Min-cut Determination for Linear Network Code in Wireless Ad Hoc Networks

Su-Kit Tang, Lingxiong, Li, and Dongyang Long

***Abstract*—In this paper, we propose an implementation neutral protocol, called Multiple Disjoint Path (MDP), for the determination of the min-cut in wireless ad hoc networks by summarizing the capacity of disjoint paths constructed from source node to sink nodes. Min-cut is essential in the formulation of data being sent using Linear network code in source node. MDP consists of virtual source routing and core heuristic. Virtual source routing is to resolve scalability, privacy and efficiency problems caused by long paths in large networks. Core heuristic is to ensure the efficiency of request packet propagation. Our simulation results reveal that MDP performs at a satisfactory level in dense networks in terms of connectivity and transmission efficiency. An opportunity for deployment of linear network code for data transmission in dynamic wireless environment can be created.**

***Index Terms*—Network Coding, Ad Hoc, Multiple Disjoint path, Routing**

I. Introduction

Network coding is proved to be able to optimize network resources for transmission in a network at a rate close to the min-cut of a network [1][2]. In Figure 1, two bits of data, *a* and *b*, sending from *s* to t_1 and t_2, are transmitted by two downlinks individually. When *a* and *b* meet at node *y*, they collide as *y* cannot transmit both at once. Encoding at node *y* is required so that delay caused by data collision between *a* and *b* can be avoided. Encoding involves loss-less transformation of two individual pieces of data into one for a single transmission instead of two. Encoded data is then recovered at sink nodes, t_1 and t_2. To some degrees the encoding reduces the transmission time for packets in entire journey if collision occurs. Usage of edges in the network are utilized and both sink nodes are able to receive the two-bit data without delay caused by data collision at node *y*.

If Linear network code [2] is used, the encoding at node *y* requires source node *s* to formulate its data into a form of linear independent equations before send. With the possession of linear independency in data being sent, data decoding at sink nodes can be achieved successfully after encoding at intermediate nodes. Therefore, the min-cut of a network is crucial in the encoding process. The min-cut determines the maximum flow of data in one transmission to all sink nodes from a source node and the linear independency of data in the encoding process. Hence, the main question is how to obtain the min-cut of the network, especially in wireless ad hoc networks, to apply Linear network code in transmission. The contribution of our paper is to find out the min-cut of a wireless ad hoc network. This specific problem we consider has not been studied previously. The concept of our solution is to construct some disjoint paths, connecting sink nodes to the source node, in the network. Different data at different paths can be transmitted at the same time. Encoding is involved if two disjoint paths intersect. By counting the minimum capacity of disjoint paths among sink nodes, the min-cut of a network is obtained. With the knowledge of min-cut, the source node can formulate its data using Linear network code. Figure 2 depicts the network in Figure 1 by disjoint paths for two bits of data, *a* and *b*, transmitting to t_1 and t_2 from *s*. It is clear that the min-cut of the network from source node *s* is 2 as there are two disjoint paths for each sink nodes, assuming that all links are of unit capacity.

In this paper, we propose MDP in an abstract way that is neutral in implementation for flexibility. MDP constructs maximal disjoint paths for a source node in one-to-many transmission by extending SMR with implicit source routing. In section II, we reviews some routing algorithms for ad hoc wireless networks that are essential to MDP. In section III, MDP will be proposed and discussed. We will show how the min-cut of a wireless ad hoc network can be determined from the disjoint paths constructed by MDP.

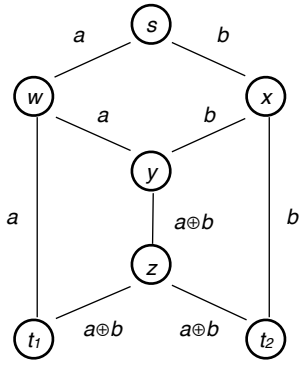

Figure 1. Transmission of Two-bit Data Using Linear Network Code.

