

Ad Hoc On-Demand Backup Node Setup Routing Protocol

YING-HONG WANG AND CHIH-CHIEH CHUANG
Department of Computer Science and Information Engineering
Tamkang University
Taipei, 251 Taiwan
E-mail: inhon@mail.tku.edu.tw
E-mail: g6190058@tkgis.tku.edu.tw

An ad hoc wireless network supports data networking without an infrastructure, so that users can use network services while continually moving. Each move of the mobile host affects the topology of the network and the transmission route. The Ad Hoc Backup Node Setup Routing Protocol (ABRP) is proposed to focus attention on the intrinsic properties of the ad hoc wireless networks. Basically, ABRP chooses some routes as backup routes from a number of routes between the source and destination. It more completely considers the quality of routing than the routing protocols proposed in the past. ABRP involves three phases: route discovery, backup node setup and route maintenance. In the route discovery phase, a node is allowed to receive duplicate requests for finding many routes to reach a destination. In the backup node setup phase, those routes, almost more than one, can be analyzed to obtain some good backup routes and can be saved in the backup route cache of the backup nodes. In the route maintenance phase when situations involving disconnection or loss of connection, these backup routes can be found rapidly. Furthermore, ABRP provides a backup route mechanism that reconnects quickly as required for ad hoc wireless networks.

Keywords: ad hoc network, wireless networks, on-demand routing, routing protocol, backup routes, mobile computing

1. INTRODUCTION

Wireless networks have been developing over a long period, and have become increasingly popular in several areas, including military, academia, business and others. *Ad hoc* networks represent one kind of mobile wireless network which provides data networking without infrastructure. Ad hoc wireless networks provide users with a network while they are continually moving. Each move of the mobile host affects the topology of the network and the route of transmission, sometimes causing a link failure. Mobile hosts communicate via radio waves. The coverage of this radio communication is limited. Accordingly, when the environment is poor or the distance between hosts is large, low-quality transmission or even disconnection may occur.

Power limits also restrict ad hoc networks. A mobile node in an ad hoc wireless network may be a notebook or a PDA. Unlike stationary equipment, such as a PC, such a mobile node usually depends on a finite energy source, normally batteries. "Power off" or "disconnection" is used to reduce the consumption of electricity. As stated above, when a mobile host detects power off or disconnection, it will be invisible in the network

Received March 7, 2003; revised June 16, 2003; accepted August 5, 2003.
Communicated by H. Y. Mark Liao.

and the network topology changes. Sometimes, link failure is inevitable; thereupon, a change in the network topology affects not only the communication among the nodes in the network but also the quality of the packet sending. Once a link fails, a packet can still be delivered quickly if backup routes are available. Otherwise, much time is taken to find a new route to the destination node.

Numerous routing protocols have been developed for ad hoc wireless networks, and they can generally be categorized as table-driven or on-demand [1]. In ad hoc networks, table-driven routing protocols attempt to maintain consistent and up-to-date routing information among nodes. Such protocols include Destination-Sequenced Distance-Vector Routing (DSDV) [2, 11], Clusterhead Gateway Switch Routing (CGSR) [3], and the Wireless Routing Protocol (WRP) [4]. 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 network topology changes. Table-driven routing protocols may generate a large overhead in a high mobility network environment, because network topology changes often to refresh the routing table.

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] and Signal Stability Routing (SSR) [9] are all on-demand routing protocols for ad hoc wireless networks. 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. The on-demand routing protocols often outperform table-driven routing protocols [1]. However, neither of the protocols involves backup routes; therefore, when a link fails, both must find a new route. Ad hoc wireless networks have many limitations, including high power consumption, low bandwidth and high error rates. Regardless of whether the table-driven or on-demand protocol is used, the dynamic mobility of an ad hoc network is the most important factor that affects a route's life cycle.

This article proposes a new ad hoc on-demand routing protocol. The remainder of the paper is organized as follows. Section 2 discusses the motivation of this research. Section 3 describes the new routing protocol. Section 4 provides the algorithm of the routing protocol. Section 5 illustrates the performance analysis of ABRP. The paper concludes by suggesting current challenges and potential directions of future work.

2. MOTIVATION

Several routing protocols for ad hoc wireless networks have been proposed to establish stable, short or fast routes. 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; so, each intermediate node need not keep route information [12]. The DSR does not require any kind of periodic message to be sent, and sets up the routes based on a source's demand. The DSR includes two major phases: route discovery and route maintenance. In the route discovery phase-a source node needs a route to the destination-the source broadcasts a route request mes-

sage with a unique request ID. When the destination node receives this request, it sends a route reply message with path information back to the source. When the other intermediate nodes receive the request, they append their address 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 an 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 the DSR can respond to a route quickly, it has a long delay when a route is rebuilt.

Lack of infrastructure and mobility are properties of ad hoc wireless networks, which still dominate routing algorithms; performance has not been greatly improved. To increase the quality of ad hoc routing the situations described must be avoided as follows:

1. Avoid excessively long routes.
2. Choose more stable routes.
3. Choose low-mobility nodes.
4. Accelerate the maintenance and re-establishing of routes.
5. Implementing backup routes.

The main idea of the five directions above is to adapt backup route mechanisms to generate a new routing protocol. The proposed method focuses on the intrinsic properties of ad hoc wireless networks and considers many factors that affect the quality of routing. When a route is required from the source node to the destination node, QoS is only slightly affected as long as the time spent searching for the route is within the tolerance period. That is, when the source node broadcasts the REQUEST packets to find a route to the destination node, the route through which REQUEST first arrives at the destination may not be the shortest path or the most stable. Both the throughput between nodes and the stability of the connection influence the order in which REQUESTs reach the destination. A short wait to allow REQUESTs to be received by the destination 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 with 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. AD HOC BACKUP NODE SETUP ROUTING PROTOCOL

The ABRP is an on-demand routing protocol that does not require any routing table. It replies with a complete route from the source to the destination on demand and sets up many backup routes for quickly reconnecting in case of failure. The ABRP allows intermediate nodes to receive and transmit the same request packet as obtained from the source to gather more information for establishing backup nodes. ABRP includes three

phases, *route discovery*, *backup node setup* and *route maintenance*, requiring three kinds of cache, *RD_request_Cache*, *Backup_Routes_Cache* and *Fresh_Routes_Cache* (see Fig. 1). The *RD_request_Cache* of a node is used to store temporary routing information in the route discovery phase. The *Backup_Routes_Cache* is used to store backup routes. The *Fresh_Routes_Cache* is used to store the fresh routes after a data transmission process is finished. Furthermore, Table 1 shows some basic protocol packets that need to be defined for the ABRP. The three phases are as follows.

