Topics in Routing and Network Coding for Wireless Networks

Maulik Desai

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2014

ABSTRACT

Topics in Routing and Network Coding for Wireless Networks

Maulik Desai

This dissertation presents topics in routing and network coding for wireless networks. We present a multipurpose multipath routing mechanism. We propose an efficient packet encoding algorithm that can easily integrate a routing scheme with network coding. We also discuss max-min fair rate allocation and scheduling algorithms for the flows in a wireless network that utilizes coding.

We propose Polar Coordinate Routing (PCR) to create multiple paths between a source and a destination in wireless networks. Our scheme creates paths that are circular segments of different radii connecting source−destination pairs. We propose a non−euclidean distance metric that allows messages to travel along these paths. Using PCR it is possible to maintain a known separation among the paths, which reduces the interference between the nodes belonging to two separate routes. Our extensive simulations show that while PCR achieves a known separation between the routes, it does so with a small increase in overall hop count. Moreover, we demonstrate that the variances of average separation and hop count are lower for the paths created using PCR compared to the existing schemes, indicating a more reliable system. Existing multipath routing schemes in wireless networks do not perform well in the areas with obstacles or low node density. To overcome adverse areas in a network, we integrate PCR with simple robotic routing, which lets a message circumnavigate an obstacle and follow the multipath trajectory to the destination as soon as the obstacle is passed.

Next we propose an efficient packet encoding algorithm to integrate a routing scheme with network coding. Note that this packet encoding algorithm is not dependent on PCR.

In fact it can be coupled with any routing scheme in order to leverage the benefits offered by both an advanced routing scheme and an enhanced packet encoding algorithm. Our algorithm, based on bipartite graphs, lets a node exhaustively search its queue to identify the maximum set of packets that can be combined in a single transmission. We extend this algorithm to consider multiple next hop neighbors for a packet while searching for an optimal packet combination, which improves the likelihood of combining more packets in a single transmission.

Finally, we propose an algorithm to assign max-min fair rates to the flows in a wireless network that utilizes coding. We demonstrate that when a network uses coding, a direct application of conventional progressive filling algorithm to achieve max-min fairness may yield incorrect or suboptimal results. To emulate progressive filling correctly for a wireless networks with coding, we couple a conflict graph based framework with a linear program. Our model helps us directly select a bottleneck flow at each iteration of the algorithm, eliminating the need of gradually increasing the rates of the flows until a bottleneck is found. We demonstrate the caveats in selecting the bottleneck flows and setting up transmission scheduling constraints in order to avoid suboptimal results. We first propose a centralized fair rate allocation algorithm assuming the global knowledge of the network. We also present a novel yet simple distributed algorithm that achieves the same results as the centralized algorithm. We also present centralized as well as distributed scheduling algorithms that help flows achieve their fair rates. We run our rate allocation algorithm on various topologies. We use various fairness metrics to show that our rate allocation algorithm outperforms existing algorithms (based on network utility maximization) in terms of fairness.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest gratitude and love to Ba, Dada and Masi for their unconditional love and care. The things I have learned from them will take me a long way. I would also like to thank my parents and brother for their love and support throughout my academic career.

Special thanks to my aunt Hema and uncle Mukesh for giving me an opportunity to come to the United States and explore new academic horizons. I would also like to thank my cousin Sejal and her husband Santosh for their help during my days as a community college student.

I am fortunate to have more than my fair share of wonderful friends. I would like to thank my friend Terence for having meaningful conversations about life, universe and everything else with me almost everyday since I have known him. Thanks to Yuriy for being the receiving end of my midnight calls every time I got a bad grade or had bad results in my research. Thanks to Ken for making himself available every time I asked him to hang out and having lighthearted conversations with me to keep my spirits up. Thanks to Qing as well, without her help I would have neither taken GRE nor written an admissions essay. Thanks to Satya for making herself available to talk during frustrating days of which there were many. Thanks to Prateek, Ravi, Richard and Kevin for hanging out with me since the days of our undergraduate studies.

I would also like to thank my labmates Patcharinee, Yitian and Robert. Also, many thanks to Kyung for his help throughout my graduate studies. It is hard to imagine a better labmate than him.

I am grateful to my thesis adviser Prof. Nick Maxemchuk for his valuable guidance,

without his insightful advice this thesis would not have been possible.

I am also thankful to my committee members Prof. Gil Zussman, Prof. Vishal Misra, Prof. Augustin Chaintreau and Dr. Thierry Klein for taking time out from their busy schedule and attending my thesis defense.

I am thankful to Prof. Bhaskar Krishnamachari and Dr. Thyaga Nandagopal. The motivation and inspiration they provided served me throughout my graduate studies.

To,
Ba, Dada and Masi.

Chapter 1

Introduction

This dissertation presents topics related to routing and network coding for wireless networks. The mechanisms discussed in this dissertation work on the wireless networks with stationary nodes. Examples of such networks include wireless sensor networks and wireless mesh networks. With some modifications, the work presented here may be adapted to the mobile ad-hoc networks where nodes in the network are not stationary. The applications of wireless networks vary from data aggregation/dissemination to natural activity monitoring. However, our main focus is concerned with the applications related to data delivery. Instead of data dissipation from a source or data aggregation towards a destination, we work with the routing schemes where the data is always travelling between a source node and a destination node.

