Dynamic backup routes routing protocol for mobile ad hoc networks

Ying-Hong Wang, Chih-Feng Chao *

Department of Computer Science and Information Engineering, Tamkang University, 151 Ying-chuan Road, Tamsui, Taipei County 251, Taiwan

Received 16 April 2004; received in revised form 8 September 2004; accepted 10 September 2004

Abstract

Mobile Ad Hoc Networks (MANETs), which provide data networking without infrastructure, represent one kind of wireless networks. A MANET is a self-organizing and adaptive wireless network formed by the dynamic gathering of mobile nodes. Due to the mobility of mobile nodes, the topology of a MANET frequently changes and thus results in the disability of originally on-the-fly data transmission routes. The dynamic properties of MANETs are therefore challenging to protocol design. To cope with the intrinsic properties of MANETs, Dynamic Backup Routes Routing Protocol (DBR²P), a backup node mechanism for quick reconnection during link failures, is proposed in this paper. DBR²P is an on-demand routing protocol and it can set up many routes to reach a destination node in a given period. Even when a link fails, those routes from the source node to the destination node can be analyzed to obtain backup routes to sustain quick reconnection. The information of backup routes can be saved in a specific on-the-route node and enables backup routes to be found immediately in situation regarding disconnection. As a result, DBR²P could more thoroughly improve the quality of routing protocol than those proposed in the past.

© 2004 Elsevier Inc. All rights reserved.
1. Introduction

The enhancement of communication and information technology has advanced the development of wireless networks. They have become increasingly important in several areas, including the military, academia, business, etc. Mobile Ad Hoc Networks (MANETs), different from the architecture of other wireless networks, need no infrastructure such as base stations or access points. A MANET provides users with a network while they are constantly moving. Each movement of a mobile node affects the topology of the network and the routes of transmission, and causes link failures sometimes. Mobile nodes communicate via radio waves. In the air, the coverage of this radio communication is limited. Accordingly, when the circumstance is poor or the distance between mobile nodes is large, low-quality transmission or even disconnection may occur.

Power limit also restricts MANETs. A mobile node in a MANET may be a notebook, a PDA, etc. Unlike stationary equipment, such as a PC and a Server, a mobile node usually depends on a finite energy source, normally batteries. “Power off” or “Suspend” state is used to reduce the consumption of electricity. As mentioned above, when a mobile node is in power off or suspend state, it will be invisible in the network and the network topology changes. Link failures are inevitable; thereupon, a change in the network topology affects not only the communication among the nodes but also the quality of the packet transmission. If backup routes are available, packets can still be delivered quickly once a link fails; otherwise, much time is taken to find a new route to the destination node.

Numerous routing protocols have been developed for MANETs, and they can be generally categorized as table-driven or proactive protocols, and on-demand or reactive protocols [1]. Table-driven routing protocols attempt to maintain consistent and up-to-date routing information among nodes in MANETs. Such protocols include Destination-Sequenced Distance-Vector Routing (DSDV) [2,11], Clusterhead Gateway Switch Routing (CGSR) [3], Wireless Routing Protocol (WRP) [4], Fisheye State Routing (FSR) [14], etc. They require each node to maintain one or more routing tables to record routing information, and propagate updated packets through the network to maintain consistent network information when the network topology changes. Table-driven routing protocols may generate a large overhead in a highly dynamic network environment because the network topology changes frequently to refresh the routing tables.
Unlike table-driven routing protocols, on-demand routing protocols create routes when a route is required only from the source node to the destination node. Ad Hoc On-Demand Distance Vector Routing (AODV) [5], Dynamic Source Routing (DSR) [6,10], Temporally Ordering Routing Algorithm (TORA) [7], Associativity-Based Routing (ABR) [8], Signal Stability Based Adaptive Routing (SSA) [9], and Neighborhood aware Source Routing (NSR) [15] are all on-demand routing protocols for MANETs. Source-initiated on-demand routing protocols frequently support route discovery to establish a route when required from a source node, and then maintain the route as the network topology changes. The issue of how to find a route rapidly and stably is frequently addressed. In many researches, the on-demand routing protocols often outperform table-driven routing protocols in highly dynamic network environments [1,6,10]. Still, MANETs have many limitations, such as high power consumption, low bandwidth, high error rates, etc. No matter either the table-driven or on-demand protocol is used, the dynamic mobile characteristic of MANETs is the most important factor that impacts the life cycle of a route.

This paper proposes a new on-demand routing protocol for MANET—Dynamic Backup Routes Routing Protocol (DBR²P). The rest of this paper is organized as follows. Section 2 discusses the motivation of this research. Section 3 describes the operation of DBR²P. Section 4 provides the algorithms of DBR²P. Section 5 illustrates the performance analysis of DBR²P. Finally, this paper concludes by suggesting current challenges and potential directions of future work.