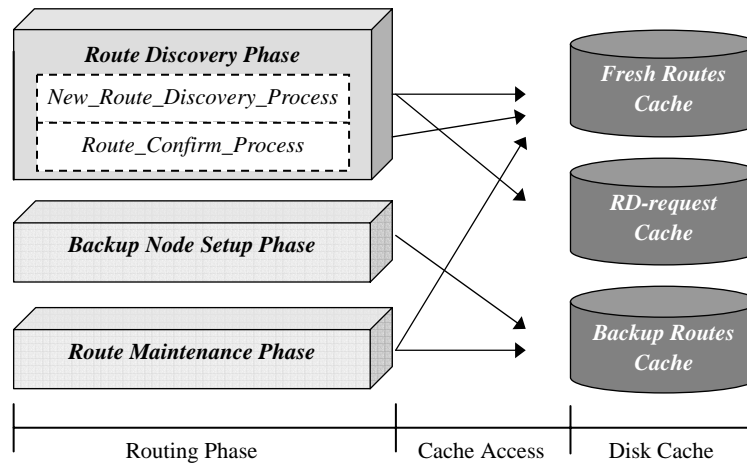


Fig. 1. The main architecture of the ABRP.

Table 1. The main protocol packets of the ABRP.

Packet Name	Function	Main Fields
Route Discovery Request (RD-request)	To find and record the route content from the source node to the destination node.	Sequence Number, Route Content
Route Confirm Request (RC-request)	To re-establish the path from the source to the destination node according to the fresh route in the <i>Fresh_Routes_Cache</i> .	Sequence Number, Route Content
Route Discovery Reply (RD-reply)	The destination node replies with the route content back to the source.	Sequence Number, Route Content
Backup Setup Packet (BS-packet)	The destination node transmits the backup information to the backup nodes to setup backup routes.	Sequence Number, Backup node, Backup Route
Route Erasure Request (RE-request)	The source node announces other nodes to erase routing information if not needed.	Sequence Number, Route Erasure
Link_Fail_Message	When a link fails, the message is used to announce the backup node along the route to replace a backup route.	Sequence Number, Route Content

3.1 Route Discovery Phase

When source node S requires the route to destination D , S enters the route discovery phase and checks whether adequate “fresh” routes to D are already available in the *Fresh_Routes_Cache* (see Fig. 2). If some “fresh” routes to D in *Fresh_Routes_Cache* are found, S runs *Route_Confirm_Process*. Otherwise, S runs *New_Route_Discovery_Process* to find a new route to the destination node.

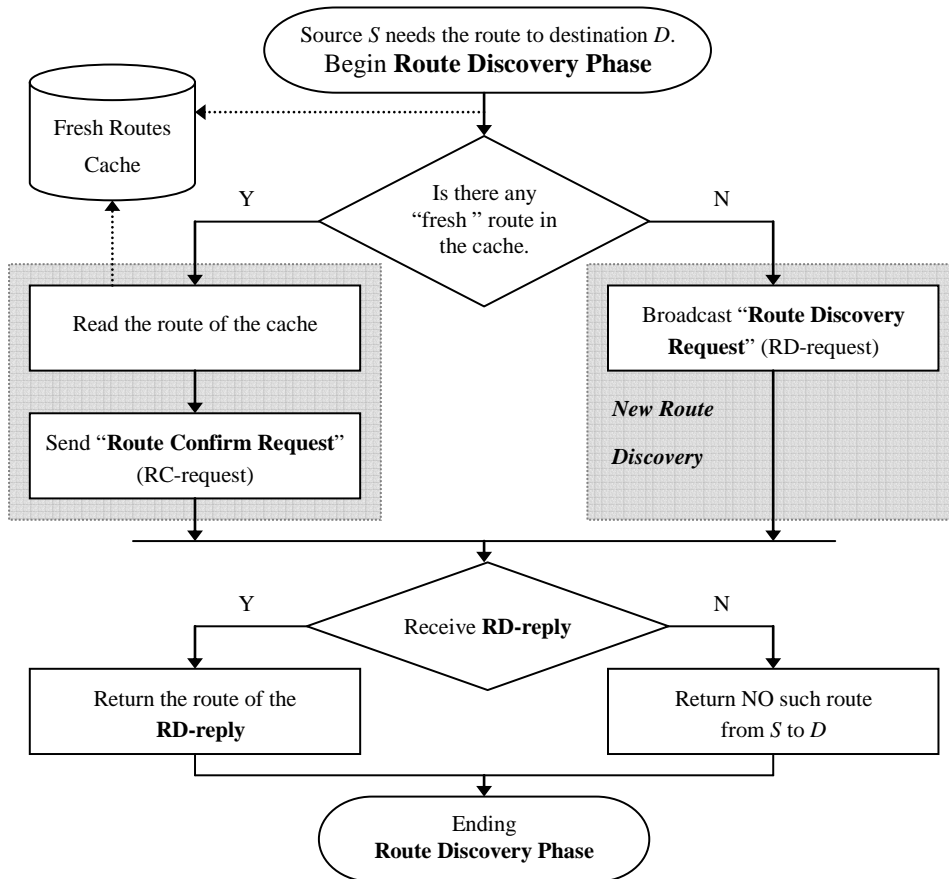


Fig. 2. Route discover phase of ABRP.

3.1.1 New_Route_Discovery_Process

Source node S broadcasts **RD-request** to nearby nodes. **RD-request** includes a sequence number field to distinguish the route discovery process from others, and a route content field for addresses along the path from S to D . After the intermediate node receives **RD-request** from an upstream node X , it inserts its address into the route content field of the **RD-request**, and then sends this modified **RD-request** to its neighboring

nodes (excluding the upstream node X). The *RD-request_Cache* of the intermediate node also records the information, including the sequence number of the **RD-request** and which neighboring nodes are sent.

