ADAPTIVE LINK CACHING FOR DYNAMIC SOURCE ROUTING - A SIMULATION STUDY

LIU YAODA

NATIONAL UNIVERSITY OF SINGAPORE

2003

ADAPTIVE LINK CACHING FOR DYNAMIC SOURCE ROUTING - A SIMULATION STUDY

LIU YAODA

(*B. Eng., Shanghai Jiaotong University, China*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgment

First of all, special thanks must go to my parents, especially my mum dwelling in my memory, for all the love and care they gave me during my journey of growing up.

I would like to express my gratitude to my supervisors, Dr. Jiang Shengming and Dr. Jiang Yuming for all their kindly help and patience through the days. The pleasure is certainly mine in having this opportunity to work under their guidance and learn from them. I really appreciate the numerous valuable advice and discussion with them.

Special thanks must go to Yih-Chun Hu from Carnegie Mellon University for his kindness of sharing his simulation model.

Also I would like to thank Institute for Infocomm Research for providing me with all facilities to carry out my research.

Work aside, I want to thank all my friends, especially my girl friend, Wei Na, for the happy hours.

Contents

List of Figures

List of Tables

Summary

In Dynamic Source Routing protocol (DSR) for Mobile Ad Hoc Networks (MANETs), caching is an important issue because it can make use of the known routing information to improve performance. In the design of caching schemes, the cache timeout, the time period that a link should stay in the cache, is very important and has high impact on the performance. However, only a few works have been done on how to determine the cache timeouts which can adapt to the change of link status.

In this thesis, an adaptive link caching scheme for DSR is proposed and evaluated through simulation. The proposed scheme suggests nodes to predict a link's lifetime, T_p, estimate the link availability, $L(T_p)$, and use $T_p \times L(T_p)$ to determine the cache timeout for the link before it is added into the cache. This cache timeout can reflect the future link status and is helpful in choosing reliable routes so that the performance of DSR can be improved.

Chapter 1

Introduction

1.1 Overview of routing in MANETs

In mobile ad hoc networks (MANETs), routing is among the most important and challenging subjects. Because MANETs are self-organizing, self-configuring and non-infrastructural, routing can only be done in a multi-hop manner, in which every node is acting not only as a node, but also as a router to forward packets for other nodes that are not within the wireless transmission range of each other. Because MANETs are bandwidth and energy constraint, routing in MANETs confronts many challenges.

Many different routing protocols have been proposed to provide MANETs with multi-hop routing solution, such as Destination- Sequenced Distance-Vector routing (DSDV) [17], Ad Hoc On-demand Distance-Vector routing (AODV) [16] and Dynamic Source Routing (DSR) [14]. In general, the current existing protocols can be categorized into two types: on-demand and table-driven routing protocols [7]. On-demand protocols attempt to find routes only when the routes are desired by a packet ready to be sent. When a source node requires a route to a destination node, a route discovery process is initiated by the source node. Once a route has been found, it is maintained until it is not required any longer or the maintaining node thinks the route is not usable any longer. On the other hand, table-driven protocols attempt to maintain consistent, up-to-date routing information for all nodes in the network. This is typically achieved through maintaining a set of routing tables and exchanging them among nodes. Changes in network topology brought about by mobility or node failures, are catered for by propagating updates throughout the network to maintain a consistent view.

Obviously, the wireless, mobile and multi-hop nature are the main causes of the complexity of routing in MANETs. Since no central administration entity exists, a node has to exchange routing information with other nodes so as to get enough information to perform the multi-hop routing, which is the origin of the signalling overhead. A lot of works have been done on how to exchange routing information. DSDV exchanges routing tables periodically. AODV deploys route discoveries to find routes and sends "hello" message periodically to maintain routes. DSR deploys route discoveries to find routes and maintains routes passively. More discussion on the routing protocols can be found in [7].

On the other hand, how to obtain and apply link status to routing in MANETs is gaining more and more attention, since it can help in maintaining routes and selecting more reliable routes. In [3] and [5], a metric called "associativity" is defined to reflect link status. Each node sends beacons periodically to signal the neighbors its existence; when receiving such beacons, the receiver increases the sender's associativity. In [8], signal stability and location stability are used to quantify link status. By measuring the signal strength of beacons, all neighbors are categorized into "strongly connected" and "weakly connected". The location stability is measured by recording the time period that the link has existed. The drawback of the above two schemes is that they do not make full use of the signal strength information. Actually, based on the obtained information of signal strength, nodes' relative velocities and distance can be estimated, and then a predicted lifetime of the link can be obtained. Several solutions have been proposed for such predictions. For example, in [13], nodes can predict links' lifetime by measuring the relative distance between the two nodes of the link. The drawback of the above scheme is that these metrics cannot reflect the possible change of link status if the nodes change their movement in the near future. A prediction-based link availability algorithm is proposed to consider the possible movement change in [9]. In this model, the link availability is defined as the time period in which an active link will continue to be available. The basic idea is to let nodes first predict a time period (T_p) that a currently active link will last from t_0 by assuming that both nodes of the link will keep their current movement unchanged. Then, the probability that the link will last from t_0 to $t_0 + T_p$ is estimated by considering the possible movement changes in the time period T_p. Lastly, a metric, $T_p \times L(T_p)$, is proposed to reflect the link availability, which is expected to reflect links' future status.

In this thesis, we study how to apply $T_P \times L(T_p)$ to provide the link caching schemes for DSR with adaptability.

1.2 Motivation

In [15], several caching schemes for DSR are studied, some are path cache, and the others are link cache. Path cache is simple for implementation and easy to manage. Link cache depends on some complex search algorithms to find the best path to the destination node, which is more difficult to implement and may require more CPU time to process. However, link cache has some strengths that we are more interested in.

First of all, link cache is more efficient in deploying the routing information obtained in the route discovery that costs a lot of bandwidth, power and CPU time. For example, if a link is observed to be broken, with path cache, a common and easy way is to remove all paths containing this link, which is not an efficient way to make use of the routing information. However, with link cache, if the same thing happens, we only remove the broken link and keep all other active links unaffected. It is obvious that link cache is better than path cache in two aspects. First, it can maintain the connectivity of the mobile ad hoc network when link breakage happens; and second, it can reduce the signaling overhead and delay caused by the route discovery for routing information which should be still available in the cache but unfortunately has been removed.

Second of all, link cache requires smaller cache capacity than path cache does. Theoretically, with a MANET of n nodes with the link cache strategy, the maximum number of links is $n \times (n - 1)$, which is the number of links that a node will possibly need to store. In this case, when the network size grows, the cache capacity demand for link cache may grow dramatically. However, this will happen only if all nodes can communicate directly with one another, which is always not the case in realistic MANETs. And active links will break because of the movement after certain period of time and such kind of link breakage will be observed and subsequently the broken link will be removed from the link cache. Besides, link cache can prevent the link cache becoming too big by giving every link a timeout after which the link will be removed from the link cache. The most important thing is that in link cache, an active link occupies only one memory entity at any time. However, in path cache a active link might be cached many times in different paths and therefore occupy multiple memory entities. In this sense, the link cache needs less cache capacity than path cache does.

Based on the above consideration, we decided to use link cache as the basis of our research.

For link caching in [15], the cache timeouts for links are determined in two ways. One is to set a single static timeout for all links. The other is to set different timeouts for different links based on a metric called link stability, which is increased when the link is used, and decreased when the link is observed to break. For the static scheme, it has been shown that either 5s or 10s are the optimal static timeouts in those cases studied and the optimal static timeouts for other scenarios could be different [15]. Since the timeouts can affect the performance a lot, an adaptive scheme which can predict the links' lifetimes is expected to perform better.

Inspired by [9] and [15], we consider to apply the metric “link availability” to maintain the route cache in DSR. In particular, we use link availability and predicted lifetimes to determine cache timeouts, which decide how long a link should stay in the link cache. Before a link is added into the link cache for DSR, its link lifetime is estimated as $T_p \times L(T_p)$ and then $lifetime + currenttime$ is used as the timeout for the corresponding link.

1.3 Methodology

To evaluate the proposed adaptive link caching scheme, a set of simulations have been conducted with NS-2 [4]. The following sections present the mobility models and the metrics used for performance evaluation. The static link caching scheme (link-static-T) [15] is also simulated for comparison, in which all links are cached for Ts.

The simulated MANET consists of 50 nodes, with 20 constant bit rate data connections in total, each transmitting at 4 packets of 64 bytes per second. A node can have at most 2 such connections. The simulation time is set to 900s and three space sizes, i.e., $700m \times 700m$, $500m \times 1500m$ and $1500m \times 1500m$, are simulated.

1.3.1 Mobility models

The following mobility models are adopted in the simulation.

- **Exponential random waypoint mobility model** [9]:

  The initial position and destination are selected uniformly over the allowed space, and the time for a node to reach the destination (i.e., epoch length) is exponential distributed with mean epoch, namely λ^{-1}. Then the node's speed is the distance divided by the epoch length. After reaching the destination, the node may pause for some time, and then repeats the above operations. For the homogeneous mean epoch cases, all nodes have the same mean epoch λ^{-1}. For the heterogeneous mean epochs, nodes may have different λ^{-1}. High mobility nodes have a smaller λ_1^{-1} and low mobility nodes have a larger λ_2^{-1}. Note that, exponential random waypoint mobility model is considered as a mobility model with exponentially distributed epochs only when the pause time equals to 0.

- **Random waypoint mobility model** [11]:

  The initial position and destination are selected uniformly over the allowed space, and a speed is selected uniformly over [0, v_{max}]. After arriving at the destination, a node waits at the destination for a pause time, and repeats the above operations.

- **Random Gauss-Markov mobility model** [12] and [15]:

  Under this mobility model, at the beginning of each deterministic interval nodes update their speeds as follows:

$$v_{x_t} = \alpha v_{x_{t-1}} + (1-\alpha)\overline{v}_x + R\sqrt{1-\alpha^2}, \tag{1.1}$$

$$v_{y_t} = \alpha v_{y_{t-1}} + (1-\alpha)\overline{v}_y + R\sqrt{1-\alpha^2}, \tag{1.2}$$

where $[v_{x_t}, v_{y_t}]$ is nodes' velocity in the interval t; R is a random variable normally distributed with mean 0 and variance δ_{v_x}; α is the weight of the velocity in the last interval. If a movement causes a node to move out of the space, the direction of the velocity is reversed.

- **Brownian mobility model**:

  Under this mobility model, nodes change speed and direction at discrete time intervals, such that at the beginning of each interval, each node chooses $r \in [0, v_{max}]$ and $\theta \in (-\pi, \pi]$ and moves with velocity vector $(r \sin\theta, r \cos\theta)$ during that interval. If this movement causes a node moving out of the allowed space, the node keeps the originally chosen velocity, but picks the intersection of the boundaries and the movement direction as destination.

1.3.2 Performance metrics

In [15], caching schemes are evaluated in terms of four performance metrics: packet delivery ratio, overhead in packets, end to end delay and path optimality. The first three metrics are also used here to evaluate the adaptive link caching scheme. In addition, path length is used to present the path optimality; overhead in bytes is used to show the overhead more precisely; number of route discoveries is used to present the number of route discoveries. The six performance metrics adopted are described as follows:

- **Packet delivery ratio (PDR)**: The fraction of packets sent by the "application layer"on a source node, which are received by the "application layer" on the corresponding destination node.

- **Overhead in packets (OiP)**: The total number of packets transmitted by the routing protocol, which does not include data packets.

- **End to end delay (DL)**: The average delay from the point when a packet is sent by the "application layer" on a source node until the point when it is received by the "application layer" on the corresponding destination node. It is computed only for packets that are successfully delivered.
- **Overhead in bytes (OiB)**: The total number of bytes transmitted by the routing protocol.
- **Number of route discoveries (NRD)**: The total number of route discoveries initialized by all nodes.
- **Path length (PL)**: The number of hops that a packet passes before it reaches the corresponding destination node.

In addition, a metric called **times of being next hop**, which is the number of times that a node is selected as an intermediate node to form source routes in DSR, is used to evaluate the adaptability of the adaptive link caching scheme to different mobility. Summing up the times of being next hop of all nodes belonging to the same class of mobility degree, the total times of being next hop of the class of nodes can be obtained. For example, when node S has a data packet addressed to node D and finds a route such as S->M->N->O->D. Suppose along this route, two nodes, M and N, are high mobility nodes; one node, O, is low mobility node. Then for this route, the times of being next hop of high mobility nodes is 2, and the times of being next hop of low mobility nodes is 1. Repeating the above operation, we can have the total times of being next hop of high and low mobility nodes. Note that we exclude the source node and destination node from the recording, because they are fixed and not affected by routing protocol.

1.4 Organization and contribution

The rest of the thesis is organized as follows:

Chapter 2 introduces the principle of the proposed adaptive link caching scheme for DSR and investigates its performance in an ideal situation. In this case, a node knows all information used to estimate link lifetime, which is further used to determine the time period that a link should stay in the link cache. Furthermore, a node is supposed to know the availability of other nodes immediately. This chapter tries to provide an overview of the proposed adaptive link caching scheme and its performance. Compared to the static link caching scheme (i.e., link-static-T) [15], the proposed adaptive link caching scheme can reflect the possible link status in the future and reduce the overhead generated for routing.

Chapter 3 proposes an adaptive link caching protocol for DSR and evaluates it with the exponential random waypoint mobility model. In reality, a node knows only the information used to estimate lifetimes of links between itself and its neighbors, and hence it can only estimate the cache timeouts for links between itself and its neighbors. To make nodes know the timeouts for other links, a scheme of exchanging timeouts among nodes is introduced into the protocol by appending the timeouts to the DSR header. Besides, to provide the protocol with adaptability to mobility models with non-exponential epochs, the ϵ measurement scheme [9] is implemented into the protocol.

Chapter 4 evaluates the proposed adaptive link caching protocol for some other mobility models whose epochs are not exponentially distributed, so as to show the adaptability of the protocol to other mobility models. We found that in most cases the protocol can adapt to mobility models and perform better than the static link caching scheme [15].

Chapter 5 concludes the thesis and discusses possible future research directions in this area.

The work in this thesis is also reported in [1] and [2].

Chapter 2

Adaptive link caching for DSR: an overview

In this chapter, we propose a new adaptive link caching scheme for DSR, which aims to manage the cached link information according to the possible link status in the future. In DSR, caching is an important source of routing information and probably the most effective way to make use of the routing information obtained through route discoveries. When it comes to cache, cache timeout, the time when a link will be removed from the cache, must be considered. However, only a few works have been done in this field. Before we present our solution, let us take a closer look at the cache timeout. When a link is about to be added into a link cache, a timeout value has to be assigned to the link. The ideal scenario is that nodes know the time point when the link will be broken at the time of adding it into the cache. Thus, this time can be used as cache timeout and the link information can represent the actual link status exactly. In reality, it is not possible to know future link status exactly in advance. So, what can we do? It is possible to obtain the historical and current link status. With these information, nodes can predict the future link status. Particularly, the lifetime of links can be predicted. Our idea is to estimate the lifetime of links first, and then set the cache timeout according to the estimated lifetime. In this chapter, we discuss the proposed scheme with a simple case in which the nodes know all information needed to predict the lifetime for every link. Based on this assumption, no timeout exchange is needed and all estimation can be done locally by the node itself. The proposed scheme is also analyzed with the simplest scenario as explained later. Firstly, it is evaluated in terms of some common performance metrics, such as packet delivery ratio, overhead, end to end delay. Secondly, its adaptability to mobility is discussed with a special scenario in which nodes move with different mobility.

