國立交通大學

網路工程研究所

碩 士 論 文

在網狀網路下利用線性組合技術實現管線式的臨機路由

Pipelined Opportunistic Routing with Linear Combination in Wireless Mesh Networks

研 究 生：林育任

指導教授：黃俊龍　教授

中 華 民 國　九 十 八　年 九 月

在網狀網路下利用線性組合技術實現管線式的臨機路由

Pipelined Opportunistic Routing with Linear Combination in Wireless Mesh Networks

研 究 生：林育任　　Student：Yu-Jen Lin

指導教授：黃俊龍　　Advisor：Jiun-Long Huang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年九月

在網狀網路下利用線性組合技術實現管線式的臨機路由

研 究 生：林育任　　指導教授：黃俊龍

國立交通大學網路工程研究所碩士班

摘　　要

臨機路由是近年來無線網路傳輸領域中的一項重要的探索方向，其可以增加在網狀網路中使用 unicast 情況時的效能。基本的臨機路由技術會使各個節點高頻率地接收到相同封包而造成網路的負擔，因此有研究提出利用排程的方法來避免收到相同的封包，但排程的方式亦存在過於浪費頻寬、以及需要大量的控制訊息之問題。利用線性組合的技術可以改進在排程中需要等待優先權較高的節點回傳訊息後才能傳送本身之封包的缺點，方法是將資料切成多個區段並對同一個區段中的封包進行線性組合，而其中該於何時傳送下一個區段及如何決定可幫忙傳送的節點則成為一個重要的議題。若每次必須等到目的地端收完某區段中的封包並回送 acknowledgment 給來源端後才能傳送新的區段的話，會造成效率的低落，若是在大範圍的網路底下，更會因為 acknowledgement 回送時間過長而使效率更差。為了增進傳送的效能，我們提出了名為 PipelineOR 的演算法來解決這個問題，主要的觀念在於當來源端傳送某段時間後，中介節點可能已經收滿足夠的封包可供解碼，此時中介節點便可以取代來源端傳送封包，藉此降低為了等待目的地端收滿全部的封包後來源端才能開始傳送下一段區塊所造成的延遲。另外因為重複的封包對於解碼並無幫助，演算法也會觀察在網路中是否有不需要用到的節點，藉由使優先權較低的節點停止傳送來降低碰撞和互搶頻寬的機會。

Pipelined Opportunistic Routing with Linear Combination in Wireless Mesh Networks

Student：Yu-Jen Lin Advisor：Jiun-Long Huang

Institute of Network Engineering College of Computer Science

National Chiao Tung University

Abstract

Recently, Opportunistic Routing has been widely explored to improve the unicast performance in wireless mesh networks. A number of recent papers introduce opportunistic routing in wireless mesh networks by utilizing schedule in order to avoid receiving duplication packets in node. Unfortunately, all of them prevent spatial reuse and thus may underutilize the wireless medium because of the node must wait until higher priority node completes. With linear combination, opportunistic routing can be implemented without complex scheduler protocol. Linear combination partitions the stream into multiple segments and combines only packets in the same segment. It is significant to decide when the next segment can be sent. In previous studies the source can send next segment only when destination receives enough packets to decode and retransmit acknowledgement to source. In this paper, we propose PipelineOR, a new protocol that uses linear combination method and select necessary forwarding nodes to improve throughput. In our algorithm, if one node receives a packet from another forwarder it also can know all packets that have stored in upstream node instead of just one re-combining packet and corresponding decoding coefficiences. In linear combination, we can only check the coefficient to determine whether forwarding nodes have enough packets to decode. If yes, send an acknowledgement to the sender. When the sender receives this acknowledgement it can send next segment packet as soon. We also check the dependency for each node and prune unnecessary node to avoid collision. Experimental results show that PepelineOR's throughput is 15%-20% higher than CodeOR.

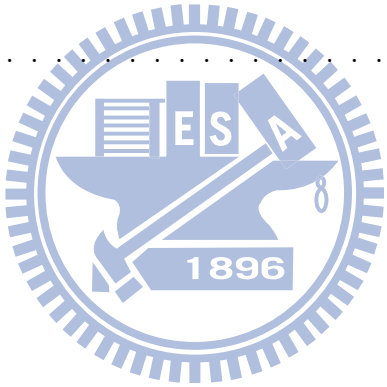
誌　　謝

首先我要感謝我的家人和朋友對我的支持與鼓勵，才能讓我能夠無後顧之憂的完成碩士學位。再來要感謝我的指導教授黃俊龍老師在我求學時間中的細心教導和關心，以及分享其求學過程中的經驗和解決問題的方式，讓我的論文能夠順利完成。感謝口試委員們對論文所提出的建議，使論文能更完整。最後感謝實驗室的學長姐、同學和學弟們，當我遇到問題時，經過與他們討論已經適時的鼓勵，讓我在研究和學習之餘，能得到壓力的紓解和前進的動力，因為有他們才能讓我在求學中多了許多樂趣和喜悅。

Contents

List of Figures

List of Tables



Chapter 1

Introduction

A wireless mesh network is a network created through the connection of wireless access points installed at each network user's locale. Each network user is also a provider, forwarding data to the next node. The networking infrastructure is decentralized and simplified because each node need only transmit as far as the next node. Figure 1.1 shows a generic mesh network topology that consists of a sender node s, a cloud of forwarding nodes, and a destination node d. Owing to interference and unreliability of link quality, routing in wireless mesh network is

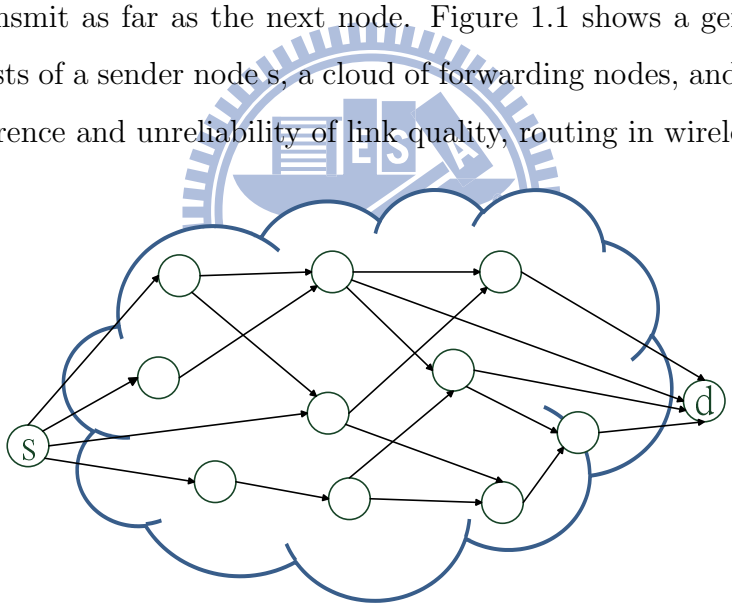


Figure 1.1: General topology for mesh network

a challenge to maximize throughput. Traditional routing focused on finding a fixed shortest path between a pair of nodes and sent data through that sequence nodes. This protocol makes the most sense when each pair of nodes is linked by a wire. For wireless networks it is not an idea approach due to the reason that wireless link with a shared communication channel is a broadcast communication medium in nature.

The main idea of opportunistic routing [3] is, instead of preselecting a single specific node to be the next-hop forwarder for a packet, and multiple nodes that closer to the destination can potentially be served as forwarder candidates. The reason is wireless broadcasting sharing

channel in nature. When the current node transmits a packet via a single-hop broadcast, all the candidate nodes that overhear the packet will coordinate with each other to determine which one would actually forward the packet according to scheduling algorithm along number of control messages. The node that is closest to the destination will perform the forwarding while the rest will simply drop the packet even they have successfully received it. For this property, opportunistic routing can take advantage of the potentially numerous, yet unreliable wireless links in the network when they actually deliver. In contrast, traditional routing in wireless networks only targets a packet to the preselected next-hop forwarder, which is the node on a preselected path towards the destination of the packet.

In [4], Chachulski et al. proposed trading structure for randomness, called MORE, for solving complex scheduling algorithm along large number of control message in opportunistic routing. The fundamental insight is that, with random mixing of coded packets, although multiple receivers overhear the same packet in a wireless broadcast neighborhood, they are able to generate linearly independent coded packets with high probability, by combining the received coded packet with existing packets in their buffers. In order to constraint of computational complexity, MORE partitions the data into multiple segments and combines only packets in the same segment. In [13], Lin et al. proposed CodeOR which allows the source to transmit a window of multiple segments concurrently and reduces packet size for decreasing the decoding delay. The primary concept is that, reducing stop-and-wait problem in large scale topology and fully utilizes network resources.

The previous researches about opportunistic routing with linear combination exist an open challenge that severely affects performance: When shall the source stop transmitting coded packets of one segment and move on to the next one? If the source stops prematurely, the destination cannot decode the entire segment. However, if the source stops too late, bandwidth resource is wasting. Unfortunately, no matter in MORE or CodeOR the latency for source move on to next segment is long and not efficient.

