

On the Route Discovery Latency of Wireless Mesh Networks

Chunhui Zhu, Myung J. Lee, Tarek Saadawi
Electrical Engineering Dept, Engineering School
City University of New York, CCNY
New York, United States of America
E-mail: {zhuc, lee, saadawi}@ccny.cuny.edu

Abstract—Designed and developed for military communications in their early stage, multi-hop wireless ad hoc network is attracting interest from commercial applications and becoming a research hot spot. Although the technology has been studied for about a decade, there still exist some problems to be solved. Route discovery latency for reactive routing protocols is one of them. In this paper, we propose a dynamic RTT-based algorithm which can dramatically reduce the route discovery latency. This algorithm can be generally applied to any source selection based reactive routing protocols. Our simulation study has also shown some interesting results about packet transmission time and route discovery latency in a contention-based channel like IEEE 802.11 which have not been discussed before.

Keywords—wireless communications; ad hoc network; mesh network; routing algorithm; route discovery latency

I. INTRODUCTION

Traditional commercial wireless networks are infrastructure-based. They usually consist of powerful, static base stations and less powerful, mobile user terminals. Communications between mobile users must go through base stations. About 10 years ago, a new infrastructure-less wireless communication model, wireless ad hoc network, started being studied. This is typically a multi-hop, peer-to-peer network which does not rely on any pre-deployed infrastructures, like base stations. The initial purpose of wireless ad hoc network was to make possible the wireless communication in the battle fields where communication infrastructures do not exist or difficult to deploy due to the hostile environment.

With the studies progressed, researchers have found many desirable features of ad hoc networks which could benefit the commercial applications. Besides of the infrastructure-free characteristic, other desirable features include flexible coverage (via extending the communication to multiple hops), efficient frequency reuse and low power consumption (a single node does not need to cover a large area), reliable data transmission (via potential multiple paths between source and destination) and etc. As a matter of fact, both IEEE WLAN (Wireless Local Area Network) and WPAN (Wireless Personal Area Network) working groups have established their own task groups, IEEE 802.11s and IEEE 802.15.5, to develop the standards for wireless mesh network, which is an alias of wireless ad hoc network in the industry. While 802.11s tries to extend the coverage of successful Wireless LAN, 802.15.5 attempts to

launch a whole new market of wireless sensor networks and personal networks through the power of mesh network.

Because nodes are usually multiple hops away from each other, routing protocols are usually needed for a source to find a route to the destination before it can send any data to the destination. There are a bunch of routing protocols have been proposed for wireless ad hoc networks so far. These routing protocols generally fall into two categories, proactive protocols and reactive protocols. Typically, proactive routing protocols discover and maintain the routes to all destinations by periodically exchanging neighbor or link state information among participating nodes. Reactive protocols, on the other hand, discover and maintain the routes on-demand, which means the protocols discover the route to a specific destination only when this route is requested.

One of the advantages (probably the biggest) of proactive protocols is that whenever a route is requested, it is immediately available. However, because of periodical information exchange, proactive protocols consume lots of network resources (e.g. bandwidth and power), generate heavy control overheads and may lead to packet collisions (if operate over a shared channel like IEEE 802.11). In contrast to proactive protocols, reactive protocols avoid these drawbacks by finding the routes on demand. By so doing, periodical message exchange is saved and the control overheads are greatly reduced. However, reactive routing protocols have their own problems. The biggest problem is their long route discovery latency.

When a data packet arrives at the routing layer of a source node running reactive protocol and the route is not currently available, the source node has to first buffer the data packet and, at the same time, start finding the route to the destination. Depending on the distance between the source node and the destination node, this route discovery process may take different amount of time. The route discovery period may be large enough to cause the continuously coming data packet to be dropped from the source node's limited sending buffer. This will be extremely true when the data rate is high and the nodes are compact devices, e.g. PDAs, which have limited amount of memory.

In this paper, we investigate the reasons of route discovery latency of reactive routing protocols and propose our RTT-based dynamic algorithm that can dramatically reduce this latency.

II. ANALYSIS OF ROUTE DISCOVERY LATENCY

Many processes and mechanisms contribute to the route discovery latency. Among them are, packet transmission time (determined by the speed of the network interface, e.g. 11Mbps for 802.11b), packet processing time at intermediate nodes and destination (determined by processor speed), network congestion status, the length of the route (number of hops) the route discovery packets take and etc.