2. Motivation

Over the past years, several routing protocols for MANETs have been proposed to establish fast and stable routes. The properties of MANETs, such as mobility and the lack of infrastructure, still dominate routing algorithms, but performance has not yet been greatly improved. The Dynamic Source Routing (DSR) protocol [6,10] is an on-demand routing protocol, and uses source routing instead of hop-by-hop packet routing. Each data packet has the list of hops in the path; therefore, each intermediate node needs not to keep route information [12]. DSR, including two major phases: route discovery and route maintenance, uses no periodic routing advertisement message, and sets up the routes based on the demand of a source node. In the route discovery phase, that is a source node needs a route to the destination node, the source node broadcasts a route request message with a unique request identification number. When the destination node receives this request, it sends a route reply message with path information back to the source node. When the other intermediate nodes receive the request, the nodes append their addresses to the source route and
broadcast this request if the request is not duplicated. Otherwise, the duplicated request will be discarded.

In the route maintenance phase of DSR, each node along the route detects the transmission of data packet by acknowledgement or passive acknowledgement. If a node does not receive the acknowledgement or hear the next hop forwarding the packet along the route, a route error packet is generated and sent to the original source node to invoke a new route discovery phase. Although DSR can respond a route quickly, it yields a long delay when a route is rebuilt.

The following directions could increase the quality of routing protocols for MANETs and avoid the aforementioned situation.

- Avoid excessively long routes;
- Choose more stable routes;
- Choose low-mobility nodes;
- Accelerate the maintenance and re-establishing of routes; and
- Implement backup routes.

According to the directions above, a new routing protocol, DBR^2P, with backup route mechanisms is generated. The proposed method focuses on the intrinsic properties of MANETs and considers many factors that affect the quality of routing. When a route is required from the source node to the destination node, quality of service is only slightly affected as long as the time spent to search for the route is within the tolerated period. That is, when the source node broadcasts the route request packets to find a route to the destination node, the route through which request first arrives at the destination node may not be the shortest path or the most stable one. Both the throughput between nodes and the stability of the connection influence the order in which requests reach the destination node. A short period wait to allow requests to be received by the destination node via some more routes will provide potential backup routes to support reconnection if a link fails. Backup route information is saved in particular on-route nodes. After the backup routes are found, nodes can be traced back whenever a disconnection or loss of connection occurs. The destination node replies the first route as the primary route; therefore, a period, $T_c$, can be specified during which more routes are gathered for backup route analysis.

3. Dynamic backup routes routing protocol (DBR^2P)

DBR^2P is an on-demand routing protocol that does not require any routing table. It replies a complete route from the source node to the destination node on demand and sets up many backup routes dynamically for quick reconnection when a link fails. DBR^2P allows intermediate nodes to receive and trans-
mit the same request packets as obtained from the source node to gather more information to establish backup nodes. Some basic protocol packets needed to be defined for DBR²P are illustrated in Table 1. DBR²P includes three phases—route discovery, backup node setup and route maintenance—requiring two kinds of cache: RD_request_Cache, and Backup_Routes_Cache (Fig. 1). A source node sets a unique request identification number for each RD-request packet from a locally maintained sequence number. The RD-request_Cache of a node is used to store temporary counters, which record how many times this node receives the RD-request packets with the same identification number, in the route discovery phase. The entity, (RD, n), is used to present this counter parameter, where RD is the sequence number of the RD-request, and n is the number of times that RD-request with the same RD has been received. The

<table>
<thead>
<tr>
<th>Packet name</th>
<th>Function</th>
<th>Main fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route discovery request</td>
<td>To find and record the route content from the source node to the destination node</td>
<td>Sequence number, destination node, route content</td>
</tr>
<tr>
<td>(RD-request)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Route discovery reply</td>
<td>The destination node replies the route content back to the source node</td>
<td>Sequence number, route content</td>
</tr>
<tr>
<td>(RD-reply)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backup setup packet</td>
<td>The destination node transmits the backup information to the backup nodes to set up backup routes</td>
<td>Sequence number, backup node, backup route</td>
</tr>
<tr>
<td>(BS-packet)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link_Fail_Message</td>
<td>When a link fails, this message is used to announce the backup nodes along the route to replace a backup route</td>
<td>Sequence number, route content</td>
</tr>
</tbody>
</table>

Fig. 1. The main architecture of DBR²P.
Backup_Routes_Cache is used to store backup routes. After the route discovery phase is initiated by a source node, the destination node may receive some routes, and then enter the backup node setup phase in order to analyze some backup nodes and backup routes. In backup node setup phase, the backup routes are sent to each backup node by BS-packets and stored in the Backup_Routes_Cache of each backup node.

3.1. Route discovery phase

When source node $S$ requires a route to destination node $D$, $S$ enters the route discovery phase to find a route to the destination node. The phase of route discovery is illustrated in Fig. 2. In this phase, source node $S$ broadcasts RD-request to nearby nodes. The RD-request is used to discover some routes to the destination node. The RD-request includes a sequence number field to distinguish the route discovery process from the others, and a route content field to record all the addresses of nodes along the path from $S$ to $D$. Initially, the address of the source node is inserted in the route content field of RD-request.

