Autonomic Tuning of Routing for MANETs

Yangcheng Huang
Ericsson Ireland Research Centre
LM Ericsson
yangcheng.huang@ericsson.com

Saleem Bhatti
School of Computer Science
University of St Andrews
saleem@cs.st-andrews.ac.uk

Sidath Handurukande
Ericsson Ireland Research Centre
LM Ericsson
sidath.handurukande@ericsson.com

***Abstract*—In Mobile Ad hoc Networks (MANETs), timers have been widely used to maintain routing (state) information. The use of fixed-interval timers is simple to implement but, in practise, may be difficult to configure in dynamic operational environments, and so may give reduced performance in the presence of frequent topology changes. This paper proposes a self-tuning timer approach within a simple control system for MANET routing protocols with the aim of allowing dynamic, autonomic, re-calibration of routing update frequencies. A novel dynamic timer algorithm is presented to automatically tune routing performance by adapting timer intervals to network conditions. Our simulation results have shown that, compared to the default fixed timer approach, the proposed algorithm could effectively improve routing throughput without manual configuration.**

I. Introduction

Mobile Ad hoc Networks (MANETs) are self-organising multi-hop wireless networks, consisting of mobile nodes connected by wireless links. There is no other network infrastructure, so the mobile nodes act as routers of traffic for other nodes in the network, as well as being sources and sinks of traffic. The nodes in MANETs are free to move arbitrarily. Also, the nodes might not take part in routing when conserving power or if they are re-started. Thus, the network topology may change randomly and rapidly, through a combination of node mobility, node power management policy and node failure. Meanwhile, each node has to maintain certain routing information to other nodes in the network. After initial start-up, timers have been widely used in neighbour detection and topology advertisements of MANET routing protocols to keep the routing information up-to-date and maintain the correctness of route selection. The two main functions that need to be performed are:

- **Neighbour Detection.** In most MANET routing protocols, periodic HELLO messages are exchanged between neighbouring nodes to detect link dynamics and to maintain node status. This is true for a wide range of protocols: proactive protocols like OLSR (Optimised Link State Routing protocol) [1], DSDV (Destination-Sequenced Distance-Vector Routing) [2], TBRPF (Topology Broadcast based on Reverse-Path Forwarding) [3]; or reactive protocols like AODV (Ad hoc On-Demand Distance Vector Routing) [4], TORA (Temporally-Ordered Routing Algorithm) [5]; or even hybrid protocols like ZRP (Zone Routing Protocol) [6],
- **Topology Advertisements.** Proactive routing protocols like OLSR [1], TORA [5] and TBRPF [3] propagate periodic network-wide *topology update* (or *topology control (TC)*) messages to advertise topology changes. In addition to initiating new link state in topology repositories of each node, for example, the topology advertisement process in OLSR removes obsolete topology state, either implicitly by assigning sequence numbers to topology advertisements, or by state time-out.

Use of timers for controlling state updates and state maintenance – the *soft-state* approach – have the following benefits:

- **Resilience.** Routing state is expected to be self-healing in the presence of network failures, so that the loss of the state would not result in more than a temporary loss of service given that connectivity exists.
- **Simplicity.** Reliable message delivery is not required, since message loss can be restored by subsequent messages. In addition, routing state would eventually expire and be removed unless periodically refreshed by the receipt of a refresh message indicating its validity. Accordingly, no explicit state removal is required.

It is clear that crucial for effective operation of a soft-state approach is the value of the timer interval used to control state updates and state maintenance. Despite the simplicity of a timer-based approach, significant concerns have been raised about the use of *fixed timer intervals* in MANET protocols. State validity of each route repository is determined by the value of these intervals. Despite its simplicity and widespread use, the fixed-interval timer approach may not provide the best routing performance under highly dynamic network conditions like MANETs. Some issues with the timer-based approach are:

- **Manual configuration.** Timers with fixed intervals have to be manually configured by administrators. The value of the timer intervals is determined mainly based on recommendations of original protocol designers or estimated by network administrators. Usually, it is not practical to perform analytical evaluations or experimental evaluations in order to configure and re-configure the intervals of various soft-state timers. Moreover, timers may be related making configuration more complex. For instance, the timer interval of HELLO messages [1], [2] should be (much) longer than the time-out interval of neighbour entries. Consequently, existing approaches have been found to be expensive in manpower, prone

to errors, and incapable of scaling to the needs of large networks. So, as an arbitrary choice, different timers often use the same (fixed) interval.