1.1 Introduction to Multipath Routing

There have been several routing schemes that achieve this task with different objectives in mind. For example a routing scheme may choose to forward packets such that they make the maximum possible progress towards their destinations. On the other hand packets may be forwarded through the minimum delay paths, or through the paths that reduce the total number of expected transmissions etc. Unfortunately, there have been few efforts

focused towards multipath routing. As the name indicates in a multipath routing scheme, the packets are forwarded towards the destinations on multiple different paths. Depending on the nature of the application there may or may not be redundant packets travelling on separate paths. Multipath routing can offer several advantages.

- It helps spread out the data packets towards the underutilized parts of the network, hence it helps alleviate congestion.
- In a network with fragile connectivity, by sending the same packets on multiple paths, one can improve the probability of data delivery. Even if the route on one of the paths fails, the data can be still delivered to the destination on other paths.
- Multipath routing improves security of data delivery as well. By sending different data packets from a flow on different paths, one can prevent an adversary from intercepting and decoding the entire set of data, by intercepting one of the paths.

Two of the noteworthy efforts exploring multipath routing are [Niculescu and Nath, 2003] and [Popa *et al.*, 2006]. In [Niculescu and Nath, 2003] a trajectory is defined using parametric equations that connects the source and the destination. Packets are forwarded through the nodes that are closer to this trajectory. If we define multiple trajectories to connect the source and the destination, this scheme may be applied as a multipath routing scheme. However, this mechanism hasn't evaluated multipath nature of the routing scheme. [Popa *et al.*, 2006] does present a multipath mechanism. The goal here is to send packets at different angles from the line connecting the source and the destination. By modifying these angles at every hop the packets are forwarded towards the destination on different trajectories. However this scheme fails to maintain high enough path separation and the hop count towards the destination may fluctuate significantly. To overcome these issues we present Polar Coordinate Routing [Desai and Maxemchuk, 2010]. Chapter 2 discusses the mechanism of this routing scheme in more detail.

1.2 Introduction to Network Coding

[Katti *et al.*, 2006] presented COPE, an implementation of a simple network coding scheme on top of a packet forwarding mechanism. They showed that a simple coding mechanism indeed helps improve throughput of a network. Since then the research interest in network coding has been reinvigorated. We present an efficient and exhaustive packet encoding algorithm to help integrate a routing scheme with network coding. This algorithm is not dependent on Polar Coordinate Routing. In fact, it can be easily integrated with any routing scheme irrespective of its objective. This helps us reap the benefits offered by both an advanced routing scheme and an enhanced packet encoding algorithm. Moreover in this thesis we also look at resource allocation problems in a wireless network with coding. Namely, we present a max-min fair rate allocation as well as a scheduling algorithms for the flows in a network with coding.

Until the implementation of network coding demonstrated in [Katti *et al.*, 2006], the work in this area has been theoretical. [Ahlswede *et al.*, 2000] demonstrated that combining packets in a single transmission can lead to improving capacity of the network. [yen Robert Li *et al.*, 2003] showed that in a multicast network transmissions of a linear combination of packets can achieve the max-flow bound on the capacity. [Koetter *et al.*, 2003] presented polynomial time algorithms for encoding and decoding packets for the linear codes. [Ho *et al.*, 2003] presented random codes and showed that probability of failure to decode a packet decreases exponentially as the length of a codeword increases for random codes. [Sundararajan *et al.*, 2008] demonstrated how a random code can be integrated with TCP.

[Li and Li, 2004] investigated how network coding can lead to throughput improvement in unicast sessions. [Wu *et al.*, 2005] presents a simple coding scheme where the packets from two unicast flows travelling in opposite directions are combined using bitwise exclusive or. The combination of these two packets is forwarded in a single transmission. Since this idea forwards more packets in fewer transmissions, it helps improve the network throughput. In COPE [Katti *et al.*, 2006] this basic idea is implemented. This basic coding scheme takes

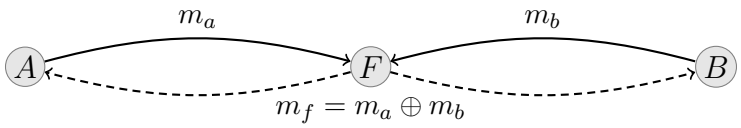


Figure 1.1: Example of the coding scheme

advantage of the broadcast nature of a wireless channel. Consider the scenario presented in figure 1.1. In this scenario we have two nodes A and B exchanging messages with each other. m_a travels from A to B and m_b travels from B to A. Since A and B are out of each other's transmission range these two messages have to be forwarded through a node F that lies in between them. Observe that in order to avoid a collision only one packet can be transmitted at a time in this network. Therefore in a conventional forwarding scheme it takes a total of four transmissions for A and B to exchange m_a and m_b with each other. However, say instead of forwarding m_a and m_b individually, node F combines these two messages and transmits the combination $m_f = m_a \oplus m_b$, where $\oplus$ stands for bitwise exclusive or. Since it is a wireless medium, both A and B will receive this packet combination. Since A already has the knowledge of m_a it will retrieve m_b as $m_b = m_f \oplus m_a$. Similarly, B can retrieve m_a as $m_a = m_f \oplus m_b$. Therefore, both A and B receive their required messages, and the whole exchange takes only three transmissions. Since this basic scheme reduces the number of transmissions from four to three, it improves the throughput of the network by 33%. In this primitive example we just combined two messages. However, if a node has k packets in its queue that are going to k different neighbors, ideally all these k packets can be combined in a single transmission, if each of these neighbors has the knowledge of $k-1$ packets other than the one it is supposed to receive. Some of the algorithms presented in this thesis are based on this simple coding technique.