In this paper, we propose a new protocol named PipelineOR standing for Pipelined Opportunistic Routing, that uses concept of the older segments are closer to destination. We also use linear combination and select necessary forwarding node to improve throughput. All of previous research about network coding with opportunistic routing focus on each forwarding node just transmits one recombining packet and decoding coefficient vector to it downstream node but ignores the number of packets stored in every current node. In our algorithm, each node not only transmits recombining packet but also transmits all packets coefficients that has received. Using this idea can help source transmits next segment soon, because if any

node knows that forwarding nodes have stored enough packet to decode, it will send acknowledge to source and the source is not necessary to send packet with the same segment. In the same case, when time passes by, the older segment will closer to destination so the number of segment flight in the network at the same time is more than other study. We also check the dependency of each node and prune unnecessary node to avoid collision.

The remainder of this paper is organized as follows. Chapter 2 presents preliminaries, including introduction of opportunistic routing, linear combination, related work and definition. Chapter 3 describes our algorithm for opportunistic routing with linear combination. Performance evaluation is presented in Chapter 4. Finally, Chapter 5 makes a conclusion for this thesis.

Chapter 2

Preliminaries

In this section, we will give some preliminaries. Section 2.1 and Section 2.2, introduce concept about opportunistic routing and linear combination, respectively. Related works for wireless opportunistic routing are presented in Section 2.3. In Section 2.4, we define notion using in this paper

2.1 The concept of Opportunistic Routing

Traditional wireless networks use routing techniques similar in wired networks. Multi-hop wireless networks typically use routing techniques similar to those in wired networks [16], [17]. These routing protocols choose the best sequence of nodes between the source and destination, and forward each packet through that sequence.

A fundamental insight of opportunistic routing work as follows. A source node wants to transmit a packet to a destination. Due to the reason that wireless channel in broadcasting is shared, so some forwarder may overhear data from another nodes. Thus, nodes between source and destination are willing to participate when data delivering. When source broadcasts the packet and some "sub-set" of the nodes will receive the packet. The nodes run a protocol to discover and agree on which nodes are in that sub-set. The node in the sub-set that is closest to the destination broadcasts the packet and other nodes cannot broadcast. Again, the nodes that receive this second transmission agree on the closest receiver, which broadcasts the packet. This process continues until the destination has received the packet.

The reason for opportunistic routing throughput better than traditional routing is that each transmission may have more independent chances of being received and forwarded. Consider the scenario in Figure 2.1. The delivery probability from the source to each intermediate is only 10%. The delivery probability from each intermediate to the destination is 100%.

Traditional routing would route all the data through the same intermediate. The high loss rate would require each packet to be sent an average of ten times before being received by the intermediate, once more to reach the destination. Opportunistic routing to be sent an average of $1/(1-(0.9)^{100})$ times before being received by the intermediate.

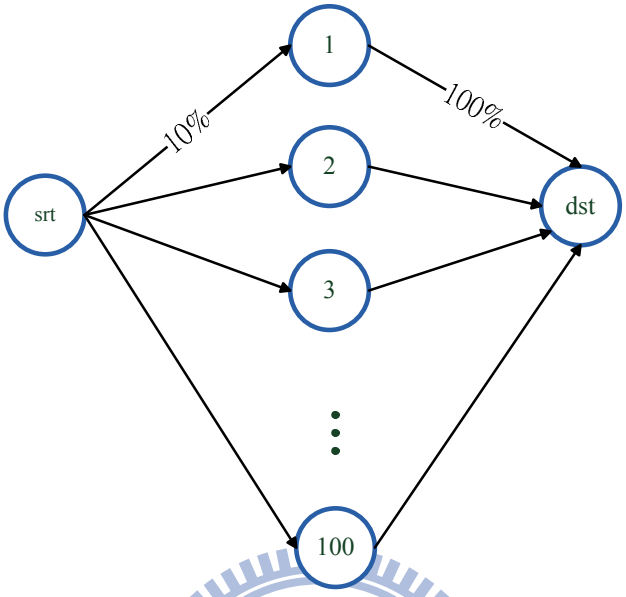


Figure 2.1: Example in which each of the source's transmissions has many independent chances of being received by an intermediate node

Packets may transmit unexpected far due to wireless network share communication channel. This is another reason for opportunistic routing might improve throughput than traditional routing. Considering Figure 2.2, in which the source is separated by a chain of nodes leading towards the destination. Delivery probability decreases with distance. Traditional routing would forward data through some sequence of the chain, for example source-C-D-destination. If a packet transmission from the source falls short of C, just reaching A, then that transmission is always wasted in traditional routing, and the source must re-send the packet. If a transmission reaches farther than C, for example all the way to D, traditional routing cannot make use of that luck. Opportunistic routing, in contrast, can often take advantage of both of these situations. In the former case, A will re-send the packet, allowing it to make some progress. In the latter case, D will forward the packet, eliminating one transmission.

2.2 Introduction to Linear Combination

In traditional communication networks, data packets delivery by store-and-forward mechanisms in which the intermediate nodes forward a copy of data packets that they have received.

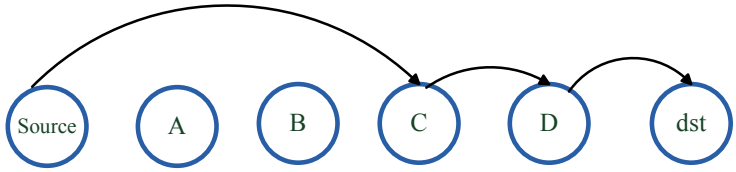


Figure 2.2: Example in which the source's transmissions may make different amounts of progress towards the destination.

With linear combination, a network node is allowed to combine several packets that it has generated or received into one or several outgoing packets instead of simply forwarding data. Combination messages can be decoded at destination if it receives sufficient data. [12] shows constructively that the linear combination can achieve the capacity of a single multicast session as well. In fact, the unreliability and broadcast nature of wireless links make wireless networks a natural setting for linear combination.

In this paper, we use random linear combination, both the combination and decoding operations can be regarded as matrix multiplication over a Galois field. The data to be transmitted from the source is divided into multiple segments and be encoded with the same segment. As shown in Figure 2.3, a coded packet p' with random network coding is a linear combination.Each p' encodes k source packets $p_0, \ldots, p_{k-1}$ in a segment with the form, $\sum_{i=1}^{k} \alpha_i p_i$ where α is a matrix composed of random coefficients in the Galois field GF(2^8).

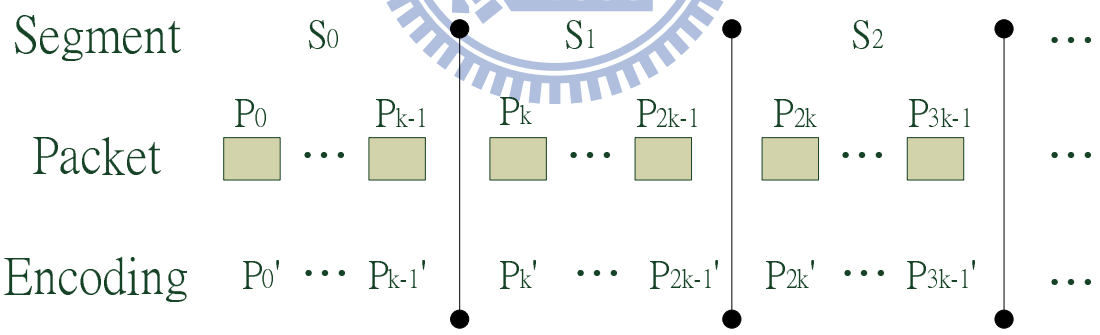


Figure 2.3: Linear network coding

For reduces redundant transmissions, forwarding node accepts an incoming packet only it is independent of existing received ones and drops the dependent packet. The forwarders can re-combination packets in buffer and broadcasting the resulting packets to downstream nodes. The re-combination operation replaces the combining coefficients accomplished with the original combined packets with another set of random coefficients. The ability of re-combination enables forwarders to avoid the severe packet redundancies in store-and-forward routing protocols. Since a coded packet carries information from not only the newly coming packet, but also existing ones that were opportunistically received. For this reason, when a

forwarding node wishes to transmit combined packets within a segment, in traditionally, it produces a coded packet p'' by combination all combined packets in its buffer belonging to the segment, namely p'_1,. . . ., $p_{m-1}{}'$, where m is the total number of coded packets in the buffer that belong to the segment:

$\sum_{i=0}^{m-1} \beta_i p'_i$

The decoding operation of k source packets at destination is equivalent to solving a linear system composed of all coded packets received . The decoding matrix represents the coefficient matrix of such a linear system. When the rank of the decoding matrix is k, the linear system can be solved ,otherwise the destination continues to receive coded packets from its neighbor until all k source packets are decoded.

2.3 Related Works

Opportunistic routing was first proposed by S. Biswas and R. Morris in [16]. In [3], ExOR is proposed as a source-based opportunistic routing protocol by effectively utilizing the wireless broadcast medium to increase network throughput. Unlike traditional best path routing protocol, which neglected the wireless broadcast advantage. ExOR selects a group of candidate nodes (instead of a single one) as the potential next-hop forwarder. The node with the smallest ETX (Expected number of Transmission times) to the destination and successively receives the current packet, will relay the packet further. As shown in Figure 2.4 introduces how to calculate ETX. To prevent unnecessary transmissions, ExOR introduces a special complex packet scheduling algorithm. Experimental results are described in the paper to verify that ExOR can significantly improve the network throughput. Many variations of opportunistic routing protocols [19], [23], [24] based on ExOR have also been proposed.