![Fig. 2. Route discover phase of DBR²P.](image-url)
A source node sets a unique request identification number for each RD-request packet from a locally maintained sequence number. When a node receives a RD-request from its neighbor, it will check whether the RD-request is received for the first time according to the sequence number of the RD-request in the records of its RD-request_Cache. There is no \(\langle \#RD, n \rangle\) entity in its RD-request_Cache, if this node receives the RD-request from a node for the first time. Then, the entity, \(\langle \#RD, 1 \rangle\), is stored in the RD-request_Cache of the node, where \#RD is the sequence number of the RD-request, and the value, 1, means this node receives the RD-request for the first time. Also, the timer, \(T_c\), is started. Then, this node inserts its address into the route content field of the RD-request, and broadcasts this modified RD-request to its neighboring nodes.

If a node receives the RD-request with duplicate sequence number from its neighboring nodes, then there is an entity, \(\langle \#RD, n \rangle\), in its RD-request_Cache, where \#RD is equal to the sequence number of this RD-request. This node continues to check whether the route content field of RD-request includes its address; if so, the node discards this RD-request to avoid the loop infinitely. On the other hand, if the address of this node is not included in the route content field of the RD-request, the node will then check whether the value of \(n\) is not smaller than three, or whether the timer, \(T_c\), is timeout; if so, the node discards this RD-request. Otherwise, the node increases the value \(n\) of \(\langle \#RD, n \rangle\) by one, and then inserts its address into the route content field of RD-request and broadcasts this modified RD-request to its neighboring nodes.

The parameter \(n\) in the \(\langle \#RD, n \rangle\) entity, stored in the RD-request_Cache of a node, is used to prevent too many upstream or downstream paths crossing this node as this node is an intermediate node of the route or backup routes. If there are too many transmissible paths of the route and backup routes crossing a node, a lot of backup routes may be invalid as this node disconnects or moves out. In addition, \(T_c\) begins when a node first receives RD-request from an upstream node, and a too long route can be avoided if a node only receives duplicate RD-request within the period \(T_c\). Moreover, the control message overhead, such as RD-request packets, can be bounded.

\(\text{DBR}^2\text{P}\) can prevent an infinite loop and remain the same method of the route discovery phase as the DSR if \(T_c\) equals zero. After the destination node \(D\) receives the first-arriving RD-request, \(D\) sends a RD-reply, whose route content field is the route of the RD-request, back to the source node \(S\), and waits for more RD-requests for a short period, \(T_c\), before entering the Backup Node Setup Phase.

### 3.2. Backup node setup phase

After the route discovery phase, the destination node \(D\) may gather many routes within a period, \(T_c\). The nodes (excluding \(S\) and \(D\)) among those routes
from $S$ to $D$ are intermediate nodes. Backup nodes are nodes with at least two different paths to their neighboring nodes in those routes from $S$ to $D$. Thus, $S$ is itself a possible backup node. The nodes in those routes that $D$ has received are compared pair wise (from beginning to end) to find whether any two paths have a section in common. A final node (excluding $D$) in such a section is a backup node. A subset of backup nodes can be gathered from any two routes. Then, all the subsets of backup nodes are joined and the BS-packet that includes each backup node and the partial paths from the backup node to $D$ is generated. $D$ then uses the BS-packet to set up the Backup Route Cache of those backup nodes separately. The BS-packet contains the sequence number of this routing process, the address of a backup node, and the path from the backup node to the destination node. The backup nodes store the partial paths from the backup node to the destination node in their Backup Route Cache after they receive the BS-packet.

3.3. Route maintenance phase

Since the wireless networks are inherently less reliable than wired networks, the mechanism of a hop-by-hop acknowledgement at the data link level can be used to provide early detection and re-transmission of lost or corrupted packets [6]. An equivalent acknowledgement signal may be available in many environments, if the wireless networks do not provide data link level acknowledgements. This type of acknowledgement is known as a passive acknowledgement [6,13]. A sender may be able to hear the next node transmitting the packet again after sending a packet to its next node along the route. For example, as shown in Fig. 3, node $X$ may be able to hear node $Y$’s transmission of the packet onto node $Z$. In addition, the transport or application level reply or acknowledgement from the destination node could also be used as an acknowledgement that the path is still working.

Fig. 3. An example of the passive acknowledgement.
In our proposed method, the mechanism of passive acknowledgement is used to detect a link failure. When a link failure is detected, a node in the route from the source to the destination cannot continue to transmit the data packet. That node will pass a "Link_Fail_Message" to an upstream node until the message reaches a backup node. After the backup node receives the Link_Fail_Message, the backup route of Backup_Route_Cache is fetched to replace the route behind the backup node, and the source node $S$ is informed to change the route. Then, $S$ sends packets along the new route. A backup route that has been fetched by the Backup_Route_Cache is labeled as a non-backup route. If Backup_Route_Cache includes no other backup route, then the node loses the qualification as a backup node. The source node will re-enter the Route Discovery Phase to establish a new route to the destination node when no available backup node exists.