[Omiwade *et al.*, 2008] and [Dong *et al.*, 2007] offer modifications to COPE that can lead to throughput improvements. There have also been several algorithms proposed that analyze various aspects of a network when this coding scheme is coupled with routing. [Sengupta *et al.*, 2007] discusses maximum throughout achieved by the network using this

scheme. [Le and Lui, 2008] gives an upper bound on the number of packets that can be encoded using this scheme. [Zhao and Medard, 2010] shows that the local fairness enforced by the MAC scheme plays an important role in the throughput improvement offered by this scheme. [Ronasi *et al.*, 2009], [Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009] provide a network utility maximization based rate control algorithm for the flows that use this coding scheme. [Seferoglu *et al.*, 2011] combines inter-session network coding with intra-session network coding in order to make this coding scheme more resilient. However, there hasn't been an exhaustive packet encoding algorithm that can be coupled with a routing scheme irrespective of the objective of the routing mechanism. Moreover, to the best of our knowledge there hasn't been a max-min fair rate allocation and scheduling algorithms for the flows in a network with coding. Therefore, as mentioned earlier in this thesis we look at these aspects of network coding. Next we describe the specifics regarding the contribution of this dissertation.

1.3 Thesis Contribution

This thesis makes contributions in the areas of routing and network coding for wireless networks. We present a multipath routing scheme. We propose an efficient packet encoding algorithm that helps integrate a routing mechanism with network coding. We also look at max-min fair rate allocation and scheduling problems for the flows in a wireless network with coding.

1.3.1 Contribution towards Multipath Routing

Firstly we present a multipath routing mechanism called Polar Coordinate Routing (PCR). This routing scheme serves several purposes as discussed in section 1.1. This routing scheme can be applied to various types of networks. Ideally, it is well suited for a dense network with a large number of nodes, such as large scale wireless sensor networks. It can also be used to route data in wireless mesh networks, and with minor modifications it can be applied to wireless ad-hoc networks as well.

The routing scheme defines multiple arcs in a network, these arcs are basically segments of a circle. The objective of the routing mechanism is to forward data on these arcs. Clearly the denser the network, the more closely packets adhere to their trajectory. We also define a non-euclidean distance metric that helps packets travel on their respective trajectories. Packet forwarding using this distance metric is similar to geographic routing schemes. We also present rules to avoid routing loops when packets are travelling on a circular trajectory.

Note that this routing scheme may be used for multiple objectives (such as successful data delivery, congestion alleviation etc). Instead of checking performance of the routing scheme individually for each application, we specify metrics that can compare two multi-path routing schemes, irrespective of the application they might be used for. We check the performance of the routing scheme using the metrics such as average path separation, fraction of nodes on various paths that are out of each other interference range, hop count etc. We run this routing scheme on a large number of random topologies and present the results.

When the flows are travelling longer hops, one of the biggest concerns in successfully reaching the destination is obstacles on the paths. We integrate our routing scheme with robotic routing [Kim and Maxemchuk, 2005], in order to help packets circumnavigate obstacles. Whenever a packet fails to make forward progress on its trajectory, the packet exits PCR and enters robotic routing. The packets circumnavigates obstacles using the rules specified by robotic routing. Once the packet travels past the obstacle and as soon as a packet realizes that it can make forward progress on its trajectory, it quits robotic routing and reenters the PCR. We specify rules on how to switch between two routing mechanisms in order to overcome obstacles.

1.3.2 Contribution towards Packet Encoding Algorithms and its Integration with a Routing Scheme

Next we work towards integrating network coding with a routing scheme. Our focus is on a coding scheme that is suggested in [Wu *et al.*, 2005] and implemented in [Katti *et*

al., 2006]. We present a novel packet encoding algorithm that searches a node's queue exhaustively in order to identify the maximum number of packets that can be combined in a single transmission. Our algorithm can be easily integrated with a routing scheme such that we can leverage the benefits offered by both an advanced routing scheme and network coding. Our packet encoding algorithm is not dependent on a particular routing scheme such as PCR. In fact it can be coupled with any routing mechanism. We demonstrate this by integrating our packet encoding algorithm with a routing mechanism where the next hop neighbors for a packet change dynamically. Our packet encoding algorithm offers the following benefits.

- Unlike prevailing packet encoding algorithms [Katti *et al.*, 2006], it searches a node's queue exhaustively.
- The algorithm runs in polynomial time. Therefore it asymptotically is faster than naïve exhaustive search.
- The algorithm can consider multiple neighbors as next hop candidates for a packet. Hence a node can forward the packet to a neighbor that helps combine more packets, thus improving the network throughput.
- Our packet encoding algorithm does not depend on a routing scheme, and hence it can be easily integrated with any routing mechanism.