When reactive routing protocols are used, a source usually initiates a Route REQuest packet (RREQ) when it needs a route to the destination but does not have one. This RREQ packet is broadcast to the whole network (sometimes via expanding ring search). Intermediate nodes re-broadcast the RREQ packet if it has not seen the packet before or the packet contains shorter route to the source. When the destination (sometimes intermediate nodes with fresh routes to the destination) receives the RREQ, it will send back (via unicast) a Route REPLY (RREP) to the source. Multiple RREQs reached the destination via different routes could lead to multiple RREPs being sent back to the source.

Even though belonging to the same reactive category, reactive protocols use different mechanisms to select the best route from multiple available routes. Generally, there are two kinds of route selection mechanisms. We refer the first kind as “source selection” because, in this case, the initiator of the RREQ selects the best route from multiple RREPs. Protocols belong to this category include AODV[2], TORA[6] and etc. The second kind is referred as “destination selection”. In this case, the destination of the RREQ selects the best route based on some predefined criteria and then sends the RREP back to the source along the route just found. Here the destination of the route request could be either the final destination or some times the intermediate nodes with fresh enough route to the final destinations. We refer both of them as “destination” hereafter. ABR[7] and SSR[8] fall into this category. We analyze the route discovery latency in the rest of this section.

When source selection protocols are used, a source usually starts a timer, T_{wait} , after sending out a RREQ, to wait for the RREPs. This T_{wait} is actually the total route discovery latency and can be described by the following formula,

$$T_{wait} = (T_{request} + T_{reply}) + T_{soft} \quad (1)$$

where $T_{request}$ is the time it takes for the first RREQ traverses from the source to the destination; T_{reply} is the time it takes for the first RREP (or the only RREP, when the protocol is destination selection based) traverses from the destination back to the source; T_{soft} is the extra waiting time after receiving the first RREQ or RREP, namely, the “soft” latency. For source selection protocols, T_{soft} happens at the source side and after T_{reply} ; for the destination selection protocols, T_{soft} happens at the destination side and before T_{reply} . In addition, note the “hard” latency, T_{hard} , is the sum of $T_{request}$ and T_{reply} . It is so named in this paper because this part of the latency is beyond the control of any routing protocols.

Before introducing our algorithm, we like to review how current reactive routing protocols deal with this problem. More specifically, we would like to see how they determine the value of T_{wait} . The simplest way is using a fixed T_{wait} just like what

DSR [4, 5] does. Obviously, a fixed timer is too simple to suit various network conditions. If the source and the destination are very close to each other, T_{wait} may be much larger than the necessary response time. If the source and the destination are far away from each other, then T_{wait} may not be large enough to collect all possible replies safely. Therefore, using a fixed T_{wait} is either a waste or unsafe. In other words, the protocols using fixed T_{wait} are not scalable. Protocols like AODV use a more advanced mechanism than fixed scheme. Since an expanding ring search approach is used to limit the flooding effect in AODV, T_{wait} in this protocol is associated with the TTL of each route request attempt. That is, when the TTL of the route request increases, the T_{wait} also increases proportionally. However, this mechanism, although makes the T_{wait} more adaptive, still cannot reduce the T_{soft} for each route request-and-reply cycle. In addition, our simulation study shows that the packet traversal time in a contention-based network does not always increase as hop count increases. It is mainly determined by the network traffic pattern and congestion situation. Therefore, increasing waiting time for larger TTL does not make sense in this situation.

The purpose of our algorithm is to reduce T_{soft} to the greatest extent yet can still guarantee the protocol will not miss the best route. We believe the hard latency, T_{hard} , provides very important information that should not be neglected. The T_{soft} must be a function of T_{hard} to avoid unnecessary latency because T_{hard} measures the real distance (in time) between the source and the destination.

III. OUR ALGORITHM AND ITS APPLICATION

The following describes our RTT -based algorithm of determining the necessary actual route reply waiting time. In this first stage of study, only the distance in hop count is considered as the route selection criterion. Therefore, when one route has less number of hops than another to the destination, it is considered better than another.