After the destination node replies a path back to the source node as the current route for sending data packet, some backup routes are established and stored in the backup nodes. Sometimes the backup routes may be incorrect if the destination node receives some inconsistent routes due to the loss of RD-request in the route discovery phase or the movement of the nodes along the backup routes in the route maintenance phase. If the current route is still alive, the situation that the backup routes are incorrect will not influence the communication of the current route. If the current route is broken and replaced by a backup route, DBR$^2$P can still operate even though the backup route is broken again. That is because the link failure will be detected and a Link_Fail_Message will be sent to find another backup node.

### 3.4. An example of using DBR$^2$P

Following the aforementioned process and principle of DBR$^2$P, this section presents an example to show how DBR$^2$P performs in a practical case. As illustrated in Fig. 4, when source node $S$ requires a route along which to send packets to destination node $D$, $S$ first enters the route discovery phase, in which it broadcasts RD-requests with a sequence number (e.g. the sequence number is $\#RD_i$) to its neighboring nodes.

When an intermediate node (e.g. node $F$) receives the RD-request, $F$ will check whether the RD-request is received for the first time. In this example, it is assumed that $F$ receives the RD-request for the first time. Therefore, there is no $\langle \#RD_i,n \rangle$ entry in $F$'s RD-Request_Cache. $F$ creates $\langle \#RD_i,1 \rangle$ entry in its RD-Request_Cache, and starts the timer, $T_c$. Then, $F$ inserts its address into the route content field of the RD-request (e.g. $S \rightarrow B \rightarrow F$) and broadcasts the modified RD-request to its neighboring nodes.

If $F$ receives the RD-request with the same sequence number, $\#RD_i$, again, $F$ will check whether its address is recorded in the route content field of the RD-Request (e.g. $S \rightarrow A$). If not, the $\langle \#RD_i,1 \rangle$ entity is modified as $\langle \#RD_i,2 \rangle$. And
then, $F$ inserts its address into the route content field of the RD-request (i.e. $S \rightarrow A \rightarrow F$) and broadcasts the modified RD-request again. The node $F$ will discard the RD-requests with the same sequence number, $\#RD_i$, if $F$ receives the duplicate RD-request in one of three situations below:

- The record in the RD-Request_Cache for the RD-request is $\#RDi, 3$.
- The timer, $Tc$, for the RD-request is time out.
- The address of $F$ has appeared on the route content field of the RD-request.

When the destination node $D$ receives the first-arriving RD-request, $D$ will reply the RD-reply packet that contains the route of the first-arriving RD-request, for example, $S \rightarrow B \rightarrow F \rightarrow I \rightarrow M \rightarrow D$. Then, within a period, $Tc$, $D$ gathers some more routes from $S$ to $D$, as follows:

- $S \rightarrow B \rightarrow F \rightarrow I \rightarrow M \rightarrow D$
- $S \rightarrow B \rightarrow F \rightarrow J \rightarrow M \rightarrow D$
- $S \rightarrow B \rightarrow G \rightarrow J \rightarrow M \rightarrow D$
- $S \rightarrow A \rightarrow E \rightarrow F \rightarrow I \rightarrow M \rightarrow D$
- $S \rightarrow A \rightarrow B \rightarrow F \rightarrow J \rightarrow N \rightarrow D$

Those routes the destination node $D$ has received are compared pair wise (from beginning to end) to find the set of backup nodes $\{S, A, B, F, J\}$. Fig. 5 shows the backup nodes and the sectional paths in common. Notably, the source node $S$ is also a backup node because $S$ can send packets to $D$ in two ways. Then, $D$ sends the BS-packet, which carries information about backup routes to each backup node. In this case, $F$ will be a backup node and stored $\{F \rightarrow I \rightarrow M \rightarrow D, F \rightarrow J \rightarrow M \rightarrow D, F \rightarrow J \rightarrow N \rightarrow D\}$ in its Backup_Route_Cache.
After the source node $S$ receives a RD-Reply from destination node $D$, it begins to send data packets to $D$. When a link fails during the delivery of packets, some nodes cannot send the packets to downstream nodes by the current path. These nodes will pass Link_Fail_Messages to upstream nodes until the messages reach the backup node. For example (Fig. 6), node $I$ leaves and a link fails from $I$ to $M$; node $I$ then sends a Link_Fail_Message back to $F$. Then, backup node $F$ checks its Backup_Route_Cache and fetches one path ($F \rightarrow J \rightarrow M \rightarrow D$) to replace the current route, informing $S$ to change the route to $D$. (The new route is $S \rightarrow B \rightarrow F \rightarrow J \rightarrow M \rightarrow D$.) Later, $F$'s Backup_Route_Cache may include no available backup route; thereupon, $F$ loses the qualification as a backup node.