We first present our packet encoding algorithm as a binary integer program. A binary integer program is NP complete. Moreover, not all the routers may have an optimization tool. Therefore we provide an additional algorithm to find an optimal packet combination. We show that finding an optimal packet combination in a node's queue is analogous to enumerating cycles in a bipartite graph. Next we extend our algorithm to consider multiple neighbors as next hop candidates for a packet. We demonstrate that considering multiple next hop candidates improves the possibility of combining more packets, hence it improves the throughput.

We also demonstrate how this packet encoding algorithm can be integrated with a routing scheme. We couple this algorithm with a routing scheme that changes a packet's next hop neighbors very dynamically. We run the routing scheme along with our encoding algorithm on a large number of topologies, and demonstrate the throughput benefits offered by this combination.

1.3.3 Contribution towards Max-min Fair Rate Allocation and Scheduling

While calculating the throughput of a routing scheme we observed that as the rates of some of the flows in the network change, the total throughput of the network changes dramatically. Therefore in order to avoid this problem, while calculating the throughput we assumed that each flow in the network is assigned the same rate. However, this need not be the case. A flow in the network may be assigned a rate in order to maximize the minimum rate each flow gets (max-min fair rates) or a flow may get its rate depending on the amount of network resources it uses (proportional fair rates). Therefore this dissertation also contributes towards a resource allocation problem, namely max-min fair rate allocation for the flows with network coding. Note that there already has been some work in the literature regarding proportionally fair rates while using coding [Seferoglu *et al.*, 2009], [Seferoglu and Markopoulou, 2009].

We couple a conflict graph based framework with simple linear programming to achieve max-min fairness for the flows in a wireless network with coding. We demonstrate how to emulate progressive filling for such a scenario. First we present our fair rate allocation algorithm from a global/centralized perspective. We show that the computational complexity of this algorithm is polynomial time. We also demonstrate that emulating progressive filling is not straight forward for a wireless network with coding. We present caveats that yield suboptimal or incorrect results if not dealt with carefully. We run this algorithm on a number of different topologies, and compare the rates assigned using our algorithm with existing max-min rate control schemes using various fairness metrics. We also present a

novel distributed version of this algorithm that achieves the same results as the centralized algorithm. We show how many messages are transmitted network wide in order for the distributed algorithm to converge to the same results as the centralized algorithm.

We also present a centralized as well as distributed version of a scheduling mechanism that helps flows achieve the rates allocated using our fairness mechanism. Our distributed algorithm works with a standard CSMA scheme, therefore it can be implemented with prevailing 802.11 standards. Hence it eliminates the need of introducing new protocols in the network. We simulate how much throughput various flows get using this scheduling mechanism. We calculate the error in achieving max-min fair rates using this scheduling scheme as well.

1.4 Outline of the Thesis

This thesis is split into five chapters. In the current chapter we provided a background and the outline of our research.

In chapter 2 we present Polar Coordinate Routing. Section 2.3 presents the mechanism of this routing scheme. In section 2.4 we compare our routing mechanism with existing multipath routing scheme and present the results. In section 2.5 we demonstrate how to integrate PCR with robotic routing in order to help packets circumnavigate obstacles. Section 2.6 presents the performance of the routing scheme when it is integrated with robotic routing.

In chapter 3 we present our packet encoding algorithm that helps us integrate network coding with any routing scheme. Section 3.3 gives the detailed explanation of our encoding algorithm. More specifically section 3.3.3 presents the problem of finding the optimal packet combination as an integer program. In section 3.3.4 we show that searching a node's queue exhaustively is analogous to enumerating cycles in a bipartite graph. In section 3.3.5 we extend this exhaustive search to include multiple next hop neighbor candidates for each packet, hence we improve the possibility of combining more packets. In section 3.4 we show how this packet encoding algorithm can be coupled with a routing scheme. In section 3.5

we present some results regarding what kind of throughput improvements we get when our packet encoding algorithm is integrated with a routing scheme.

In chapter 4 we present our max-min fair rate allocation algorithm for the flows in a wireless network with coding. In section 4.3 we present our rate allocation algorithm in a centralized fashion. Section 4.3.4 discusses the complexity of this algorithm. In section 4.3.5 we apply our rate allocation algorithm to a few topologies and compare the results with existing rate control algorithm. In section 4.5.1 we present our rate allocation algorithm in a distributed fashion. For a few different scenarios we also simulate how many messages are transmitted network wide in order for the flows to achieve the same rates as the centralized algorithm. In section 4.4 we present our centralized scheduling algorithm. Section 4.5.2 presents the scheduling algorithm in a distributed fashion. We simulate what type of rates the flows achieve using this scheduling scheme and present the error in achieving the true max-min fair rates.

Finally in chapter 5 we conclude our study.

Chapter 2

Polar Coordinate Routing for Multiple Paths in Wireless Networks

2.1 Introduction

In wireless networks geographic routing techniques follow the most direct path to a destination. However, there are instances where the direct path is not sufficient and multiple paths are needed to connect a source and a destination. Multiple paths offer several advantages.