If a node receives the **RD-request** with the same sequence number from its neighbor Y , then it checks whether the route content of **RD-request** includes its address; if so, the node discards this **RD-request**. Otherwise, the node inserts its address into the **RD-request**, and then checks whether Y is in the *RD-request_Cache*. If so, the node clears Y from the *RD-request_Cache*, and then forwards the **RD-request** again to its neighboring nodes specified in the *RD-request_Cache*. If Y is not in the *RD-request_Cache*, the node forwards the **RD-request** to the neighbor nodes in the *RD-request_Cache*. If *RD-request_Cache* includes no record of downstream neighboring nodes, the **RD-request** will be discarded.

Unlike the DSR, a node only discards the duplicate **RD-request** when its *address* is present in the route content field of the **RD-request** or when no other downstream node is recorded in its *RD-request_Cache*. Hence, the ABRP can prevent an infinite loop. After the destination node D receives a **RD-request**, D sends the route content of the first-arriving **RD-request** back to the source node S , and waits a short period for more **RD-requests** for a short period, before entering the backup node setup phase.

3.1.2 Route_Confirm_Process

If a “fresh” route is available to the destination in the *Fresh_Routes_Cache*, the source node S adds the fresh route from S to D to the **RC-request**, and then transmits **RC-request** along this route. When it receives the **RC-request**, an intermediate node checks its *Fresh_Route_Cache* to determine whether any other “fresh” route to D is included. If a “fresh” route is available, the node copies **RC-request** and puts the route in the route content field of the **RC-requests** before transmitting the **RC-request** along this fresh route. If no “fresh” route is available, **RC-request** is transmitted downstream according to its original route content field. Eventually, after D receives the **RC-request**, **RD-reply** is sent back to S , and S sends packets by this original route.

3.2 Backup Node Setup Phase

After the route discovery phase, the destination D may gather many routes within a period T_c . The nodes of those routes which D received are compared pair wise from beginning to end to find whether any two paths have a section in common. The final node, excluding destination D , in such a section is the “*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-packets** that include each backup node and the partial path from the backup node to the destination node are generated. The destination node then uses **BS-packet** to separately set up the *Backup_Route_Cache* of those backup nodes, where 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. 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

When a link fails, a node cannot continue to transmit. The node sends an alert message, *Link_Fail_Message*, to an upstream node along the reverse current route. The *Link_Fail_Message* is used to announce the backup node along the route to replace the backup route. The alert message will not be passed by an upstream node until the message is returned to a backup node. After the backup node receives the message of a link failure, 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, node *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 has lost the identity of the backup node. Under such circumstances, no backup node exists. The source node will receive the link failure message and re-enter the route discovery phase to establish a new route to the destination.

After the destination node replies with a path back to the source as the current route for sending data packets, some backup routes are established and stored in the backup nodes. If the current route is still alive, the situation that any node along the backup routes moves will not influence the communication of the current route. If the current route is broken and replaced by a backup route, the ABRP can still work even though a section of this backup route has failed. That is because the link which failed will be detected and an alert message will be sent to find another backup node.

When *S* does not have the route to *D*, *S* will store the usable routes into the *Fresh_Routes_Cache* and broadcast **RE-request** to announce all backup nodes that this data transmission process is ending. The **RE-request** packet contains the sequence number of this transmission process for distinguishing it from the other processes. When the backup node receives **RE-request**, it will also save remnant backup routes from *Backup_Route_Cache* in the *Fresh_Routes_Cache*.

3.4 An Example of ABRP

The process and the principle of ABRP are described above. This section presents an example to show how the ABRP performs in a practical case (see Fig. 3). When

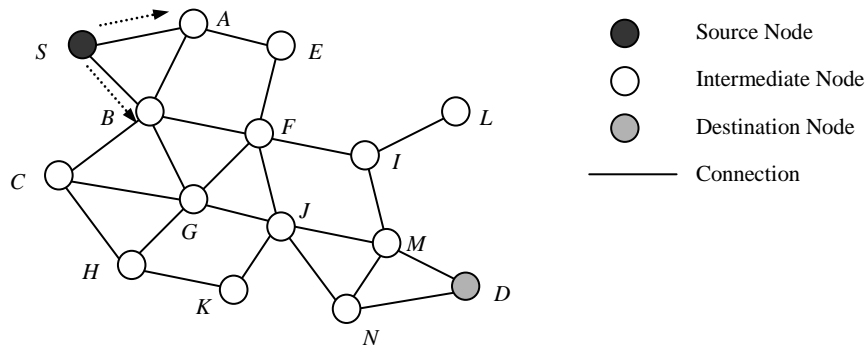


Fig. 3. An example of ABRP.

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-request** with a sequence number, $\#SN$, to its neighboring nodes.

When an intermediate node (F for example) receives the **RD-request** (from B for example), F checks whether its address is recorded in the route content field of the **RD-request**. If not, F inserts its address into the route content field and sends the **RD-request** to its neighboring nodes E , G , I and J , after recording \langle sequence number, downstream node \rangle pair into its **RD-Request_Cache** (which is in this case, $\langle\#SN, E\rangle$, $\langle\#SN, G\rangle$, $\langle\#SN, I\rangle$, and $\langle\#SN, J\rangle$). If F receives the **RD-request** with the same $\#SN$, without its address in the route content field from one node G , then F will also insert its address into the route content field and send this **RD-request** to the nodes E , I , and J in the **RD-Request_Cache**, but not to G since an infinite loop will occur if F sends **RD-request** to G again. F will discard the **RD-request** if **RD-request** already contains F or has no \langle sequence number, downstream node \rangle pair that is recorded in the **RD-Request_Cache**.