4. The algorithms of DBR$^2$P

To evaluate the performance of DBR$^2$P, we constructed a simulator using a Java platform. The algorithms of the Java programs are discussed in this section. It is worth to note that $Tc$ is a critical parameter that influences the
performance of DBR$^2$P. $T_c$ begins with the $D$'s first receiving RD-request from $S$ and then analyzing some routes after a period $T_c$, during which gathering is completed. If $T_c$ is larger, more redundant backup routes are processed, and more delay in the route maintenance phase occurs. Also, the control message overhead in the network is increasing because too much transmission of RD-requests. A small $T_c$ will reduce the number of analyzable routes the destination node has gathered, and will be useless for reconnection when a link fails in the route maintenance phase. To avoid receiving and sending redundant RD-requests during the route discovery phase, the intermediate nodes must stop receiving any RD-request after $D$ has received its last RD-request before the end of $T_c$. The three approaches for an intermediate node to stop receiving any RD-Requests are as follows:

1. The destination node broadcasts messages to each node.
2. All nodes are set to a synchronous clock in order to possess consistent period, $T_c$.
3. The intermediate nodes only receive RD-request from $S$ in a period, $T_c$.

However, the first approach increases the network overhead, and the second is very difficult to implement. Requiring only that an adequate $T_c$ be chosen for DBR$^2$P, the last approach is the most convenient one to implement. Hence, we use the third approach to design the algorithms of DBR$^2$P.

Moreover, DBR$^2$P also allows an intermediate node to receive and broadcast the duplicated RD-request less than $n$ times, where $n$ is three, in the period, $T_c$. The parameter $n$ affects the amount of control messages spreading in the network. It also influences how many backup nodes are selected and how many backup routes are established. If the value of parameter $n$ is 1, that is the intermediate node will discard the duplicated RD-request instantly, the process of an intermediate node in the route discovery node in the route discovery phase will be similar to the DSR. In addition, it is scarcely influential if the value of parameter $n$ is large because the amount of control messages, such as the redundant RD-requests, can be restricted by the period, $T_c$.

The main DBR$^2$P algorithms of each phase, which are route discovery, backup node setup and route maintenance, are illustrated respectively in the following section.

4.1. Algorithms of route discovery phase

When a source node $S$ requires a route to the destination node $D$, $S$ will enter the route discovery phase. In the route discovery phase, the processes can be divided into three situations including source node procedure, interme-
mediate node procedure and destination node procedure. During the operation of
S, S runs SOURCE_NODE_ROUTE_DISCOVERY(S, D) to find a new route
to D.

In Procedure SOURCE_NODE_ROUTE_DISCOVERY(S, D), a timer is
started and the RD-request packets are broadcasted to the neighboring nodes
of S. During the period, S waits for the RD-reply packet from the D. If S re-
ceives a RD-reply in the period, the route from S to D is established. Other-
wise, a Link_Fail_Message meaning no route is found from S to D will
return to S. The details of Procedure SOURCE_NODE_ROUTE_DISCOV-
ERY(S, D) are as follows:

Procedure SOURCE_NODE_ROUTE_DISCOVERY(S, D)
//S: the source node
//D: the destination node
{
    start the timer;
    create a new RD-request;
    broadcast the RD-request;
    while (not timeout or not receive a RD-reply)
        wait for a RD-reply;
    end while
    If (receive a RD-reply)
        return the route of the RD-reply;
    else
        return ERROR;
    end if
}

When an intermediate node receives the first-arrival RD-request from an
upstream node, it executes Procedure RECEIVE_RD-request (RD-request),
starts the timer and modifies the RD-request, to insert its address, Node-
Addr, into the route content field of the RD-request. The intermediate node
then broadcasts this modified RD-request to its neighboring nodes. The en-
tity, (#RD,1), is recorded in the RD-request_Cache of this intermediate
node, where #RD is the sequence number of the RD-request and the value,
1, means this is the first time the node receives the RD-request of the se-
quence number.

If a node receives the duplicate RD-request that it has received before, the
node will discard the RD-request in one of situations below:

- The value $n$ of (#$RD$,n) in the RD-Request_Cache is not smaller than three.
- The timer, $Tc$, for the RD-request is timed out.
The address of this node has appeared on the route content field of the RD-request.

Otherwise, the node will increase the value $n$ of $\langle #RD, n \rangle$ by one and broadcast the modified RD-request again after its address is inserted into the route content field of the RD-request. The details of Procedure RECEIVE_RD-request($RD\text{-}request$) are as follows:

**Procedure** RECEIVE_RD-request ($RD\text{-}request$)

- $#RD$: the sequence number of a RD-request
- $Tc$: a timer for each route discovery process which is separated by $#RD$
- $\langle #RD, n \rangle$: value $n$ is how many times the node receives the RD-request whose sequence number is $#RD$
- $NodeAddr$: the address of this intermediate node

```
if (this RD-request is received for the first time)
    insert $\langle #RD, 1 \rangle$ into the $RD\text{-}request\ Cache$;
    start the timer, $Tc$;
    insert the $NodeAddr$ into the RD-request;
    broadcast the RD-request;
else
    if ($NodeAddr$ is in this RD-request)
        discard the RD-request;
    else if ($n \geq 3$ or timeout)
        discard this RD-request;
    else
        $\langle #RD, n \rangle \leftarrow \langle #RD, n + 1 \rangle$;
        insert $NodeAddr$ into the RD-request;
        broadcast the RD-request;
end if
```