- **Unawareness of network conditions.** Fixed-interval timers do not allow for differences in network conditions, such as node velocity and link loss rate, which impact directly routing performance. Questions may arise regarding the configuration of timer intervals. For example: does the default value of timer intervals work well against all types of link failures under various scenarios? Or: given requirements on system consistency, how do we determine the value of the timer intervals in order to achieve the best balance between performance and overhead? For instance, in order to keep low the traffic overhead due to control messages, topology advertisement intervals of MANET routing protocols are usually set to relatively large values, e.g. 5s in OLSR [1]. In a high-density network with fast mobility, the change rate of topology is relatively high. However, topology changes would not be advertised until the update timer expires. Under such circumstances, topology changes might be too frequent to be captured by *periodic* updates.
  Another consequence of such unawareness is resource wastage. In real-world scenarios, the nodes' mobility is more likely to be intermittent. Also, there might be only a fraction of the node population moving during a certain time period. Therefore, keeping a constant refresh rate, i.e. maintaining fixed timer values, for all nodes may lead to unnecessary resource consumption: of network capacity, due to extra transmissions; of CPU and memory usage due to additional processing; and, of course, this impacts battery usage.
- **Balancing throughput and overhead.** It is commonly believed that a smaller timer interval for refreshing soft-state could speed up adaptation to changes at the expense of increased overhead. However, there are no studies on *how much* it could improve the consistency against the amount of traffic overhead. Especially, this question is critical to MANET protocols. On one hand, topology changes require effective signalling (i.e. smaller timer intervals) to maintain routes and so maximise the throughput. On the other hand, the resource constraints of MANETs require minimum control overhead to reduce channel contention and battery consumption.

There have been several adaptive routing proposals for mobile networks. For instance, Sharma et al [7] proposed a scalable timer approach to constrain the volume of control traffic in a per-session based timer approach, where the refresh interval is adjusted according to the fixed available bandwidth (pre-allocated for control traffic) and the amount of state to be refreshed. Benzaid et al [8] presented an approach that adjusts refresh frequency based on node mobility and the multi-point relay (MPR) status of its neighbouring nodes. Boppana et al [9] proposed an Adaptive Distance Vector (ADV) routing algorithm by adopting flexible route update strategies according to network conditions.

These approaches are usually targeted to reduce control overhead. In addition, the performance of these proposals depends primarily on the accuracy of network measurement [8], [9]. That is, information for the purpose of controlling routing updates is dependent upon accurate estimates of real-time network/traffic characteristics, and in practise these may not be available, either from additional services (e.g. network management or measurement function), or from the application itself. These approaches also lead to increased systems complexity.

For example, let us consider ADV [9]. The operations in zone maintenance and continuous network monitoring not only introduce extra processing overhead but also increase the complexity in configuration and implementation. The performance of ADV is determined by constant trigger thresholds, which need to be manually configured. Furthermore, the performance bounds of these proposals are not well-defined. For example, in ADV, the route update frequency increases quickly with node mobility, which brings larger overheads than periodic updates. Also, since only partial route information is maintained, ADV takes longer for a new connection to find a valid route [9].

A. Our Work and Contribution

This paper proposes a solution to these problems by providing a self-tuning method for MANET routing protocols, to adjust refresh timer intervals based on network conditions. One of the objectives of our solution is to increase the throughput while limiting the control message overhead. Compared with existing approaches, our proposal could effectively improve routing performance *without* manual configuration, or reliance on any external services for information. To demonstrate adaptable, autonomic control, a dynamic timer algorithm is proposed as part of a feedback control system to adjust the timing of routing state updates and maintenance for DSDV (Destination Sequenced Distance-Vector Routing) [2], a classical MANET routing protocol based on the Bellman-Ford algorithm. The protocols behaviour (i.e. parameter values) is tuned according to the status of hosting environments such as node mobility and channel loss rate, in order to improve routing performance and eliminate the need for manual tuning. Our contributions are:

- **Independent operation.** The operations of the proposed algorithm are independent of network measurement or node mobility detection. Through internal monitoring of link change events, we propose a simple method in detecting network conditions and tuning timer intervals.
- **Simple configuration and implementation.** Compared to our recent improvements in OLSR [10], the adaptability process in this paper is inspired by feedback-based control theory to be fully automated, introducing no extra control parameters. The proposed algorithm can be introduced incrementally into existing protocols.
- **Demonstration of operation.** We have shown through simulations that the proposed self-tuning algorithm out-

performs a traditional MANET distance vector routing protocol.