When destination node D receives the first-arriving **RD-request**, node D will reply with 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 T_c , D may gather some more routes from S to D , as follows:

$$\begin{aligned} S &\rightarrow B \rightarrow F \rightarrow I \rightarrow M \rightarrow D \\ S &\rightarrow B \rightarrow F \rightarrow J \rightarrow M \rightarrow D \\ S &\rightarrow B \rightarrow G \rightarrow J \rightarrow M \rightarrow D \\ S &\rightarrow A \rightarrow E \rightarrow F \rightarrow I \rightarrow M \rightarrow D \\ S &\rightarrow A \rightarrow B \rightarrow F \rightarrow J \rightarrow N \rightarrow D \end{aligned}$$

Those routes the destination D received are compared pair wise from beginning to end to find the set of backup nodes, $\{S, A, B, F, J\}$, and the sectional paths in common. Notably, source node S is also a backup node, because S can send packets to destination D in two ways. Then, node D sends **BS-packet**, which carries information about backup routes to each backup node. In this case, F will be a backup node and store $\{F \rightarrow I \rightarrow M \rightarrow D, F \rightarrow J \rightarrow M \rightarrow D, F \rightarrow J \rightarrow N \rightarrow D\}$ in its **Backup_Route_Cache**.

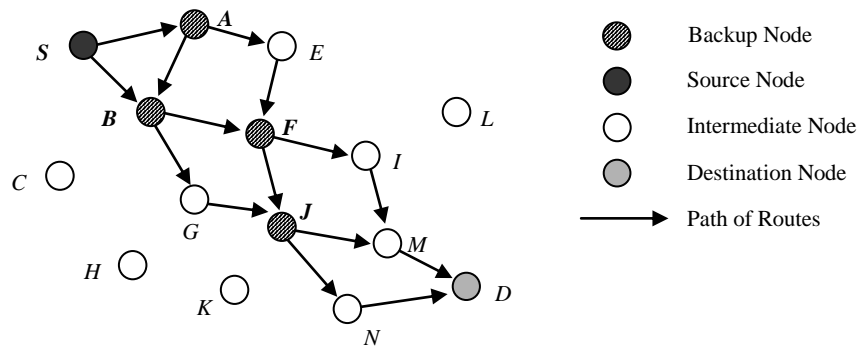


Fig. 4. An example of a route tree and backup node subset in ABRP.

After source node *S* receives **RD-reply** from destination node *D*, it begins to send packets to *D*. When a link fails while packets are being sent, some nodes cannot send the packet to downstream nodes by the current path. These nodes will not pass link failure messages to upstream nodes until such messages are returned to the backup node (See Fig. 5). For example, node *I* leaves and a link fails from *I* to *M*; node *I* then sends an alert message back to backup node *F*. Then, 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 backup route, whereupon *F* loses its identity of a backup node.

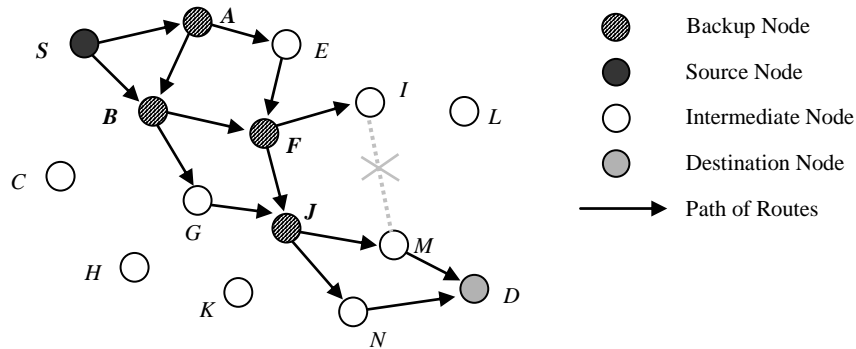


Fig. 5. An example of link failure.

4. THE ALGORITHMS OF ABRP

The details of ABRP algorithms are spelled out in this section. Notably, T_c is a critical parameter that influences the performance of ABRP. A small value for T_c will reduce the number of analyzable routes the destination node gathers, and will be helpless for reconnecting when a link fails in the route maintenance phase. If T_c is too large, some redundant backup routes are processed, and some delay in the route maintenance phase arises. Also, the control message overhead in the network increases because of too many **RD-requests**. To prevent the intermediate nodes from receiving and sending redundant **RD-requests** during the route discovery phase, they must stop receiving any **RD-request** when the destination node enters the backup node setup phase. There are some other ways for an intermediate node to stop receiving **RD-requests** as follows:

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

The first approach increases the network overhead, and the second is extremely difficult to implement. The last way is the most convenient to implement, requiring only that an adequate T_c be chosen for the ABRP. Hence, we use the third way for designing

the algorithms of ABRP. In ABRP, each node receives only **RD-request** from in a period T_c . The main ABRP algorithms of each phase, which are route discovery, backup node setup and maintenance, are illustrated separately in the following.

4.1 Algorithms of Route Discovery Phase

When a source node requires a route to destination D , source node S enters the route discovery phase. In the route discovery phase, the processes can be divided into three types including source node operation, intermediate node operation and destination node operation. During the operation of the source node, node S checks whether “fresh” routes to D are available in the *Fresh_Routes_Cache* first. If some “fresh” routes are found, node S runs $\text{ROUTE_CONFIRM}(S, D, \text{FreshRoute})$ to re-establish a route to D , where *FreshRoute* is a set of fresh routes fetched from the *Fresh_Routes_Cache*. Otherwise, S runs $\text{NEW_ROUTE_DISCOVERY}(S, D)$ to find a new route to the destination node.

The function $\text{ROUTE_CONFIRM}(S, D, \text{FreshRoute})$ is used to check whether the routes of *FreshRoute* which are passed by the protocol $\text{SOURCE_NODE_ROUTE_DISCOVERY}(S, D)$ are still available. The source node S starts a timer and delivers **RC-requests** which contain and pass along each route of *FreshRoute*. During the period, S waits for the **RD-reply** packet from destination D . If S receives a **RD-reply** in the period, the route of **RD-reply** is returned to S , and the route from S to D is re-established. Otherwise, the function $\text{NEW_ROUTE_DISCOVERY}(S, D)$ is to be executed to find a new route to the destination node.

