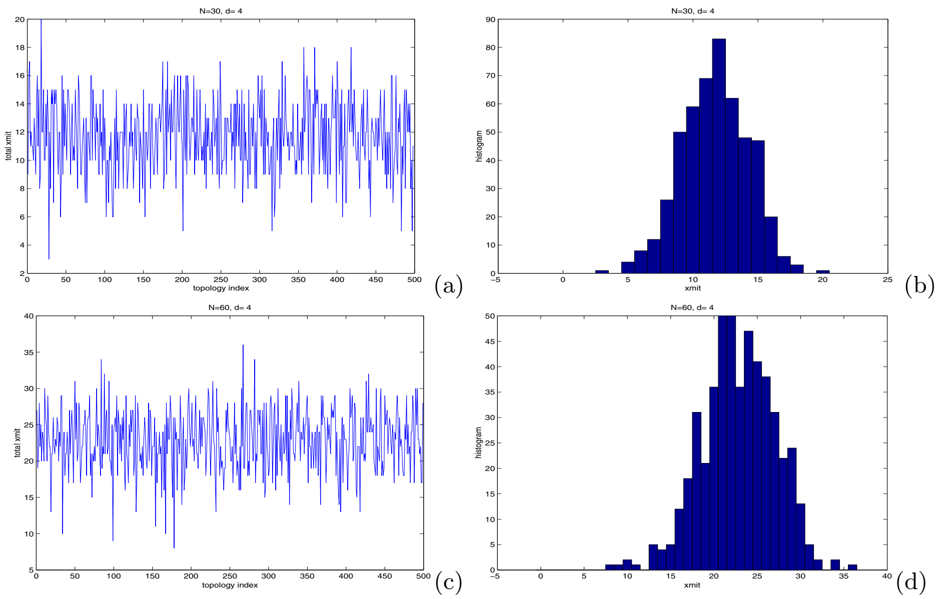

Fig. 4. (a) transmission attempts for D=4, m=30; (b)histogram of transmission time for D=4, m=30; (c) transmission attempts for D=4, m=60;(b) histogram of transmission time for D=4, m=60;

to a new gateway. This might force all downstream vehicles to execute a secondary handoff procedure. The aggregated delay will be far too long for certain applications, such as unmanned aerial vehicle control.

For simplicity, our analysis only consider the topology changes caused by the movement of one node. The node detected the topology change is denoted as node x, and the node that causing the topology change is denoted as y. To keep the multicast tree intact, the recovery protocol must satisfy two conditions: (1) the current 2-hop neighbor of node x must be reconnected, and (2) node y must be covered by some other nodes.

4.1 Recovery Analysis for MPR

In the worst case, MPR protocol would have to recalculate the entire MPR set whenever there is a topology change. Thus theoretically MPR is capable of self-healing to topology change and fault-tolerant. The retransmission policy could be integrated to MPR for dropped packets.

When node y moves out of its current position, all nodes that use y as MPR relaying node must update their MPR set, including x. These nodes will solicit to their current neighbor to obtained new 2-hop information. They will then execute the MPR algorithm to establish a new local MPR set to assure connectivity. This procedure requires a message cost of $D.M^2$ where D is the maximum node degree and M is the maximum size of 2-hop neighbor.

To reconnect y to a new R-node, y will broadcast a "request-to-join" packet to its new neighbors. All nodes that receive this packet will execute the MPR algorithm to include y in their multicast tree, which will count for another $D.M^2$ message exchange.

4.2 Recovery Analysis for DLGM-s

The handoff procedure for DLGM is briefly described in [17], here we reexamine the basic mechanism and provide some recent results. Still assuming that node y leaves its current neighbor, the method in [17] could not fix the topology error until some packet delivery failures in y's neighbors. This is because the DLGM algorithm does not specify a fixed R-node among mobile nodes, hence there is no fixed upstream node for any nodes. According to RTS/CTSACK signaling procedure, y's upstream node x will find that node y fails to respond to the MRTS packet and thus can determine y's absence. However with the deferred ACK and reduced *ctslist* in the R-node, it would take several packets drop before x realized that y is missing. A improvised procedure allow y's neighbor to act at the earliest time to reduce the packet drops in the transient period:

- let node $z \in N(x)$ share a non-empty neighbor as y: $N(y) \bigcap N(z) \neq \emptyset$.
- If $\exists w \in N(y) \cap N(z)$ such that $seq(w) < seq(local) + 1$, node z can presumably decide x's absence.
- x will remove y from $N(z)$.
- X broadcast y's absence.
- For each node u in $N(x)$, y will be removed from the $N()$ of u's neighbor $N(u)$.