The rest of the paper is organised as follows. Section II gives some background information on existing distance-vector routing algorithms. Section III gives the detailed description of the proposed algorithms. Section IV introduces the simulation configurations used in this study. Section V presents our observations based on simulations in *ns2*. Conclusions are summarised in Section VI.

II. Distance Vector Routing

A. Protocol Overview

The Destination Sequenced Distance-Vector (DSDV) [2] routing protocol is a table-driven proactive routing algorithm based on the Bellman-Ford routing algorithm. Each node in the network maintains a routing table that records distance vectors, i.e. the number of hops to all of the possible destinations within the network and the corresponding next-hop nodes.

The main improvement made to the basic Bellman-Ford algorithm is the loop-free property by marking nodes with sequence numbers, which are used to distinguish stale routes from new ones. A DSDV update packet contains an unique sequence number (SN). The transmitter assigns this SN, and the receiver selects the packet with the highest SN, i.e. the most recent route. The route labelled with the most recent sequence number is always used. In the event that two updates have the same sequence number, the route with the best metric (i.e. lowest number of hops) is used in order to optimise the path. An advantage of DSDV is that in relatively stable networks like a Wireless Personal Area Network (WPAN), incremental updates are sent to avoid extra traffic. Its main disadvantage is that in fast changing networks, like mobile networks, as the number of incremental update packets increases rapidly, then full dumps are preferred, or DSDV requires *bidirectional links* to operate, so that links are treated as symmetric in terms of metrics and so routing state is reduced. In summary, the salient DSDV functions are:

- **Neighbour sensing.** New neighbours can be detected by exchanging periodically the routing tables. There are two proposals for link breakage detection: either by use of the layer-2 protocol, or by use of a time-out (if no routing table updates have been received for a period from an existing neighbour). When a link to a next hop is broken, any route through that next hop is immediately assigned a metric of ∞ so that it should not be selected for data delivery.
- **Topology update.** DSDV requires each node to advertise its own routing table by broadcasting its entries to each of its current neighbours *locally*. In order to reduce the amount of state information carried in each update and help alleviate the potentially large amount of topology update traffic, DSDV employs two types of update packets. *Full updates* carry all available routing information and might require multiple network protocol data units (NPDUs). Full updates can be transmitted relatively infrequently when no movement of mobile nodes occurs. *Incremental updates* carry only information changed since the last full update. The size of incremental updates is smaller than that of full updates, and therefore can fit into a standard NPDU. When movement becomes frequent, the size of incremental updates increases, and approaches the size of a NPDU. Then, a full update can be scheduled so that the next incremental update will be smaller.

B. Convergence Analysis

Here, we analyse the route convergence latency L, i.e. the period from the occurrence of the change (i.e. when route inconsistency occurs) to the time the nodes in the network update their route state repositories (i.e. converging to route state consistency again). A summary of definitions is given in Table I.

A node running a distance-vector protocol detects link changes and updates its route tables. These link changes are advertised by exchanging route tables between the neighbouring nodes. So, the route convergence latency is the sum of link detection latency, L_d, and route advertisement latency, L_a.

We assume that:

- The message exchange events in each node are independent. The intervals of message broadcasting in any two nodes conform to a uniform random distribution.
- The change event of links is an independent, identically distributed Poisson process with arrival rate λ. The assumption is reasonable, if the node degree is small and the nodes are moving randomly so that the process of route change is also random.

TABLE I
Definitions in Convergence Analysis

L	Route convergence latency
L_d	Link detection latency
L_a	Route advertisement latency
r	Route table update interval
d_i	Time gaps between successive messages sent by neighbouring nodes
m	Route length [hops]

Let r be the update interval. Let d_i $(i = 1,2,3,...m)$ be the time gaps between successive messages sent by neighbouring nodes. If we assume $d_i \sim U(0,r)$, then:

$$E(L_a) = \sum_{i=1}^{m} E(d_i) = m.\frac{r}{2} \quad (1)$$

So, we can see that the route advertisement latency, L_a, is directly proportional to the refresh interval, r.