In the function of $\text{NEW_ROUTE_DISCOVERY}(S, D)$, a timer is started and **RD-request** packets are broadcast to the neighboring nodes of S . During the period, S waits for the **RD-reply** packet from D . If S receives a **RD-reply** in the period, the route of **RD-reply** is returned to S immediately; that is, the route from S to D is re-established. Otherwise, an error message, that no route from S to D was found, is returned to S .

The details of $\text{SOURCE_NODE_ROUTE_DISCOVERY}(S, D)$ are as follows:

Protocol $\text{SOURCE_NODE_ROUTE_DISCOVERY}(S, D)$

```
// S: source node
// D: destination node
// FreshRoute: a set of fresh routes from S to D
{
  if (There are any fresh routes from S to D in Fresh_Routes_Cache) then
    FreshRoute = all the fresh route from S to D;
    return ROUTE_CONFIRM(S, D, FreshRoute);
  else
    return NEW_ROUTE_DISCOVERY(S, D);
  end if
}
```

Procedure $\text{ROUTE_CONFIRM}(S, D, \text{FreshRoute})$:

```
// FRi: a fresh route of FreshRoute
{
```

```

start the timer;
for (each  $FR_i$  of FreshRoute)
    send RC-request with fresh route to  $D$ ;
next
while (not timeout or receive RD-reply)
    wait for RD-reply;
end while
If (receive RD-reply) then
    return the route of RC-request;
else
    return NEW_ROUTE_DISCOVERY( $S, D$ );
end if
}

```

Procedure NEW_ROUTE_DISCOVERY(S, D)

```

{
start the timer;
broadcast RD-request;
while (not timeout or receive RD-reply)
    wait for RD-reply;
end while
If (receive RD-reply) then
    return the route of RC-request;
else
    return ERROR;
end if
}

```

During operation of the intermediate node, an intermediate node may receive a **RD-request** or a **RC-request**. When an intermediate node receives a first-arrival **RD-request** from an upstream node X , it executes RECEIVE_RD-request (RD-request) and starts the timer and modifies the **RD-request**, which inserts this node's address, $NodeAddr$, into the route content field of the **RD-request**. The intermediate node then sends this modified **RD-request** to its neighboring nodes (excluding the upstream node X). Some $\langle \#RD, N_i \rangle$ pairs are recorded in the **RD-request_Cache** of this intermediate node, where $\#RD$ is the sequence number of the **RD-request** and N_i is each neighboring nodes are sent. The $\langle \#RD, UpNode \rangle$ pair is also stored in the **RD-request_Cache**, where $UpNode$ is the node's address where the RD-request comes from.

If an intermediate node receives the **RD-request** again and it has the same $\#RD$ and is sent from its neighbor Y , then it will discard the **RD-request**; $NodeAddr$ is in the RD-request, **RD-request_Cache** is empty and timeout. Otherwise, the node modifies the **RD-request**, which inserts its address, $NodeAddr$, into the **RD-request**. Then, the intermediate node deletes the $\langle \#RD, Y \rangle$ pair if this pair is in the **RD-request_Cache**, and delivers the modified **RD-request** to the remaining neighboring nodes of **RD-request_Cache**.

The details of RECEIVE_RD-request (RD-request) is as follows:

```

Protocol RECEIVE_RD-request (RD-request)
// #RD: sequence number of RD-request
// NodeAddr: address of this intermediate node
// DownNode: a set of downstream neighboring nodes
// UpNode: node where the RD-request comes from
// Ni: a variable of node
{
  if (this RD-request is received for the first time) then
    start the timer;
    insert the NodeAddr into the RD-request;
    for (each node Ni of {DownNode – UpNode})
      send the RD-request to Ni;
      insert <#RD, Ni> pair into RD-request Cache;
    next
    insert <#RD, UpNode> pair into RD-request Cache;
  else
    if (not timeout) then
      if (NodeAddr is in the RD-request or RD-request Cache is empty)
        discard this RD-request;
      else if (<#RD, UpNode> is in RD-request Cache) then
        delete <#RD, UpNode>;
      end if
      insert NodeAddr into the RD-request;
      for (each node Ni of <#RD, DownNode> in RD-request Cache)
        deliver the RD-request to Ni;
      next
    else
      discard this RD-request;
      delete each <#RD, DownNode> in RD-request Cache;
    end if
  end if
}

```

In addition, when an intermediate node receives a **RC-request**, it executes RECEIVE_RC-request(RC-request) to check if there is any fresh route to D in its *Fresh_Routes_Cache*. If so, the intermediate node fetches those fresh routes and duplicates the **RC-request**. The original partial route of the duplicated **RC-request** is replaced by the fresh route from this node to D . The intermediate node then sends the original **RC-request** and any duplicated **RC-requests** along their routes.

The details of RECEIVE_RC-request (RC-request) is as follows:

```

Protocol RECEIVE_RC-request (RC-request)
// PR: partial route from this node to D of the RC-requests
// FreshRoute: a set of fresh routes from this node to D
// FRi: a fresh route of FreshRoute
{
  PR ← partial route from this node to D of the RC-requests;
  send the RC-request along the PR;
  if (there are any fresh routes to D in Fresh_Routes_Cache) then
    FreshRoute = all the fresh route from S to D;
    for (each FRi of {FreshRoutes – PR})
      RC-request' ← the duplicate of RC-request;
      replace the PR of RC-request' with FRi;
      send the RC-request' along the FRi;
    next
  end if
}

```

After destination node *D* receives a **RD-request** or a **RC-request**, node *D* executes DESTINATION_RECEIVE_request (RD-request or RC-request) to send the route content of the first-arriving **RD-request** or **RC-request** back to source node *S*, and starts a timer in order to wait for more routes. After one period, the destination node enters the backup node setup phase.