A. The algorithm

When a source node initiates a RREQ, it starts a timer called $T_{wait_scheduled}$ to wait for the RREPs. The value of this initial timer is estimated based on some sort of adaptive algorithm like the one used in AODV (which will be described later). $T_{wait_scheduled}$ must have a “safe” initial value because we do not want to miss the RREPs if they are under the coverage of the TTL of the RREQ. At the same time, the source records the sending time of this RREQ, T_1 .

Upon receiving the first RREP, the source node again records the receiving time, T_2 . The time difference between T_2 and T_1 is then the RTT (Round Trip Time) of the fastest RREQ/RREP pair. Please note that this RTT is what we called T_{hard} , which is not under our control.

$$T_{hard} = RTT = T_2 - T_1 \quad (2)$$

Now that we have the first RREP at hand, we would like to compare its RTT with the previously scheduled timer $T_{wait_scheduled}$. A big difference between them would reveal that our initial estimation is not accurate; hence, the actual waiting time should be adjusted to adapt to the network situation. We

propose the following algorithm to recalculate the actual waiting time a source node would like to spend.

$$T_{wait_actual} = (1 + \alpha) RTT \quad (3)$$

where α is a constant between 1 and 3. Our recommended value for α is 2. The selection of α is discussed later in the simulation analysis section.

Now we compare T_{wait_actual} with $T_{wait_scheduled}$:

If T_{wait_actual} is smaller than $T_{wait_scheduled}$, then instead of waiting until pre-scheduled timer $T_{wait_scheduled}$ to expire, the source node waits only T_{wait_rest} from this moment, which

$$T_{wait_rest} = T_{wait_actual} - RTT = \alpha * RTT \quad (4)$$

In addition, if an RREP with smaller hop count is received during this extra waiting period, the source node will take the route reported by the later RREP even without waiting T_{wait_rest} to expire. Again, the reason will be discussed later in the simulation analysis section.

If, however, T_{wait_actual} is greater than $T_{wait_scheduled}$, then the original $T_{wait_scheduled}$ will be used and T_{wait_actual} will be disregarded.

As a summary, after receiving the first RREP, the source node calculates the actual time it is going to wait for more RREPs using formula (3), and compare the result with the pre-scheduled waiting time. It will then choose to use the smaller one of these two and wait another T_{wait_rest} period given by (5) from this moment.

$$\begin{aligned} T_{wait_rest} &= \alpha * RTT && (T_{wait_actual} < T_{wait_scheduled}) \\ &= T_{wait_scheduled} - RTT && (T_{wait_actual} \geq T_{wait_scheduled}) \end{aligned} \quad (5)$$

where $1 < \alpha \leq 3$.

B. Example application

Although our algorithm can be applied to any reactive source-selection ad hoc routing protocols, we select Multicast Ad hoc On-demand Distance Vector routing protocol (MAODV) as our benchmark protocol. MAODV and its unicast brother AODV share the same mechanism of determining the route reply waiting time. Therefore, we use the word MAODV and AODV alternatively in this paper.

In AODV, after sending out a RREQ packet, the source waits for RING_TRAVERSAL_TIME to collect all possible RREPs. Specifically,

$$\begin{aligned} \text{RING_TRAVERSAL_TIME} &= 2 * \text{NODE_TRAVERSAL_TIME} && (6) \\ &*(\text{TTL_VALUE} + \text{TIMEOUT_BUFFER}) \end{aligned}$$

where NODE_TRAVERSAL_TIME is the time it takes for a packet be transmitted from one node to its direct neighbor. Its recommended value is 40 milliseconds. The TIMEOUT_BUFFER is designed to provide extra cushion for timeout so that the timer will not expire if a RREP is on its way to the source but suffers from network congestions. This parameter can be omitted by setting its value to zero while the recommended value is 2 second. Note here our $T_{wait_scheduled}$ equals to the RING_TRAVERSAL_TIME.

For a specific network, both NODE_TRAVERSAL_TIME and TIMEOUT_BUFFER are constant. Therefore, RING_TRAVERSAL_TIME will only adapt to TTL_VALUE. However, for each TTL value (that is, each RREQ/RREP cycle), the RING_TRAVERSAL_TIME is also a fixed value.