Our previous work [11] has revealed the expected link detection rate as follows[1].

$$E(L_d) = r + \frac{e^{-r\lambda} - 1}{\lambda} \quad (2)$$

[1]We assume no packet loss. Please refer to [11] for more details.

So, the expected route convergence latency, $E(L)$, is:

$$\begin{aligned} E(L) &= E(L_d)+E(L_a) \\ &= r+\frac{e^{-r\lambda}-1}{\lambda}+m.\frac{r}{2} \end{aligned} \quad (3)$$

Based on this expression, the relationship between route convergence latency L and update interval r is shown in Fig 1 for three values of λ (0.05, 0.50, 1.00) with $m = 4$.

In summary, the analysis above shows that the update interval has a significant impact on route convergence. Specifically, the convergence speed of routes is approximately inversely proportional to the route update intervals. So, tuning the update interval may result in improvements in routing performance.

One simple method to improve the route convergence speed is through reducing the route update interval. However, increasing the frequency of control message dissemination in this way increases control overhead and resource consumption (e.g. bandwidth, CPU, memory, and battery power). So, it is necessary to find efficient methods in tuning update intervals in order to improve route convergence without introducing excessive overhead.

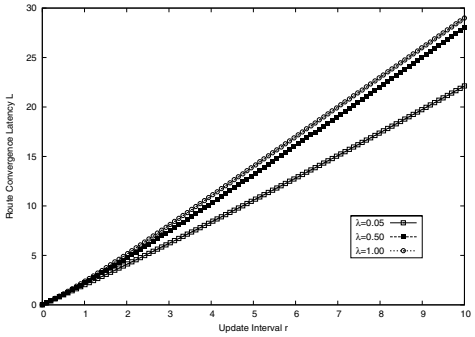


Fig. 1. Route Convergence Latency vs. Update Interval ($m = 4$)

III. Self-tuning Distance-Vector Routing

In this section, we propose a self-tuning distance-vector algorithm (sDV) to adapt dynamically the route update intervals to network conditions. It has been inspired by control-theoretic adaptive mechanisms similar to those widely adopted in the Internet, e.g. Additive Increase Multiplicative Decrease (AIMD) of TCP, which is used to adjust sending rates in response to network congestion. Our approach in this algorithm uses an *Additive-Increase Additive/Multiplicative-Decrease (AIAMD)* controller to adapt the route update interval, r, to the conditions of node mobility and data traffic.

Briefly, if there are more link change events during the previous update period (i.e. $\lambda_n > \lambda_{n-1}$), the update interval is reduced *additively*, so that the failure rate during the next update cycle should be smaller, considering the reduced update interval. However, if the *variances* between link change rates during previous update periods increase (i.e. $\lambda_n - \lambda_{n-1} > \lambda_{n-1} - \lambda_{n-2}$), this indicates a larger change and so we need to react more quickly and to reduce the update interval more aggressively (i.e. halving).

Whenever there is a reduction in the occurrences of link change events (i.e. $\lambda_n < \lambda_{n-1}$), the update interval is incremented *additively*, in order to reduce the routing overhead and reduce resource wastage. The additive increase element of the algorithm is inversely proportional to the update interval r and the number of route changes.

In addition, a deliberate random variation (jitter) in timing [12] of update generation, is employed in order to reduce simultaneous packet transmissions from multiple nodes (i.e. synchronisation).

In Fig 2, we see the simple control model we have. If we consider the Neighbour Detection control loop (right side of Fig 2), we see that the current value of the update interval, r_{nb}, is used by the neighbour detection function, as input to the timer tuning controller. Here, Algorithm 1 (with definitions in Table II) is used to update the value of r_{nb}, based on the rate of link change events (λ_n). If the current update period has seen more frequent changes than in the previous update period, then we have the interval reduced. However, if the link change rate is slowing, then we have the interval increased gradually. Note that we also make the algorithm sensitive to both the size of the routing table (*size_of_rtbl*) and the number of route changes (*rtbl_chg_cnt*) and use these factors to control the value of the additive (increase/decrease) element. For example, we use a smaller additive increase element when the table is large, which facilitates a quicker response to network changes.