The details of DESTINATION_RECEIVE_request (RD-request or RC-request) is as follows:

```

Protocol DESTINATION_RECEIVE_request (RD-request or RC-request)
// #RD: sequence number of RD-request
// Route[ ]: array for storing the routes of RD-requests
{
  start the timer;
  i ← 1;
  while (not timeout)
    Route[i] ← route string of the RD-request or RC-request;
    if (i = 1) then
      reply RD-reply with Route[1] to the source node;
    end if
    i ++;
  end while
  run BACKUP_NODE_SETUP(Route[ ]);
}

```

4.2 Algorithms of Backup Node Setup Phase

The backup node setup protocol, $\text{BACKUP_NODE_SETUP}(Route, \#RD)$, has two procedures: $\text{FIND_BACKUP_NODE}(Route)$ and $\text{SETUP_BACKUP}(Backup, \#RD)$, where $Route$ is an array of routes the destination received and $Backup$ is an array of backup information including backup node and partial backup routes from backup node to destination.

The goal of $\text{FIND_BACKUP_NODE}(Route)$ is used to compare pair wise (from beginning to end) to find the backup nodes and the partial backup routes. After the $\text{FIND_BACKUP_NODE}(Route)$ is executed, the information of backup routes and backup nodes will be returned. According to the returned information, the function $\text{SETUP_BACKUP}(Backup, \#RD)$ then sends the **BS-packets** that include each backup node and the partial path from the backup node to the destination node to separately set up the *Backup_Route_Cache* of those backup nodes.

The details of protocol $\text{BACKUP_NODE_SETUP}(Route, \#RD)$ are spelled out as follows:

Protocol $\text{BACKUP_NODE_SETUP}(Route[], \#RD)$

```
// Backup[ ]: an array of backup information, Backup[ ].node is the field of backup node
// and Backup[ ].route is the partial backup routes from backup node to destination
// Route[ ]: an array for storing the routes of RD-requests
// #RD: sequence number of RD-request
{
  Backup[ ][ ] ← FIND_BACKUP_NODE(Route[ ]);
  run SETUP_BACKUP(Backup[ ],#RD);
}
```

Procedure $\text{FIND_BACKUP_NODE}(Route[])$

```
// 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]) then
```

```

while (Route1[flag1] = Route2[flag2] and
        (flag1 < length of Route1[ ] or flag2 < length of Route2[ ]))
    flag1 ← flag1 + 1;
    flag2 ← flag2 + 1;
end while
if (flag1 <> length of Route1[ ] and flag2 <> length of Route2[ ]) then
    BackupNodeTemp ← Route1[flag1];
    BackupRoutTemp ← path from the (flag1 – 1)th node
                       to the end of RouteTemp1;
    if (BackupRoutTemp is not in Backup[ ].route) then
        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) then
        Backup[ ].node ← BackupNodeTemp;
        Backup[ ].route ← BackupRoutTemp;
    end if
    end if
    flag2_start ← flag2;
    flag2 ← flag2 – 1;
end if
    flag2 ← flag2 + 1;
end while
    flag1 ← flag1 + 1;
end while
next
next
return Backup[ ];
}

```

```

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

```

After the backup nodes receive the **BS-packet**, they execute 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 RECEIVE_BS-packet(BS-packet) are spelled out as follows:

```

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

```

4.3 Algorithm for Maintenance Phase

When a link fails, a node cannot continue to transmit. That node will pass “*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 a backup node from *Backup_Route_Cache* and replaces the current route behind the backup node. The backup node also informs source node *S* of changing the route 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 does not have the identity of “BACKUP_NODE”, it will re-enter the route discovery phase to establish a new route to the destination.

```

Protocol LINK_FAIL(Link_Fail_Message)
// S: source node
// D: destination node
{
  if (is backup node) then
    fetch backup route from Backup Route Cache;
    instead the current route with backup route;
    send new route back to source node;
    label the backup route fetched as “Non_Backup_Route”;
    if (there is no “Backup_Route” in Backup Route Cache) then
      cancel the identity of “BACKUP_NODE”;
    end if
    Return the new backup route;
  else if (this node is source node but not “BACKUP_NODE”) then
    return NEW_ROUTE_DISCOVERY(S, D);
  else
    send Link_Fail_Message to the upstream node of the current route;
  end if
}

```

5. PERFORMANCE EVALUATION

5.1 The Simulation

To evaluate the performance of our ad hoc backup node setup routing protocol, we constructed a simulator using Java. This simulator allowed us to observe and measure the performance of ABRP under a variety of conditions. The parameters in our simulation are given as follows.

- The number of mobile hosts is 50.
- The mobility speed is from 0 to 30 kilometer per hour.
- Each transceiver has a range of 50 meters.
- Data transmission rate is 2 MB/sec.
- The size of a route discovery packet is 2 KB.
- The size of data packet is a constant 30 KB.
- The area of simulation is $500 \times 500 \text{ m}^2$.
- The simulation time is 200 seconds.

Each node is initially placed at a random position within the simulation area. Source nodes and destination nodes were chosen randomly with uniform probabilities. To simulate node mobility, each node randomly chooses a new location to move to and a velocity between 0 and 30 kilometer per hour at which to move there. If a moving node protrudes from the simulation area, its direction will change. All source nodes and destination nodes were chosen randomly in the simulation. A traffic generator was used for the sources to simulate constant bit rate packet delivery. After sending a packet to its next node, the packet is dropped if the sender does not receive an acknowledgement from the next node.

In the simulation, the routing protocol selection is DSR, DSDV and our proposed ABRP. 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 there is a link failure, the data transmission is broken before finding a new transmission route. The DSDV [2, 11] is a table-driven routing protocol based on the classical Bellman-Ford routing mechanism [13, 14]. 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 round 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 was 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. A comparison of some of the characteristics

Table 2. A summary of ABRP, DSR and DSDV.

Performance parameters	ABRP	DSR	DSDV
Route establishment	On-demand	On-demand	Table-driven
Route metric	Freshest and shortest path	Freshest and shortest path	Shortest path
Periodic messages	None	None	Route tables
Beaconing requirements	None	None	Yes
Loop-free	Yes	Yes	Yes
Backup routes	Yes	None	None
Table required	None	None	Yes
Cache required	Yes	Yes	None

of ABRP, DSR and DSDV are shown in Table 2. Moreover, we assigned the timer of ABRP a period of 20ms.