- Sending data through multiple paths to a destination increases reliability of data delivery.
- In a congested network, setting up multiple paths may reduce congestion in the network by spreading out packets towards the underutilized parts of the network.
- When data is segmented into multiple parts, and each data segment is transmitted to the destination on a separate path, multiple paths prevent an adversary from intercepting the complete set of data.

While multiple paths between a source and a destination offer a few advantages, they also require precautions. For example, if the paths between the source and the destination are close to each other, there will be interference between these paths. On the other hand, when these paths are spread out in a network in order to maintain a higher separation, the total number of hops may increase beyond an acceptable level.

Unfortunately existing solutions for multipath routing in wireless networks do not necessarily offer a good solution in terms of path separation, average number of hops etc. There is some work in the wireless networking domain that shows how to forward messages on a trajectory, however this solution has not been tested for multiple paths. Moreover, it does not offer a good solution to circumnavigate obstacles and the areas with low node density.

In this chapter we present a simple way to form circular arcs between a source destination pair. We present a simple non−euclidean distance metric using which messages can be forwarded through the nodes that are closest to these arcs. We also show that the arcs maintain a high level of separation, which reduces the possibility of interference. We integrate our message forwarding scheme with simple robotic routing, so that it can circumnavigate the areas with obstacles and low node density. We also demonstrate that using our non−euclidean distance metric we can continue forwarding messages along the predefined trajectory even after the obstacle is crossed.

This chapter is organized as follows. In section 2.2 we give a short overview of existing work on multi−path routing in wireless networks. Section 2.3 demonstrates the functionality of PCR. In section 2.4 we show our simulation results and compare our results with existing schemes. In section 2.5 we integrate PCR with simple robotic routing, which gives it an ability to overcome obstacles. In section 2.6 we show the performance of PCR in adverse conditions like areas with obstacles and low node density. Finally, we summarize our study in section 2.7.

2.2 Related Work

Greedy forward routing [Finn, 1987] is one of the simplest forms of routing in wireless networks where location information of nodes is available. In greedy forward routing a node chooses its next hop neighbor such that it is geographically closest to the destination among all the neighbors. While it is not useful in creating multiple paths between a source destination pair, in our simulations we use this technique as a benchmark and compare the performance of PCR with greedy forward routing.

Biased geographic routing (BGR) [Popa *et al.*, 2006] is geared towards reducing congestion in a network, and it offers a simple way of forwarding messages through multiple paths. In this method a source initially transmits its message at an angle θ. A node located in that direction receives this message and forwards it to a node that is located at an angle $\theta_{new} = \theta - \frac{K}{d^2}$, where K is a constant and d is the distance between the current node and the destination. As the message gets forwarded on every hop, the value of θ decreases, which forms an arc. Eventually θ will become zero and from that point on the message will be forwarded directly to the destination. By choosing different values for initial angle θ, it is possible to set up multiple paths between a source destination pair.

While this scheme can be helpful in setting up multiple paths in a network with high node density, its performance is very poor in a network with low node density. If a message sender does not find a receiver at the angle θ, it will send messages to a node that is far from the desired path. Since this method does not have a way to specify a particular path, if a message is not delivered at the correct angle θ, it will wander away from the desired path or come too close to the greedy forwarding path. This would either lead to increasing the total number of hops or increasing the interference with the other paths, both of which are highly undesirable scenarios. Moreover, if the initial θ and constant K are not chosen correctly, a message may spiral around the destination before it gets delivered. If a message runs into obstacles this method does not propose anyway to circumnavigate them.

Trajectory based forwarding (TBF) [Niculescu and Nath, 2003] is a method that defines a trajectory in terms of parametric equations, and it lets a message travel along this trajectory.

For example, if a message travels on a straight line that passes through a point (x_1, y_1) and has a slope α, TBF will represent this trajectory as $X(t) = x_1 + t\cos(\alpha)$ and $Y(t) = y_1 + t\sin(\alpha)$. Each node in TBF will choose its next hop neighbor such that the messages travel along this trajectory. If we form several trajectories between a source and a destination we can achieve multipath routing. However, performance of TBF is not tested for multipath routing. Naturally, not all the trajectories can be ideal for multipath routing. For example, if two trajectories overlap each other, it may lead to very high interference and disrupt the performance of the network. Therefore, it is necessary to come up with a way to define paths that would maintain a large separation between one another. Furthermore, in TBF each message has to include the type of trajectory and also all of the parameters that define the trajectory, which can amount to a very high overhead.

Unlike BGR, TBF attempts to circumnavigate an obstacle by estimating the size of it. TBF proposes a technique in which whenever a node cannot forward a message greedily along the trajectory, it assumes that there is an obstacle next to it. The node tries to estimate the diameter (Δ) of the obstacle and attaches it to the message in terms of a parameter of the trajectory. This way the message will try to travel around the estimated obstacle. Each node forwarding a message in this mode determines if it can forward the message greedily along the original trajectory. If the node cannot forward the message greedily along the trajectory, it is assumed that the message is still trying to overcome the obstacle and the current process continues, otherwise the node quits the algorithm and forwards the message greedily along the trajectory. If the estimation of obstacle diameter is too high, a lot of nodes will unnecessarily end up performing the calculations for exit points. An overestimation of Δ can also mean that the message won't be able to travel along the trajectory even if the obstacle is crossed. On the other hand, an underestimation of Δ can result into a message spiraling around the obstacle multiple times before it actually gets back on the trajectory.