2.1 Principle of adaptive link caching

This section introduces the principle of the adaptive link caching scheme. Basically, in this scheme a node does a prediction based cache timeout estimation for each link when it is about to add the link into its link cache.

Before we delve into the details on how to estimate cache timeouts, let us look at what information we have and what can be done. When a node receives data packets or beacons from a neighbor node, it can measure the signal strength and then estimate the distance and relative velocity [13]. More recently in [6], with the help of global positioning system, a node can also get information about the distance and relative velocity. Now that we have such movement information, what can be done? At first, we assume that nodes will not change the current movement. Based on this assumption, we can predict the future movement of the nodes with their historical movement and then predict the lifetime of links, T_p. Due to the dynamic nature of MANETs, T_p itself can not reflect the real lifetime of links especially in high mobility scenarios. Considering the possible movement change of nodes, the probability that the link will continue to be available in the predicted time period, $L(T_p)$, must be introduced. Particularly, given a currently available link and T_p estimated based on historical and/or current status of the link, $L(T_p)$ can be estimated. The estimation of $L(T_p)$ is based on the knowledge of the mobility model. For different mobility models, different $L(T_p)$ can be obtained for the same T_p. The combination of the predicted lifetime and the probability as $T_p \times L(T_p)$, can be expected to reflect the future link status better than the lifetime itself.

Now let us go further to the prediction of $T_p \times L(T_p)$. Assuming that nodes know all necessary information such as relative velocities and distances between all nodes, they can estimate the lifetime of links that they noticed and assign timeouts according to the estimated lifetimes. By assigning different timeouts to different links, which can reflect the actual lifetime of the link, nodes' cached link information can reflect the current link status better and then do better routing. In other words, links with longer lifetimes will stay in the link cache for longer time than those with shorter lifetimes, and at the same time, links with longer lifetimes will have higher probability to be selected to form source routes than those with shorter lifetimes. The scheme can improve the performance of the network, such as, reducing the overhead caused by broadcasting route requests and error rate caused by broken links. Simulation results that verify this will be shown in later sections.

The $T_p \times L(T_p)$ estimation procedure is as follows. Once a node notices an active link that does not exist in its own link cache, the lifetime of this link is estimated. Then the estimated lifetime value plus the current time is used as the cache timeout for the link. Finally, the link and its cache timeout are stored in the link cache for future use .

2.2 An application of adaptive link caching in DSR

This section presents an application of the above link caching scheme for DSR. Here, T_p is calculated using the measurement-based link lifetime prediction algorithm proposed in [13], which predicts the lifetime of the link by measuring the distance between the two nodes of the link. $L(T_p)$ is estimated using the Prediction-based Link Availability Estimation algorithm proposed in [9]. For cases in which all nodes have the same mean epoch λ^{-1}, called homogeneous mean epoch, $L(T_p)$ is estimated as

$$L(T_p) \approx (1 - e^{-2\lambda T_p})(\frac{1}{2\lambda T_p} + \epsilon) + \lambda T_p p e^{-2\lambda T_p}, \tag{2.1}$$

where, p is the probability that two nodes move closer after they change their movements, and ϵ is an adjustment to the link availability estimation.

In [10], this algorithm has been extended to support different mean epochs for the two nodes of a link, called heterogeneous epoch, as

$$\begin{aligned} L(T_p) &\approx \frac{e^{-(\lambda_1+\lambda_2)T_p}}{2(\lambda_1+\lambda_2)T_p}\{pT_p^2(\lambda_1+\lambda_2)^2 - 2T_p(\lambda_1+\lambda_2)\epsilon \\ &\quad -2+2e^{(\lambda_1+\lambda_2)T_p}[1+T_p(\lambda_1+\lambda_2)\epsilon]\}, \end{aligned} \tag{2.2}$$

where, ${\lambda_1}^{-1}$ and ${\lambda_2}^{-1}$ are the mean epochs of the two nodes of a link, and the other variables are the same as those in (2.1).

For simplicity, the ϵ can be set to 0, with which (2.1) is simplified to

$$L(T_p) \approx (1-e^{-2\lambda T_p})\frac{1}{2\lambda T_p} + \lambda T_p p e^{-2\lambda T_p}, \tag{2.3}$$

and (2.2) is simplified to

$$L(T_p) \approx \frac{e^{-(\lambda_1+\lambda_2)T_p}}{2(\lambda_1+\lambda_2)T_p}\{pT_p^2(\lambda_1+\lambda_2)^2 - 2 + 2e^{(\lambda_1+\lambda_2)T_p}\}, \tag{2.4}$$

There are two assumptions for the $L(T_p)$ estimation:

- Epochs are exponentially distributed with λ^{-1}.
- Node mobility is uncorrelated.

2.3 Simulation results

This section presents some simulation results with two kinds of mobility models and for simplicity the ϵ is set to 0 in this chapter. We first evaluate the adaptive link caching scheme with the exponential random waypoint mobility model. The homogeneous mean epoch case is studied in Section 2.3.1 and the heterogeneous mean epoch case is studied in Section 2.3.2. Lastly, the performance based on random waypoint mobility is presented in Section 2.3.3.

2.3.1 Homogeneous mean epoch

This section evaluates the proposed adaptive link caching scheme with homogeneous mean epoch, in which nodes move within the space of $500m \times 1500m$ according to the exponential random waypoint mobility model. For comparison, the results of the static link caching scheme (link-static-T) [15] is also presented. As shown in Figs. 2.1 and 2.2 , the adaptive link caching scheme performs as well as, if not better than link-static-T.

Fig. 2.1 (a) presents the results of the adaptive scheme in terms of packet delivery ratio. Among link-static-Ts, link-static-2 performs best in terms of packet delivery ratio, reaching 99.3%, a little higher than 98.9% achieved by link-static-5. Note that, the results in [15] reported that for the random waypoint mobility model [11], link-static-5 (i.e., static timeout equals to 5s) performs best in terms of packet delivery ratio. However, this does not stand here anymore. This shows that one single static timeout may not always work best. While the adaptive link caching scheme, without such static setting, performs a little worse than link-static-2, but better than all other link-static-Ts and achieves the packet delivery ratio of 99.1%.

Figs. 2.1 (b) and (d) present the performance in terms of overhead. Although link-static-2 performs best among link-static-Ts in terms of packet delivery ratio, it does not perform best in terms of overhead. It generates overhead of 58379 packets and 3003434 bytes, however, link-static-10 generates overhead of only 27435 packets and 1658613 bytes. Thus, a conclusion can be drawn that the static scheme with a specific static timeout may not performs best in terms of both packet delivery ratio and overhead. Besides, since with all static timeout simulated the static scheme can always achieve the packet delivery ratio higher than 95%, the performance in terms of overhead is more interesting to us. The same result has also been reported in [15]. However, it it interesting to find that the adaptive link caching scheme performs even better than link-static-10. It generates overhead of only 16871 packets and 988684 bytes, which is much less than those generated by link-static-10. The reason is explained below.

In DSR, overheads include two kinds of route request packets: route requests initialized by source nodes and those relayed by the intermediate nodes. In this scenario, as shown in Fig. 2.1 (d) compared to Fig. 2.2 (d), most of the overhead packets belong to the first class of route requests. For the static timeout of 10s, among about 17000 overhead packets, 14323 route requests belong to the first class. However, the adaptive scheme reduces the number of such overhead packets to 7595. To explain the decreasing of initialized route requests, let us look at the relationship between the timeout settings and the number of the first class of route requests. If a link that will actually exist for 20s is manually assigned the timeout of 5s, if in the last 15s, this link is needed in forming a route, a route request is initialized, which is not necessary if we set the timeout dynamically to

20s. If a link that will actually exist for 5s is manually assigned the timeout of 20s, if in the last 15s, this link is used in forming a route, route errors happen, which will not happen if we set the timeout dynamically to 5s. Since the number of route requests equals to the number of route discoveries, we conclude that the adaptive scheme can decrease the number of route discoveries so as to reduce the total overhead.

With a closer look at link-static-T's performance in packet delivery ratio and overhead, we find there is a tradeoff between these two performance metrics. With the increase of T, link-static-T performs worse and worse in terms of packet delivery ratio; but in terms of overhead it performs better and better until T reaches some value (10s in this scenario), after which it performs worse and worse. There seems to be some kind of best T with which link-static-T can perform fairly well in these two performance metrics for a specific scenario, for example, 4s in this case. A metric called "packet delivery ratio / overhead in bytes" can help us investigate the performance in these two metrics and find a best T. Larger value of this metric means better performance. As shown in Fig. 2.1 (c), the best static timeout is 4s. On the other hand, it is not easy, if possible, to find this T for every scenario. However, the proposed scheme does not need to worry about the setting of T and can achieve better performance than link-static-T with the best T.

Fig. 2.2 (a) presents the results in terms of end to end delay. We can see that the adaptive scheme also performs well in this metric. Particularly, with the adaptive link caching scheme, packets experience mean delay of 42ms, while

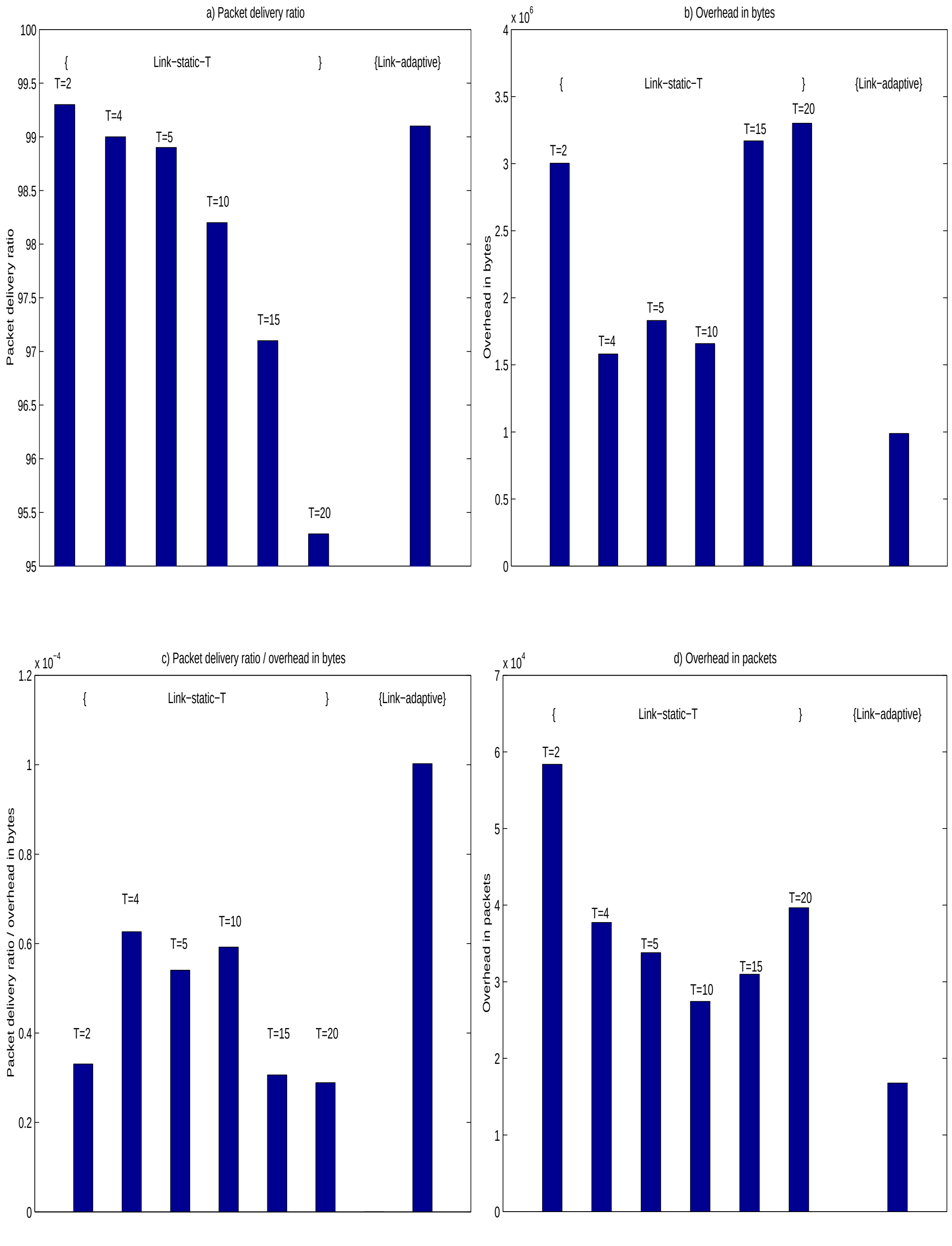

Figure 2.1: Overhead vs packet delivery ratio (exponential random waypoint mobility model (pause time = 0s), homogeneous mean epoch ($\lambda^{-1} = 60$), $500m \times 1500m$)