The performance metrics to be observed are:

- **Control Message Overhead:** the number of necessary control messages for all nodes in the network to maintain the routing table or for a source to establish and maintain a route to destination. Notice that the ABRP is to establish not only the route to the destination but also the backup routes. In this performance metric, two situations of ABRP are simulated. One is the situation that the number of control messages are gathered only while a source receives the RD-reply packet. The other is the situation that all the control messages in the simulation.
- **Data Throughput:** the number of data packets passed through the network in the simulation time.
- **Average Transfer Latency:** the average interval from the time the unicasting of a source node was initiated to the time this node finishing its unicasting.

5.2 Results

Fig. 6 shows the control message overhead incurred by DSR, DSDV and ABRP. Both DSR and ABRP have a constant increasing rate of control message overhead. The DSDV has an almost constant amount of control message overhead in varied mobility. Because of the control message overhead metric, two situations of ABRP were simulated. The ABPR1 in Fig. 6 shows the simulation result of the first situation when the number of control messages were gathered only while a source receives the RD-reply packet. The ABPR2 shows the simulation result of another situation where all the control messages in the simulation need an action. Because the ABRP provides the mechanism of backup routes and requires more RD-request packets for establishing backup routs, the control message overhead in both situations of ABRP is larger than DSR. Although the performance of ABRP in this metric is not as good as DSR, this result may be a challenge to the network traffic. The RD-request packets for a destination node to gather more routes do not spring up all over the network. And, this shortcoming can be improved if we adjust the value of the timer of ABRP.

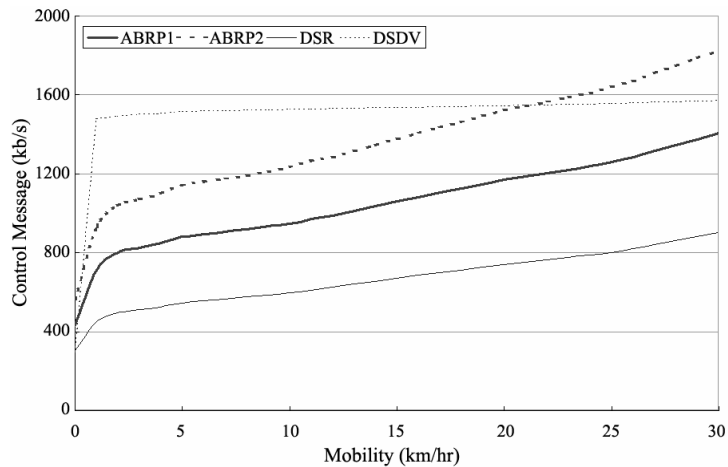


Fig. 6. Performance comparison of control message overhead.

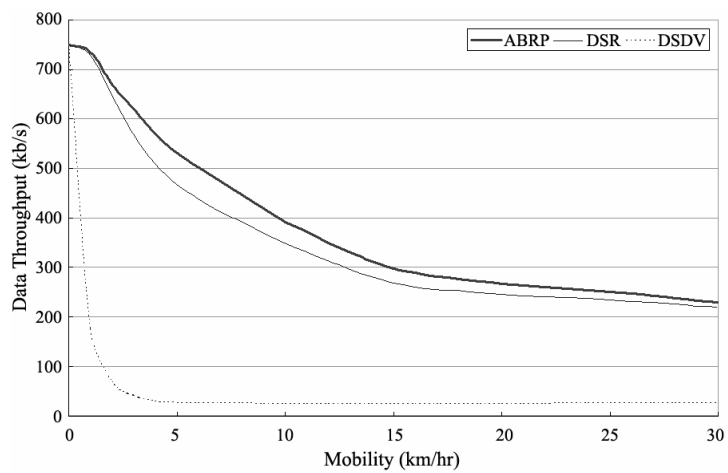


Fig. 7. Performance comparison of data throughput.

Fig. 7 shows the simulated result of data throughput. The poor performance of DSDV can be attributed to the frequent updating of control messages as mobility speed increases. The figure reveals that ABRP has a little higher throughput than DSR. In DSR, a route must be re-established when this route is broken, and it results in a decrease of throughput before a new route is established. Because the ABRP provides a mechanism for backup routs, the decrease of throughput when a link fails is not obvious. During the route failure, the data packets can be re-transmitted in a short time if a backup route is found and replaced.

Fig. 8 shows the average transfer latency comparison of DSR, DSDV and ABRP. Both DSR and ABRP have better performance than DSDV. In DSDV, the mechanism of table-driven routing results in frequent route reconnection when links fail, and the aver-

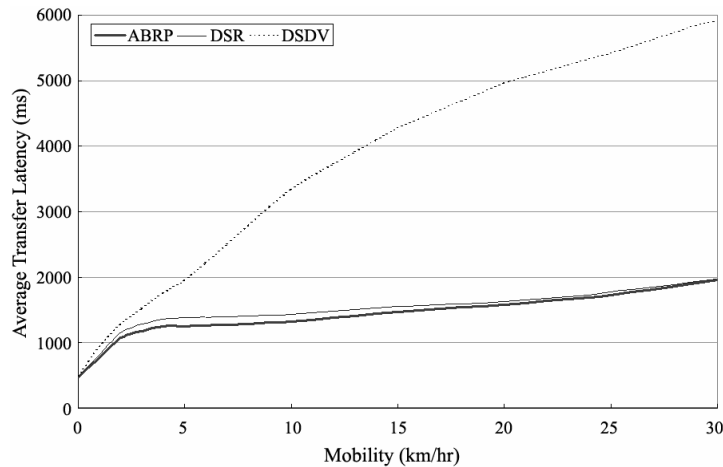


Fig. 8. Performance comparison of average transfer latency.

age latency of transmission increases as mobility speed increases. The ABRP has a better performance than DSR in low mobility because the ABRP provides the mechanism of backup routes for shortening the delay of reconnection when a link fails. When links fail, for an increase in mobility speed, the probability of reconnecting the transmission route successfully by using backup routes decreases. The resulting performance for average transfer latency is close to that for the DSR in high mobility. Increasing the period of the timer of the ABRP can increase the total number of backup routes and improve the performance of transfer latency. But, it will result in higher network traffic because of increasing control message overhead.