When the timer ($T_{wait_scheduled}$) expires, the source checks whether it has found the route. If it has, it will start using the route to route packets; if it has not, it will increase the TTL_VALUE by TTL_INCREMENT and starts another route-searching attempt using the new TTL value. This expanding ring search process continues until the NET_DIAMETER is reached.

As a conclusion, AODV uses the source selection mechanism and an adaptive approach to calculate the reply waiting time for different TTL values.

IV. SIMULATION DESIGN AND ENVIRONMENT

We implemented our algorithm using OPNET version 8.0. The simulated network contains 50 nodes. These nodes are initially placed randomly within a fixed-size L x L area. The communication radius of the nodes is set to 100 meters. A random way point mobility model is used. Six moving speeds ranging from 0 m/s to 1m/s are simulated. The MAC layer model is the OPNET implementation of IEEE 802.11b WLAN model. The speed of the Wireless LAN is set to 1Mbits/sec.

In this study, we only examine the multicast performance of the protocol. Therefore, there is only multicast traffic in the network. Nodes join the group within a short time period (0~60 second, we refer this as *joining period* hereafter) after the simulation starts. This simulates the scenario that after a multicast session is announced, nodes join the group to participate in the session. Nodes can choose to generate data packets to the session after the joining period or they can just join to listen. At the time nodes join the group, they also schedule a random time to leave the group. Data packets are sent by both group members and non-members to the group at randomly selected time throughout the simulation. Data packets are 64 bytes in length with constant bit rate and the number of data packets transmitted per session is a geometric distribution with average value 3,000.

We tested our algorithm in two different scenarios. In the first scenario, 50 nodes are placed in an area of size 500m x 500m. An area of this size provides high enough node density so that almost every node can reach all other nodes in one or multiple hops. Simulations show that there are few, if any, partitions of the multicast tree throughout the entire simulation process. In the second scenario, the area is increased to 850m x 850m. All other environment parameters remain the same. In an area of this size, node density is much lower than the previous one (the area is about 3 times of the previous one). This area size allows the nodes to be partitioned into several small groups.

For each scenario, 10 simulations were run with different seeds. All simulations were run for 300 seconds. The results shown below are the average values of 10 simulation runs for each moving speed and scenario.

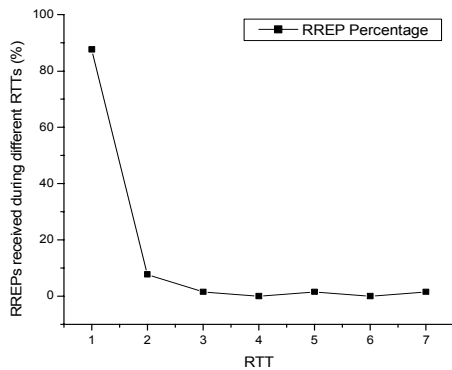


Figure 1. The percentage the better RREPs fall into different *RTT* slots

V. SIMULATION RESULTS AND ANALYSIS

In our study, simulations were run for two general purposes; first, to explore the delay characteristics of route discovery and determine the key parameters like α ; second, to examine the performance of our algorithm.

A. Delay characteristics and α

In order to determine the value of α , we initially set its value to 10. This means that after receiving the first RREP, the source waits 10 times of the *RTT* of the first RREQ/RREP pair to see whether there will be better routes to arrive. With this setting, we are safe to say that we will not miss any better routes. To describe the problem better, we chop the extra waiting time after the first RREP into ten *RTT* slots. In Figure 1, we plot the percentage distribution that better routes fall into different the *RTT* slots. For example, 87.69% of the best routes arrive during the first *RTT* period after the source receives the first RREP. If we sum the first two histograms, we will notice that 95.38% of the best routes arrive within the first two *RTTs* after the first RREP is received. This is the reason we recommend the value of α as 2. The result of Figure 1 is a summary of two node densities at six different moving speeds.

Normally, a source will receive multiple route replies after sending out a RREQ. Therefore, it is also interesting to see which RREP is the best one – the first RREP, the second RREP, and so on. To explore this, we count the number of the first RREPs. If no RREPs better than the first RREPs are received during the extra 10 *RTT* periods, the first RREPs will be considered as the best ones. If a better route is received after the first RREP (“first better RREP” hereafter), it will be considered the best one unless an even better route is received. Surprisingly, we observed that after receiving the first better RREPs, there are not any more routes received are even better than the “first better RREP”s.