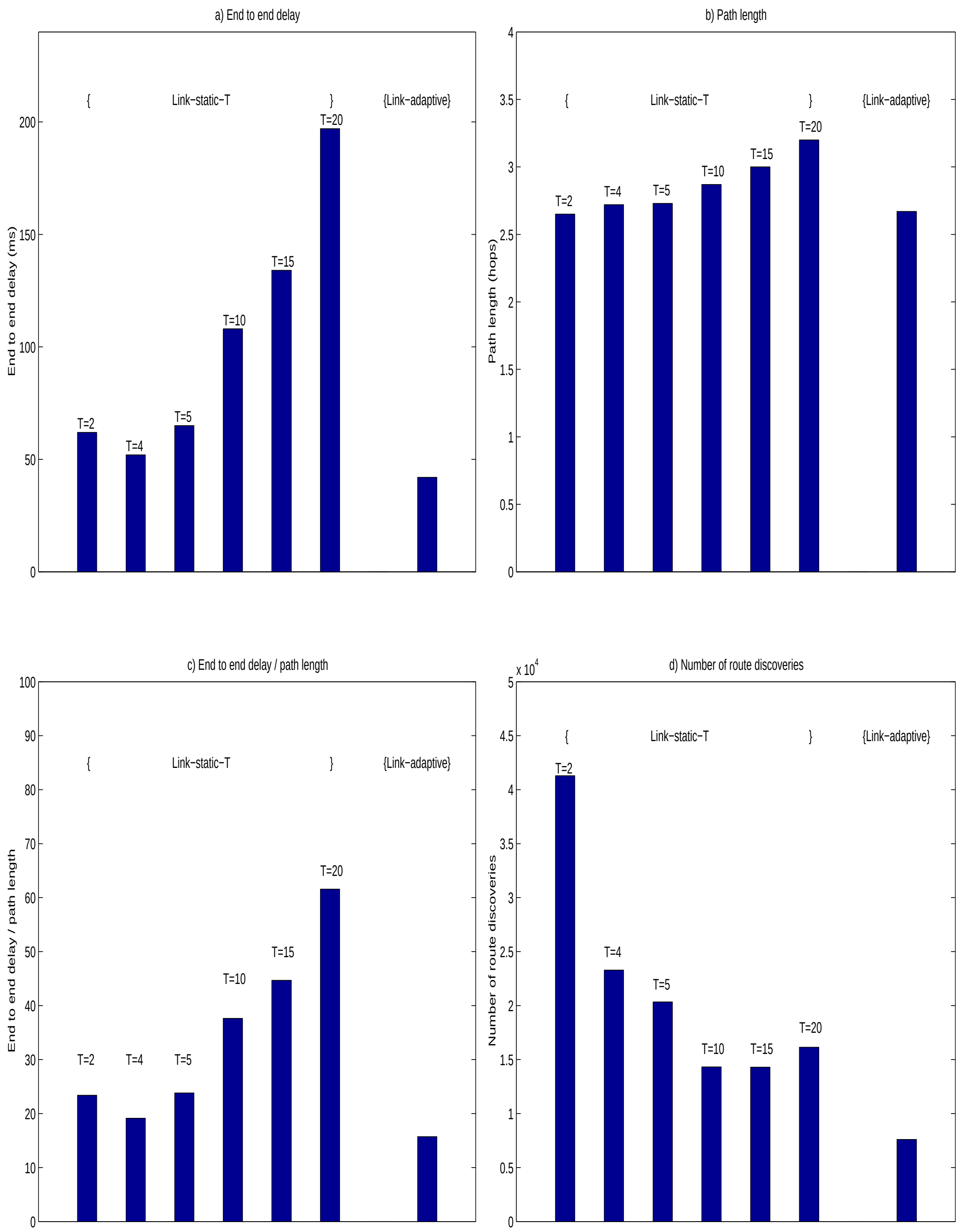

Figure 2.2: Delay vs path length (exponential random waypoint mobility model (pause time = 0s), homogeneous mean epoch ($\lambda^{-1} = 60$), $500m \times 1500m$)

with link-static-Ts, packets experience mean delays ranging from 52ms (at static timeout 4s) to 197ms (at static timeout 20s). The reason is as follows. DSR introduces two kinds of delay: the time spent in waiting for a route discovery to complete before a packet can be sent, and the time spent in detecting (through retransmission and acknowledgement) route errors and performing salvages. It is clear that the more route discoveries performed, the longer time spent in waiting for them to complete; the more route errors, the longer time spent in salvaging route errors. Since the adaptive scheme can reduce the number of route discoveries and route errors as we discussed above, it is not surprising that it can achieve better performance in delay.

Fig. 2.2 (b) presents the results in terms of path length. The adaptive scheme performs worse than the static scheme with a relatively small static timeout. The reason is as follows. For the static scheme, the smaller timeout set, the more route discoveries performed. With more route discoveries performed, more route information can be found and hence the a shorter path can be found. With a small static lifetime, such as 2s, although DSR does not perform well in terms of other metrics, it does get the latest routing information so as to get shorter paths. But with the rise of the static timeout, the performance of the static scheme becomes worse and worse.

Fig. 2.2 (c) presents the performance in terms of delay per hop. It is shown that the adaptive scheme can also reduce the delay per hop compared to link-static-Ts.

2.3.2 Heterogeneous mean epochs

This section discusses the adaptability of the adaptive link caching scheme to heterogeneous mean epochs. In heterogeneous mean epochs, nodes with large mean epoch, λ_2^{-1} are called low mobility nodes; those with small mean epoch, λ_1^{-1} are called high mobility nodes. Low mobility nodes cause less topological change than high mobility nodes and links consisting of low mobility nodes are more stable than those consisting of high mobility nodes. Although we do not change the routing selection scheme by using the adaptive link caching scheme, we can still expect it to choose better route since the proposed scheme makes the cached routing information reflect the link status better. Better routes here mean those that are more stable, which is more important than other metrics in wireless situation in some sense.

With different combinations of λ_1^{-1} and λ_2^{-1}, different time periods are simulated. For the case of $\lambda_1^{-1} = 60s$ and $\lambda_2^{-1} = 250s$, the simulation time is 900s; for the case of $\lambda_1^{-1} = 60s$ and $\lambda_2^{-1} = 500s$, the simulation time is 1500s.

Table 2.1 lists the distribution of nodes' times of being next hop. The results show that the adaptive scheme can adapt to node's mobility and select low mobility nodes as next hop more frequently. For both spaces, link-static-5 selects high mobility nodes slightly more frequently than low mobility nodes. However, using the adaptive scheme, low mobility nodes are selected much more frequently than high mobility nodes, for example, with the space of $700m \times 700m$, 72.95% of nodes selected as next hop are low mobility nodes.

Table 2.1: Distribution of times being next hop vs space sizes: heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$), exponential random waypoint mobility model (pause time=0s)

	$700m \times 700m$				$500m \times 1500m$			
	Static T=5s		$T_p \times L(T_p)$		Static T=5s		$T_p \times L(T_p)$	
	LMN	HMN	LMN	HMN	LMN	HMN	LMN	HMN
TBNH	1491.6	1658	1721.2	638.1	3053.4	3184.2	3350.5	1645.2
PBNH	47.36	52.64	72.95	27.05	48.95	51.05	67.07	32.93

(LMN = Low Mobility Node, HMN = High Mobility Node, TBNH = Times of being next hop, PBNH = Percentage of being next hop)

Table 2.2 presents the network performance for heterogeneous mean epochs. The adaptive scheme outperforms link-static-5, especially in terms of overhead in packets. For example, for the space of $700m \times 700m$, the adaptive link caching scheme generates only 3041 overhead packets, while link-static-5 generates 21418 overhead packets, more than 7 times of 3041. Since low mobility nodes cause less network topology change, in some sense, links consisting of low mobility nodes are more reliable than those consisting of high mobility nodes. By selecting low mobility nodes as next hop more frequently, the adaptive link caching scheme can be expected to achieve better performance than link-static-T.

Table 2.3 investigates the effect of mobility on nodes' being next hop. By setting $\lambda_1^{-1} = 60s$ for both scenarios and $\lambda_2^{-1} = 250s, 500s$ respectively, we found that the lower mobility the nodes are, the more frequently they will be selected

Table 2.2: Performance versus space sizes: heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$), exponential random waypoint mobility model (pause time=0s)

	$700m \times 700m$		$500m \times 1500m$	
	Static T=5s	$T_p \times L(T_p)$	Static T=5s	$T_p \times L(T_p)$
PDR	99.97	99.98	99.26	99.40
OiP	21418	3041	29716	12092
NRD	20176	2416	19238	5720
DL	7.983	7.960	50.3	33.7
PL	1.7543	1.7990	2.7366	2.734
OiB	926374	167548	1770297	703264

(For details on metrics, refer to Section 1.3.2)

as next hop.

From all the above results, we can make a conclusion that the adaptive scheme selects low mobility nodes as next hops more frequently so as to form more stable routes.

2.3.3 Random waypoint mobility model

Different from the last two sections, random waypoint mobility model is used to simulate the nodes' movement in this section. Note that non-exponential epoch is the feature of the random waypoint mobility model, while exponential epoch is an assumption of $L(T_p)$ estimation [9] which was part of our adaptive link caching

Table 2.3: Distribution of being next hop versus mobility: heterogeneous mean epochs ($\lambda_1^{-1} = 60s$), exponential random waypoint mobility model (pause time=0s), $1500m \times 1500m$

	$\lambda_2^{-1} = 250s$				$\lambda_2^{-1} = 500s$			
	Static T=5s		$T_p \times L(T_p)$		Static T=5s		$T_p \times L(T_p)$	
	LMN	HMN	LMN	HMN	LMN	HMN	LMN	HMN
TBNH	4208.8	4466.3	4107.3	3890.2	3053.4	2576.8	3350.5	1579.4
PBNH	48.52	51.48	51.36	48.64	54.23	45.77	67.96	32.04

(LMN = Low Mobility Node, HMN = High Mobility Node, TBNH = Times of being next hop, PBNH = Percentage of being next hop)

scheme. It is interesting to observe that the adaptive link caching scheme can perform well in this case. Figs. 2.3 and 2.4 present the performance of the adaptive scheme in comparison with those of link-static-T.

As mentioned in Section 2.3.1, the results in [15] reported that for random waypoint mobility model [11], link-static-5 (i.e, static timeout equals to 5s) performs best in terms of packet delivery ratio. The results are reproduced as shown in Fig. 2.3 (a), where link-static-5 achieves packet delivery ratio of 99.66%. However, our adaptive link caching scheme performs even better than link-static-5, achieving the packet delivery ratio of 99.77%.

The other results achieved by the adaptive scheme are similar to those in Section 2.3.1, thus here we only discuss the results briefly. In terms of metrics except for path length, the adaptive link caching outperforms link-static-T.

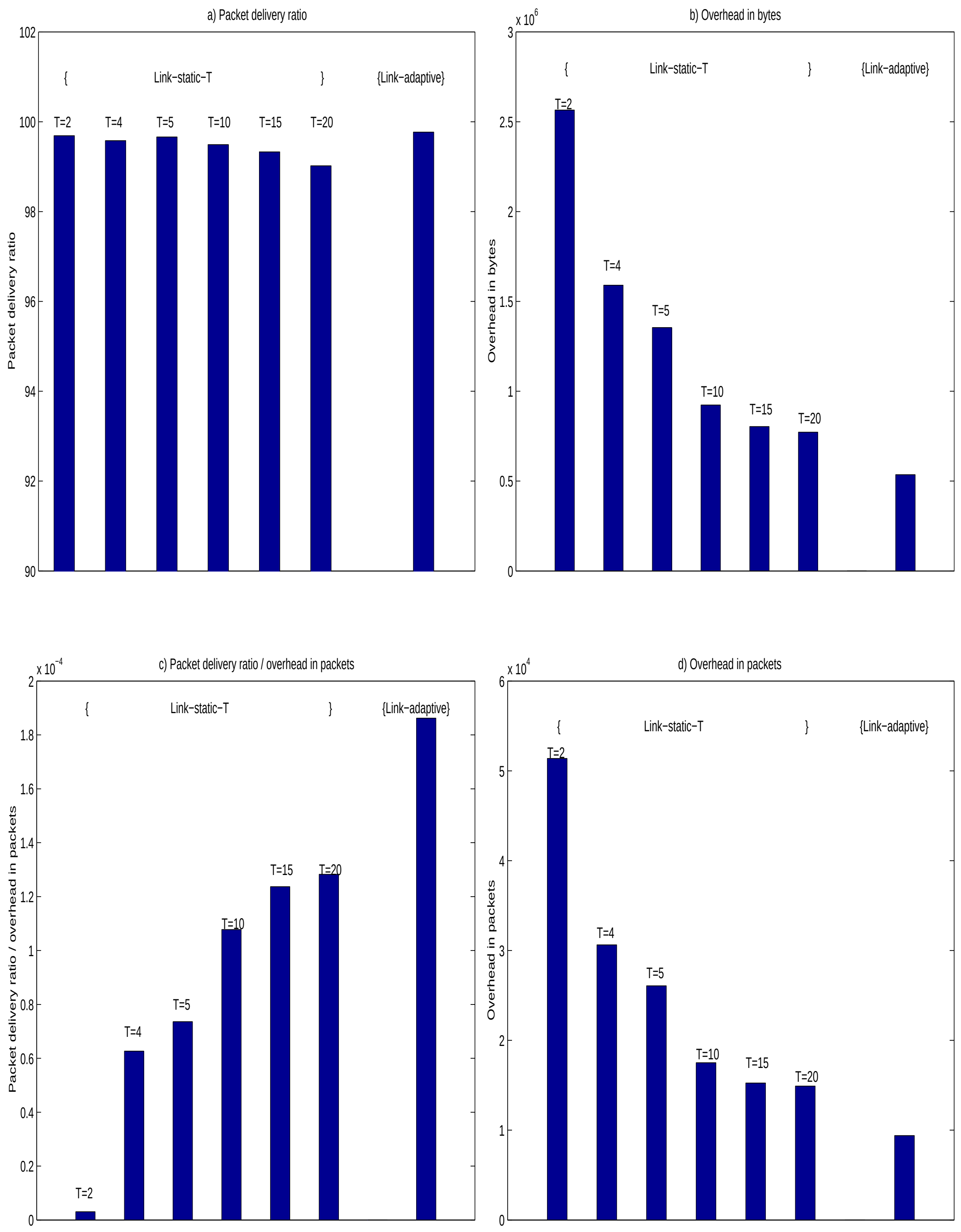

Figure 2.3: Overhead vs packet delivery ratio (random waypoint mobility model, $500m \times 1500m$)

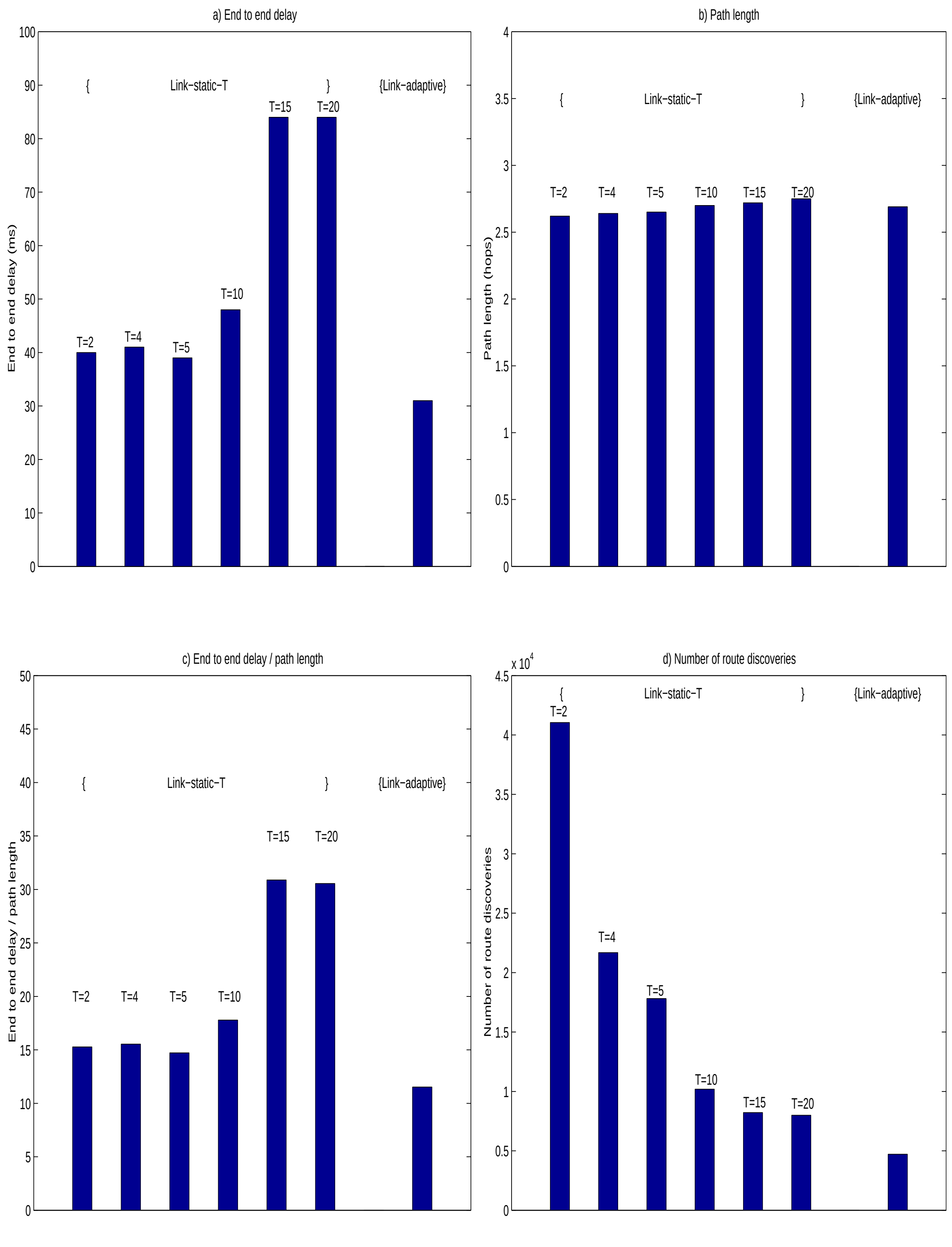

Figure 2.4: Delay vs path length (random waypoint mobility model, $500m \times 1500m$)

2.4 Conclusion

In this chapter, we propose and evaluate an adaptive link caching scheme for DSR in an ideal case, where nodes can know all information needed to estimate $T_P \times L(T_p)$ immediately. Compared to link-static-T [15], the adaptive link caching scheme achieves good performance in all metrics since it can choose more reliable routes by selecting low mobility nodes as next hop more frequently. Although the evaluation is not complete, it somewhat shows us the potential performance of the proposed caching scheme. An adaptive link caching protocol based on this scheme and a more detailed performance evaluation will be given in the following chapters.