After the destination node $D$ receives a RD-request, $D$ executes Procedure DESTINATION_RECEIVE_request ($RD\text{-}request$) to send the route content of the first-arriving RD-request back to the source node $S$, and to start a timer in order to wait for more routes. After the period, $D$ will enter the Backup Node Setup Phase. The details of Procedure DESTINATION_RECEIVE_request($RD\text{-}request$) are as follows:

**Procedure** DESTINATION_RECEIVE_request ($RD\text{-}request$)

- $#RD$: the sequence number of the RD-request
- $Tc$: a timer for each route discovery process which is separated by $#RD$

```
```
4.2. Algorithms of backup node setup phase

The backup node setup procedure, BACKUP_NODE_SETUP(Route, #RD), has two sub-procedures: FIND_BACKUP_NODE(Route) and SETUP_BACKUP(Backup, #RD), where Route is an array of routes the destination receives and Backup is an array of backup information including backup node and partial backup routes from backup node to destination node.

The goal of FIND_BACKUP_NODE(Route) is used to compare pair wise (from beginning to end) to find the backup nodes and the partial backup routes. After the FIND_BACKUP_NODE(Route) is executed, the information of backup routes and backup nodes will be returned. According to the returned information, the function, SETUP_BACKUP(Backup, #RD), then sends the BS-packets that include each backup node and the partial path from the backup node to the destination node to set up the Backup_Route_Cache of those backup nodes respectively. The details of Procedure BACKUP_NODE_SETUP(Route, #RD) are as follows:

Procedure BACKUP_NODE_SETUP(Route[], #RD)
// Backup[]: an array of backup information, Backup[].node is the field of backup node
// Route[]: an array for storing the routes of RD-requests
// #RD: the sequence number of RD-request
{
    Backup[] ← FIND_BACKUP_NODE(Route[]);
    run SETUP_BACKUP(Backup[], #RD);
}
Procedure FIND_BACKUP_NODE(Route[])
// RouteTemp1, RouteTemp2, BackupNodeTemp, BackupRoutTemp: variables
// Route1[], Route2[]: array for storing the route temporarily
// #N: the number of Route[]
{
  for (RouteTemp1 ← Route[1] to Route[#N − 1])
    for (RouteTemp2 ← Route[2] to Route[#N])
      Route1[] ← transfer RouteTemp1 to an array;
      Route2[] ← transfer RouteTemp2 to an array;
      flag1 ← 1;
      flag2_start ← 1;
      while (flag1 < length of Route1[])
        flag2 ← flag2_start;
        while (flag2 < length of Route2[])
          if (Route1[flag1] = Route2[flag2])
            while (Route1[flag1] = Route2[flag2] and
                (flag1 < length of Route1[] or flag2 < length of Route2[]))
              flag1 ← flag1 + 1;
              flag2 ← flag2 + 1;
          end while
        end while
      end if
    end for
  end for
  BackupNodeTemp ← Route1[flag1];
  BackupRoutTemp ← path from the (flag1 − 1)th node to the end of RouteTemp1;
  if (BackupRoutTemp is not in Backup[].route)
    Backup[].node ← BackupNodeTemp;
    Backup[].route ← BackupRoutTemp;
  end if
  BackupRoutTemp ← path from the (flag2 − 1)th node to the end of RouteTemp2;
  if (BackupRoutTemp is not in Backup[].route)
    Backup[].node ← BackupNodeTemp;
    Backup[].route ← BackupRoutTemp;
  end if
end if
flag2_start ← flag2;
flag2 ← flag2 − 1;
end if
end if

end while
next
next
return Backup[];
}

Procedure SETUP_BACKUP (Backup[], #RD)
{
    for each entry of Backup[]
set up BS-packet with Backup[].node, Backup[].route, and #RD;
send the BS-packet to Backup [].node;
    next
}

After the backup nodes receive the BS-packet, they execute Procedure RECEIVE_BS-packet(BS-packet) to store the partial routes of BS-packet into their Backup_Route_Cache and set their mode to “BACKUP_NODE”. The details of Procedure RECEIVE_BS-packet(BS-packet) are as follows:

Procedure RECEIVE_BS-packet (BS-packet)
{
    set up the mode of “BACKUP_NODE”;
    insert BS-packet’s backup route strings into Backup Route Cache;
}

4.3. Algorithms of route maintenance phase

When the source node S sends a data packet, the header of the data packet includes the route to the destination node D and the sequence number. When an on-the-route node receives a data packet, it transmits the data packet to the next node and hears the transmission of the packet of next node onto another node. If the node cannot hear any transmission of the packet in a period nor receive the acknowledgement from its downstream node, it will assume that the link fails.