Figure 2 plots the number of first RREPs and the first better RREPs for both dense and sparse network situations at six different speeds. The two solid lines represent the percentages that the first RREPs are the best ones (90.15% on average) and the first better RREPs are the best ones (9.85%) separately. As we mentioned above, these two percentages add up to 100% of the best routes. In other words, either the first RREP or the first

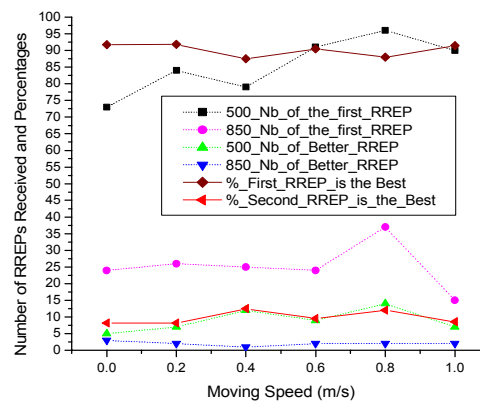


Figure 2. Number of the first RREPs and the first better RREPs and their percentages to be the best routes

“first better RREP” is the best one. No other routes will be even better. Therefore, in our algorithm, after a source receives the “first better RREP”, it will stop waiting for more RREPs because there will not be any of this kind based on our simulation result.

Some protocols simply pick the first arrived RREP and believe it is the best route. Our simulation shows even though only about 10% later arrived RREPs are better than the first ones, they do shorten the route (in hop counts) by fair high percentages (30%~50%). This indicates that it is worthwhile to wait for the better routes in most of the cases.

B. Performance examination

In this sub-section, we are going to compare the performance of our *RTT*-based algorithm with AODV’s algorithm. In order to calculate the estimated waiting time for AODV, we set the `NODE_TRAVERSAL_TIME` to its default value, 40 ms. The `TIMEOUT_BUFFER` is set to 0. The `TTL_VALUE` in (6) starts with initial value 2 and increases by 2 every time when the searching range increases.

We first compare the overall waiting time of AODV’s algorithm and that of our *RTT*-based algorithm to see how much we can improve. The waiting times of different moving speeds for both scenarios are compared in Figure 3. Simulations show that our algorithm’s overall waiting times are about 5.15% of that of AODV’s. Specifically, it is about 3.9% for sparse network scenarios and 6.4% for dense network scenarios. Because AODV’s algorithm is based on estimation (which depends on the distance in hops between the source and the destination) and our algorithm is based on measurement, our algorithm adapts better than AODV to different network densities and congestion conditions.

For a detailed comparison, we would like to see how much our algorithm could improve for different distance (hops). This result is plotted in Figure 4 using a logarithmic scale for the waiting time.

The first observation from Figure 4 is that while the waiting time of AODV’s algorithm increases with the distance between the source and destination nodes, our *RTT*-based waiting time

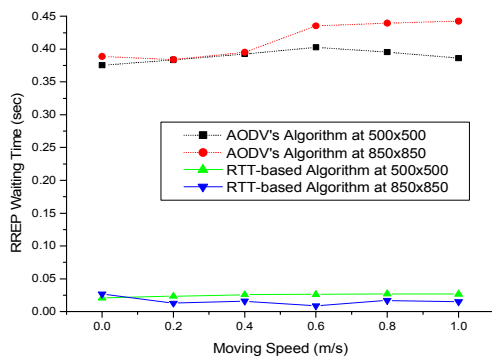


Figure 3. Comparison of waiting time

does not. This is especially true in the case of a sparse network. In the dense scenario, we see our waiting time goes higher with the increase of hop count up to five hops and goes lower beyond five hops. In the sparse scenario, the waiting time goes down with the increase of hop count beyond three hops. Two conclusions can be drawn from this observation. First, in general, the larger the hop counts are, the more waiting time our algorithm can save; second, the assumption that packet traverse time increases with distance is not true in a contention-based channel like IEEE 802.11, especially in a sparse network. Therefore, the AODV's algorithm may not be meaningful. This result can be further explained in the second observation of this figure.