Chapter 3

An adaptive link caching protocol for DSR

In this chapter, we develop an adaptive link caching protocol for the real situation in which a node only knows the relative distance and velocities of neighbor nodes. In this case, a node only estimates $T_p \times L(T_p)$ for links between itself and its neighbor nodes. Since cache timeouts for all known links are necessary, a mechanism to exchange the information between nodes is introduced in Section 3.1. In order to make the protocol more adaptive, an implementation of ϵ measurement [9] is also introduced in Section 3.2. The performance of this protocol is investigated in comparison with the results given in Chapter 2.

3.1 Protocol specification

This section introduces the adaptive link caching protocol. In this protocol, we first let nodes estimate the timeout for links between themselves and their neighbors. Then we add these timeouts into the DSR header so as to exchange them between nodes. Here each node maintains a list of timeouts, one timeout for one link in its own cache, which are used to manage the link cache.

3.1.1 Header format

The header in our protocol is the same as those in DSR except that some extra bytes are added to contain the timeout information. Wherever there is a route in the header, more clearly, in route request, route reply and data packets, the corresponding timeout information should be added.

Let us illustrate the amendment made to source route option first. Suppose we have a data packet with a DSR header, which contains a source route n hops long, the amended source route option is shown in Fig 3.1. There are not only a set of addresses, but also timeouts of the links along the route. If we add one byte for each timeout value, $n-1$ bytes of timeout information are added into the source route option. For comparison, the original source route option in DSR header is presented in Fig 3.2.

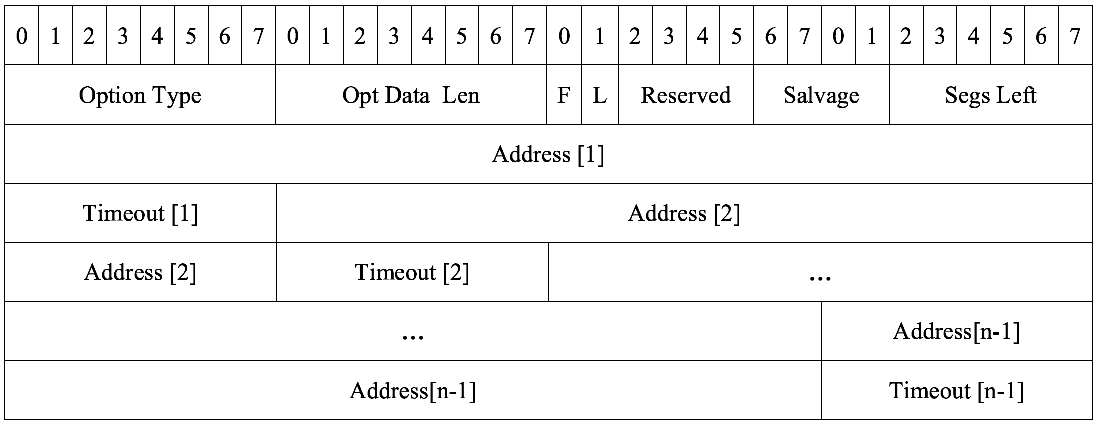

Figure 3.1: Amended DSR Source Route option in a DSR Options header

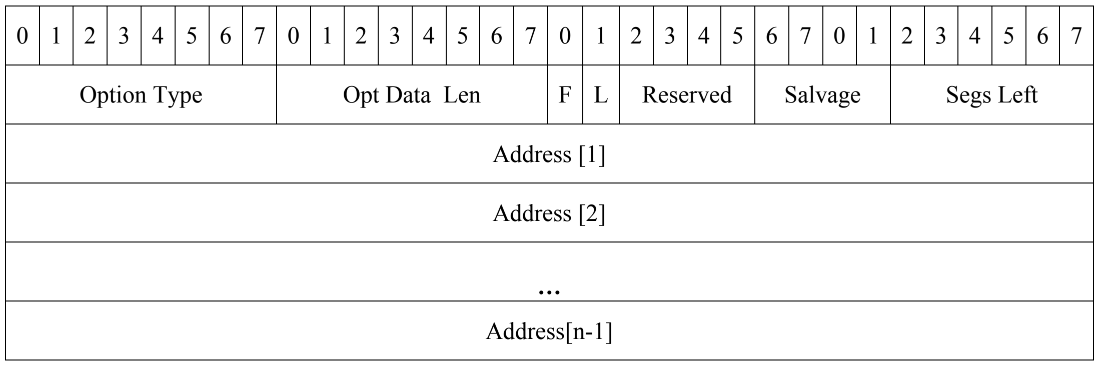

Figure 3.2: Original DSR Source Route option in a DSR Options header

In Fig 3.1, $Address[i]$ is the i^th intermediate node along the source route; $timeout[1]$ is the timeout of the link between originating node and $Address[1]$, which is estimated at $Address[1]$ normally; $Timeout[i]$ is the timeout of the link between $Address[i-1]$ and $Address[i]$.

In the route request and route reply options, the same structures of addresses and timeouts are used.

3.1.2 Detailed operation

In this section, we only introduce the operations related to timeouts. Generally, these operations includes: timeout estimation, extracting timeouts from cache, adding timeouts to DSR header, extracting cache timeouts from DSR headers and cache management. For details of other operations in DSR, please refer to [14].

In our protocol, the most important operation is the estimation of timeouts, which is the only source of timeout information. Here we introduce when

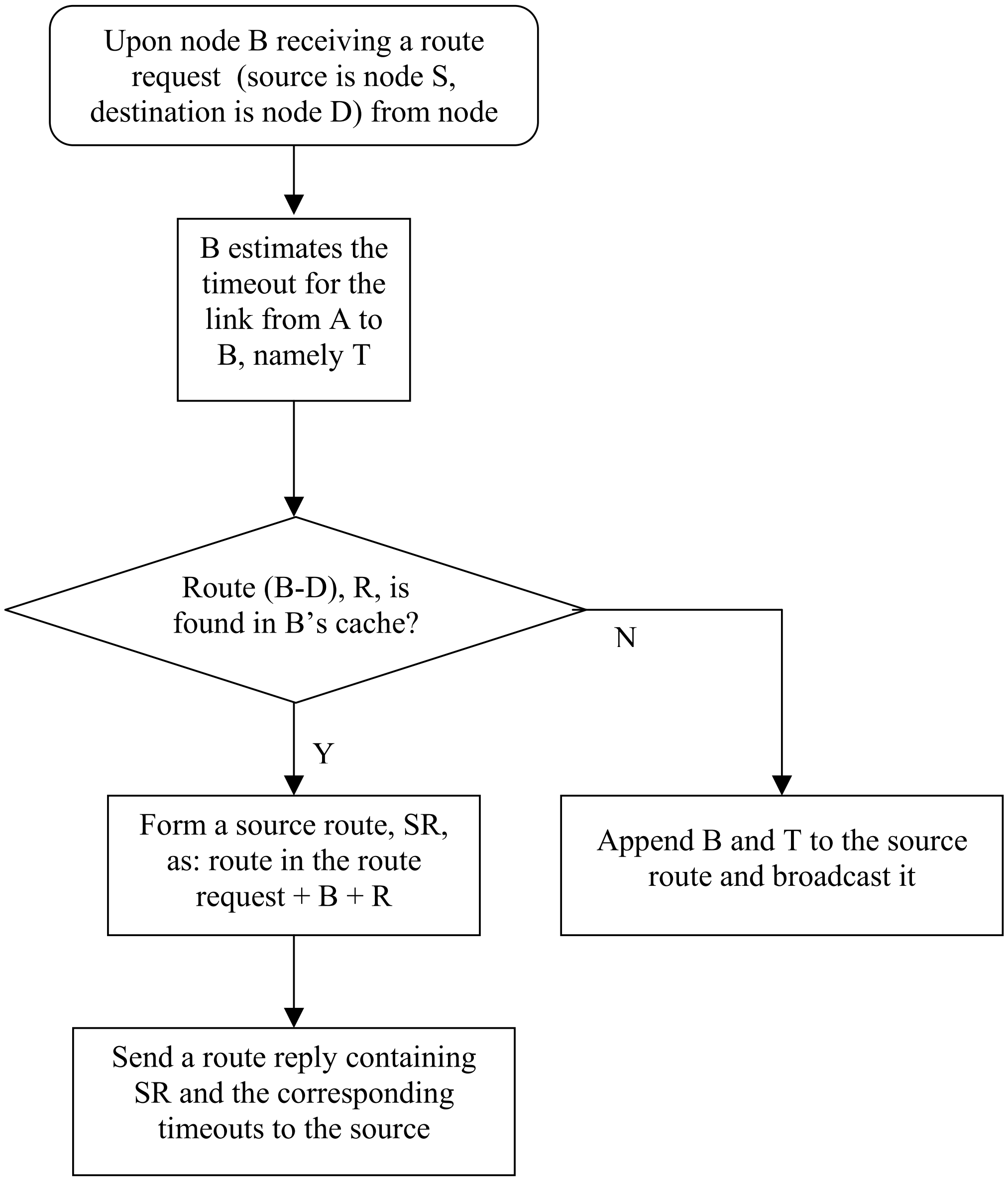
Upon node B receiving a route request (source is node S, destination is node D) from node
B estimates the timeout for the link from A to B, namely T
Route (B-D), R, is found in B's cache?
N
Y
Form a source route, SR, as: route in the route request + B + R
Append B and T to the source route and broadcast it
Send a route reply containing SR and the corresponding timeouts to the source

Figure 3.3: Flow chart of route request processing

and who to do timeout estimation. To know the way to estimate timeouts, please refer to Section 2.2.

A node estimates timeouts only when it receives a route request. Fig. 3.3 illustrates the route request processing, for more details on it, please refer to Appendix A. When receiving a route request from its neighbor, a node estimates the timeout for the link between itself and the neighbor. Then the node searches its own link cache for a route from itself to the destination of the source route. If some route is found, a route from the source to the destination can be formed, and then a route reply containing the source route and the corresponding timeouts is sent back to the source of the route request. If no route is found in the link cache, the node appends its own address and the estimated timeout, and then broadcasts the route request.

So far, we have estimated timeouts and added them into the DSR header. To make a node know the timeouts for every link in its cache, whenever it receives packets containing source routes, these timeouts must be extracted and cached as well as the links along the source route, which can be used to form source routes for sending data packets or creating route reply. If a link is already in the cache, the two timeouts are compared and the larger one is selected as the timeout for the link.

The timeouts stored in the link cache are also used in maintaining the link cache. If the timeout for a link is T, at time point T, the link will be removed from the link cache.

3.2 An implementation of ϵ measurement

To make the protocol more adaptive, we implement the ϵ measurement proposed in [9], which can enable the estimation of $L(T_p)$ given by (2.1) and (2.2) more applicable to other mobility models.

After a node has a T_p prediction on a link at time t_0, it measures the time period (T_r) during which this link will continue to be available. If this link is still available at $t_0 + T_p$, the node sets $T_r = T_p$ and does another prediction. After we obtain the information of T_p and T_r, a measured link availability, $L_m(T_p)$, is calculated by T_r/T_p. With $L_m(T_p)$, a measured ϵ, ϵ_m, can be estimated as

$$\epsilon_m \approx \frac{L_m(T_p) - 0.5\lambda T_p e^{-2\lambda T_p}}{1 - e^{-2\lambda T_p}} - \frac{1}{2\lambda T_p}, \tag{3.1}$$

for homogeneous mean epoch [9], and for heterogeneous mean epoch

$$\epsilon_m \approx \frac{L_m(T_p) - 0.25(\lambda_1 + \lambda_2)T_p e^{-(\lambda_1+\lambda_2)T_p}}{1 - e^{-(\lambda_1+\lambda_2)T_p}} - \frac{1}{(\lambda_1 + \lambda_2)T_p}. \tag{3.2}$$

Repeating the above operation, a node can have a series of $\epsilon_{m,j}$ and ϵ can be estimated by

$$\epsilon = \frac{\sum_{j=1}^{N} \epsilon_{m,j}}{N}, \tag{3.3}$$

where, N is the number of $\epsilon_{m,j}$ samples.

In our implementation, each node maintains a list of T_p and t_0, one T_p and one t_0 for one link between itself and one of its neighbors. As mentioned in Section 3.1.2, when a node receives a route request from its neighbor, it predicts a T_p and estimates a $L(T_p)$ for the link between itself and the neighbor, and then records the T_p and t_0.

To get T_r, each node needs to know the time when the link breaks down, T_b. With T_b, T_r is estimated as:

$$T_r \approx T_b - t_0. \tag{3.4}$$

As for the measurement of T_b, several means can be used. Firstly, if the physical layer can detect the link status and report them to the DSR, for example, by checking the signal strength of packets, the T_b can be easily obtained. Secondly, if the upper layer can detect and report the breaking down of a link, for example, by retransmission for a maximum times, the T_b can also be obtained. Thirdly, DSR itself can also detect it by sending beacons periodically. However, to simplify the implementation, in our simulation, a simpler way is used to accomplish this objective. That is, a node estimates the distance between itself and its neighbors once per second to find the time that a link breaks.