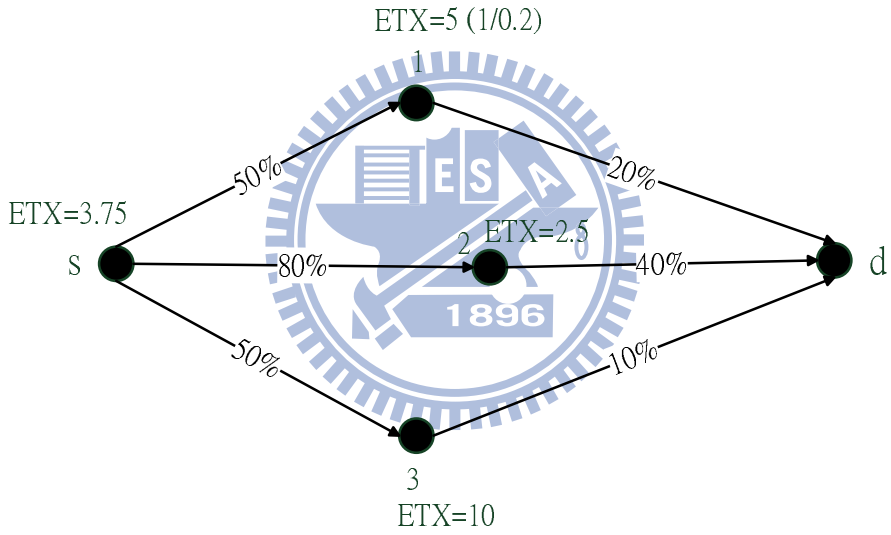


Figure 2.4: ETX calculates

Chachulski et al. [4] have proposed a practical opportunistic routing protocol based on linear combination, referred to as MORE In particular, MORE is MAC-protocol independent and avoids duplicates by randomly mixing packets before forwarding. By this way, MORE is feasible to be implemented in practice, and achieves higher throughput than ExOR. However, MORE uses a stop-and-wait protocol with a single segment, which is not efficient utilizing the delay-bandwidth product in large-scale networks.

In [13], Lin et al proposed using small segment size to decrease decoding delay and transmitting multi-segment to improve throughput. Since the destination needs to wait to accumulate a sufficient number of combined packets before it is able to decode the segment, a smaller segment size is beneficial towards reducing such a delay, which is important to real-time mul-

timedia applications. With CodeOR, one may use a smaller segment size, since the number of segments that are in flight in the network adapts to its estimated delay-bandwidth product. In other work, CodeOR allows the source to transmit sliding windows of multiple segments using opportunistic routing, and with segmented linear combination. The time for source to move on next segment is dependent on source receives E-ACK that sent from destination. In large-scale networks CodeOR also has long delay time from destination sends acknowledge to source. The number of segment in flight in the network at the same time dependent on sliding windows size. In other word, it may be inefficient when forwarding nodes closer to destination have sufficient packets to decode but source does not transmit new segment.

With the observation that error probabilities of symbols are much lower than that of packets on a wireless link, MIXIT [10] improves the performance of MORE by operating network coding on symbols rather than on packets as in MORE, where a packet consists of multiple symbols and a symbol is the smallest transmission unit over the wireless link. With such a simple modification, MIXIT can utilize correct symbols in a corrupted packet, and therefore attains higher throughput than MORE. However, as an extension to MORE, MIXIT adopts the same stop-and-wait paradigm, and hence shares the same drawback as MORE in large-scale networks. Halloush et al. [7] proposes multi-segment mixing to improve single segment encoding. However, it will increase the computational complexity.

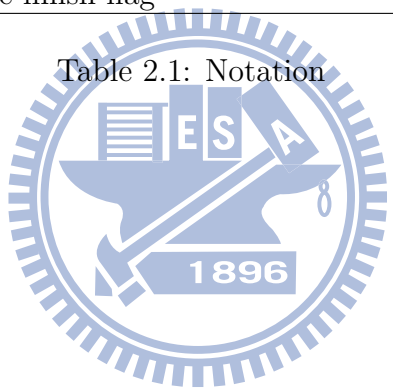
A number of recent papers [18], [22], [21] have used different variants of optimization frameworks to extend MORE to scenarios involving multiple sessions. They require the transmission of a large number of control messages in a timely and reliable fashion, which may not be practical in loss wireless networks. More importantly, none of these papers has raised efficient algorithm on the constraints of the stop-and-wait paradigm on session throughput and care about forwarding node selection.

2.4 Definition

In this section we define notion used in this paper. The k means partition size of one segment and segment number is defined S_i. The localMaxETXHost means the node has maximum ETX that stores some decoding packets. Every sub-net has its localMaxETXHost, so we define globalMaxETXHost for the maximum localMaxETXHost in network. Former is defined former localMaxETXHost. Total-Segment and Finish-Flag means total segments for source can transmit and process finish flag, respectively.The notations are shown in Table 2.1.

Notation	**Meaning**
s	Source of the mesh network
d	Destination of the mesh network
S_i	Segment number
k	Partition size of one segment
$localMaxETX$	Maximum ETX node that stores some decoding packets
$globalMaxETX$	The maximum $localMaxETX$
$StoreforMaxETX$	Recoding $localMaxETX$ in each node
$Former$	The former $globalMaxETX$
$Total-Segment$	The total segment number
$Finish-Flag$	The finish flag

Table 2.1: Notation



Chapter 3

PipelineOR : Pipelined Opportunistic routing with Linear Combination in Wireless Mesh Network

3.1 Motivation

For previous researches, we are inspired by the following observation. When nodes on the multiple paths between the source and destination have sufficient data in segment i, it can represent the source has produced enough combining packets for segment i. Therefore, the source can transmit next segment i+1 to avoid stop-and-wait condition acutely. As shown in Figure 3.1, if node 2 and 3 have received a sufficient number of combined packets in segment i, source may start to transmit the next segment i+1 after receiving acknowledge from node. In MORE, however, s continues to transmit segment i even when node 2 and 3 have obtained all the required coded packets in this segment, until the end-to-end acknowledge from destination. Hence, the latency for source moves on to the next segment will become serious in large scale network. In CodeOR [13], it uses sliding windows technique to reduce latency for source stops current segment and sends the next segment. Actually, the resource is wasting due to the reason that all nodes store the same segment. Source moves on next segments when it receives E-ACK from destination. In our algorithm, we improve these drawback. For example, if node 3 notifies that has sufficient packets to decode in forwarding node then node 3 will send acknowledgement to source to transmit next segment. Segment may different in each node because the nodes closer to destination have sufficient packets to decode, so nodes that far to destination can clear buffer and receive new segment.

We also notice that collision and bandwidth contention may decrease performance, so

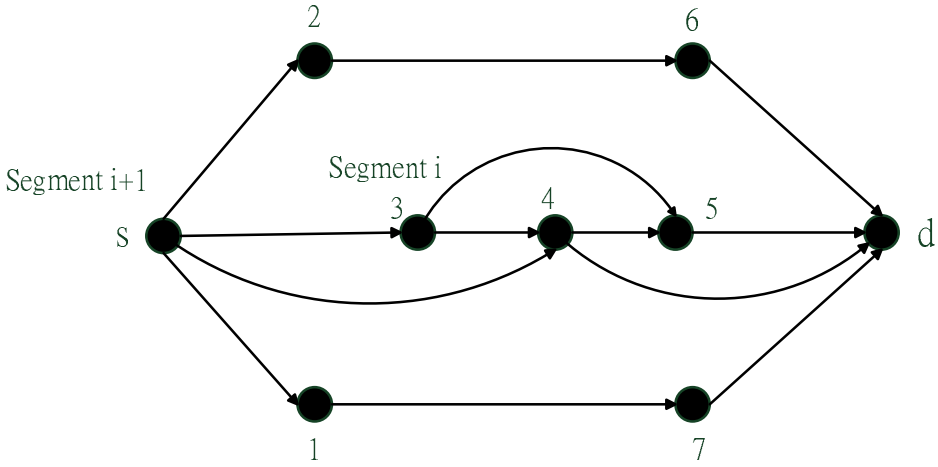


Figure 3.1: General topology for many segment flight in the network

we want to find unnecessary nodes for network topology. In some case, we set the network topology randomly even each node has independent chance to receive packet but redundant nodes will always transmit the same packet for another. The same packet is invalid for decoding, so we prune these nodes to improve throughput.

3.2 Protocol Design

In this section,we will introduce our algorithm detail. In follow ,any two nodes, i and j, let i>j denote that node j is closer to the destination than node i. In another word, j has a smaller ETX (expected transmission count) than i and j is the downstream node for i. The destination ETX is defined as zero.

To simplify the protocol design, a node transmits segments sequentially, PipelineOR ensures that the smaller ETX nodes receive a sufficient number of coded packets of segment i such that this node never needs to transmit segment i again. The critical challenge in PipelineOR is how does nodes determine that they have received a sufficient number of coded packets from its upstream nodes on a general random topology and how to determine the next segment can be store in a node buffer?

In particular, the downstream nodes of a node receive packets with different rates because they have different distances to the sender. Hence, they receiving different coded packets in a segment at the same times. For example, in Figure 3.2, node 1, 2, 3 receive different packets due to transmission loss. Thus, node s may wait so long for some of one downstream node to complete receiving all the coded packets in segment i especially in poor quality network.