[Kim *et al.*, 2005], [Kuhn *et al.*, 2003], [Karp and Kung, 2000], [Bose *et al.*, 1999] and [Kranakis *et al.*, 1999] also present algorithms to circumnavigate obstacles. These

algorithms are primarily based on forwarding messages to the nodes that form a planar graph. Robotic routing [Kim and Maxemchuk, 2005] is another technique that lets a message circumnavigate obstacles, however it does not require us to pre-compute planar graphs. We integrate PCR with robotic routing to overcome obstacles.

2.3 Polar Coordinate Routing (PCR)

Some of the objectives that are necessary for a good multipath routing scheme include,

- Trajectories created by multipath routing scheme should maintain a known separation among each other to reduce interference.
- While the trajectories should be far from each other to reduce interference, the total number of hops should not increase too much.
- Message overhead to define a trajectory should be low.
- If a message encounters an obstacle, it should be able to circumnavigate the obstacle, and continue traveling on the trajectory.

In Polar Coordinate multipath routing a trajectory is represented by an arc that is a segment of a circle. The center of this arc lies on the bisector of the line segment connecting the source and the destination (figure 2.1(a)). The source and the destination are basically two end points of this arc. A message from a source to the destination travels on this arc. If we choose a different point on the bisector as the center, we can obtain another arc with a different radius connecting the source destination pair. Using this technique we can form multiple paths. We choose trajectories as circular arcs since they do not overlap each other. Furthermore, it is very easy to maintain a large separation between two trajectories as we will show in this section. The overhead of defining an arc is also relatively low. In order to define an arc, the only thing that has to be included in a message is the location of the source, destination and the center of the arc.

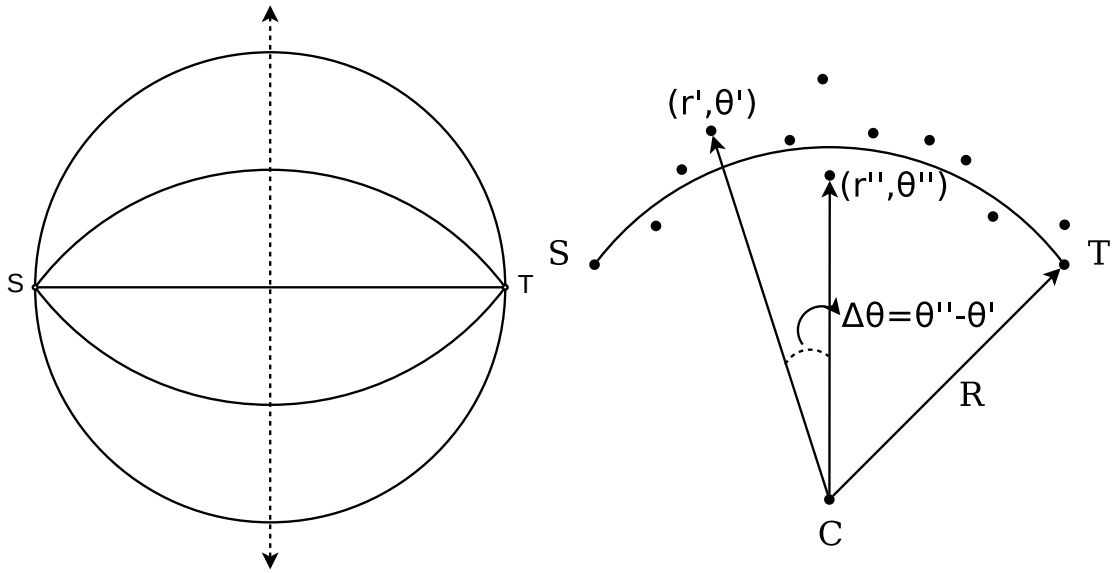
Before we formally introduce a method of defining arcs, we mention some of the assumptions that we made.

- Each node in the network is aware of its location in cartesian coordinate system.
- A node is also aware of the location of its one hop neighbors.
- The source node knows the location of the destination.
- Each node is equipped with some basic computational resources that can perform simple arithmetic operations.

PCR defines arcs with different radii that connect a source destination pair as shown in figure 2.1(a). The objective is to send messages through the nodes that are close to these arcs. It is relatively easy to formulate this problem in a network where the nodes are localized according to the polar coordinate system. Say the arc connecting the source and the destination has a radius R. Also, for the simplicity let's assume that the center of the arc C has coordinates $(0, 0)$ in the polar coordinate system. Hence, the goal of PCR is to send messages through the nodes that are R distance away from the center C. Moreover, PCR also has to make sure that a node selects its next hop neighbor such that the message travels the maximum angular distance. In other words the quantity $\Delta\theta$ in figure 2.1(b) has to be maximized.