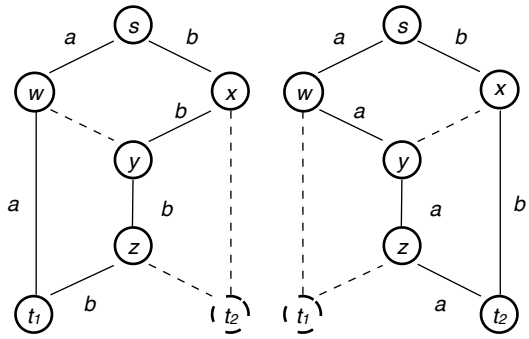

Figure 2. Transmission of Two-bit Data by Disjoint Paths.

This work was partially sponsored by the Natural Science Foundation of China (Project No. 60273062, 60573039) and the Guangdong Provincial Natural Science Foundation (Project No. 04205407, 5003350)

Mr. Su-Kit Tang is a Doctor candidate of the Sun Yat-Sen University, GuangZhou, China. He is also with the Macao Polytechnic Institute, Macao, China (phone: 853-599-6440; email: sktang@ipm.edu.mo)

Mr. Lingxiong Li is a Doctor candidate of the Sun Yat-Sen University, GuangZhou, China (email: lilx@mail2.sysu.edu.cn)

Professor Dongyang Long is with the Sun Yat-Sen University, GuangZhou, China (email: issldy@mail.sysu.edu.cn)

Lastly, we conclude this paper. For declaration, this paper does not consider aspects such as resource and energy allocation, but focuses on optimally constructing disjoint paths in order to assist for min-cut determination. We also do not consider the encoding of linear network code from an optimality perspective as it is out of scope of this paper.

II. Related works

MDP builds maximal disjoint paths for a source node in one-to-many transmission by extending SMR with implicit source routing. Therefore, DSR is essential to MDP. DSR [3] is a single path routing algorithm that setups a path from sender to destination by forwarding the address of each node in a path to the destination. Sender will maintain a route cache that it has learned. If a route is found in the cache, it uses the route to transmit the packet. If a route is not found, it initiates a route request and the route request will be propagated in the ad-hoc network until it reaches the destination. When the destination receives the route request, it will return the route record (The path), in which it contains all intermediate nodes' addresses along the trip, to the sender. Significantly, DSR suffers a scalability problem that the propagation of route record may involve unexpected size of source routing information. This creates impact on limited wireless bandwidth. Therefore, Hu proposed the use of implicit source routing in wireless ad hoc networks that preserves the advantage of source routing while avoiding the associated per-packet overhead in most cases [4]. It introduced a cache table in each participating nodes. This cache table maintains a list of next hop addresses for each particular routing path, indexed by a flow identifier. A source may then send any packets headed by a flow identifier in lieu of a source route. This avoids of the overhead caused by the source route. In Split Multipath Routing (SMR) [5], an extension to DSR, disjoint paths are constructed by spreading request paths in a network. Its objective, which is different from ours, is to build maximally disjoint paths for load balancing transmissions over the network. Data traffic is split into multiple routes to avoid congestion and to use network resources efficiently. Similar to DSR, SMR is designed for one-to-one transmission. It also suffers from unexpected length of route record in large networks.

III. The Protocol

To determine the min-cut of a network in dynamic wireless environment, we propose a Multiple Disjoint Path (MDP) construction algorithm which setups disjoint paths from sink nodes to a source node. MDP constructs disjoint paths based a virtual source routing concept and the core heuristic. Virtual source routing is to resolve scalability, privacy and efficiency problems caused by long paths in large networks. Core heuristic is to ensure the efficiency of request packet propagation among multiple sink nodes.