Owing to the property of linear combination, we can only check the decoding coefficient carried in each packet to determine whether packets are independent or not. Inspired by this

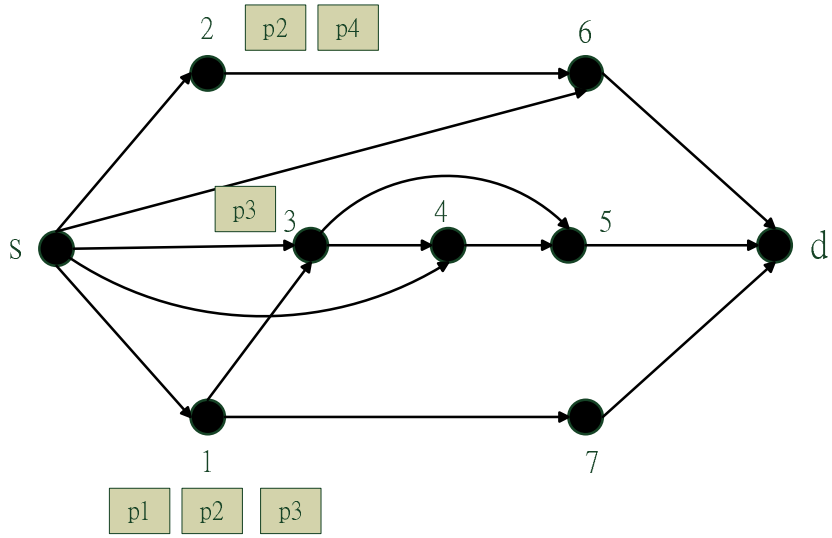


Figure 3.2: Independent chances for node receives packets

work, we propose when current node broadcasts data packets to another forwarder, each packet not only includes combining data and decoding vector but carries all packets corresponding coefficient stored in its buffer. As shown in Figure 3.3, node 2 has two packets in buffer and each packet has its decoding coefficient. When node broadcasts, each re-combining packet

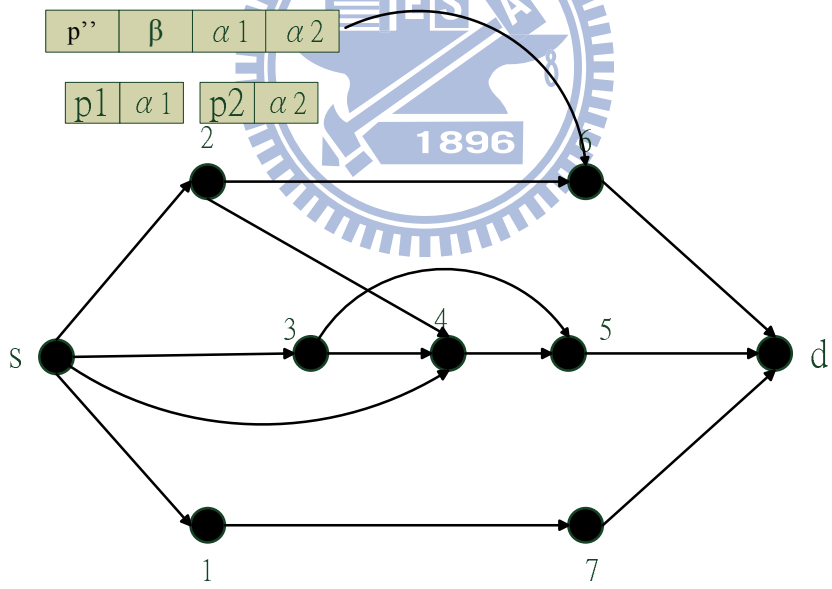


Figure 3.3: Packet carries extra coefficients

carries another two packets coefficients and extra message shown in Table 3.1. SrcAddress means which node broadcasts packets and which node that in ForwarderList will receive packet. The initial value of globalMaxETXHost and Former node are source because in the beginning only source has data to broadcast.

Node that receive the packets will calculate total independent coefficients by Gaussian Elimination and record which node transmits independent packets to this node in localMax-

$globalMaxETXHost$	$Former$
ForwarderList	SrcAddress
SegmentID	All packet coefficient
$Total - Segment$	k

Table 3.1: Extra broadcasting message

ETXHost. The localMaxETXHost will be change if the node that has larger ETX transmits independent packets to node. This value will more and more closer to destination when time passes by. If total independent coefficients are equal than k (partition size of one segment), meaning the forwarder have sufficient packets to decode. In this time, the node will transmit ACKs to source. In order to prevent that one forwarder node may become a bottleneck, so we set that source must receive half of downstream node replying ACKs at least. In this condition, source can move on to next segment. The replying message as shown in Table 3.2.

$localMaxETXHost$	$Former$
SrcAddress	SegmentID

Table 3.2: Reply message

For example, as shown in Figure 3.4, if k is 5 and node 6 has received three independent packets (i.e. dependent packets are invalid for decoding), the next time it receive re-combining packet from node 2 and knows node 2 has two packets in it buffer and these two packets are independent for node 6. Therefore the localMaxETXHost is node 2 and node 6 can send ACKs to source. The replying message will stop when it transmit to Former. We assume that node 3 and node 4 have be reply ACKs message. The globalMaxETXHost is be changed to node 2, because node 2 has the largest ETX value.

The Former node is stored in each node and the value changed when node receives globalMaxETX that has smaller ETX than its Former. As shown in Figure 3.5, if node 6 receives a globalMaxETX and its ETX smaller than zero, so Former in node 6 is changed to 2. The Former node chains each node from source to destination because the Former is more and more closer to destination. Finally, a node stops transmitting segment i if globalMaxETX has smaller ETX value for current node.

3.3 Improvements of PipelineOR

3.3.1 Buffer Management

In previous describes, we know the routing method for our algorithm. The concept is source can transmit new segment sequence if it receives ACKs. Unfortunately buffer in each node

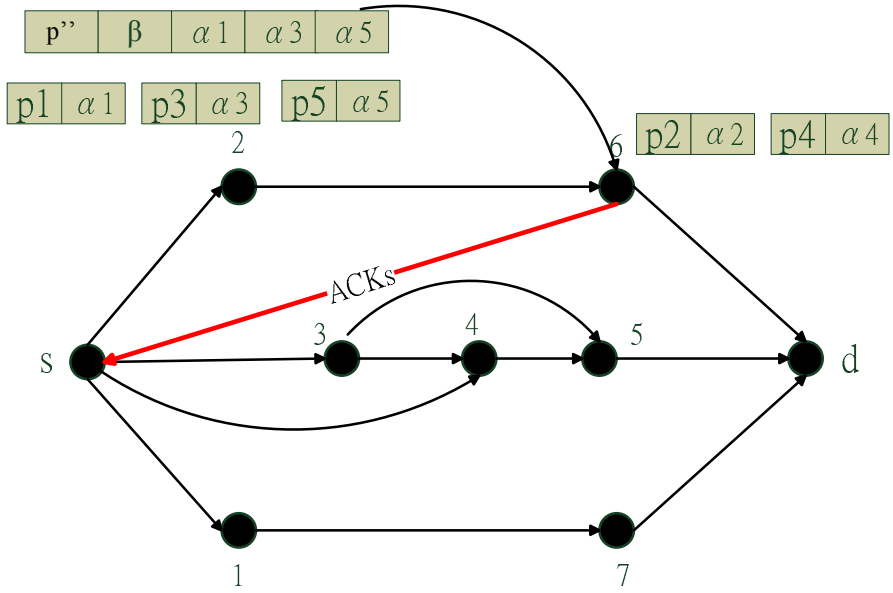


Figure 3.4: When node 6 receives sufficient packets to decode, it send ACKs to source

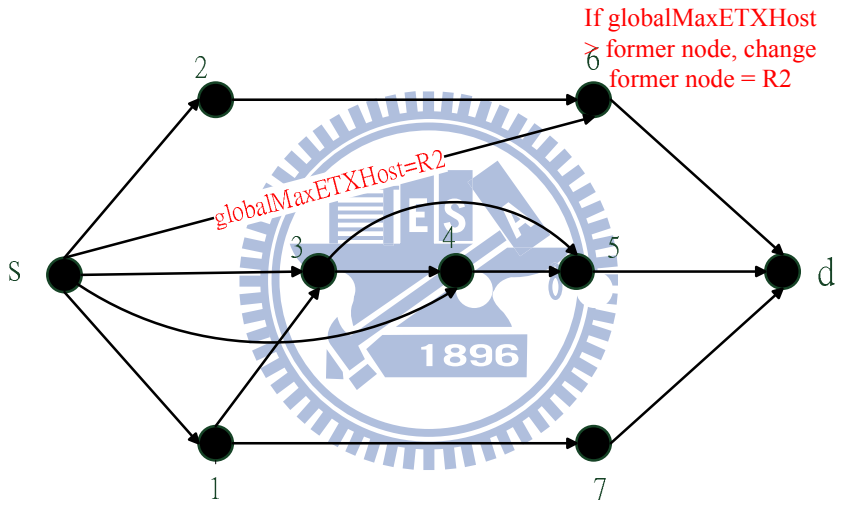


Figure 3.5: The condition for *Former* node is changed

is not infinite, so how to make the performance increase is important. If buffer size is 6, for example, meaning every node can store 6 segments at most. With CodeOR, segment stored in node dependent on source transmits which segments. If source transmits six segments from zero to five, then all of node just can save these segments. It wastes bandwidth because of some segments have stored in node that closer to destination but the far nodes cannot clear buffer and receive new segment packets.The time for total segments transmit over also be late for this condition. So we do not using this method for each node stores the same segment. Our method is describing following:

First, when current node broadcasts packet to another forwarder then the forwarder can receive packet if buffer in node is not full.In other condition, when node receives packet but no empty buffer to store data. We compare ETX of receiving node and globalMaxETXHost

to decide packet dropped or stored. If ETX of globalMaxETXHost smaller than node number means packet can be stored because the older segment has been transmitted closer to destination. Hence, we can clear buffer before globalMaxETXHost. For example, if s want to broadcast new segment to other forwarding node, as shown in Figure 3.6, the current globalMaxETXHost is node 5 so the node from 1 to 4 can clear buffer and store new segment packet. By this way can help more segments in flight in the network and improve performance.Especial in scale-up and poor link quality environment our algorithm is more efficient because we do not wait long delay time from destination send ACKs to source.