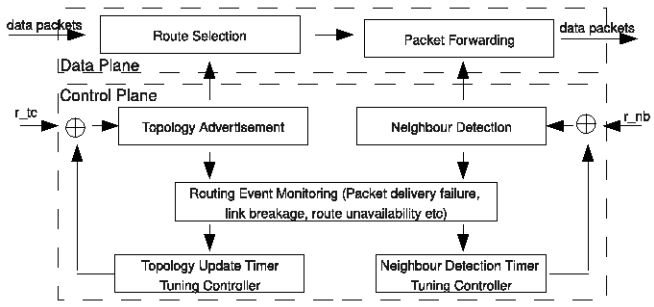


Fig. 2. Self-tuning Routing Support

TABLE II
Definitions in Self-tuning Algorithm

r	Update interval
link_chg_cnt	Number of link changes within current refresh period
size_of_rtbl	Size of route table
rtbl_chg_cnt	Number of changes of route entries within current refresh period
λ_n	Link change rate of current refresh period
λ_{n-1}	Link change rate of previous refresh period
λ_{n-2}	Link change rate of the period before previous

IV. Performance Analysis

We integrate our proposed algorithms with the DSDV implementation which runs in version 2.31 of *ns2* [13] and uses

Algorithm 1 Self-tuning Algorithm

Calculate(link_chg_cnt)
Calculate(rtbl_chg_cnt)

$\lambda_n \Leftarrow \frac{link\_chg\_cnt}{r}$

if $(\lambda_n > \lambda_{n-1})$ **then**
  if $(\frac{\lambda_n - \lambda_{n-1}}{\lambda_{n-1} - \lambda_{n-2}} > 1)$ **then**
    $r \Leftarrow \frac{r}{2}$
  else
    $r \Leftarrow r - \frac{1}{size\_of\_rtbl.r}$
  end if
end if

if $(\lambda_n < \lambda_{n-1})$ **then**
  $r \Leftarrow r + \frac{\lambda_{n-1}}{\lambda_n} \cdot \frac{1}{rtbl\_chg\_cnt.r}$
end if

$\lambda_{n-2} \Leftarrow \lambda_{n-1}$
$\lambda_{n-1} \Leftarrow \lambda_n$
$r \Leftarrow r.(0.75 + Random(0, 0.25))$ {to prevent synchronisation}

the ad-hoc networking extensions provided by CMU [14]. The detailed configuration is shown in Table III. We use a network consisting of n nodes: $n = 20$ to simulate a low-density network, $n = 50$ to simulate a high-density network. Nodes are placed in a 1000×1000 m^2 field. All simulations run for 100s. The radio range (radio radius) was 250m. We use the Random Trip Mobility Model, "... a generic mobility model that generalizes random waypoint and random walk to realistic scenarios ..." [15] and performs perfect initialisation. Unlike other random mobility models, Random Trip reaches a steady-state distribution without a long transient phase and there is no need to discard initial sets of observations. The mean node speed, v, ranges between 1m/s to 30m/s. For example, when the mean node speed is 20m/s the individual node speeds are uniformly distributed between 0m/s and 40m/s. The average node pause time is set to 5s.

TABLE III
MAC/PHY LAYER CONFIGURATIONS

MAC Protocol	IEEE 802.11
Radio Propagation Type	TwoRayGround
Interface Queue Type	DropTailPriQueu
Antenna Model	OmniAntenna
Radio Radius	250m
Channel Capacity	2Mbits
Interface Queue Length	50

A randomly distributed Constant Bit Rate (CBR) traffic model is used which allows every node in the network to be a potential traffic source and destination. The rate of each CBR flow is 10kb/s. The CBR packet size is fixed at 512 bytes. There are at least $n/2$ data flows that cover almost every node.

For each datum point presented, 100 random mobility scenarios are generated. The simulation results are thereafter statistically presented with the mean of the metrics and the error bars at $\pm$ 1 standard deviation. This reduces the chance that the observations are dominated by a certain scenario which favours one protocol over another.