When some sink nodes initiate communication with a source node, request packets will be created and broadcasted by sink nodes. Request packets spread across the wireless network until they reach the source node. As request packets travel, intermediate nodes, receiving these packets, are required to process and rebroadcast the packets. A simple processing job on request packets involved in a node is to extend a virtual path by assigning sequential virtual address of a virtual path to the node. Since communication in wireless network is done by broadcasting, duplicated request packets of a sink node may be received. Nodes have to make correct and accurate decision on request packet processing and forwarding.

We assume for simplicity that communication request is initiated by sink nodes. A source node will accept requests from sink nodes. We also assume that the clocks of all nodes are synchronized so that data transmission is handled in time unit. One time unit is a block of time of some length and it is not specified for flexibility of implementation in networks of different conditions.

A. Model and Definitions

We first define our model of communication system in wireless ad hoc networks.

Definition 1 (The network) We model a wireless ad hoc network as a directed graph $G = (V, E)$, where V is a finite set of nodes and E is a finite set of directed links. Each node in the network has a sufficiently large random number generator, $\mathscr{G}$. The node set V contains a node s, called the source node, which only has out-going edges. The node set T, which is subset of V, contains all sink nodes. The node set $U = V - s - T$ contains intermediate nodes that make up paths to connect the source node s to some sink nodes in T. In our model, an edge, denoted by $e(a, b)$ for $(a, b) \in V$, is a acyclic link from node a to node b. We also assume for simplicity that each edge has a unit capacity.

Definition 2 (Request packet) A request packets, r_i in a finite set of request packets R, is generated by $t_i \in T$ for communication with s. We denote a request packet by r_i: $\{t_i, s, b_i, vid_i, c_i\}$, where t_i is the initiator of the request packet, s is the source node the packet destines at, b_i is the time unit the packet travels, vid_i, at the first node in a virtual path, is a unique virtual path id, and c_i is a hop count that can give the length of the virtual path and the position of a node in a path. vid_i and c_i also act as a virtual address of a node when they come together as it is unique in the path. For simplicity and without lost of generality, we assume that a request packet r_i will travel from a sink node t_i to a source node s at the time unit b_i.

Definition 3 (Virtual path) A virtual path, p, is a subset of V, which contains a collection of nodes $\{s, (u_1,...,u_n) \in U, t \in T\}$ that a request packet $r \in R$ has visited. For a virtual path, $p_i \in P$, p_i must be unique among all paths for t such that $p_i \cap p_j = \{s, t\}$, for $i \neq j$ and $(p_i, p_j) \in P$. This is the disjoint property of a virtual path. Two virtual paths, p_i and p_j, are parallel, if they run to two different sinks with intersecting nodes u_k, such that $p_i \cap p_j = \{s, u_k\}$, for $i \neq j$, some k and $(p_i, p_j) \in P$.

Definition 4 (Virtual path table) Each node $u \in U$ will maintain a virtual path table, denoted by VPT_u, for all received request packets, $r \in R$, with unique sink nodes until they timeout. We assume for simplicity that the request packet timeout in VPT_u is flexibly defined to some time units for different network environment.

B. Virtual Source Routing

The virtual source routing (VSR) constructs virtual paths that contain virtual addresses of intermediate nodes a request packet has visited. Each virtual path is identified by a unique virtual path id, *vid*, which is generated by the first node after sink node. Along with a hop counter of a virtual path, c, virtual addresses of a node in a path can be determined. A node is now identified by a virtual address. No address privacy issues would be raised among nodes if this is a concern. As a request packet travels, c in virtual address is incremented at each node. This maintains the sequential order of virtual addresses in a path virtually. A

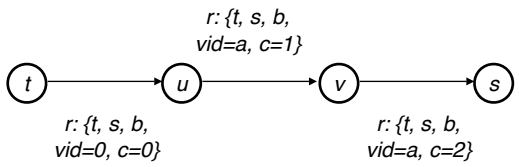

Figure 3. Virtual Source Routing Operation

virtual path is always determined by *vid* and nodes in the same path would have the same *vid*.