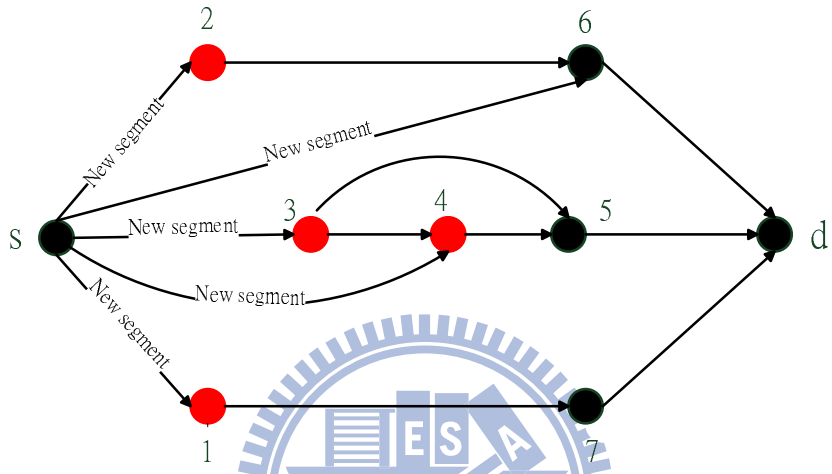


Figure 3.6: When new segment packet transmitted and buffer is full

The best buffer size is the same with sliding window size. Similar to TCP, the size of the sending window should approximately equals the delay-bandwidth. CodeOR [13] has present how to calculate to sending window size. In this paper, we focus on how to replace the segment in buffer.

3.3.2 Reduce redundant packets transmission

Collision and bandwidth contention is a influence for performance. On the other hand, if we decrease node transmission that may avoid collision happening. In random network topology, as shown in Figure 3.7, some node like node 1, 2, 3 may always receive the same packets but the same packets are invalid for decoding. In general solution, as shown in [4], we calculate how many packets does forwarder send dependent on transmission probability between nodes. For any two nodes, i and j, let i>j denote that node j is closer to the destination than node i, meaning j has a smaller ETX than i. Let p_{ij} denote the transmission probability in sending a packet from i to j. Let n_i be the expected number of transmissions that forwarder i must make to route one packet from the source to the destination. In this section, we focus on

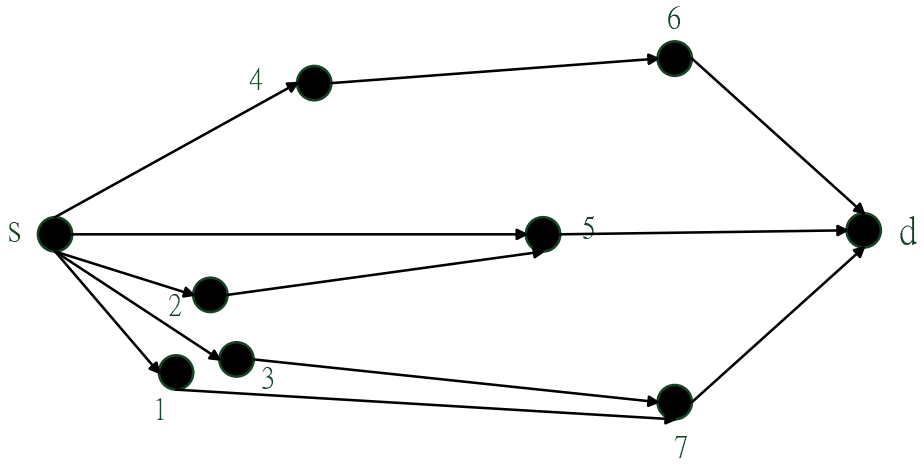


Figure 3.7: Random nework topology

delivering one packet from source to destination. The number of packets that a forwarder j must forward to transmit one packet from source to destination is calculating in following. The expected number of packets that j receives from nodes with higher ETX is $\sum_{i>j} n_i p_{ij}$. For each packet j receives, j should forward it only if no node with lower ETX metric hears the packet. This happens with probability $\prod_{k<j}(1-p_{ik})$. Thus, in expectation, the number of packets that j must forward, denoted by Lj, is:

$L_j = \sum_{i>j} n_i p_{ij} \prod_{k<j}(1-p_{ik})$

We assume $L_s = 1$ because the source generates the packet. Now, consider the expected number of transmissions a node j must make. J should transmit each packet until a node with lower ETX has received it. Thus, the number of transmissions that j makes for each packet it forwards is a geometric random variable with success probability (1-$\prod_{k<j}(1-p_{jk})$). This is the probability that some node with ETX lower than j receives the packet. Thus, the expected number of transmissions that j must make is:

$n_j = L_j/(1-\prod_{k<j}(1-p_{jk}))$

We first prune the node i if its n_i smaller than a threshold. In addition, we also set a counting number CNT, if packets from upstream nodes are dependent then CNT+1. The bandwidth is wasting, when some nodes always transmit the same packets to receiver. Therefore, we define a threshold h and $time$ if CNT over h or over $time$ the node does not receive new packet. This node will reply to upstream node and let it to stop transmit packet. More importantly, We can know how many remainder packets that downstream node may receive, As shown in Figure 3.8, node 7 has two packets and node 1 has three packets and node 7 included in node 2. We could presume node 1 just re-encoding one packet that independent for node 7. In this case if node 7 has receive this independent packet and through specific time, we stop node 1 avoid it transmits redundant data to it downstream node 7. Each node can know how many

packet that it upstream nodes can receive. Therefore, current node can decide whether stop upstream node.

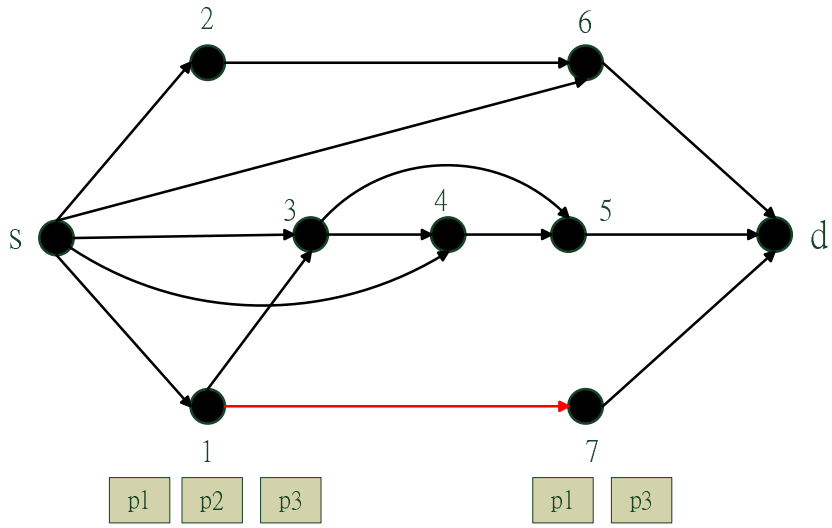


Figure 3.8: How Many Remainder Packet between a pair nodes

3.3.3 Faster Packet Combining

In traditional operations in finite field, for example in GF(7), shows in Table 3.3: With linear

Operation	Result
Addition	3 + 5 = 1
Subtraction	3 - 5 = 5
Multiplication	3 * 5 = 1
Division	3 / 5 = 2

Table 3.3: Operations in finite field

combination we present a packet as a sequence of symbol (i.e. one byte) in a finite field. Therefore the finite field is in GF(2^8) and primitive polynomial is f(x) =$x^8 + x^4 + x^3 + x + 1$. Let be the root of f(x) then the set 1, α, α^2,. . ., α^7 can be used as a basis for GF(2^8) so any byte can be presented as Table 3.4 . In GF(2^8), addition and subtraction operations are using

ASCII	Binary	Binary
0	00000000	0
1	00000001	1
2	00000010	α
3	00000011	α+1
...	...	...
255	11111111	$\alpha^7+\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha+1$

Table 3.4: Byte presentation in GF(2^8)

bit-wise xor, for example, $00000010 \oplus 00010110 = 00010100$.Multiplication is multiplying two operand and mod primitive polynomial. Division detail sees in [5].Actually, in multimedia applications the computing of previous is slowly so using lookup table [6] is important for this. For example, we build a Table 3.5 for GF(7) and the multiplication is changed to $3 * 4 = 5^5 * 5^2 = 5^7 = 5^1 = 5$.The Division is change to $6/3 = 5^3/5^5 = 5^{-2} = 5^4 = 2$.