Fig. 3.4 illustrates the ϵ_m measurement implementation in our simulation. For more details on it, please refer to Appendix B. At the beginning of every second, a node checks the links between itself and its neighbors to update ϵ. If a link has a flag called Link_dead, which means that the link has expired before the last check, nothing needs to be done. The objective of this flag is to prevent multiple ϵ_m estimations for one link broken. If the link does not have this flag, T_r is estimated as

$$T_r \approx CURRENT\_TIME - t_0. \tag{3.5}$$

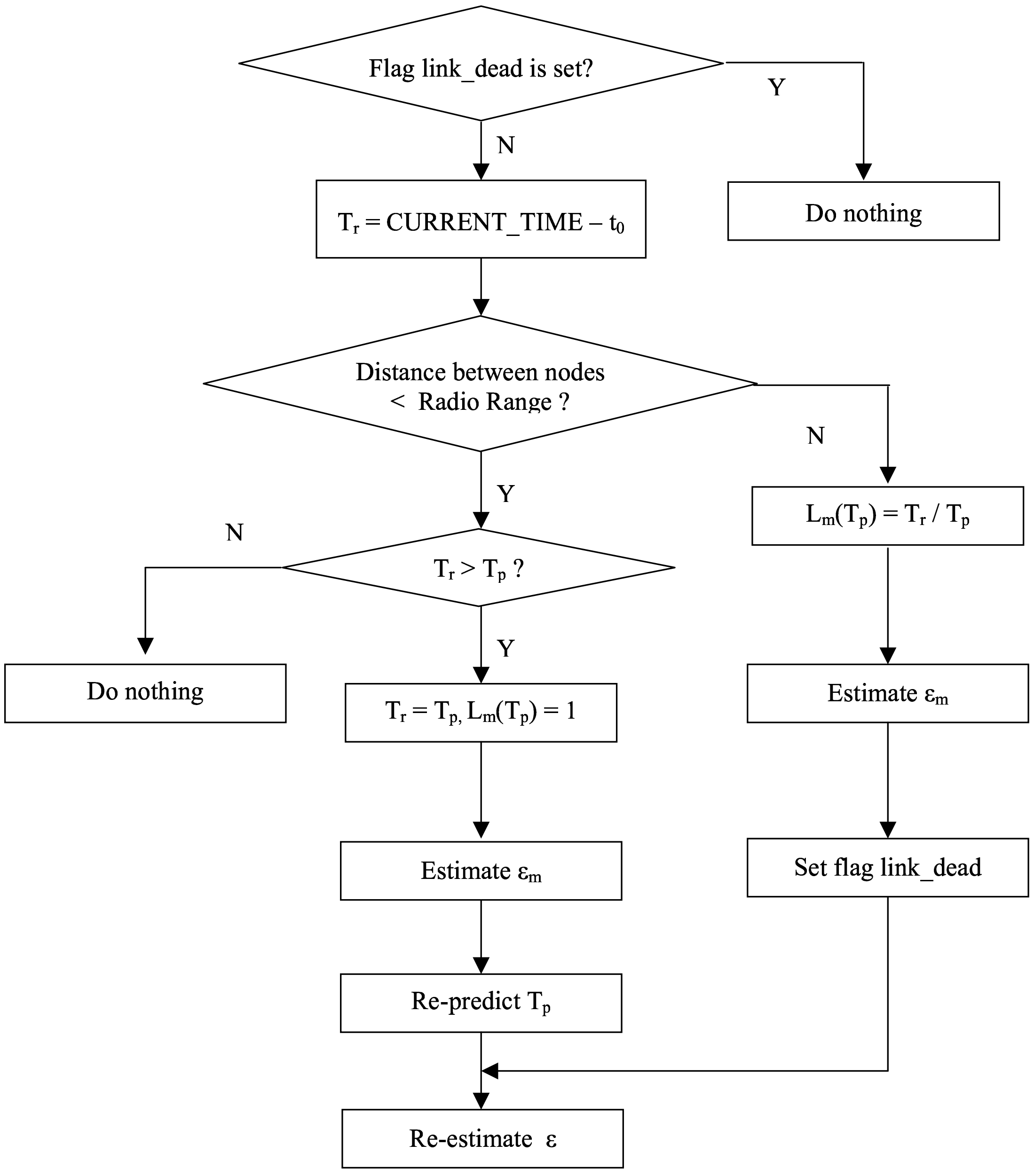

Figure 3.4: Flow chart of ϵ measurement

and the distance between the two ends of the link is calculated and compared to the radio transmission range. Distance greater than the transmission range means that the link is broken. If the link is broken, ϵ_m is calculated as (3.1) and (3.2). If the link is not broken and T_r is greater than T_p, the node sets $T_r = T_p$, estimates ϵ_m with $L_m(T_p) = 1$ and does another T_p prediction. If the measured ϵ_m is less than 0, it is set to 0. Lastly, the measured ϵ is calculated as (3.3).

With ϵ measurement, different nodes have different ϵ, which adapts to the node's mobility. We can expect high mobility nodes to have small T_r and ϵ and consequently next time when they do T_p prediction, a small $L(T_p)$ will be given for the same T_p. After reaching steady state, for the same T_p, high mobility node will have small timeout value and low mobility node will have large timeout value.

3.3 Simulation results

This section presents simulation results for the exponential random waypoint mobility model without pause. This part of results tries to show the impact of the delay caused by exchanging timeouts on the performance through comparison with the results presented in Chapter 2. The results for the homogeneous mean epoch are presented in Section 3.3.1, and Section 3.3.2 presents the results for heterogeneous mean epochs. In this section, we abbreviate ideal link-adaptive scheme proposed in Chapter 2 and link-adaptive protocol to **AD** and **ADP**, respectively.

3.3.1 Homogeneous mean epoch

Table 3.1 lists the results given by the proposed protocol in comparison with the results presented in Section 2.3.1.

We observe that the proposed protocol performs worse than the ideal link-adaptive scheme proposed in Chapter 2. For example, the proposed protocol achieves the packet delivery ratio of 99.08%, while the ideal link-adaptive scheme achieves that of 99.12%. The protocol generates overhead of 22505 packets and 1275098 bytes, while the ideal link-adaptive scheme achieves that of 16781 packets and 988684 bytes. The proposed protocol makes data packets to suffer a delay of $50ms$, while the ideal link-adaptive scheme achieves a delay of $42ms$. The reason is that timeouts exchange introduces some latency and consequently the timeout information in the proposed protocol is not as accurate as that in the ideal link-adaptive scheme.

However, compared to link-static-T, the proposed protocol performs fairly well. As shown in Table 3.1, in terms of packet delivery ratio and path length, the proposed protocol outperforms all link-static-Ts except for link-static-2; in terms of overhead in packets and bytes, and delay, the link-adaptive protocol outperforms all link-static-Ts. Most importantly, similar to the ideal link-adaptive scheme, the proposed protocol does not need static timeout settings and hence it does not need to worry about the tradeoff between packet delivery ratio and overhead mentioned in Section 2.3.1.

Table 3.1: Performance: exponential random waypoint mobility model (pause time = 0s), homogeneous mean epoch ($\lambda^{-1} = 60$)**,** $1500m \times 500m$

Metrics	Static timeout					$T_p \times L(T_p)$	
	$2s$	$4s$	$5s$	$10s$	$20s$	AD	ADP
PDR	99.3	99.0	99.0	98.2	95.3	99.12	99.08
DL	62	66	65	108	197	42	50
PL	2.65	2.72	2.73	2.87	3.20	2.67	2.72
NRD	41284	23285	20345	14323	16141	7595	8288
OiP	58379	37735	33797	27435	39659	16781	22505
OiB	3003434	1580672	1830622	1658613	3300721	988684	1275098

(For details on metrics, refer to Section 1.3.2)

3.3.2 Heterogeneous mean epochs

In this section, the link-adaptive protocol's adaptability to mobility is evaluated by times of being next hop. In Chapter 2, we found that the ideal link-adaptive scheme chooses low mobility nodes more frequently to form source routes. Here, we find the proposed protocol also has this preference. Table 3.2 and Table 3.3 present link-adaptive protocol's being next hop performance in comparison with link-static-5's and the ideal link-adaptive scheme's respectively. Table 3.4 presents the corresponding network performance.

Table 3.2 shows us that the proposed protocol chooses low mobility nodes as next hop more frequently than high mobility nodes compared to link-static-5, especially for the small space such as $700m \times 700m$. For the space of $1500m \times$

Table 3.2: Distribution of being next hop vs space sizes: exponential random waypoint mobility model (pause time=0s), heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$)

	$700m \times 700m$				$500m \times 1500m$			
	Static T = 5s		ADP		Static T = 5s		ADP	
	LMN	HMN	LMN	HMN	LMN	HMN	LMN	HMN
TBNH	1491.6	1658	1835	564	3053.4	3184.2	3640	1628
PBNH	47.36	52.64	76.5	23.5	48.95	51.05	69.1	30.9

(LMN = Low Mobility Node, HMN = High Mobility Node, TBNH = Times of being next hop, PBNH = Percentage of being next hop)

$500m$, among nodes selected as next hops by link-static-5, 48.95% are low mobility nodes, while among nodes selected as next hops by the protocol, 68.6% are low mobility nodes. It is interesting to find from Table 3.3 that the proposed protocol performs even better than the ideal link-adaptive scheme in terms of the preference of selecting low mobility nodes as next hop. This can be explained as follows. Because of the ϵ measurement, low mobility nodes are expected to have larger ϵ after reaching steady state. With larger ϵ, links consisting of low mobility nodes will have larger timeouts. As a result, they will stay in the cache for longer time and gain more chances to be selected as next hop.

From Table 3.4, we can find that with the preference for low mobility nodes, the proposed protocol performs better than link-static-5 in all performance metrics, especially with the space of $700m \times 700m$. Inherited from the

Table 3.3: Distribution of being next hop vs space sizes: exponential random waypoint mobility model (pause time=0s), heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$)

	$700m \times 700m$				$500m \times 1500m$			
	AD		ADP		AD		ADP	
	LMN	HMN	LMN	HMN	LMN	HMN	LMN	HMN
TBNH	1721.2	638.1	1835	564	3350.5	1645.2	3640	1628
PBNH	72.95	27.05	76.5	23.5	67.07	32.93	69.1	30.9

(LMN = Low Mobility Node, HMN = High Mobility Node, TBNH = Times of being next hop, PBNH = Percentage of being next hop)

ideal link-adaptive scheme, the proposed protocol also improves the overhead performance a lot. With the space of $500m \times 1500m$, it achieves overhead of 14856 packets and 803546 bytes, while link-static-5 achieves that of 29716 packets and 1770297 bytes. However, compared to the ideal link-adaptive scheme, the proposed protocol performs worse because of the latency mentioned in Section 3.3.1. For example, the proposed protocol achieves the packet delivery ratio of 99.30%, while the ideal link-adaptive scheme achieves that of 99.40%.

3.4 Conclusion

In this chapter, we developed a protocol to deploy the adaptive link caching scheme proposed in Chapter 2. A mechanism of timeout exchange between nodes is proposed and a measurement of ϵ is introduced into the proposed protocol.

Table 3.4: Performance versus space sizes: exponential random waypoint mobility model (pause time=0s), heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$)

	$700m \times 700m$			$500m \times 1500m$		
	Static	$T_p \times L(T_p)$		Static	$T_p \times L(T_p)$	
	5s	AD	ADP	5s	AD	ADP
PDR	99.97	99.98	99.98	99.26	99.40	99.31
NRD	20176	2416	3436	19238	5720	7874
DL	7.98	7.96	7.98	50.3	33.7	40.3
PL	1.75	1.80	1.77	2.74	2.73	2.73
OiP	21418	3041	5128	29716	12092	15012
OiB	926374	167548	320635	1770297	703264	1043282

(For details on performance metrics, please refer to Section 1.3.2)

The simulation results show that the delay for exchanging timeouts has trivial impact, but the proposed protocol performs as good as the adaptive link caching scheme if all nodes move according to exponential random waypoint mobility model without pause. More simulation results on the adaptability of this protocol to other mobility models will be presented in Chapter 4.

Chapter 4

Adaptability to mobility models

Chapter 3 only evaluates the proposed adaptive link caching protocol with scenarios in which nodes move according to the exponential random waypoint mobility model without pause. This mobility models conforms to the assumptions of the $L(T_p)$ estimation given by (2.1) and (2.2). However, in reality, nodes may move according to other mobility models, with or without pause. Therefore, the adaptability of the proposed protocol to mobility models affects its applicability in reality. In Chapter 2, this adaptability has been discussed briefly for an ideal situation, where a node can estimate timeouts for every link and ϵ is set to 0 for simplicity.

The adaptive link caching protocol is expected to have some adaptability to mobility models due to the use of ϵ measurement. In this chapter, to investigate the adaptability of the proposed protocol to various mobility models, a set of simulations are conducted by considering the nature of the mobility models. Mobility models with non-exponential epoch distribution under investigation include: the exponential random waypoint with positive pause (Section 4.1), the original random waypoint (Section 4.2), the random Gauss-Markov (Section 4.3) and the Brownian mobility models (Section 4.4).

4.1 Exponential random waypoint mobility model with pause

Table 4.1: Performance: exponential random waypoint mobility model (pause time = 5s), homogeneous mean epoch ($\lambda^{-1} = 60s$), $1500m \times 500m$

Metrics	Static T=5s						$T_p \times L(T_p)$
	$2s$	$4s$	$5s$	$10s$	$15s$	$20s$	
PDR	99.4	99.1	99.0	98.2	96.1	95.1	98.8
DL	57	62	59	119	172	220	51
PL	2.92	2.96	2.94	3.15	3.29	3.54	3.03
RD	42064	23661	20064	14430	15252	16494	9043
OiP	61500	39520	35815	29787	35006	39498	23274
OiB	3270866	2133570	1932737	2247102	4191732	4620977	1362362

(For details on metrics, refer to Section 1.3.2)

This section investigates the proposed protocol's adaptability to the exponential random waypoint mobility model with positive pause. Firstly, link-static-Ts and the proposed protocol are compared for a specific pause time. Secondly, the adaptability to pause times is studied. Lastly, the adaptability to heterogeneous mean epochs is discussed, in which low and high mobility nodes have the mean epochs of 250s and 60s respectively.

Table 4.1 presents the results of link-static-Ts and the proposed protocol for the pause time of $5s$. For link-static-T, the tradeoff between packet delivery ratio and overhead discussed in Section 2.3.1 recurs here, that is, among link-static-Ts, link-static-2 performs best in terms of packet delivery ratio, path length and delay, while link-static-10 performs best in terms of overhead in packets and route discoveries, and link-static-5 performs best in terms of overhead in bytes. Link-static-5 can be said to perform best in link-static-Ts, achieving relatively good performance in all performance metrics. However, the adaptive link caching protocol performs even better than link-static-5, achieving overhead of 1402694 bytes (only 75% of those obtained by link-static-5), and packet delivery ratio of 98.8% (99.8% of 99.0% obtained by link-static-5).

Figs 4.1 and 4.2 present the results of the proposed protocol against various pause times in comparison with those of link-static-5. For all pause times simulated, link-static-5 achieves higher packet delivery ratio and lower path length than the proposed protocol as shown in Fig. 4.1 (a) and Fig. 4.2 (b). However, in all other performance metrics, the proposed protocol outperforms link-static-5 for all pause times simulated. In addition, Fig 4.1 (c) shows that the proposed protocol performs better in terms of the overhead generated for each percent of packets delivered successfully. Fig. 4.2 (c) shows that the proposed protocol performs better in terms of the delay suffered per hop. For the explanation of these results, please refer to Section 2.3.1 because the results are similar.