VSR creates a virtual path, *p*, by spreading request packet *r*. When a sink node, *t*, initiates a transmission session, it broadcasts a request packet *r*. A node, *u*, first receiving *r*, as indicated by *vid*=0, will randomly generate a sufficiently large number by $\mathscr{G}$ for *vid*. *vid* is assumed to be unique as it is sufficiently large and collision-free. The hop count, *c*, indicates the position of this node in a virtual path as well as the length of the path at source nodes. Neighbor nodes receiving *r* will verify whether *r* is new at current time unit by *vid* and *c*. If it is new, *c* are incremented and *r* will be rebroadcasted. Otherwise, *r* will be ignored. This avoids the broadcast storm problem discussed in [6]. This processing repeats all the way in a virtual path at nodes receiving *r* until the source node is reached. Thus, a virtual path *p* is created. Figure 3 shows an overview of the virtual source routing operation.

In Figure 3, a virtual path, *p*, is constructed by *s* in a transmission session initiated by *t*. First, *t* broadcasts a request packet *r* that containing *vid=0,* and *c=0*. Nodes *u* receives *r* and rebroadcast *r* after generating *vid=a* and incrementing *c* by one. Subsequent node *v* receiving *r* will increment the path variable *c* by one, and rebroadcast *r*. Gradually, source node *s* will receive *r* containing *vid=a*, and *c=2*, which indicates that a 2-hop virtual path *p*, identified by *vid=a*, from node (*a, 2*). The virtual path $\{t \rightarrow u \rightarrow v \rightarrow s\}$ is constructed.

A node receiving a request packet *r* will propagate and maintain *r* in its virtual path table for a transmission session until *r* expires. Since *vid* and *c* can uniquely identify a node in a path, quick routing decision can be made. A node is also needed to maintain a virtual path table in cache for a period of time. It records all virtual paths that the node resides until timeouts.

C. Core Heuristic

We define disjoint paths as non-intersecting paths, running from a sink node to a source node. These paths do not consist any common intermediate nodes if they are parallel. Figure 2a illustrates two disjoint paths running from *s* to t_1. To ensure that paths are disjoint with each others for a sink node, a path discovery heuristic is required to verify the originality of a request packet. This enables a node to determine if a request packet should be accepted when there are many of same originality. Note that parallel virtual paths are constructed by request packets from different sink nodes. They are assumed to be of no conflict of interest, so they would not be considered in the heuristic. The heuristic includes the following rules.

Rule 1 (Non-broadcast storming) A request packet r_i: $\{t_i, s_i, b_i, vid_i, c_i\}$ will be accepted if virtual path table VPT_u does not show r_j: $\{t_j, s_j, b_j, vid_j, c_j\}$ where $i \neq j$, $t_i = t_j$, $s_i = s_j$ and $b_i = b_j$.

A request packet r_i is used to initiate a communication from a sink node t_i to a source node s_i at a time unit b_i. When a node *u* receives r_i, *u* can determine if r_i should accept it by looking up its VPT_u. If r_j is not found in VPT_u such that $t_i = t_j$, $s_i = s_j$ and $b_i = b_j$, r_i will be accepted, processed and rebroadcasted.

Rule 2 (Non-intersecting) A request packet r_i: $\{t_i, s_i, b_i, vid_i, c_i\}$ may be accepted if virtual path table VPT_u shows r_j: $\{t_j, s_j, b_j, vid_j, c_j\}$ where $t_i = t_j$, $s_i = s_j$, $b_i = b_j$, $vid_i = vid_j$ and $c_i \neq c_j$.

Request packets are running directionless and a node *v* may receive request packets, r_i and r_j, initiated by the same sink node, $t_i = t_j$, and destined at the same source node, $s_i = s_j$, at same time unit, $b_i = b_j$. Since vid_i and vid_j are collision-free, r_i and r_j are shown to be running their own paths as vid_i would be different from vid_j. r_i and r_j are originated by the same communication request but of different paths. Running both request packets through node *v* would lead to a single delivery path only. Therefore, node *v* can only take either r_i or r_j. However, the evaluation of request packets raises an optimization issue in this situation.