5^0	1
5^1	5
5^2	4
5^3	6
5^4	2
5^5	3
5^6	1

Table 3.5: Change table for GF(7)

More advance, we can using two table, ltable = [0,3,1,2](array start from 1) and atable = [1,3,4,2](array start from 0) to calculate result show in Figure 3.9. For example, 3*4 =

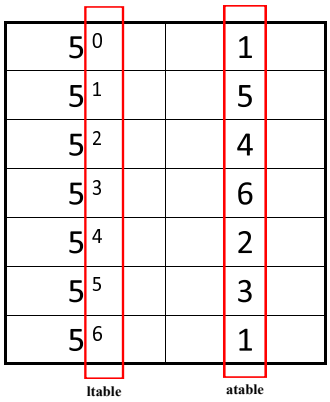


Figure 3.9: Lookup Table in GF(7)

ltable[3]+ltable[4]=3 then takes 3 to atable[3]=2. In GF(2^8) the ltable and atable are shown in Figure 3.10.

```
unsigned char ltable[256] = {
    0x00,0xff,0xc8,0x08,0x91,0x10,0xd0,0x36,0x5a,0x3e,0xd8,0x43,0x99,0x77,0xfe,0x18,
    0x23,0x20,0x07,0x70,0xa1,0x6c,0x0c,0x7f,0x62,0x8b,0x40,0x46,0xc7,0x4b,0xe0,0x0e,
    0xeb,0x16,0xe8,0xad,0xcf,0xcd,0x39,0x53,0x6a,0x27,0x35,0x93,0xd4,0x4e,0x48,0xc3,
    0x2b,0x79,0x54,0x28,0x09,0x78,0x0f,0x21,0x90,0x87,0x14,0x2a,0xa9,0x9c,0xd6,0x74,
    0xb4,0x7c,0xde,0xed,0xb1,0x86,0x76,0xa4,0x98,0xe2,0x96,0x8f,0x02,0x32,0x1c,0xc1,
    0x33,0xee,0xef,0x81,0xfd,0x30,0x5c,0x13,0x9d,0x29,0x17,0xc4,0x11,0x44,0x8c,0x80,
    0xf3,0x73,0x42,0x1e,0x1d,0xb5,0xf0,0x12,0xd1,0x5b,0x41,0xa2,0xd7,0x2c,0xe9,0xd5,
    0x59,0xcb,0x50,0xa8,0xdc,0xfc,0xf2,0x56,0x72,0xa6,0x65,0x2f,0x9f,0x9b,0x3d,0xba,
    0x7d,0xc2,0x45,0x82,0xa7,0x57,0xb6,0xa3,0x7a,0x75,0x4f,0xae,0x3f,0x37,0x6d,0x47,
    0x61,0xbe,0xab,0xd3,0x5f,0xb0,0x58,0xaf,0xca,0x5e,0xfa,0x85,0xe4,0x4d,0x8a,0x05,
    0xfb,0x60,0xb7,0x7d,0xb8,0x26,0x4a,0x67,0xc6,0x1a,0xf8,0x69,0x25,0xb3,0xdb,0xbd,
    0x66,0xdd,0xf1,0xd2,0xdf,0x03,0x8d,0x34,0xd9,0x92,0x0d,0x63,0x55,0xaa,0x49,0xec,
    0xbc,0x95,0x3c,0x84,0x0b,0xf5,0xe6,0xe7,0xe5,0xac,0x7e,0x6e,0xb9,0xf9,0xda,0x8e,
    0x9a,0xc9,0x24,0xe1,0x0a,0x15,0x6b,0x3a,0xa0,0x51,0xf4,0xea,0xb2,0x97,0x9e,0x5d,
    0x22,0x88,0x94,0xce,0x19,0x01,0x71,0x4c,0xa5,0xe3,0xc5,0x31,0xbb,0xcc,0x1f,0x2d,
    0x3b,0x52,0x6f,0xf6,0x2e,0x89,0xf7,0xc0,0x68,0x1b,0x64,0x04,0x06,0xbf,0x83,0x38
};

unsigned char atable[256] = {
    0x01,0xe5,0x4c,0xb5,0xfb,0x9f,0xfc,0x12,0x03,0x34,0xd4,0xc4,0x16,0xba,0x1f,0x36,
    0x05,0x5c,0x67,0x57,0x3a,0xd5,0x21,0x5a,0x0f,0xe4,0xa9,0xf9,0x4e,0x64,0x63,0xee,
    0x11,0x37,0xe0,0x10,0xd2,0xac,0xa5,0x29,0x33,0x59,0x3b,0x30,0x6d,0xef,0xf4,0x7b,
    0x55,0xeb,0x4d,0x50,0xb7,0x2a,0x07,0x8d,0xff,0x26,0xd7,0xf0,0xc2,0x7e,0x09,0x8c,
    0x1a,0x6a,0x62,0x0b,0x5d,0x82,0x1b,0x8f,0x2e,0xbe,0xa6,0x1d,0xe7,0x9d,0x2d,0x8a,
    0x72,0xd9,0xf1,0x27,0x32,0xbc,0x77,0x85,0x96,0x70,0x08,0x69,0x56,0xdf,0x99,0x94,
    0xa1,0x90,0x18,0xbb,0xfa,0x7a,0xb0,0xa7,0xf8,0xab,0x28,0xd6,0x15,0x8e,0xcb,0xf2,
    0x13,0xe6,0x78,0x61,0x3f,0x89,0x46,0x0d,0x35,0x31,0x88,0xa3,0x41,0x80,0xca,0x17,
    0x5f,0x53,0x83,0xfe,0xc3,0x9b,0x45,0x39,0xe1,0xf5,0x9e,0x19,0x5e,0xb6,0xcf,0x4b,
    0x38,0x04,0xb9,0x2b,0xe2,0xc1,0x4a,0xdd,0x48,0x0c,0xd0,0x7d,0x3d,0x58,0xde,0x7c,
    0xd8,0x14,0x6b,0x87,0x47,0xe8,0x79,0x84,0x73,0x3c,0xbd,0x92,0xc9,0x23,0x8b,0x97,
    0x95,0x44,0xdc,0xad,0x40,0x65,0x86,0xa2,0xa4,0xcc,0x7f,0xec,0xc0,0xaf,0x91,0xfd,
    0xf7,0x4f,0x81,0x2f,0x5b,0xea,0xa8,0x1c,0x02,0xd1,0x98,0x71,0xed,0x25,0xe3,0x24,
    0x06,0x68,0xb3,0x93,0x2c,0x6f,0x3e,0x6c,0x0a,0xb8,0xce,0xae,0x74,0xb1,0x42,0xb4,
    0x1e,0xd3,0x49,0xe9,0x9c,0xc8,0xc6,0xc7,0x22,0x6e,0xdb,0x20,0xbf,0x43,0x51,0x52,
    0x66,0xb2,0x76,0x60,0xda,0xc5,0xf3,0xf6,0xaa,0xcd,0x9a,0xa0,0x75,0x54,0x0e,0x01
};
```

Figure 3.10: The table for GF(2^8)

The detailed steps of PipelineOR are described in the following Algorithm. Algorithm 1 illustrates all initial value. In Algorithm 2 and Algorithm 3, describing the action for node broadcast and reply message. The condition for each node receives packet describing in Algorithm 4 and Algorithm 5.

Algorithm 1: Initial all value using in our algorithm

1: Initialize $localMaxETXHost$, $globalMaxETXHost$ and $Former$ are source.
2: Initialize two array named dataArray[ ][ ]=0 and checkArray[ ][ ]=0.
3: Initialize $Total - Segment$, this value depend on packet size,k and total data size.
4: Clear buffer in each node.
5: Initialize Finish-Flag=0.

Algorithm 2: Node in broadcasts case

1: **if** Node occupies channel, and has data in buffer **then**
2: **if** Current number is source **then**
3: Encoding the data using fast packet combining method and set table 3.1 into transmitting packet except extra packet coefficient.
4: **else**
5: Encoding the data using fast packet encoding method and set table 3.1 into transmitting packet.

Algorithm 3: Node in replies case

1: **if** CheckArray[ ][ ] is equal to k and
the ETX of node is smaller than $globalMaxETXHost$ **then**
2: Replying ACKs(Reply message in table 3.2) to $Former$.
3: **if** Node receives new message **then**
4: Replying ACKs to $Former$.
5: **if** All Segment has decoded in destination **then**
6: Replying $Finish - Flag$.

Algorithm 4: Node in receives broadcasting packet case

1: **if** Node receives packet and node in forwarding list **then**
2: **if** Buffer in node is not full **then**
3: Using Gaussian Elimination checks the packet is independent or not.
4: **if** Packet is independent **then**
5: Storing this packet in dataArray[ ][ ].
6:
7: Check extra packet coefficient are independent or not.
8: **if** Extra packet is independent **then**
9: Storing packet in checkArray[ ][ ].
10: **if** CheckArray[ ][ ] is equal to k and the ETX of node is smaller than $globalMaxETXHost$ **then**
11: Replying ACKs to $Former$.
12: **if** The $globalMaxETXHost$ is smaller than $StoreForMaxETXHost$ **then**
13: Change $StoreForMaxETXHost$ value to $globalMaxETXHost$
14: **if** The $globalMaxETXHost$ is smaller than $Former$ in each node and current ETX of node is smaller than $globalMaxETXHost$ **then**
15: Change $Former$ value to $globalMaxETXHost$
16: **else**
17: **if** The ETX of node larger than $globalMaxETXHost$ **then**
18: New segment can clear older segment and be stored in buffer.
19: Repeat step 3-15.