However, when the pause time gets longer, the advantage of the proposed

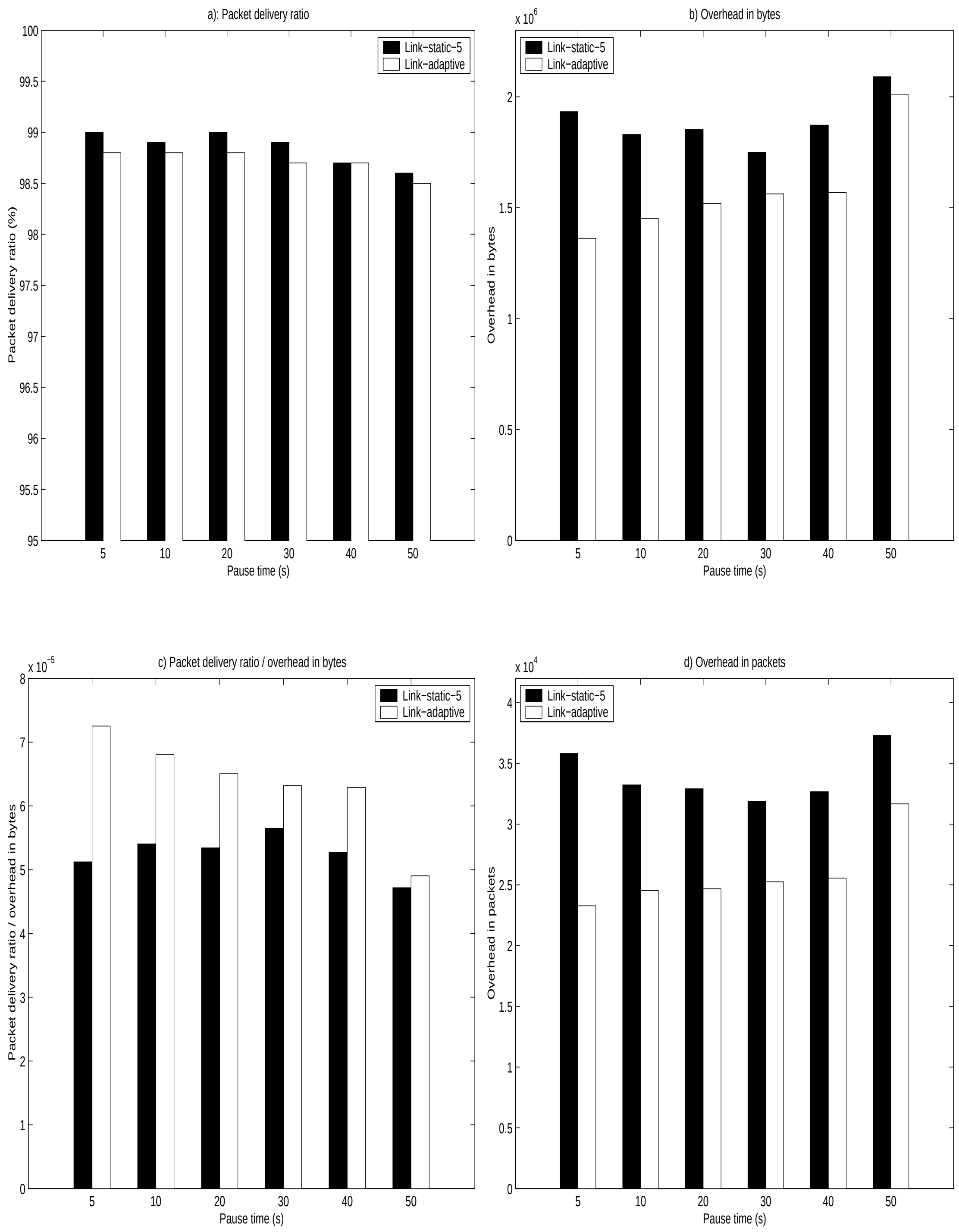

Figure 4.1: Packet delivery ratio vs overhead (exponential random waypoint mobility model, $500m \times 1500m$)

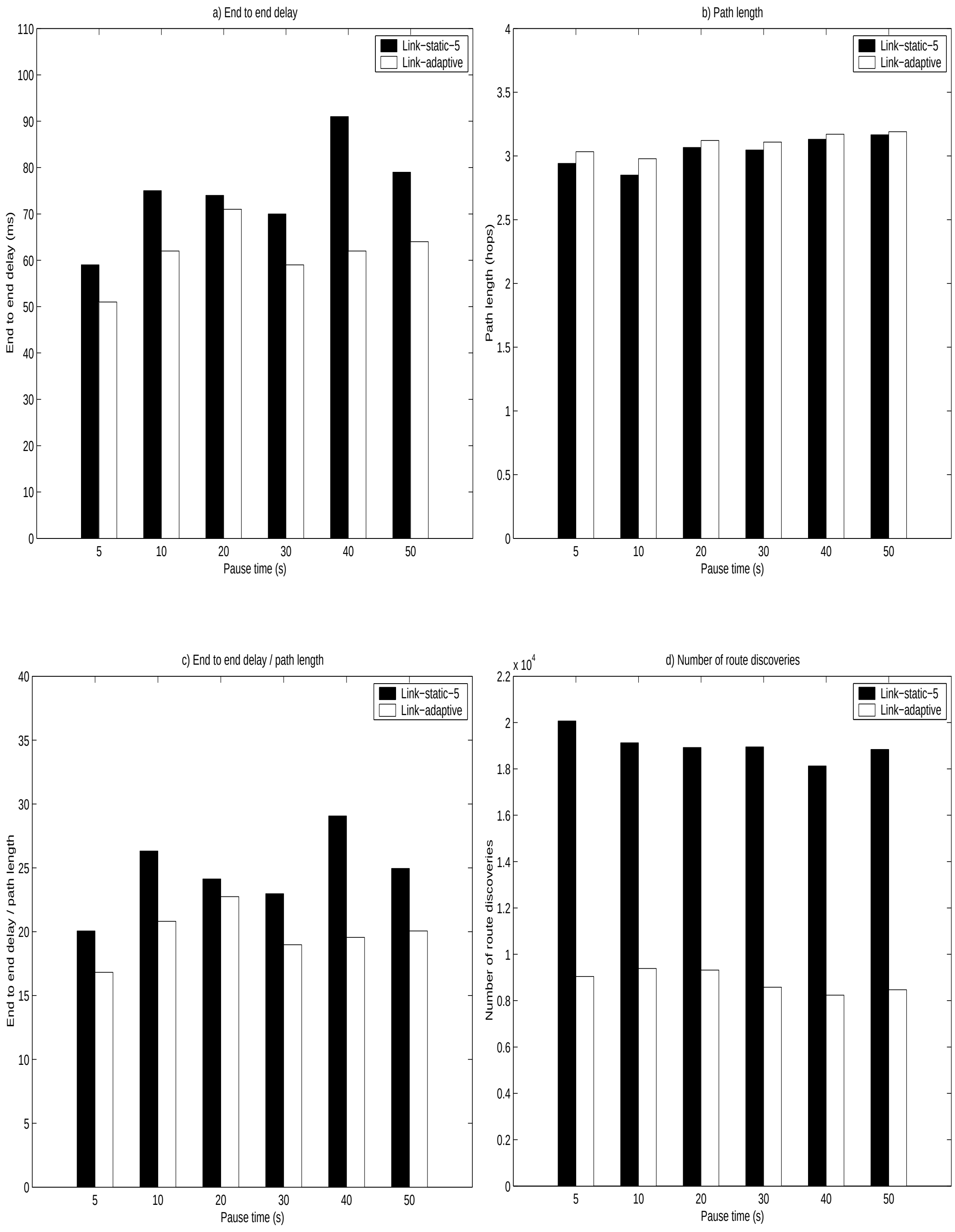

Figure 4.2: Delay vs path length (exponential random waypoint mobility model, $500m \times 1500m$**)**

protocol in terms of overhead gets weaker. For example, if the pause time equals to $50s$, the proposed protocol generates 2008398 overhead bytes, just a little less than 2097307 bytes generated by link-static-5. Note that the proposed protocol is based on the estimation of $T_p \times L(T_p)$ and $L(T_p)$ is based on knowledge of the mobility model. In our estimation of $L(T_p)$, although we used the measurement of ϵ to provide it with some adjustment, we do not consider pause time. Thus when the pause time increases, the accuracy of the estimation is being lost gradually. With a fairly large pause time, even the adjustment cannot compensate the inaccuracy of the $L(T_p)$ estimation in this scenario. Because of this inaccuracy, $T_p \times L(T_p)$ can not reflect the future link status anymore and then protocol performs worse and worse, but still better than link-static-5.

Our protocol selects low mobility nodes as next hop more frequently than link-static-5 does as shown in Table 4.2. Particularly, with our protocol a low mobility node is selected 3623 times, almost 2 times of 1868 times obtained by a high mobility node, while with link-static-5 low mobility nodes and high mobility nodes are selected almost the same times, 3251 and 3446 respectively.

Table 4.3 presents the performance of the proposed protocol in heterogeneous mean epochs in comparison with link-static-5. By selecting low mobility nodes more frequently, our protocol achieves better performance than link-static-5 in all performance metrics except for packet delivery ratio and path length.

Table 4.2: Distribution of being next hop: exponential random waypoint mobility model (pause time = 5s), heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$), $500m \times 1500m$

	Static T=5s		$L(T_p) \times T_p$	
	LMN	HMN	LMN	HMN
Times of being next hop	3251	3446	3623	1868
Percentage of being next hop	49	51	66	34

Table 4.3: Performance: exponential random waypoint mobility model (pause time = 5s), heterogeneous mean epochs ($\lambda_1^{-1} = 60s$, $\lambda_2^{-1} = 250s$), $500m \times 1500m$

	PDR	OiP	OiB	RD	DL	PL
Static T=5s	99.25	32470	1777762	19309	70	2.92
$T_P \times L(T_p)$	99.04	20520	1261654	6743	52	2.94

(For details on metrics, refer to Section 1.3.2)

4.2 Random waypoint mobility model

This section investigates the adaptability of the proposed protocol to the random waypoint mobility model. In this case, the movement can be changed by two parameters, max speed and pause time. Here, we first run the simulations with movement patterns generated for five max speeds, i.e., 5, 10, 20, 30 and 40m/s, and six pause times, i.e., 0, 5, 10, 20, 30 and 40s.

Figs. 4.3 and 4.4 present the results of link-static-5 and the proposed protocol against various max speeds. Generally, compared to link-static-5, the

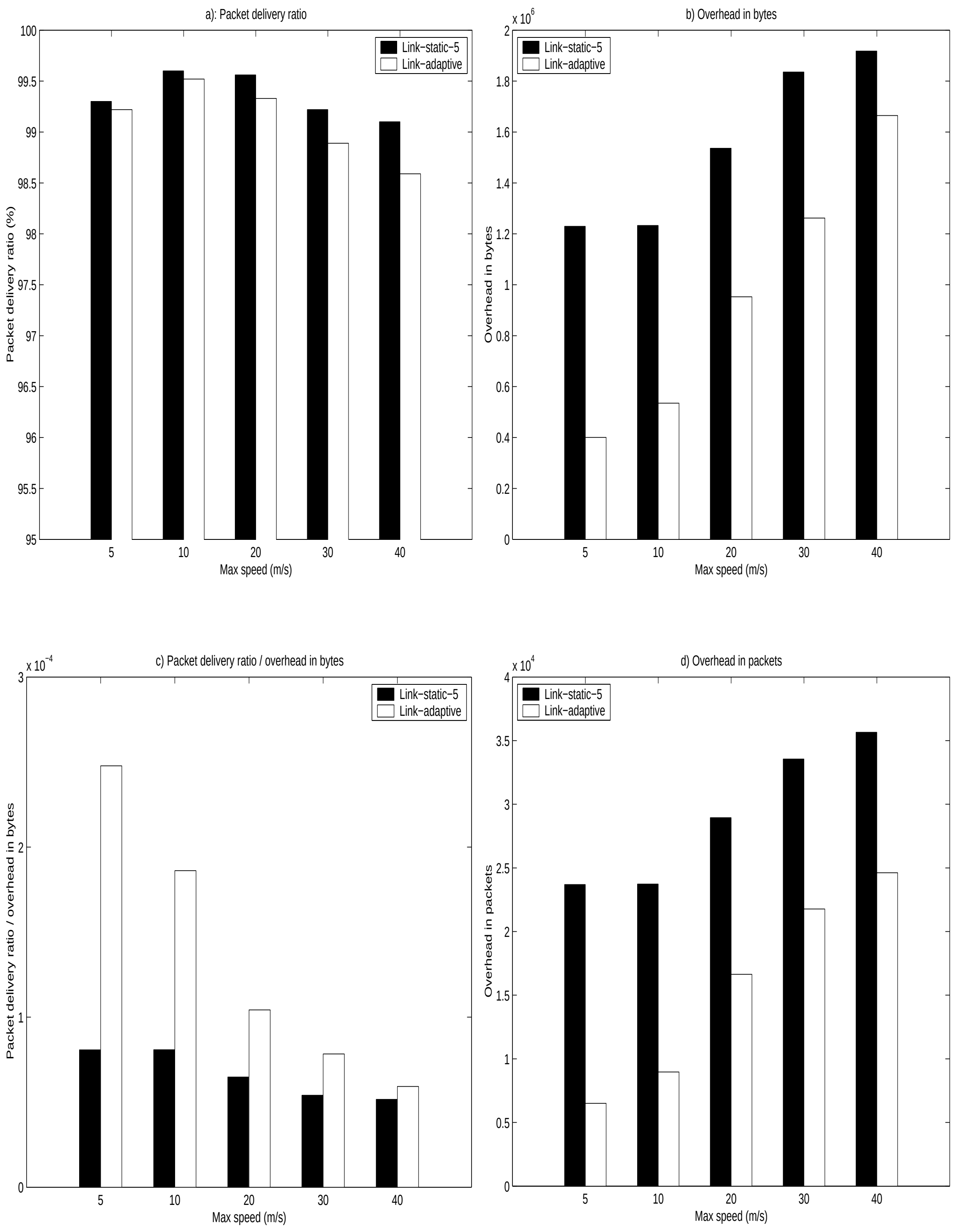

Figure 4.3: Packet delivery ratio vs overhead: random waypoint mobility model (pause time = 5s), $500m \times 1500m$