Rule 2.1 (Selection criteria) A request packet r_i of (vid_i, c_i) will be accepted if virtual path table VPT_u shows r_j of (vid_j, c_j) where $vid_i = vid_j$, $c_i \neq c_j$ and $c_i < c_j$, such that r_i replaces r_j.

In this paper, the term *optimal* is defined to be minimal hop count as the amount of energy consumed in a transmission would be less, assuming that the energy consumption for one broadcast operation is same for all nodes. The shorter the path is, the less the energy it consumes, in one transmission. Based on this rationale, node *v* will accept r_i as $c_i < c_j$. This will trigger a rebroadcast of request packet with updated information for the change of the path segment constructed previously if r_i comes after r_j. In case of $c_i = c_j$, a request packet from a farthest distance is in preference comparatively. This can encourage the construction of shortest path in a dense area. However, determination of the distance of neighbor nodes is out of scope of this research.

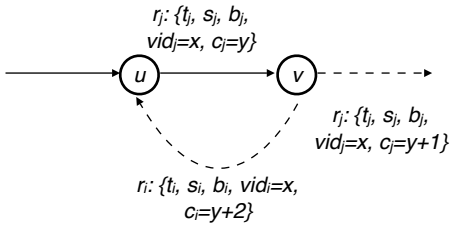

Figure 4. A node *u* rejecting a loop-back request packet r_i from node *v* during path construction in MDP algorithm

Rule 3 (Loop-free) A request packet r_i: $\{t_i, s_i, b_i, vid_i, c_i\}$ will not be accepted if virtual path table VPT_u shows r_j: $\{t_j, s_j, b_j, vid_j, c_j\}$ where $t_i = t_j$, $b_i = b_j$, $vid_i = vid_j$ and $c_i > c_j$.

Suppose that node *u* is somewhere after node *v* in a path construction from sink node *t* to source node *s* at time unit *b* in Figure 4. Request packet r_i of (vid_i=*x*, c_i=*y*+2) from node *v* is running into node *u*. VPT_u shows r_j of (vid_j=*x*, c_j=*y*) where $vid_i = vid_j$ and $c_i > c_j$. From Rule 2.1, node *u* will not accept r_i. In addition, a condition $vid_i = vid_j$ holds. We can see that r_i and r_j are of same originality and it is a loop.

Rule 4 (Shadow path) A request packet r_i: $\{t_i, s_i, b_i, vid_i, c_i\}$ will be accepted if virtual path table VPT_u shows r_j: $\{t_j, s_j, b_j, vid_j, c_j\}$ where $t_i = t_j$, $s_i = s_j$, $b_i = b_j$, $vid_i = vid_j$, and $c_i < c_j$, such that r_i replaces r_j.

A shadow path is a path that is an extension of a segment of another path. Suppose that a request packet r_i of (vid_i, c_i)

is running into a node v along with another request packet r_j of (vid_j, c_j). If r_i and r_j are running in with two different paths, $vid_i \neq vid_j$, recalled that vid is a sufficiently large random number and collision-free. However, it is contradicting and $vid_i = vid_j$. r_i and r_j are actually one delivery path. The selection criteria is same as Rule 2.1 as only one request packet is allowed.