In each simulation, we measure each CBR flow's throughput and control traffic overhead and then calculate the mean performance of each metric as the result of the simulation. Throughput is considered as the most straight-forward metric for MANET routing protocols [16], and is computed as the amount of data transferred (in bytes) divided by the simulated data transfer time (the time interval from sending the first CBR packet to receiving the last CBR packet). Considering the broadcast nature of the control message delivery, the control Overhead is calculated by summing the size of all the control packets *received* by each node during the simulation period.

To make clear the benefit of our approach, we also evaluate the percentage of (traffic / overhead) reduction (POR). Let T be the average throughput and O be the control overhead. Then the percentage of throughput reduction is:

$$T_{POR} = \frac{T_{r=r_0} - T_{sdv}}{T_{r=r_0}} \tag{4}$$

The percentage of overhead reduction is:

$$O_{POR} = \frac{O_{r=r_0} - O_{sdv}}{O_{r=r_0}} \tag{5}$$

V. OBSERVATIONS

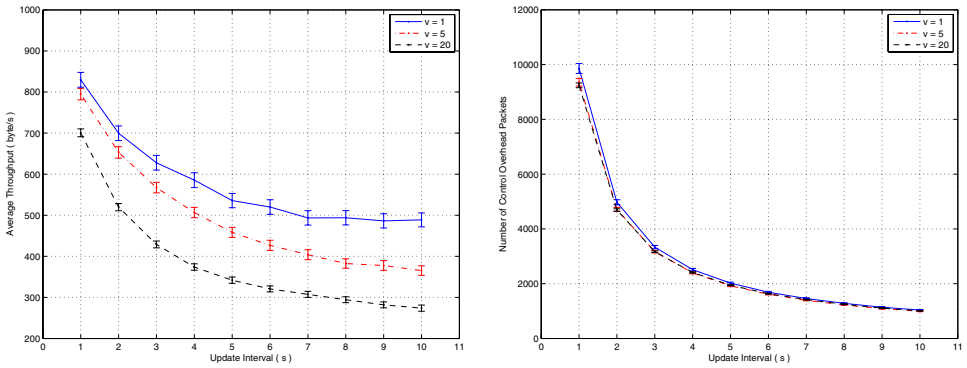
In this section, we compare the routing performance of the proposed adaptive routing algorithms with that of an unmodified version of DSDV, and present the observations under the variation of selected parameters, such as node velocity and node density. In each of Figs. 3, 4, 5 and 6, subfigure (a) plots Average Throughput on the vertical axis, subfigure (b) plots the Number of Control Overhead Packets on the vertical axis, and subfigure (c) plots the POR on the vertical axis. *intVal* refers to the update interval of DSDV.

For the DSDV configurations where $intVal \geq 2$, the throughput values are much smaller than the case where $intVal = 1$ (Fig. 3(a) and 4(a)). Since we are interested in increasing the throughtput, we did not consider the configurations where the throughtput is lower (i.e. $intVal \geq 2$). For similar reason, in Fig 5(c) and 6(c), we only consider the DSDV configuration where $intVal = 1$. For reference purposes, we have plotted the performance of the configuartion where $intVal = 2$ in Fig. 5(a), 5(b), 6(a) and 6(b).

A. Impact of Update Intervals on Routing Performance

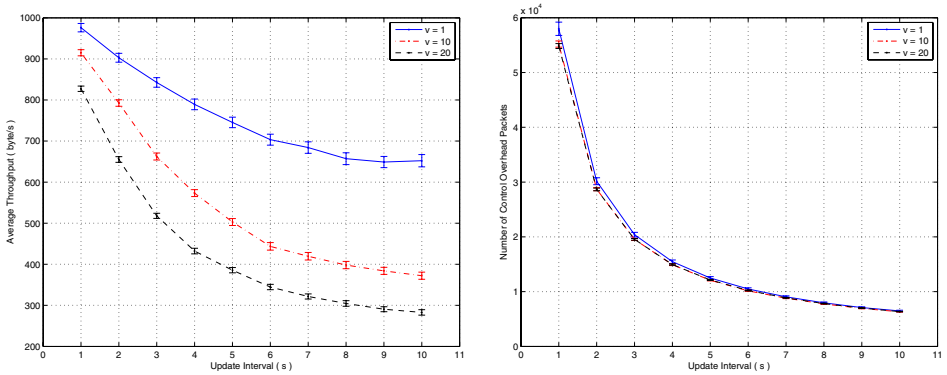
From Fig 3(a) and Fig 4(a) we can see that the average throughput decreases with the increase of update intervals across different values of node velocity, v. This matches well our analytic results in Section II-B.