When a link fails, a node cannot continue to transmit packets and it will pass a Link_Fail_Message to an upstream node along the reverse current route. If a backup node receives the Link_Fail_Message, this backup node fetches another backup node from Backup_Route_Cache and replaces the current route behind it. The backup node also informs the source node S that the route has been changed, and labels this fetched backup route as “Non_Backup_Route”. If Backup_Route_Cache includes no other backup route, then the node has lost the identity of “BACKUP_NODE”. If the source node receives the Link_Fail_Message and loses the identity of “BACKUP_NODE”,...
it will re-enter the Route Discovery Phase to establish a new route to the destination node.

When a node receives a Link_Fail_Message, it executes Procedure LINK_FAIL(Link_Fail_Message), shown as follows:

**Procedure** LINK_FAIL (Link_Fail_Message)

// $S$: the source node
// $D$: the destination node

{ if (is a backup node)
    fetch a backup route from *Backup Route Cache*;  
    instead the current route with the backup route; 
    send a new route back to the source node; 
    delete the fetched backup route; 
    if (there is no available *backup route* in *Backup Route Cache*)
        cancel the identity of “BACKUP_NODE”;
    end if
    replace the new backup route;
else if (this node is the source node but not a “BACKUP_NODE”)
    return NEW_ROUTE_DISCOVERY($S, D$);
else
    send a Link_Fail_Message to the upstream node of the current route;
end if
}

5. Performance evaluation

5.1. Simulation environment

To evaluate the performance of DBR$^2$P, we constructed a simulator using a Java platform. This simulator allows us to observe and measure the performance of DBR$^2$P under a variety of conditions. The parameters in our simulation are given as follows:

- The area of simulation is in $500 \times 500$ m$^2$.
- The number of mobile nodes is 50.
- Nodes in the simulation move according to the “random waypoint” model [16] with 5 s pause time.
- Each transceiver of a node has a range of 50 m$^2$.
- The bandwidth for transmission data is 2 Mbps.
- The size of route discovery packet is 2 Kbytes.
- The size of data packet is constant and 30 Kbytes.
• 20 pairs of source and destination nodes are transmitting and receiving data packets simultaneously in the simulation.
• The simulation time is 200 s.
• The average mobility speed of all nodes is from 0 to 30 km/h.

Each node is initially placed at a random position within the simulation area. Source nodes and destination nodes are chosen randomly with uniform probabilities. To simulate node mobility, each node randomly chooses to move to a new location in the velocity between 0 and 30 km/h. If a moving node protrudes from the simulation area, its direction will be changed. All source nodes and destination nodes are chosen randomly in the simulation. A traffic generator is used for the sources to simulate constant bit rate packet delivery. The packet is lost if the sender receives no acknowledgement from the next node after sending a packet to its next node.

In order to evaluate the performance of DBR²P, two other routing protocols, DSR and DSDV, are chosen for comparison. The DSR [6,10] is an on-demand routing protocol; a mobile node initiates a route discovery phase to establish a route and then performs a route maintenance phase to maintain the established route. If a link fails, the data transmission will be broken before the node finds a new transmission route. The DSDV [2,11] is a table-driven routing protocol based on the classical Bellman–Ford routing mechanism [12]. Each mobile node maintains a routing table in which all the possible destinations within the network and the numbers of hops to each destination are recorded. Each entry is marked with a sequence number for a node to distinguish old routes from new ones and to avoid routing loops. The routing table updates are periodically transmitted all around the network in order to maintain the consistency of each node’s table. During periods of infrequent movement, this kind of route update packet, full dump, is transmitted with all available routing information. Another kind of smaller packet, incremental, is used to modify the information that has changed since the last full dump.

In DSDV, new route broadcasts contain the address of the destination, the number of hops to reach the destination, the sequence number of the information received regarding the destination, and a new sequence number unique to the broadcast [2]. The route labeled with the most recent sequence number and smaller metric is used. The comparisons of some characteristics of DBR²P, DSR and DSDV are shown in Table 2.

In the route discovery phase of DBR²P, the period of timer, $T_c$, will influence the amount of RD-requests the destination node receives. It will also influence how many backup routes the destination node can obtain in the backup node setup phase. In the first two metrics of performance evaluation, two situations of the timer, $T_c$, are assigned; one is in a period of 20 ms, and the other is in a period of 50 ms.
The performance metrics to be observed are shown as follows:

- **Control message overhead**: the number of necessary control messages for all nodes in the network to maintain the routing table or a source node to establish and maintain a route to the destination node. Notice that DBR²P is to establish not only the route to destination node but also the backup routes. In this performance metric, two situations of DBR²P are simulated. One is the situation that the timer, $T_c$, is 20 ms (marked as DBR²P1). The other one is the situation that the timer, $T_c$, is 50 ms (marked as DBR²P2).

- **Data throughput**: the amount of data packets that have passed through the network in the simulation time. Similarly, two situations of DBR²P are simulated in this performance metric. One is the situation that the timer, $T_c$, is 20 ms (marked as DBR²P1). The other one is the situation that the timer, $T_c$, is 50 ms (marked as DBR²P2).