Algorithm 5: Node in receives replying packet case

1: **if** Node receives $Finish - Flag$ **then**
2: Replying $Finish - Flag$ to source by shortest path.
3: **if** $localMaxETXHost$ is less than $StoreForMaxETXHost$ **then**
4: Change $StoreForMaxETXHost$ value to $localMaxETXHost$.
5: **if** Source receives a message for Segment i has sufficient packet to decode in forwarding nodes **then**
6: Broadcasting new segment.

Chapter 4

Performance evaluations

4.1 Simulation Model

In this chapter, we use the Omnet++ network simulation [15] to evaluate the performance of our scheme. Each experiment are performed over 802.11 in Omnet++. We develop a customized discrete event simulator, which implements randomized network coding, wireless opportunistic routing protocols. In our simulation, two nodes are regarded as neighbors only if the link quality between them is sufficient to achieve a transmission success probability higher than 0.1.

We conduct two experiments on random topology and grid topology shown in Figure 4.1 and 4.2. In our experiments, we focus on the time for destination has sufficient to decode all segments. For simplify to observe, node only transmits one packet for each time. Each topology with 100 nodes that are deployed in a square of size 2000 * 2000. The source nodes for these two topologies are host[0], destination nodes are host[16] and host[99], respectively. We set the data packet size to 2048 bytes and ACKs packets to 16 bytes in most experiments. In addition, we set the number of segment is 6,12,18,24,30 for the purpose of illustration. Buffer in each node can store 6 segment at most and each segment divided into 16 packets. We assume that a node occupies the wireless channel in a local neighborhood during the transmission time of a packet. However, the wireless channel is released and can be used by other nearby nodes during the time for this packet is over.

As shown in Table 4.1, we define the parameter using in our experiment.

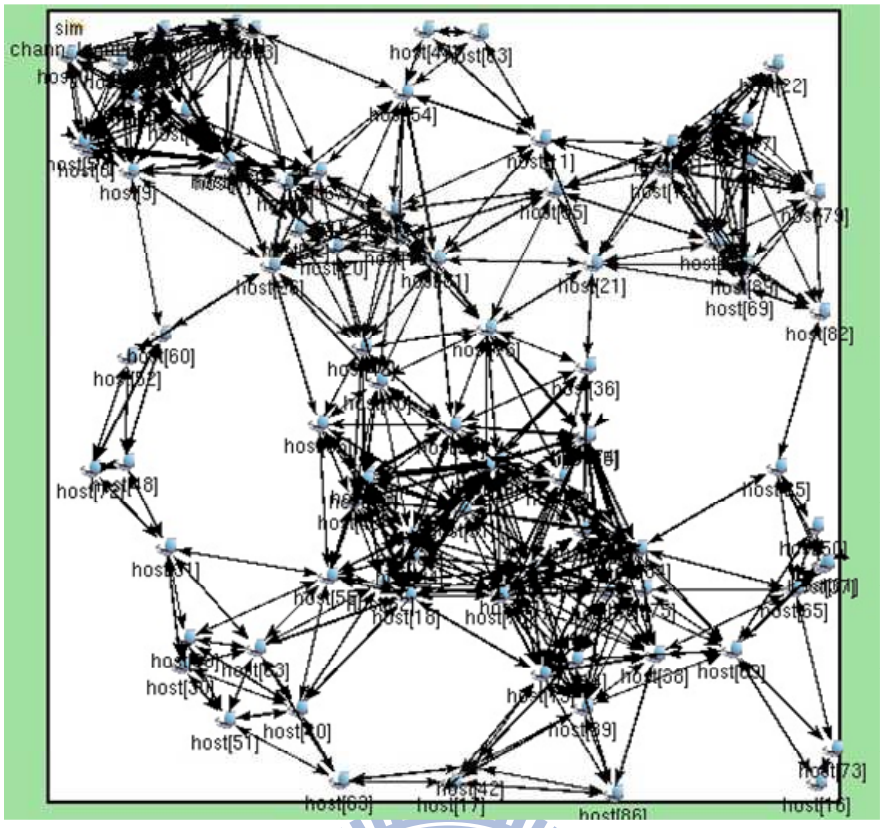


Figure 4.1: Random topology for evaluation

parameter	value
k	16
buffer size	6
sliding window size	6

Table 4.1: Notation

4.2 Experimentation

A. Transmission Time of Different Segment Number

In this section, we compare the transmission time of PipelineOR, CodeOR and MORE over a different segments. As shown in Figure 4.3, x-coordinate means the total segments that a source transmitting to destination and y-coordinate means the time that destination receives all segments. In this experimentation, the loss rate for each is 15%. We observe that PipelineOR achieves significantly lower transmission time than MORE and CodeOR. The PepelineOR's transmission time in average is 15% higher than CodeOR and 30% higher than MORE. In addition, when segment is small, the transmission time is equal or lower than MORE. The reason is, the overhead for nodes must reply control message to upstream node may neutralize the advantage. In grid topology simulation that node can decode faster due

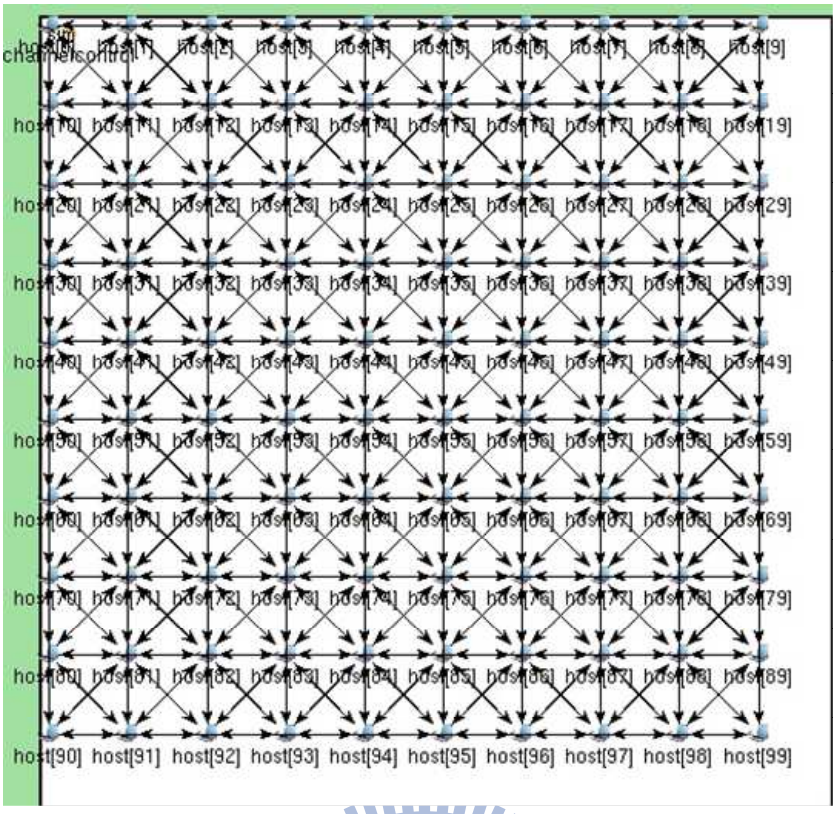


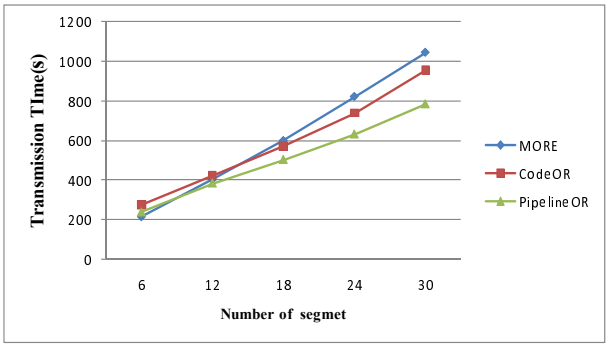
Figure 4.2: Grid topology for evaluation

to the transmission probability of each pair node in grid topology are better than random topology. Therefore, each node can receive innovate packet faster.

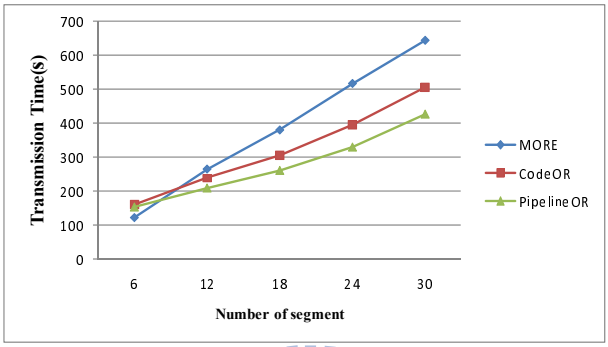
B. Time of Different Transmission Rate

In this section, we let each node has lower transmission rate with 30 segments. The x-coordinate means that miss rate for each node receive the packets and y-coordinate means the time that destination receives 30 segments. In Figure 4.4, PipelineOR also has better throughput than others. The infection for PipelineOR is lower than other two protocol when miss rate is increasing due to in PipelineOR each node can cooperation. According our algorithm, we can let the node has lower ETX value help source transmit segment, so the time for source move on to next segment is faster than others. In MORE, it must ensure that destination has decoded segment then source node could move on to the next segment. The time for destination receive all segment is dependent on transmission rate entirely. The same problem is happen in CodeOR. CodeOR calculates a probability for node stop transmit some segment. This probability also depends on transmission rate entirely.