As expected, Fig 3(b) and Fig 4(b) show that the control overhead of DSDV is approximately inversely proportional to the route update intervals.



(a) Throughput (b) Overhead

Fig. 3. Impact of Update Intervals on DSDV Performance ($n = 20$)



(a) Throughput (b) Overhead

Fig. 4. Impact of Update Intervals on DSDV Performance ($n = 50$)

B. Self-tuning Distance Vector Routing

As shown in Fig 5 and Fig 6, the proposed sDV algorithm achieves as good performance as standard DSDV with smaller update interval (i.e. $intVal = 1$) but with a much lower overhead. For example, the overhead of sDV is ∼15-20% lower than standard DSDV, while the throughput degradation is only ∼2-5% (Fig 5(c) and Fig 6(c)). It is worth noting that the throughput of sDV exceeds that of standard DSDV (i.e. $T_{POR} < 0$), with a significant overhead reduction of ∼22.5% in low-density networks (Fig 5(c)) and ∼15.5% in high-density networks (Fig 6(c)). This demonstrates clearly the benefits of our *sDV* algorithm described in Section III. The control overhead is reduced because the route update interval is adjusted automatically to a large value when there are no network changes.

Compared to standard DSDV with a larger interval (i.e. $intVal = 2$), sDV shows good adaptability to node mobility. With the increase in node mobility, the throughput drop of sDV is less significant than standard DSDV with larger interval. For example, as shown in Fig 6(a), when the node velocity increases from 10m/s to 20m/s, sDV has a ∼10% performance drop, while standard DSDV (i.e. $intVal = 2$) has a drop of ∼20%. This matches our expectations. When the link change rate increases, the *sDV* algorithm increases the route update rate so that route inconsistency can be recovered quickly.

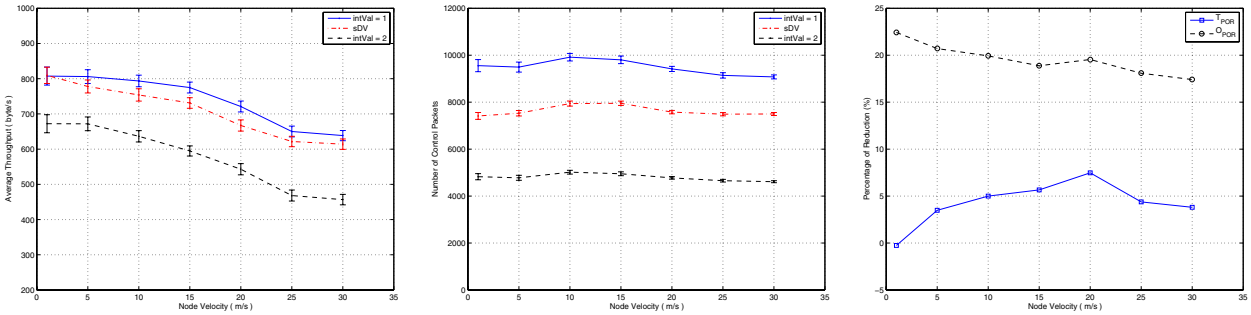
On the other hand, sDV has more overhead than standard DSDV with larger interval (i.e. $intVal = 2$). As shown in Fig 5(b) and Fig 6(b), the overhead of sDV is ∼40% more than standard DSDV with larger interval. However, such overhead increase introduced by sDV is still much smaller than the reducing update interval to $1s$ with DSDV.

To summarise, the simulation results show that *sDV* improves the standard DSDV proactive routing algorithm in terms of the balance of throughput and overhead under the conditions simulated.