- **Average transfer latency**: the average interval from the time the unicasting of a source node is initiated to the time this node finishes its unicasting.

### 5.2. Results and analysis

The control message overhead incurred by DBR²P, DSR and DSDV is shown in Fig. 7. Both DBR²P and DSR have the constant increasing rate of control message overhead, and the DSDV has an almost constant amount of control message overhead in varied mobility. Due to the control message overhead metric, two situations of DBR²P were simulated. The DBR²P1 in Fig. 7 shows the simulation result of first situation that the timer, $T_c$, is 20 ms. The DBR²P2 shows the simulation result of another situation that the timer, $T_c$, is 50 ms. Because the DBR²P provides the mechanism of backup routes and requires more RD-request packets for establishing backup routes, the control message overhead in both situations of DBR²P is larger than DSR. In DBR²P, the RD-request packets for a destination node to gather more routes do not spring up all over the whole network. Moreover, in the two results of DBR²P1
and DBR²P2, the control message overhead is not much increased if we increase the period of $T_c$. The amount of RD-request packets can be bounded in the route discovery phase because each intermediate node only receives and re-broadcasts the duplicate RD-request three times at most. The difference of the control message overhead amount between DBR²P1 and DBR²P2 is caused by the BS-packets for preparing different backup routes.

The simulated result of data throughput is shown in Fig. 8. The poor performance of DSDV can be attributed to frequent update control messages as mobility speed increases. Fig. 8 also shows that DBR²P has a higher throughput than DSR. In DSR, a route must be re-established when this route is broken, and it results in the decrease of throughput before a new route is established. Because DBR²P provides the mechanism of backup routes, the decrease of throughput while a link fails is not obvious. While the route fails, the data packets can be re-transmitted within a short time if a backup route is found and replaced. Moreover, the throughput of DBR²P2, which is the situation of larger $T_c$, is a little lower than DBR²P1 and DSR when the mobility is low. That is because the DBR²P2 pays more overhead to set up more backup routes, but the utilization of backup routes is not frequent during the low mobility situation. However, the utility of more backup routes in the situation of larger $T_c$ (DBR²P2) is more obvious than the situation of smaller $T_c$. According to the result of Fig. 8, the mechanism of backup routes is still helpful to the performance of data throughput in high mobility even though the control message is rising.
Furthermore, from the two results discussed above, the timer, $T_c$, should be adjusted dynamically according to the circumstances of networks, such as the mobility of nodes, the number of backup routes the source node needs, or the demand of service of each individual user. And, the value of the timer, $T_c$, can be determined according to the result of evaluations or the actual network statistics.

Fig. 9 shows the average transfer latency comparison of DBR$^2$P, DSR and DSDV. Both DBR$^2$P and DSR have better performance than DSDV. In DSDV, the mechanism of table-driven routing results in route reconnection frequently when links fail, and the average transfer latency of transmission increases as mobility speed increases. DBR$^2$P has better performance than DSR in low mobility because DBR$^2$P provides the mechanism of backup routes to shorten the delay of reconnection as a link fails. When links fail, with the increasing mobility speed, the probability to reconnect the transmission route successfully by using backup routes is decreasing. It results in the performance of average transfer latency being close to the DSR in high mobility. Increasing the period of timer, $T_c$, of DBR$^2$P can enhance the total number of backup routes and the performance of transfer latency. However, it will result in heavier network traffic because of increasing control message overhead.

According to aforementioned results, the performance of DBR$^2$P is better than the DSR in low mobility environment. Although the performance of DBR$^2$P is close to the performance of DSR when the mobility is high, the performance of DBR$^2$P is better than the DSDV protocol. Furthermore, both on-demand routing protocols, DBR$^2$P and DSR, can provide shortest routes.
during the route discovery phase. The time cost of DBR²P and DSR for establishing a current route is approximately equal. However, in DBR²P, the destination node continues to receive more routes to establish backup routes. The mechanism of backup routes, that the DSR does not provide, can improve the performance effectively.

6. Conclusions

In this paper, a new on-demand routing protocol for MANETs, DBR²P, is proposed. It could provide a backup node mechanism for quick reconnection when a link fails and set up many routes to reach a destination in a given period. Even when a link fails, these routes, almost more than one, from the source node to the destination node can be analyzed to obtain backup routes to support reconnection. DBR²P is proposed to focus attention on the intrinsic properties of MANETs and the results of the simulation experiment show that DBR²P has good performance.

In the wireless environment, DBR²P could provide more reliable data transmission for applications, such as the mobile learning, mobile commerce and mobile entertainment, [17–19]. Improvement on the stability of routes could still be considered in the future for DBR²P [20]. Issues such as quality of service and multicast could be addressed to enhance the capability of DBR²P. Moreover, in order to enlarge the scale of applications for DBR²P, supporting
hierarchy and heterogeneous interfaces in MANETs [21] could be considered in future researches.

References