These calculations are straightforward in the cartesian coordinate system as well as the polar coordinate system. For the sake of simplicity we demonstrate these calculations in the cartesian coordinate system.

The idea behind PCR is similar to geographic routing. However, unlike geographic routing, PCR defines a non−euclidian distance metric, which allows messages to travel on an arc instead of following a direct path. In this section we develop a new distance metric and describe how to travel along an arc using this metric.



(a) (b)

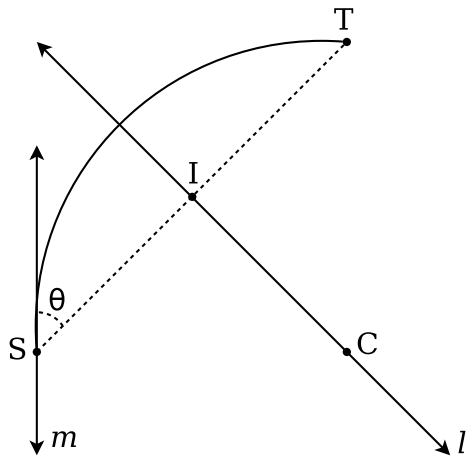
Figure 2.1: Polar Coordinate Routing

2.3.1 Arc Specifications

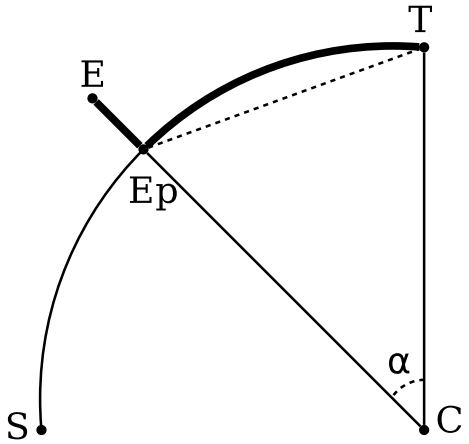
Given a source (S) and a destination (T), we can draw a line segment $\overline{ST}$; let line l be the bisector of $\overline{ST}$. We can pick any point along line l, which can represent the center of a circular segment connecting S and T, let us call this point C. Note that C does not have to be represented by a node. The position of C along l defines the curvature of an arc. For example, if I represents the intersection of l and $\overline{ST}$,the closer the C to I, the sharper the curve.

A more elegant way to define an arc is by its radius. If the distance between S and T is d, let's represent the radius of the circle as $R = a\frac{d}{2}$, where $a \in [1, \infty)$. Let m be the line passing through S and tangent to arc $\overset{\frown}{ST}$, and θ be the angle between m and $\overline{ST}$, then $a = \frac{1}{\sin\theta}$ (figure 2.2(a)). Hence, by controlling the value of a, it becomes easy to control the curvature of the arc. For example, for a semicircle $\theta = \frac{\pi}{2}$, hence a would be 1.

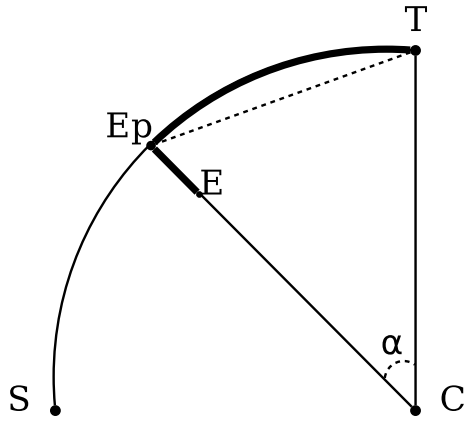
Thus we can define arcs with different curvatures and traverse messages along these arcs.



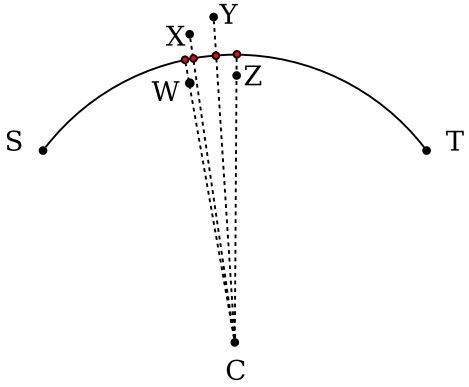
(a) Arc Specifications



(b) A Non−Euclidean Distance Metric



(c) A Non−Euclidean Distance Metric



(d) In PCR a node is selected as a next hop neighbor only if it is closer to the destination along the arc

Figure 2.2: Polar Coordinate Routing

2.3.2 Non-Euclidean Distance Metric

We define a new non-euclidean distance metric, which gives this scheme the ability to forward messages along an arc. Our goal is to define a metric such that a message not only remains close to the arc, but it also travels as far as possible along the arc.