VI. Conclusions

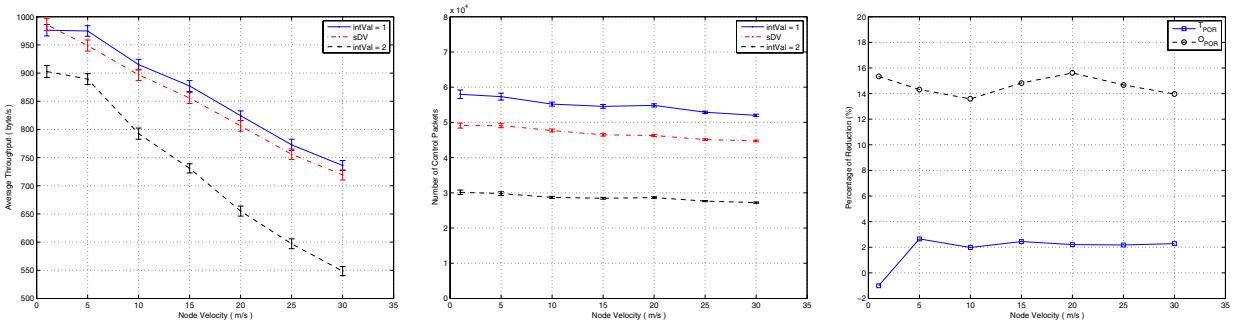
This paper proposes a self-tuning method for automated timer configuration, to adjust timer intervals of routing protocols based on network conditions. In particular, a novel dynamic timer algorithm is presented to automatically tune routing performance by adapting timer intervals to network conditions. Our simulation results have shown that, compared to the default fixed-timer interval approach, the proposed algorithms could effectively improve routing performance without manual configuration.

The original data, the source code and the scripts used in this study are all available from the authors on request.



(a) Throughput (b) Overhead (c) Percentage of Reduction

Fig. 5. sDV Performance ($n = 20$)



(a) Throughput (b) Overhead (c) Percentage of Reduction

Fig. 6. sDV Performance ($n = 50$)

VII. Acknowledgements

The authors would like to thank Brian Lee and Anne-Marie Bosneag for their valuable comments on this paper. Yangcheng Huang and Sidath Handurukande are funded by European Commission's Marie Curie Host Fellowships program.

References

[1] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, a. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol," in *Proceedings of the IEEE INMIC*, 2001.

[2] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *Proceedings of the ACM SIGCOMM Conference on Communications Architectures, Protocols and Applications*, London, UK, 1994, pp. 234–244.

[3] B. Bellur and R. G. Ogier, "A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, New York, NY, USA, March 1999, pp. 178–186.

[4] C. E. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, July 2003.

[5] V. D. Park and M. S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Kobe, Japan, April 1997, pp. 1405–1413.

[6] Z. J. Haas and M. R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," Internet Draft (draft-zone-routing-protocol-02.txt), June 1999.

[7] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, "Scalable Timers for Soft State Protocols," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Washington, DC, USA, 1997, p. 222.

[8] M. Benzaid, P. Minet, and K. Agha, "Integrating Fast Mobility in the OLSR Routing Protocol," in *Proceedings of the Fourth IEEE Conference in Mobile and Wireless Communications Networks*, Stockholm, Sweden, September 2002.

[9] R. V. Boppana and S. Konduru, "An Adaptive Distance Vector Routing Algorithm for Mobile Ad Hoc Networks," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, USA, April 2001, pp. 1753–1762.

[10] Y. Huang, S. Bhatti, and S. Sorensen, "Adaptive MANET Routing for Low Overhead," in *Proceedings of the First IEEE WoWMoM Workshop on Adaptive and DependAble Mission- and bUsiness-critical mobile Systems, WoWMoM 2007*, Helsinki, Finland, 2007.

[11] Y. Huang and S. Bhatti, "Model Based Analysis of Soft State Signaling Protocols," in *Proceedings of the London Communications Symposium*, London, UK, September 2005.

[12] T. Clausen, C. Dearlove, and B. Adamson, "Jitter Considerations in Mobile Ad Hoc Networks (MANETs)," Internet Draft (draft-ietf-manet-jitter-04.txt), December 2007.

[13] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/.

[14] Wireless Extensions to ns-2, http://www.monarch.cs.rice.edu/cmu-ns.html.

[15] J. Y. L. Boudec and M. Vojnovic, "Perfect Simulation and Stationarity of a Class of Mobility Models," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, Miami, USA, 2005.

[16] S. Corson and J. Macker, "Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," IETF, RFC 2501, January 1999.