The second observation we have is that our *RTT*-based waiting time in the dense scenario is larger than that in the sparse scenario for the same hop count. This stems from the fact that the denser the network is, the more contentions the flooding mechanism generates when sending RREQs. When contentions are detected at the MAC layer, nodes have to backoff exponentially before re-transmitting the packets. This leads to longer delay in a dense network than that in a sparse network. The contentions happen when different nodes try to rebroadcast the RREQ they just received at a highly correlative time or when some nodes try to reply while others try to rebroadcast at the same time. In a sparse network, the chance of this kind of contention is less than that in a dense network, and hence packets may experience less traverse delay between the source and the destination. This observation also explains the first observation; when the destination is far away from the source, the effect of flooding is less severe so that the packets can travel faster. This phenomenon is referred as the “broadcast storm”. Our work in [9] discussed the problem and provided a highly efficient solution to the problem.

VI. SIMULATION RESULTS AND ANALYSIS

In this paper, we have discussed the problem of route discovery latency of reactive ad hoc routing protocols. To avoid the “blind” estimation of the route discovery time, our algorithm measures the *RTT* of the first RREQ/RREP pair and uses it as a rule to measure the quality of later received route replies. We showed by simulation that 95% of the better route replies would arrive during the first two *RTT* periods after the source node receives the first RREP. In addition, either the first RREP

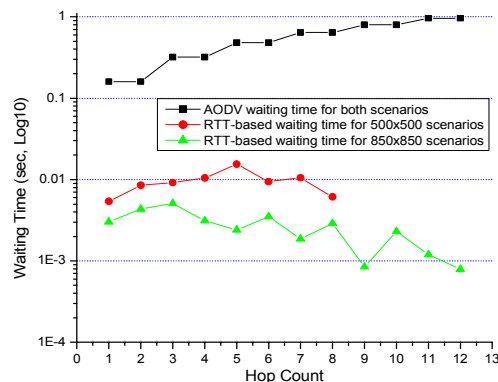


Figure 4. Waiting time comparison for different node distance

or the “first better RREP” carries the best route; not other routes will be the best route. Our algorithm waits up to three *RTTs* (if a better route is not received) or less (if a better route is received) to determine the best route to the destination. The simulation results also showed that the primary reason for route discovery delay is the backoff time of the contention led by the flooding of RREQs. This delay does not always grow as the distance between the source and the destination increases. By comparing with one of the reactive routing protocols, we showed our algorithm is much better than the “blind estimation” algorithms and can adapt to different network densities. Another advantage of our algorithm is its simplicity and, hence, scalability. Only local clock information is needed to calculate the dynamic waiting time. Very little processing power is needed for the source node. The destination node does not need to involve in the process.

ACKNOWLEDGMENT

This work is supported by Samsung Advanced Institute of Technology.

REFERENCES

- [1] Elizabeth M. Royer, Charles E. Perkins, *Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing (draft-ietf-manet-maodv-00.txt)*
- [2] Elizabeth M. Royer, Charles E. Perkins and Samir R. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing (RFC 3561)*
- [3] Elizabeth M. Royer, Charles E. Perkins, *Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol*
- [4] David B. Johnson, Davis A. Maltz, *Dynamic Source Routing in Ad Hoc Networks*, Mobile Computing, T. Imielinski and H. Korth, Eds., Kulwer, 1996, pp. 152-81.
- [5] David B. Johnson, Davis A. Maltz, Yih-Chun Hu, Jorjeta G. Jetcheva *The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks*, <draft-ietf-manet-dsr-06.txt>.
- [6] VD Park and MS Corson, *A highly adaptive distributed routing algorithm for mobile wireless networks*, Proc. INFOCOM'97, Apr. 1997
- [7] C.-K. Toh, *Long-lived Ad-Hoc Routing based on the concept of Associativity*, March 1999 IETF Draft, 8 pages.
- [8] R. Dube et al., "Signal Stability based adaptive routing for Ad Hoc Mobile networks", IEEE Personal. Comm., Feb. 1997, pp. 36-45.
- [9] C. Zhu, M. Lee, T. Saadawi, *A Border-aware Broadcast Scheme for Wireless Ad Hoc Network*, IEEE CCNC, Las Vegas, January, 2004.