The above procedure will generate exactly $M(x)$ messages where $M(x)$ is x's degree. The $M(x)$ messages will trigger updating process in the original $N^2(x)$ neighbor. It can be shown that the N^2 information around x's two-hop neighbors will remains consistent after this procedure. Compare to the procedure in [17], the topology repair occurs almost immediately after a node left. Re-connecting y to its new neighbors remain the same. This stage of updating requires $2 * M^2 + M + 1$ packets. Thus the overall message complexity is still in the order of $O(M^2)$ per single node topology change.

5 Conclusion

We proposed a completely distributed, reliable MAC-layer algorithm for wireless multicast. The uniqueness of our method is that each node decides its behavior (to relay a message or not) based on the realtime neighborhood status. Our simulations show that this method is more efficient than other schemes especially when the network topology is highly dynamic.

References

1. Wu, J.: Dominating-Set-Based Routing in Ad Hoc Wireless Networks with Unidirectional Links. Trans. Parallel and Distributed Systems 13(9), 866–881 (2002)
2. Dai, F., Wu, J.: An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks. IEEE Trans. Parallel and Distributed Systems 15(10), 908–920 (2004)
3. Qayyum, A., Viennot, L., Laouiti, A.: Multipoint relaying for flooding broadcast messages in mobile wireless networks. In: Proceedings of the Hawaii International Conference on System Sciences (HICSS 2002) (January 2002)
4. Wan, P.-J., Alzoubi, K.M., Frieder, O.: Distributed construction of connected dominating set in wireless ad hoc networks. In: Proceedings of Infocom 2002 (2002)
5. Guha, S., Khuller, S.: Approximation Algorithms for Connected Dominating Sets. Algorithmica 20(4), 374–387 (1998)
6. Alzoubi, K.M., Wan, P.-J., Frieder, O.: Distributed Heuristics for Connected Dominating Set in Wireless Ad Hoc Networks. IEEE ComSoc/KICS Journal on Communication Networks 4(1), 22–29 (2002)
7. Stojmenovic, I., Seddigh, M., Zunic, J.: Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. In: Proc. IEEE Hawaii Int. Conf. on System Sciences (January 2001)
8. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: Above the clouds: A berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28 (2009)
9. Pering, T., Want, R., Rosario, B., Sud, S., Lyons, K.: Enabling Pervasive Collaboration with Platform Composition. In: Tokuda, H., Beigl, M., Friday, A., Brush, A.J.B., Tobe, Y. (eds.) Pervasive 2009. LNCS, vol. 5538, pp. 184–201. Springer, Heidelberg (2009)
10. Lyons, K., Pering, T., Rosario, B., Sud, S., Want, R.: Multi-display Composition: Supporting Display Sharing for Collocated Mobile Devices. In: Gross, T., Gulliksen, J., Kotzé, P., Oestreicher, L., Palanque, P., Prates, R.O., Winckler, M. (eds.) INTERACT 2009, Part I. LNCS, vol. 5726, pp. 758–771. Springer, Heidelberg (2009)
11. Li, X., Zhang, H., Zhang, Y.: Deploying Mobile Computation in Cloud Service. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom. LNCS, vol. 5931, pp. 301–311. Springer, Heidelberg (2009)
12. Chun, B., Maniatis, P.: Augmented Smartphone Applications Through Clone Cloud Execution. In: Proceedings of USENIX HotOS XII (2009)
13. Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., Jeong, S.: Securing elastic applications on mobile devices for cloud computing. In: Proceedings of the ACM Workshop on Cloud Computing Security, pp. 127–134 (2009)
14. Angin, P., Bhargava, B., Helal, S.: A Mobile-Cloud Collaborative Traffic Lights Detector for Blind Navigation. In: Eleventh International Conference on Mobile Data Management, pp. 396–401 (2010)
15. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Algorithms for Energy-Efficient Multicasting in Static Ad Hoc Wireless Networks. Mobile Networks and Applications (MONET) 6(3), 251–263 (2001)
16. Ramani, S., Savage, S.: Syncscan: Practical Fast Handoff for 802.11 Infrastructure Networks. In: Proc. of IEEE7 INFOCOM (March 2005)
17. Wang, J., Wang, X.: An energy-efficient,distributed wireless multicast protocol based on concurrent CTS and N^2 connectivity. Wireless Network 16, 2031–2048 (2010)