Furthermore, both of the on-demand routing protocols, ABRP and DSR, can provide the shortest routes during the route discovery phase. The time cost of ABRP and DSR for establishing a current route is approximate, and a route can be provided when the destination node receives the first request. However, in ABRP the destination node keeps on receiving more routes for establishing backup routes. The mechanism of backup routes that the DSR does not provide, can effectively improve the performance.

6. CONCLUSIONS

The ABRP is an on-demand routing protocol in a mobile ad hoc wireless network and addresses how to reconnect quickly when the transmission route fails. Our proposed ABRP protocol provides a backup node mechanism to reconnect quickly as required for ad hoc wireless networks. According to the proposed ABRP, many routes can be found to reach a destination in a given period. Those routes, almost always more than one, from the source node to the destination node can be analyzed to obtain some good backup routes to support reconnection when a link fails. In the route discovery phase of ABRP, the source node can acquire a route to the destination quickly, because a destination node is allowed to send back an **RD-reply** with information on the route for the first-arriving **RD-request**. The simulation illustrates that our proposed scheme has good performance

in an ad hoc wireless network.

The ABRP will be helpful in providing more reliable data transmission for some applications in the wireless environment, such as mobile learning, mobile commerce and mobile entertainment [15-17]. For ABRP, many ways of improving the stability of the routes can still be considered in the future [18]. For example, the intermediate nodes can choose the most stable connection to the downstream neighboring node first. Issues such as QoS and multicast will be addressed to enhance the capability of the ABRP. Moreover, supporting hierarchy and heterogeneous interfaces in ad hoc wireless networks [19] can also be considered in order to enhance the scale of application for ABRP.

REFERENCES

1. E. M. Royer and C. K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communication*, Vol. 6, 1999, pp. 46-55.
2. C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *Computer Communication Review*, Vol. 24, 1994, pp. 234-244.
3. C. C. Chiang, "Routing in clustered multihop, mobile wireless networks with fading channel," in *Proceedings of IEEE SICON '97*, 1997, pp. 197-211.
4. S. Murthy and J. J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *Mobile Networks and Application, Special Issue on Routing in Mobile Communication Networks*, Vol. 1, 1996, pp. 183-197.
5. C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)*, 1999, pp. 90-100.
6. D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad-hoc wireless networks," in T. Imielinski and H. Korth, (eds.), *Mobile Computing*, Kluwer Academic Publishers, Boston, 1996, pp. 153-181.
7. V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of the Conference on Computer Communications (IEEE INFOCOM '97)*, Vol. 3, 1997, pp. 1405-1413.
8. C. K. Toh, "A novel distributed routing protocol to support ad-hoc mobile computing," in *Proceedings of the IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, 1996, pp. 480-486.
9. R. Dube *et al.*, "Signal stability based adaptive routing (SSA) for ad-hoc mobile networks," *IEEE Personal Communication*, Vol. 4, 1997, pp. 36-45.
10. X. Hong, K. Xu, and M. Gerla, "Scalable routing protocols for mobile ad hoc networks," *IEEE Network Magazine*, Vol. 16, 2002, pp. 11-21.
11. A. Boukerche, A. Fabbri, and S. K. Das, "Analysis of randomized congestion control in DSDV routing," in *Proceedings of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000, pp. 65-72.
12. Y. J. Yi, M. Gerla, and T. J. Kwon, "The selective intermediate nodes scheme for ad hoc on-demand routing protocols," in *Proceedings of the IEEE International Conference on Communications*, Vol. 5, 2002, pp. 3191-3196.

13. L. R. Ford Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, 1962.
14. H. Li and D. Yu, "Comparison of ad hoc and centralized multihop routing," in *Proceedings of 5th International Symposium on Wireless Personal Multimedia Communications*, Vol. 2, 2002, pp. 791-795.
15. T. Plagemann, V. Goebel, C. Griwodz, and P. Halvorsen, "Towards middleware services for mobile ad-hoc network applications," in *Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2003, pp. 249-255.
16. M. Milenkovic, S. H. Robinson, R. C. Knauerhase, D. Barkai, S. Garg, A. Tewari, T. A. Anderson, and M. Bowman, "Toward internet distributed computing," *Computer*, Vol. 32, 2003, pp. 38-46.
17. E. Callaway, P. Gorday, L. Hester, J. A. Gutierrez, M. Naeve, B. Heile, and V. Bahl, "Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks," *IEEE Communications Magazine*, Vol. 40, 2002, pp. 70-77.
18. D. A. Maltz, J. Broch, J. Jetcheva, and D. B. Johnson, "The effects of on-demand behavior in routing protocols for multihop wireless ad hoc networks," *IEEE Journal on Selected Areas in Communications*, Vol. 17, 1999, pp. 1439-1453.
19. J. Broch, D. A. Maltz, and D. B. Johnson, "Supporting hierarchy and heterogeneous interfaces in multi-hop wireless ad hoc networks," in *Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms, and Networks*, 1999, pp. 370-375.



Ying-Hong Wang (王英宏) received the B.S. degree in Information Engineering from Chung Yuan University, Taiwan, in 1986 and the M.S. and Ph.D. degrees in Information Engineering from the Tamkang University, in 1992 and 1996, respectively. From 1988 to 1990, he worked in the Product Development Division of Institute of Information Industry (III). From 1992 to 1996, he was a lecturer in the Department of Information Engineering of Tamkang University. Beginning fall 1996, he is an Associate Professor in the Department of Information Engineering of Tamkang University.

He has approximately 100 technological papers published in International journals and international conference proceedings; he also joins many international activities: been on program committee, workshop chair, session chair and so on. He had been invited as Visiting Researcher at The University of Aizu, Japan, from January to March 2002. His current research interests are software engineering, multimedia database system, wireless multimedia, and mobile agent.



Chih-Chieh Chuang (莊智傑) was born in Kaohsiung, Taiwan on August 27, 1975. He received the B.S. degree from the Department of Computer Science and Information Engineering, Tamkang University, Taiwan. Currently, he is a Ph.D. candidate in the Department of Computer Science and Information Engineering at Tamkang University. His major research interests include wireless networks, ad hoc routing, mobile communication and knowledge management.