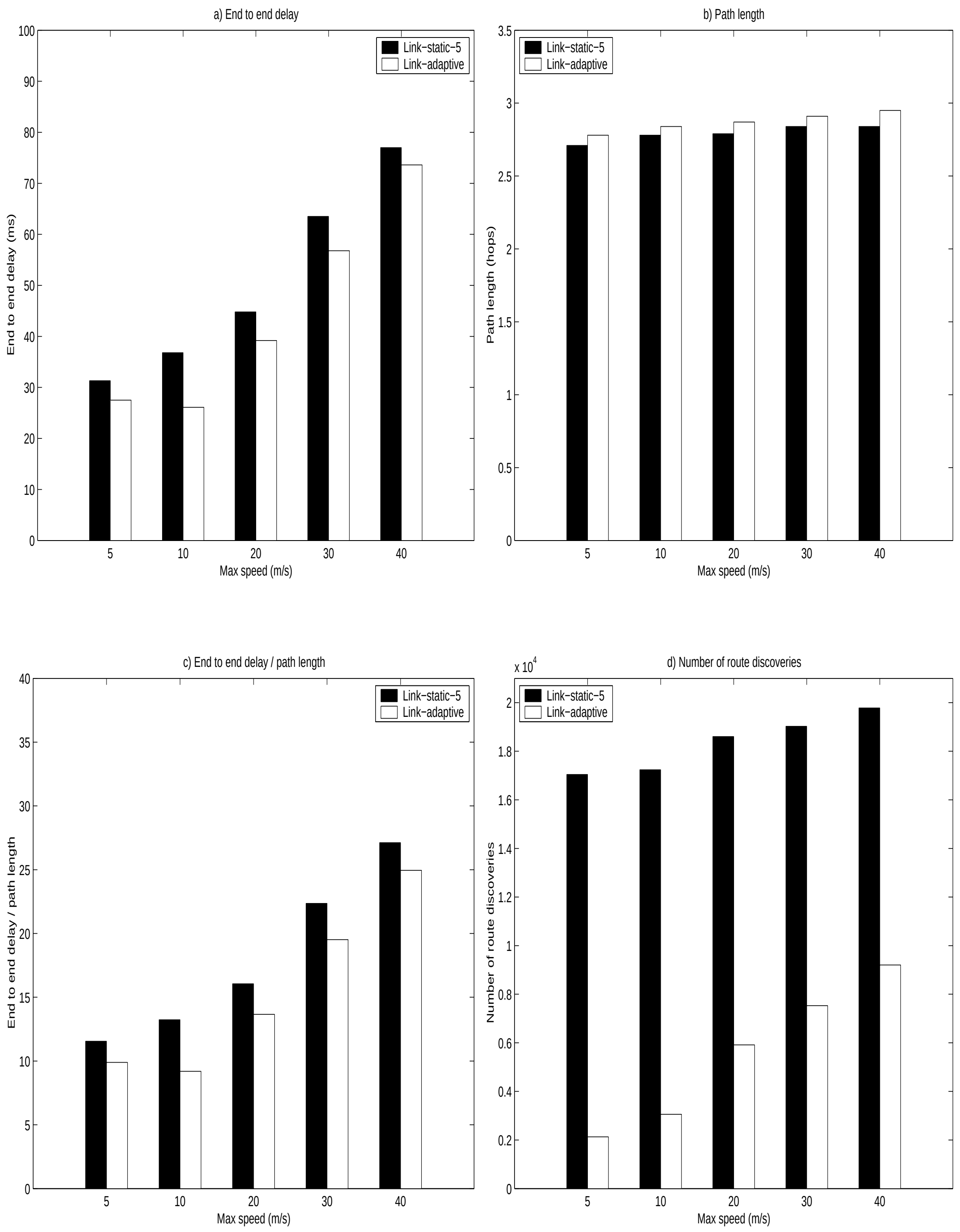

Figure 4.4: Delay vs path length: random waypoint mobility model (pause time = 5s), $500m \times 1500m$

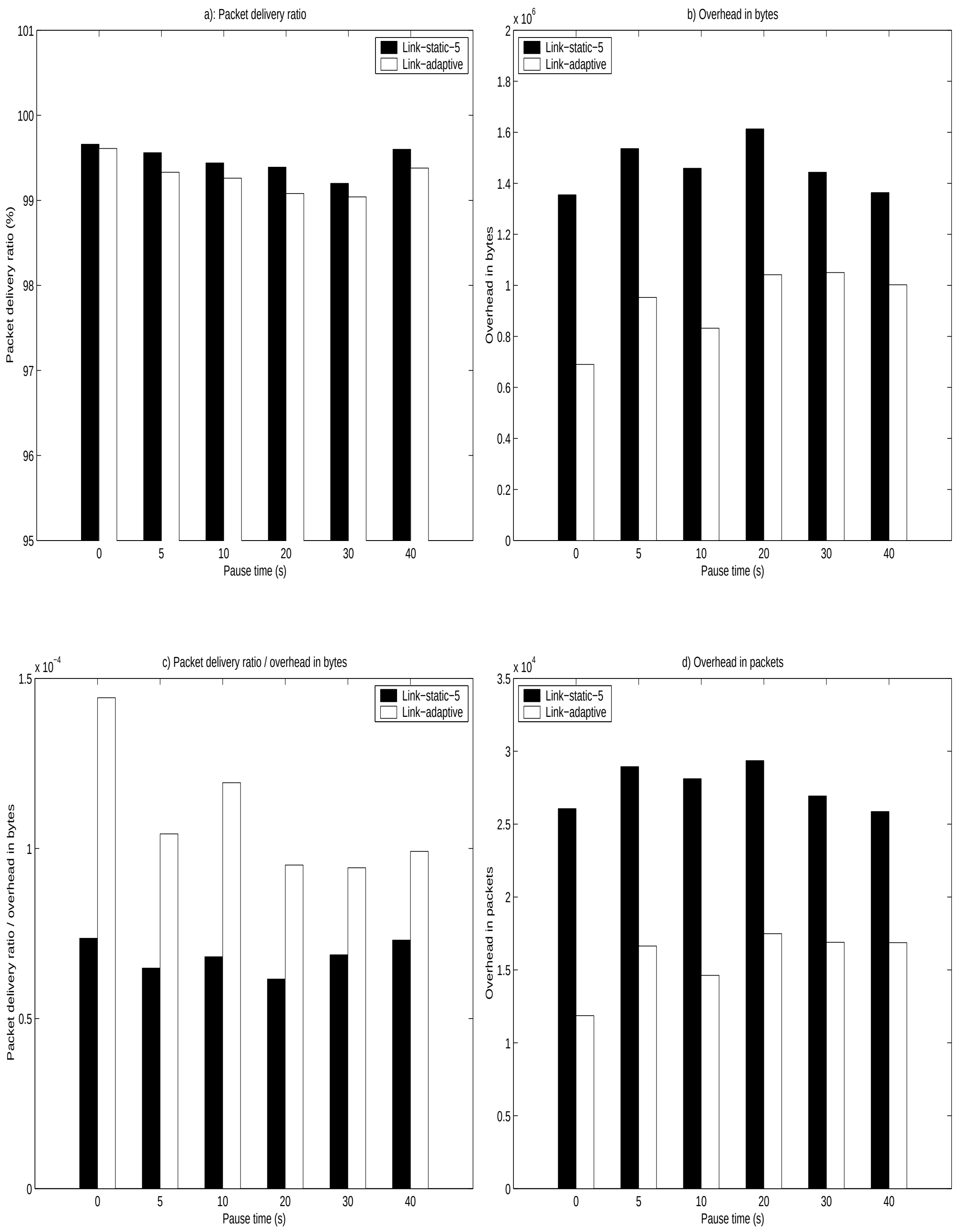

Figure 4.5: Packet delivery ratio vs overhead (random waypoint mobility model, $500m \times 1500m$, Max speed = 20m/s)

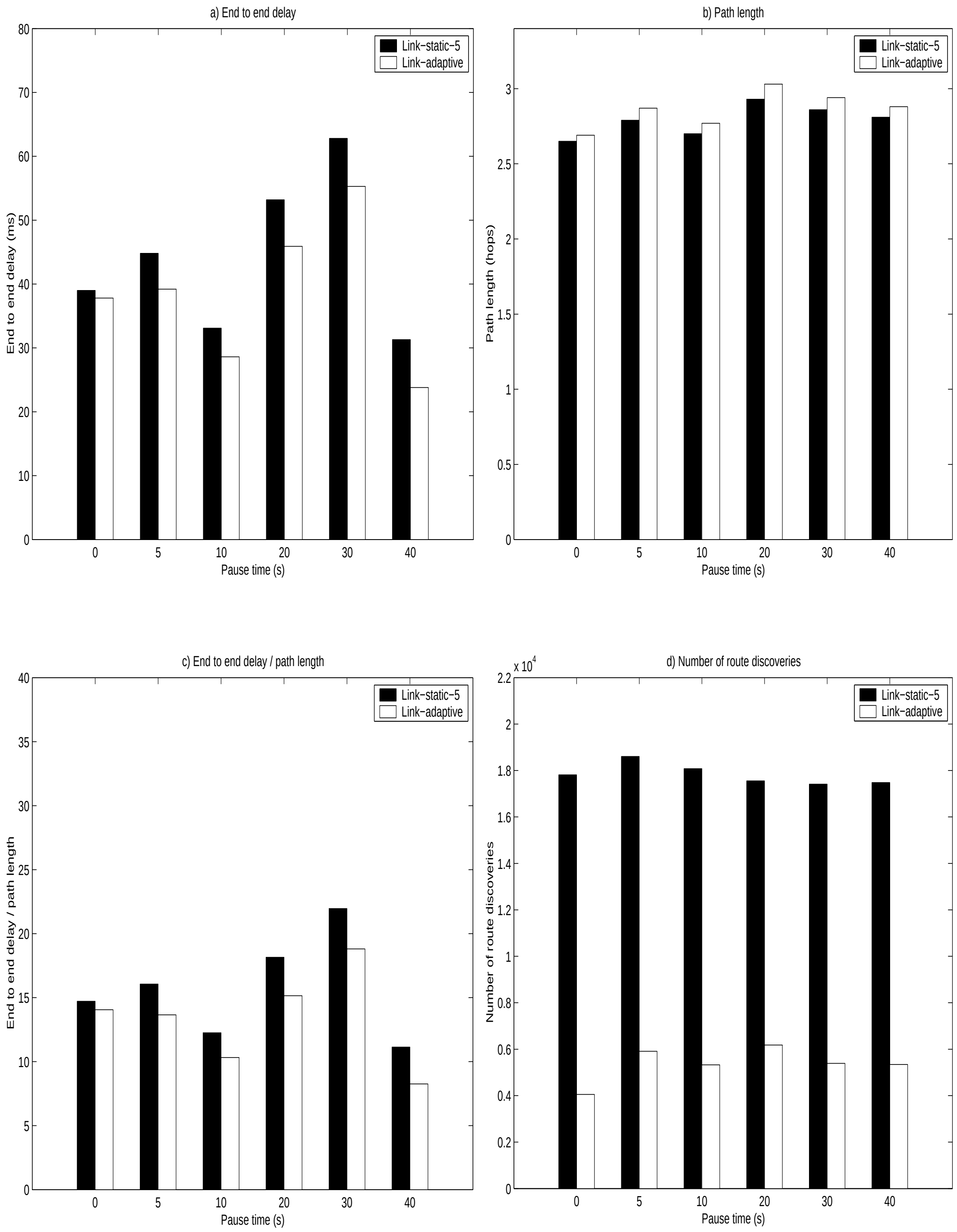

Figure 4.6: Delay vs path length (random waypoint mobility model, $500m \times 1500m$, Max speed = 20m/s)

proposed protocol adapts to speeds well. For all speeds simulated, link-static-5 performs a little better than the proposed protocol in terms of packet delivery ratio and path length, while the protocol outperforms link-static-5 in other performance metrics. We can say that the proposed protocol can perform well with various speeds if nodes move according to random waypoint mobility model. However, we also find that with the increase of max speed, the advantage of the proposed protocol is losing gradually. For example, at the max speed of $5m/s$, the proposed protocol generates only 32.6% of overhead bytes generated by link-static-5, but at the max speed of $40m/s$, it generates 86.8% of overhead bytes generated by link-static-5.

Figs. 4.5 and 4.6 present the performance of link-static-5 and the proposed protocol against various pause times. Results similar to those in Section 4.1 are obtained. With all pause times simulated, link-static-5 performs better in terms of packet delivery ratio and path length, but worse in terms of other metrics than the proposed protocol. However, with the increase of pause time, the proposed protocol's advantage becomes weaker because of the breach of the $L(T_p)$ estimation assumption.

4.3 Random Gauss-Markov mobility model

First of all, let us talk about the simulation limitation of this mobility model. In the simulation of random Gauss-Markov mobility model, if a relatively long interval is simulated, nodes will move out of the allowed space frequently. As mentioned in Section 1.3.1, if a movement cause a node moving out of the space, the direction of the velocity is reversed. If the $speed \times interval$ is large enough compared to the space size, the node may move out of the allowed space even if the direction of the velocity is reversed. This is the reason that we can not simulate long movement intervals.

In this section, we simulate homogeneous mean epochs (i.e., $10s$, $20s$) and heterogeneous mean epochs (i.e., $10s$ for high mobility nodes, $20s$ for low mobility nodes). With the interval of $10s$ and $20s$, we set the mean epoch in $L(T_p)$ estimation to 10 and 20 respectively.

In our simulation, $\overline{v}_x = \overline{v}_y = 0\ m/s$, $\delta_{v_x} = \delta_{v_y} = 10.4835769\ m/s$ and $\alpha = 0.9$. These parameters are chosen to be equal to those used by Yih-chun in his implementation [15].

Table 4.4 presents the results for the interval of $10s$ and $20s$. In these cases, the proposed protocol performs a little better than link-static-5.

In heterogeneous mean epochs, our protocol selects low mobility nodes as next hop more frequently than link-static-5. Particularly, with our protocol low mobility nodes are selected 3967 times per node, and high mobility nodes are selected 3283 times per node, while with link-static-5 high mobility nodes are selected more frequently than low mobility nodes. In this case, our protocol performs better than link-static-5 in terms of packet delivery ratio and path length, but worse in terms of overhead and delay as shown in Table 4.6.

Table 4.4: Performance: random Gauss-Markov mobility model, homogeneous interval, $1500m \times 500m$

	Interval = 10s		Interval = 20s	
	Static T = 5s	$T_p \times L(T_p)$	Static T = 5s	$T_p \times L(T_p)$
PDR	63.39	68.41	67.89	73.65
DL	1657	1799	1292	1360
PL	4.25	3.84	4.60	4.11
NRD	23560	20790	26787	18245
OiP	100808	901460	111398	102664
OiB	8528065	7635748	10753878	8571029

(For details on metrics, refer to Section 1.3.2)

4.4 Brownian mobility model

Similar to random Gauss-Markov mobility model, there is also a limitation within Brownian mobility model. As mentioned in Section 1.3.1, if a movement causes a node to move out of the allowed area, the node picks the intersection of the boundaries and the direction of the velocity as the destination. It seems that nodes could keep themselves in the allowed area with this mobility model, but in fact a node may still move out of the space. When a node arrives at the destination on the boundaries and chooses a velocity directing it out of the space, the current position will be chosen as the destination again, which is not allowed in NS-2. Our solution is to reverse the direction of the velocity if the above situation happens. This introduces the limitation we have discussed in Section

Table 4.5: Distribution of being next hop: random Gauss-Markov mobility model, heterogeneous intervals ($\lambda_1^{-1} = 10s$, $\lambda_2^{-1} = 20s$), $1500m \times 500m$

	Static T =5s		$L(T_p) \times T_p$	
	LMN	HMN	LMN	HMN
Times of being next hop	3837	4518	3967	3283
Percentage of being next hop	46	54	55	45

Table 4.6: Performance: random Gauss-Markov mobility model, heterogeneous intervals ($\lambda_1^{-1} = 10s$, $\lambda_2^{-1} = 20s$), $1500m \times 500m$

	PDR	OiP	OiB	NRD	DL	PL
Static T =5s	65.09	106363	9313377	25361	1561	4.54
$T_p \times L(T_p)$	71.15	116302	9721711	20187	1804	3.94

(For details on metrics, refer to Section 1.3.2)

4.3, that is to say, large interval such as $30s$ can not be simulated.