Based on the rules discussed above, we have the pseudo-code of MDP presented below. The MDP will take a request packet r as a parameter to implement the Rule 1 by lines 10 to 11; Rule 2 by lines 12 to 16; Rule 3 by lines 12, 17 to 19 and Rule 4 by lines 12, 17, 20 to 22. Depending on the type of a node, MDP(r) will do corresponding tasks. If it is a sink node, it initiates r and broadcast r. If it is an intermediate node, it follows the core heuristics to process r. If it is a source node, it accepts request packet r if it is not from a shadow path.

```
Algorithm MDP(r)
1.   if this is a sink node then
2.     t ← this
3.     s ← source node (destination)
4.     b ← now
5.     vid ← 0
6.     c ← 0
7.     broadcast r = (t, s, b, vid, c)
8.   else if this is an intermediate node then
9.     result = VPT_lookup(r)
10.    if result = null then
11.      VPT_add(r)
12.    else if result = path_found then
13.      if r and result are from diff paths then
14.        if r is better than result then
15.          VPT_replace(r)
16.        end if
17.      else if r and result are same path then
18.        if r.c > result.c then
19.          do nothing...loop
20.        else if r is better than result then
21.          VPT_replace(r)
22.        end if
23.      end if
24.    end if
25.    broadcast r = (t, s, b, vid, c + 1)
26.  else if this is a source node then
27.    result = VPT_lookup(r)
28.    if r and result are not shadow then
29.      VPT_add(r)
30.    end if
31.  end if
```

D. Min-cut Determination

To determine the min-cut of a network using MDP, it is simply by looking at the capacity of disjoint paths from each sink nodes. By taking the minimum capacity of paths from sink nodes, the min-cut can be determined. Since a virtual path is assumed to be a unit capacity in this paper, the min-cut of a network would be the minimum number of disjoint paths per a sink node among all sink nodes. In Figure 1, s is supposed to receive four request packets (four virtual paths) of four different vid, in which two are from t_1 and two are from t_2. Therefore, taking the minimum number of disjoint paths per sink node among all sink nodes, the min-cut of the network is found to be two. It implies that 2 bits of data can be sent to all sink nodes in one time unit, if linear network code is used.

IV. Performance Evaluation

To evaluate the performance of MDP, we implemented MDP in ns2 [7], an open source network simulator, and

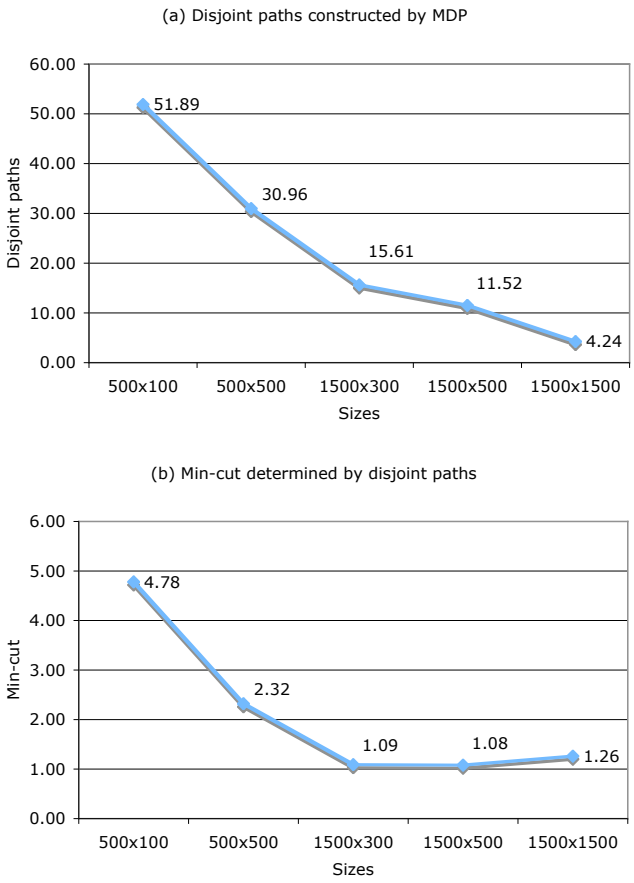

Figure 5. Performance of MDP in Networks of Different Sizes

conducted a set of simulations using the following settings: IEEE 802.11b standard at MAC layer implementation; 50 nodes randomly placed on areas of $500\times100m^2$, $500\times500m^2$, $1500\times300m^2$, $1500\times500m^2$ and $1500\times1500m^2$ for different network density environments; 1000 randomly generated scenarios for each area; Five sink nodes and one source node are assigned. Since this study is new in the research field, our experiment is designed to evaluate how MDP performs in environments of different network density. In our simulation, a run was conducted for each scenario at each area and collected data was averaged over 1000 scenarios. The metrics we consider are:

(a) Disjoint Paths constructed: The number of disjoint paths constructed from all sink nodes to the source node.
(b) Min-cut: The min-cut of the network from the source node, based on the disjoint paths constructed.

We observe that MDP constructs more disjoint paths in dense networks. This happens because MDP propagates its packets by broadcasting and nodes will get involved in packet delivery. Connectivity among nodes is comparatively higher. In Figure 5a, the curve runs downwards from higher density area to lower density area, so the number of disjoint paths MDP can construct gets less. However, there are still more than one disjoint path running to each sink node in area of size $1500\times1500m^2$, in which only 2.83 sink nodes can connect to the source node on average. It is due to the network partitioning problem caused in large area. We also observe that the min-cut in dense networks is higher than those in sparse networks. Figure 5b shows that the min-cut drops from 4.78 to 1.26. Likely to the connectivity of sink nodes, the efficiency of data transmission follows the downwards trend. From the aspect of connectivity and

Network size (m^2)	Density (m^2 per node)	Disjoint paths (Figure 5a)	Min-cut (Figure 5b)
500x100	1000	51.89	4.78
500x500	5000	30.96	2.32
1500x300	9000	15.61	1.09
1500x500	15000	11.52	1.08
1500x1500	45000	4.24	1.26

Table 1. Summary of MDP performance

transmission efficiency, MDP shows satisfactory performance in dense networks comparatively. A summary of our simulation results is shown in Table 1.

V. Conclusion

In this paper, we proposed a multiple disjoint paths construction algorithm, called Multiple Disjoint Path (MDP), to determine the min-cut of a network by taking the minimum number of disjoint paths per sink node among all sink nodes. Using Linear network code, a source node can formulate its data into a form of linear equations with linear independency so that data recovery at sink nodes can be achieved successfully. Our simulation reveals that, from the aspects of connectivity and transmission efficiency, data transmission using Linear network code would perform better in dense networks than sparse networks. This is because the min-cut of a network is better as the density of a network is higher. In addition, as an extension to MDP with implicit source routing, MDP does not suffer from the scalability, privacy and efficient problems caused by long paths in large networks. Thus, MDP creates an opportunity for deployment of Linear network code in data transmission in dynamic wireless environment, especially in dense networks.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Leung, "Network information flow", *IEEE Trans. Information Theory*, vol. 46, no. 4, July 2000, pp. 1204-1216.

[2] S.-Y. R. Li, R. W. Leung, and N. Cai, "Linear network coding", *IEEE Trans. Information Theory*, vol. 49, no. 2, February 2003, pp 371-381.

[3] D. Johnson, D. Maltz & Y. Hu, "Dynamic Source Routing in Ad Hoc Wireless Networks", *Internet Draft*, draft-ietf-manet-dsr-10.txt, work in progress, July 2004.

[4] Y.-C. Hu & D. Johnson, "Implicit Source Routes for On-Demand Ad Hoc Network Routing", *Proc. of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*, Long Beach, CA, USA, October 2001, pp.1-10.

[5] Lee, S.-J., Gerla, M., "Split Multipath Routing with Maximally Disjoint Paths in Ad Hoc Networks", *IEEE International Conference on Communications*, Vol. 10, 2001

[6] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network", *ACM/IEEE Mobicom*, Seattle Washington, USA, August 1999, pp. 151-162.

[7] Network Simulator (NS-2), Available at *http://www.isi.edu/nsnam/ns/index.htm*.