C. The Total Packet Transmission

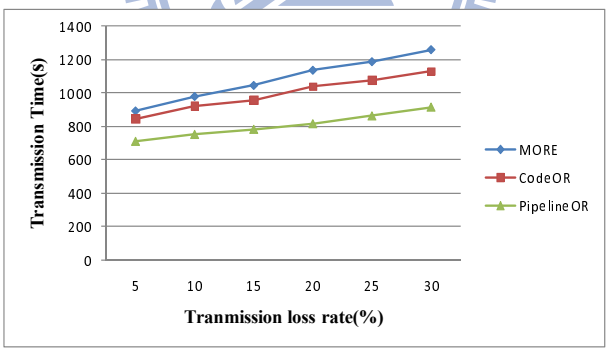


(a) Random topology transmission time

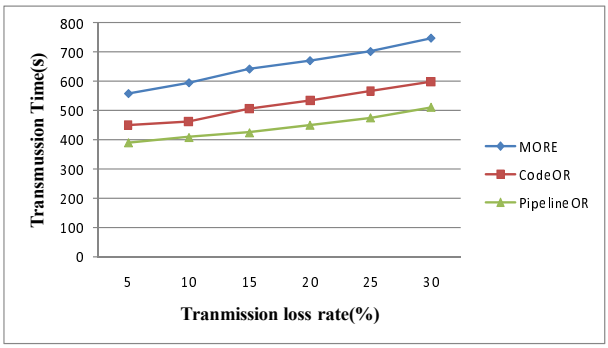


(b) Grid topology transmission time

Figure 4.3: Transmission time

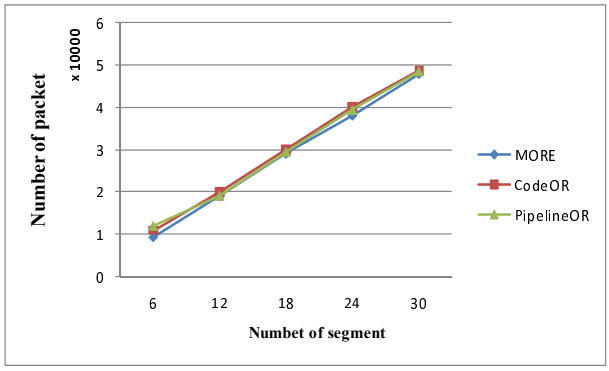


(a) Time of different loss rate in random topology

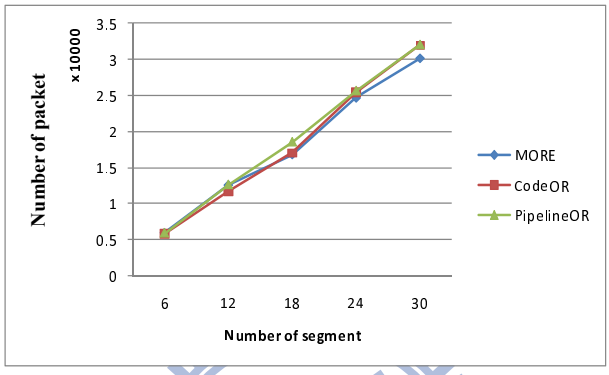


(b) Time of different loss rate in grid topology

Figure 4.4: Time of different loss rate



(a) Number of packet transmissions in random topology



(b) Number of packet transmissions in grid topology

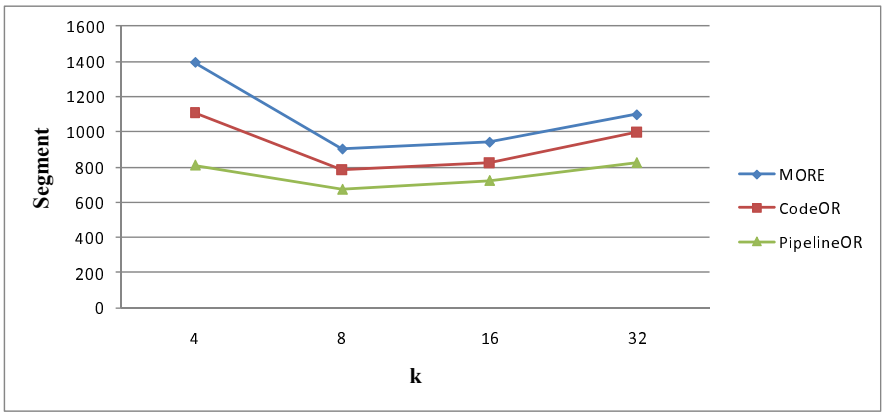
Figure 4.5: Number of packet

As shown in Figure 4.5, compares the number of data transmissions by these three protocols. We observe that PipelineOR reduces slightly data transmissions because we stop some node to transmit redundant packets even n_i is not zero. Why the total data transmissions are so close but the time for these three protocols are different? The reason is all of these three protocols are calculate how many packets does each node must transmit. Therefore, the total data transmissions have not large gap in simulation. The time for different protocols transmit segment is different due to the time for source move on the next segment.

D. The throughput of different k

In MORE [4], it has explored the performance for various batch sizes k. In this section, we show the variation for different value of k in these three protocols. As shown in Figure 4.6, we set the data size is 768K and loss rate is 15% in random topology. We can observe, when k is 8 that has best throughput. When k is too large, the encoding and decoding time will increase. When k is small, the segment size will become too large to reduce the throughput.

This result comforms with MORE.



The throughput of different k

Figure 4.6: The throughput of different k

Chapter 5

Conclusion

In this paper, we propose a approach for improve the throughput of prior opportunistic routing protocols based on network coding that degrades in a large-scale network. PipelineOR, that using cooperation for each node to ensure forwarder has sufficient to decode. Hence, source can move on new segment soon, because another forwarder can help the source to transmit older segment. For reduce bandwidth contention and improve performance, we also prune redundant nodes that have low contribution. We show that PipelineOR significantly outperforms existing approaches in network throughput while data is large. Unlike existing protocols, PipelineOR is able to achieve higher throughput by different segment can transmit continuous if buffer is not fully.

Bibliography

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 2000.

[2] S. Biswas and R. Morris. Opportunistic Routing in Multi-hop Wireless Networks. *Hotnets-II*, 2003.

[3] S. Biswas and R. Morris. ExOR: Opportunistic Multi-Hop Routing for Wireless Networks. In *Proceedings of ACM SIGCOMM*, 2005.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading Structure for Randomness in Wireless Opportunistic Routing. In *Proceedings of ACM SIGCOMM*, 2007.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. 2001.

[6] http://www.samiam.org/galois.html.

[7] M. Halloush and H. Radha. Network coding with multi-generation mixing. In *Proceedings of IEEE Information Sciences and Systems. 42nd Annual Conference*, 2008.

[8] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The Benefits of Coding over Routing in a Randomized Setting. In *Proceedings of IEEE International Symposium on Information Theory*, 2003.

[9] Z. K, L. W, and Z. H. On end-to-end throughput of opportunistic routing in multirate and multi hop wireless networks. In *Proceedings of IEEE INFOCOM*, 2008.

[10] S. Katti and D. Katabi. MIXIT: The Network Meets the Wireless Channel. *ACM Workshop on Hot Topics in Networks*, 2007.

[11] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in The Air: Practical Wireless Network Coding. In *Proceedings of ACM SIGCOMM*, 2006.

[12] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear Network Coding. *IEEE Transactions on Information Theory*, 2003.

[13] Y. Lin, B. Li, and B. Liang. CodeOR: Opportunistic Routing in Wireless Mesh Networks with Segmented Network Coding. In *Proceedings of IEEE ICNP*, 2008.

[14] D. S. Lun, M. Medard, and R. Koetter. Network Coding for Efficient Wireless Unicast. In *Proceedings of International Zurich Seminar on Communications (IZS)*, 2006.

[15] http://www.omnetpp.org/.

[16] C. E. Perkins and P. Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOMM*, 1993.

[17] C. E. Perkins and E. M. Royer. Ad hoc On-Demand Distance Vector Routing. *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[18] B. Radunovic, C. Gkantsidis, P. Key, and P. Rodriguez. An Optimization Framework for Opportunistic Multipath Routing in Wireless Mesh Networks. In *Proceedings of IEEE INFOCOM*, 2008.

[19] E. Rozner, J. Seshadri, Y. Mehta, and L. Qiu. Simple Opportunistic Routing Protocol for Wireless Mesh Networks. *IEEE Workshop on Wireless Mesh Networks*, 2006.

[20] Y. Yan, B. Zhang, H. T. Mouftah, , and J. Ma. Practical Coding-Aware Mechanism for Opportunistic Routing in Wireless Mesh Networks. In *Proceedings of IEEE ICC*, 2008.

[21] X. Zhang and B. Li. Dice: a Game Theoretic Framework for Wireless Multipath Network Coding. In *Proceedings of ACM MobiHoc*, 2008.

[22] X. Zhang and B. Li. Optimized Multipath Network Coding in Lossy Wireless Networks. In *Proceedings of IEEE ICDCS*, 2008.

[23] Z. Zhong, J. Wang, and S. Nelakuditi. Opportunistic Any-Path For- warding in Multi-Hop Wireless Mesh Networks. *USC CSE Technical Report*, 2006.

[24] A. Zubow, M. Kurth, and J. Redlich. Multi-Channel Opportunistic Routing in Multi-Hop Wireless Networks. In *Proceedings of IEEE European Wireless Conference*, 2007.