We simulate the homogeneous intervals of $10s$ and $20s$ and heterogeneous intervals (i.e., $10s$ and $20s$). The settings of λ^{-1} in $L(T_P)$ estimation is the same as those in Section 4.3.

Table 4.7 presents the results for homogeneous intervals. With the intervals of both $10s$ and $20s$, the proposed protocol achieves almost the same performance as link-static-5 does.

For the heterogeneous intervals, our protocol keeps on selecting low mobility nodes more frequently as shown in Table 4.8. With our protocol low mobility nodes and low mobility nodes are selected as next hop 5656 and 4820 times re-

Table 4.7: Performance: Brownian mobility model, homogeneous interval, $1500m \times 500m$

	Interval = 10s		Interval = 20s	
	Static T = 5s	$T_p \times L(T_p)$	Static T = 5s	$T_p \times L(T_p)$
PDR	98.42	98.62	97.04	97.10
DL	96.2	97.3	172.2	142.1
PL	3.20	3.23	3.72	3.70
NRD	17716	13489	19228	12773
OiP	39686	49405	48153	47398
OiB	2410106	3216090	3009948	3136681

(For details on metrics, refer to Section 1.3.2)

spectively, while with link-static-5 low mobility nodes and high mobility nodes are all selected 7632 times.

With this preference, our protocol performs better than link-static-5 in terms of packet delivery ratio, delay, route discoveries and path length, but worse than link-static-5 in terms of overhead as shown in Table 4.9.

4.5 Conclusion

In this chapter, we found that for those mobility models with non-exponential epoch, the proposed protocol can perform well in most cases. Specifically, our protocol can adapt to speeds well, but can be affected by pause times and mean epochs. The shorter the pause time is, the better our protocol performs; the

Table 4.8: Distribution of being next hop: Brownian mobility model, heterogeneous intervals ($\lambda_1^{-1} = 10s$, $\lambda_2^{-1} = 20s$), $1500m \times 500m$

	Static T =5s		$L(T_p) \times T_p$	
	LMN	HMN	LMN	HMN
Times of being next hop	7632	7632	5656	4820
Percentage of being next hop	50	50	54	46

Table 4.9: Performance: Brownian mobility model, heterogeneous intervals ($\lambda_1^{-1} = 10s$, $\lambda_2^{-1} = 20s$), $1500m \times 500m$

	PDR	OiP	OiB	NRD	DL	PL
Static $T = 5s$	97.87	45216	2794743	18499	122	3.47
$T_p \times L(T_p)$	98.25	79232	5401734	17767	116	3.38

(For details on metrics, refer to Section 1.3.2)

longer the mean epoch is, the better our protocol performs.

Chapter 5

Conclusions

This thesis proposes an adaptive link caching scheme for DSR and evaluates it through simulation in comparison with a static link caching scheme [15]. To make the cached link information reflect the link status, we determine cache timeouts for links in the cache through a lifetime prediction and a link availability estimation, i.e., $T_p \times L(T_p)$, which assumes exponentially distributed epochs.

We found that for mobility models in which nodes moves with exponential distributed epochs, the proposed scheme can choose more reliable routes and improve the performance, especially the performance in terms of overhead. For other mobility models with non-exponential epochs, we observed that the proposed scheme still can choose more reliable routes and improve the performance. That is, if nodes have relatively long movement intervals and short pauses, the proposed scheme performs much better than the static scheme. On the other hand, if nodes have relatively short movement epochs and long pauses, although the proposed scheme can still achieve performance improvement, the improvement is less than that achieved with long intervals and short pauses. Particularly, for the exponential random waypoint mobility model with positive pauses, the proposed scheme can improve the performance for all pauses simulated. For the random waypoint mobility model, the proposed scheme can improve the performance for all pauses and speeds simulated. For the random Gauss-Markov mobility model, the proposed scheme performs slightly better than the static scheme. However, for the Brownian mobility model, the proposed scheme performs slightly better than the static scheme only in terms of packet delivery ratio.

However, there are still a lot of issues we have not covered in this thesis. The most important one is the impact of errors that may occur when measuring nodes' mobility parameter, specifically during the T_p prediction. In this thesis, we assume that nodes can always measure their neighbors' mobility parameter accurately and then predict T_p correctly. However, in reality, measurement errors may happen due to the imperfectness of physical channel (e.g., noise, channel fading, etc). Unfortunately, in the T_p prediction mechanism adopted in our scheme [13], the author assumed the physical channel is always perfect. Another assumption of this T_p prediction mechanism, which can also bring in errors into the measurement, is that the nodes are assumed to know the transmission power of all other nodes and all nodes keep their transmission powers constant. Unfortunately, this is not the case in reality . For example, recently power consumption has drawn a lot of attention, and a lot of work has been done on how to adjust the transmission power of mobile nodes dynamically according to the channel condition and battery level. If such kind of techniques are used, we cannot expect this T_p prediction mechanism to provide accurate information on nodes' movement.

Below, I will briefly discuss some potential solutions to these problems.

When the signal to noise ratio is very low or channel fading is severe, the signal strength detected cannot represent the distance, and a node will fail to estimate the relative velocity between the two nodes of a link. Subsequently, $T_p \times L(T_p)$ will fail to show the actual lifetime of a link. However, when the signal to noise ratio is high and the channel fading is moderate, the ϵ adjustment in $L(T_p)$ estimation can alleviate the inaccuracy in measuring the mobility parameters. Anyway, for this factor, the credibility for T_p prediction, $0 \leq \alpha \leq 1$, should be measured or estimated. Then the actual lifetime of a link under estimation should be amended as $\alpha \times T_p \times L(T_p)$. More studies on α is required.

As an alternative for T_p prediction [6], a scheme has been proposed to predict T_p with the help of GPS. With the information on location and velocity provided by GPS, the T_p can be estimated more accurately than with the signal strength measurement based T_p prediction. Note that, the $L(T_p)$ estimation does not depend on the methods of the T_p prediction.

The following topics can also be further studied for $T_p \times L(T_p)$ application. Firstly, in this thesis we assume that all nodes know the mean epoch used in $L(T_p)$ estimation, so a dynamic measurement of the mean epoch should be useful to provide the proposed scheme with more adaptability. Secondly, so far we focus on the application of $T_p \times L(T_p)$ to link caching in DSR. In which way $T_p \times L(T_p)$ can be used in other places to improve network performance is another interesting issue.

Bibliography

[1] Y.D. Liu, S.M. Jiang, Y.M. Jiang, D.J. He, "An adaptive link caching scheme for on-demand routing in MANETS", *THE 14TH IEEE Int Sym on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2003, Beijing, China.

[2] S.M. Jiang, Y.D. Liu, Y.M. Jiang, "Provisioning of Adaptability to Variable Topologies for Routing Schemes in MANETs", the extended abstract is accepted by IEEE JSAC Special Issue on Quality-of-Service Delivery in Variable Topology Networks, and the full paper is invited.

[3] C.K. Toh, "Wireless ATM & Ad-hoc Networks", Kluwer, Nov. 1996

[4] The network simulator (NS-2), http://www.isi.edu/nsnam/ns/

[5] C.K. Toh, "Associativity-based routing for ad-hoc networks", *Wireless Personal Communications*, Mar. 1997, pp. 103-139

[6] W. Su and M. Gerla, "IPV6 flow handoff in ad hoc wireless networks using monility predication", *Proc. IEEE GLOBOCOM*, pp. 271-275, Dec 1999.

[7] E.M. Royer and C.K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks", *IEEE Personal Communications*, pp. 46-55, April 1999.

[8] R. Dube, C. Raia, K-Y Wang and S. Tripathi, "Signal Stability based adaptive routing (SSA) for ad hoc networks", *IEEE Personal Communications*, pp. 36-45, Feb 1997.

[9] S.M. Jiang, D.J. He, and J.Q. Rao. "A Prediction-based Link Availability Estimation for Mobile Ad Hoc Networks", *IEEE INFOCOM*, pp. 1745–52, 2001. http://citeseer.nj.nec.com/498937.html.

[10] S.M. Jiang, D. He and J. Rao, "A prediction-based link availability estimation for routing metrics in MANETs", submitted to a journal. Its early version was presented at IEEE INFOCOM 2001, http://www.cwc.nus.edu.sg/ network/publications.html

[11] D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", in *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pp. 153-181. Kluwer Academic Publishers, 1996.

[12] B. Liang, Z.J. Haas, "Predictive distance-based mobility management for PCS networks", *Proc. IEEE INFOCOM*, Mar 1999.

[13] D.J. He, S.M. Jiang and J.Q. Rao, "Link availability prediction model for wireless ad hoc network", *Proc. 2000 International Conference on Distributed Computing System Workshop*, pp. D7-D11, April 2000.

[14] D.B. Johnson, D.A. Maltz and Y.C. Hu, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", *IETF Internet Draft, draft-ietf-manet-dsr-09.txt*, April 2003.

[15] Y.C. Hu and D.B. Johnson. "Caching Schemes in On-Demand Routing Protocols for Wireless Ad Hoc networks". *Proc. the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00)*, pp. 231–242, August 2000.

[16] C.E. Perkins, E.M. Belding-Royer and S.R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", *IETF Internet Draft, draft-ietf-manet-aodv-13.txt*, February 2003.

[17] C.E. Perkins and P. Bhagwat. "Highly dynamic Destination- Sequenced Distance-Vector routing (DSDV) for mobile computers". *Proc. the SIGCOMM 94 Conference on Communications Architectures, Protocols and Applications*, pp. 234244, August 1994.

Appendix A

Source Code for $T_p \times L(T_p)$ Estimation

For the description of $T_p \times L(T_p)$ Estimation, refer to Section 2.2. The source code is as follows:

```
    double LinkCache::find_timeout(ID a, ID b) //added by liuyaoda
{
    double lifetime = 0.0; // $T_p$
    double probability = 0.0;// $L(T_p)$
    double epsilon = 0.0;
    Node *fromnode, *tonode, *node;
        /*one node of the link*/
    fromnode = Node::get_node_by_address(a.addr);
        /*the other node of the link*/
    tonode = Node::get_node_by_address(b.addr);
```

```
        /*the node itself*/
node = Node::get_node_by_address(net_id.addr);
        /*predict the link's lifetime*/
lifetime = ((MobileNode *) fromnode)->lifetime((MobileNode *) tonode);
#ifdef MULTI_EPOCH //for heterogeneous mean epochs
        double lambda1 = 0.0;// The lambda for fromnode
        double lambda2 = 0.0;// The lambda for tonode
        if(a.addr < 25){
                lambda1=60.0;}
        else if (24 < a.addr < 50){
                lambda1=250.0;}
        if(b.addr < 25){
                lambda2=60.0;}
        else if (24 < b.addr < 50){
                lambda2=250.0;}
        lambda1=1.0/lambda1;
        lambda2=1.0/lambda2;
        /*estimate L(Tp)*/
        probability = exp(-(lambda1+lambda2)* lifetime) * (0.5 *
                        pow(lifetime,2) * pow((lambda1+lambda2),2) - 2
                        -2 * lifetime * epsilon * (lambda1+lambda2) +
                        2 * exp((lambda1 + lambda2) * lifetime) * (1 +
                        lifetime * epsilon * (lambda1 + lambda2))) / (2 *
                        (lambda1 + lambda2) * lifetime);
    #endif
    #ifndef MULTI_EPOCH //for homogeneous mean epochs
        double lambda=0.0;
        lambda=1.0/60.0;
        /*get the ε of the estimating node*/
        epsilon=((MobileNode *) node)->epsilon;
        /*estimate the L(T_p)*/
        probability = (1-exp(-2 * lifetime * lambda)) * (1.0/ (2.0 * lambda *
                    lifetime) + epsilon) + lifetime * lambda* exp(-2 *
                    lifetime *lambda) / 2.0;
    #endif
        /*estimate T_p × L(T_p)*/
    lifetime = probability * lifetime;
    if (lifetime < lc_minlifetime)
        lifetime = lc_minlifetime;
    return CURRENT_TIME + lifetime;
}
```

Appendix B

Source Code for ϵ Estimation

For descriptio of ϵ Estimation, refer to Section 3.2. The source code is as follows:

```
void LinkCache::periodic_checkCache() {
    for(c = 0; c <= LC_MAX_NODES; c++) {
        Link *v = lcache[c].lh_first;
        for( ; v; v = v− >ln_link.le_next) {
                /*find the links between myself and my neighbors*/
            if (c == net_id.addr || v− >ln_dst == net_id.addr){
                Node *nodea, *nodeb, *node;
                /*one node of the link*/
                nodea = Node::get_node_by_address(c);
                /*the other node of the link*/
                nodeb = Node::get_node_by_address(v− >ln_dst);
                /*the node itself*/
                node = Node::get_node_by_address(net_id.addr);//the node itself
                /*estimate the distance between the two node*/
```

```
double distance = ((MobileNode *) nodea)- >distance((MobileNode
                *) nodeb);
/*the time that the links has been in the cache*/
double Tr=CURRENT_TIME-v- >ln_insert-1;
/*the predicted lifetime*/
double Tp=v- >lifetime;
if (((MobileNode *) nodea)- >distance((MobileNode *) nodeb)>250.0){
/*the link is not available now*/
  if (v- >flag_dead==0){//the link was available at the last check
    v- >flag_dead=1;
     if (Tr<Tp){
       #ifdef MULTI_EPOCH//heterogeneous mean epochs
/*mean epoch for one end of the link*/
          double lambda1 = 0.0;
/*mean epoch for the other end of the link*/
          double lambda2 = 0.0;
          if(c<25){
            lambda1=60.0;}
          else if (24 < c < 50){
            lambda1=250.0;}
          if(v- > ln_dst < 25){
            lambda2=60.0;}
          else if (24 < v- > ln_dst < 50){
                lambda2=250.0;}
            lambda1=1.0/lambda1;
            lambda2=1.0/lambda2;
/*estimate ϵ*/
            double epsilong = ((Tr/Tp) - 0.25 * (lambda1 + lambda2)
                 * Tp * exp (-(lambda1 + lambda2) * Tp)) / (1 -
                exp(-(lambda1 + lambda2) * Tp)) -1 / ((lambda1 +
                 lambda2)*Tp);
        #endif
        #ifndef MULTI_EPOCH//homogeneous mean epoch
/*mean epoch for both ends of the link*/
            double lambda=1.0/60.0;
/*estimate ϵ*/
            double epsilong=((Tr / Tp) - 0.5 * lambda * Tp *
                exp(-2 * lambda * Tp)) / (1 - exp(-2 * lambda * Tp))-
                1 / (2 * lambda * Tp);
        #endif
        if(epsilong<0 ){
            epsilong=0;
        }
/*small Tp is not suitable for the ϵ estimation*/
        if (Tp > 2){
            ((MobileNode *) node)->update_lifetime(Tr);
```

```
                        ((MobileNode *) node)− >update_epsilong(epsilong);
                     }
                  }
               }
            }
            /*what to do if the link is still available*/
            if (((MobileNode *) nodea)− >distance((MobileNode *) nodeb)<250.0){
               if($T_r$>$T_p$){
            /*re-predict lifetime*/
                  double lifetime = ((MobileNode *) nodea)
                                    -> lifetime((MobileNode *) nodeb);
            /*reset lifetime*/
                  v -> setlifetime(lifetime);
               }
            }
         }
      }
   }
stat.reset();
}
```