Let $D(E,T)$ represent the non−euclidean PCR distance metric between the location of a network node E and the destination T. As shown in figures 2.2(b) and 2.2(c), let point E_p be the intersection of $\overline{EC}$ and $\overset{\frown}{ST}$. We define $D(E,T)$ as the length of $\overline{EE_p}$ plus the length of the arc $\overset{\frown}{E_pT}$. Now, it is possible to show that euclidean distance between E_p and T (d(E_p,T)), and the length of $\overset{\frown}{E_pT}$ are both one to one and increasing functions of α, where α is the angle between $\overline{EC}$ and $\overline{CT}$. Therefore, we can write $D(E,T)$ as,

$$D(E,T) = d(E,E_p) + d(E_p,T).$$

In a network where nodes are forwarding packets according to PCR, a node X will calculate the distance $D(N_x,T)$ among all its neighbors N_x and the destination T. Node X sends its messages to a node that yields the smallest value of $D(N_x,T)$. The optimization criterion of $min\{D(N_x,T)\}$ can be broken down into two parts as $min\{d(N_x,N_{xp})+d(N_{xp},T)\}$. Therefore, nodes in PCR do not necessarily pick the next hop neighbor that is closest to the arc, nor do they select the neighbor that is farthest along the arc. In PCR a node selects its next hop neighbor which gives the minimum aggregate value of both these criterions: closest to the arc and the farthest along the arc.

It is a common practice in geographic routing to forward a packet to a node that is closer to the destination compared to the current node. Hence, in geographic routing a node chooses its next hop neighbor that is closer to the destination. Following this rule prevents a message to be forwarded in the backward direction and hence it avoids the routing loops. We also adhere to this principle in PCR. In figure 2.2(d), if node X is looking to forward its messages to one of its neighbors, it will only consider nodes Y or Z, since their projections on the arc are closer to the destination compared to the projection of X. Note that even though node W seems geographically closer to the destination than

X, it will not be considered as a next hop candidate since it's projection is farther to the destination than the projection of X.

2.3.3 Separation between Arcs

So far we have established how to define an arc, and how to navigate a message along it. Sometimes it is useful to predict what kind of results we are going to get from an arc. For example, it may be helpful to estimate what proportion of nodes on an arc is going to interfere with the nodes on other paths. If we can estimate the average separation between two arcs in advance, we can make an educated guess regarding how sharp an arc should be to yield a low interference with the neighboring arcs.

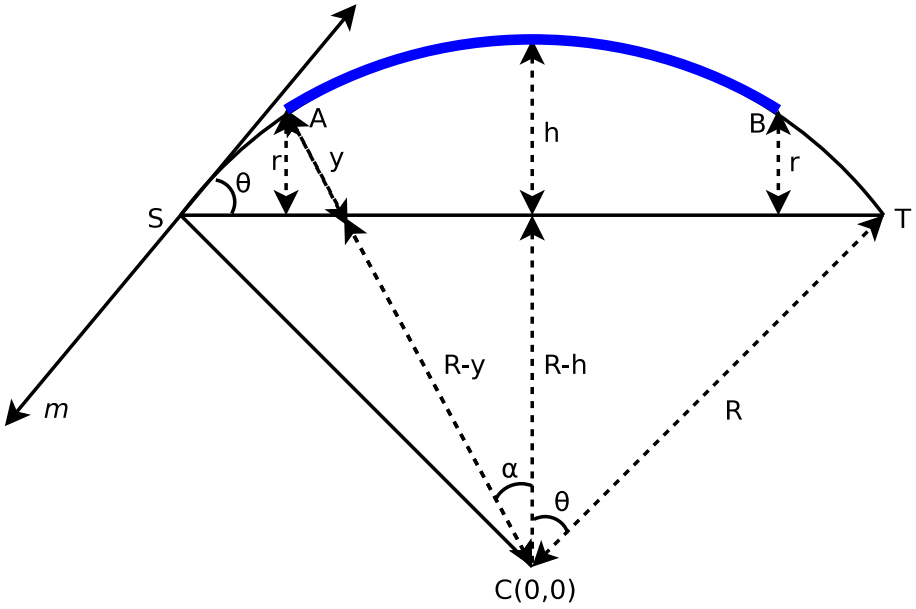


Figure 2.3: Estimation of nodes from an arc out of range of greedy forwarding path

Consider a network where all the nodes have the same transmission radius r. In figure 2.3 for a source (S) and a destination (T), let line segment $\overline{ST}$ represent the greedy forwarding path, and $\overset{\frown}{ST}$ represent the path created by PCR. Moreover assume that $\overset{\frown}{ST}$ is formed at an angle θ with respect to the $\overline{ST}$, which yields an arc radius of R. All the points on $\overset{\frown}{AB}$ in

figure 2.3 are more than a transmission radius away from all the points on $\overline{ST}$. Therefore, probability that the nodes lying along $\overset{\frown}{AB}$ will interfere with the nodes on greedy path is very low. Assuming that the number of hops to reach T from S along $\overset{\frown}{ST}$ is proportional to the length of this arc, we can estimate the proportion of nodes that will not interfere with the nodes on the greedy forwarding path as $\frac{length\ of\ \overset{\frown}{AB}}{length\ of\ \overset{\frown}{ST}}$. Since the paths formed by PCR are circular it is very easy to calculate the length of these arcs. Length of $\overset{\frown}{ST}$ would be $\pi \times 2\theta$, while length of $\overset{\frown}{AB}$ would be $\pi \times 2\alpha$, where $\alpha = \cos^{-1}(\frac{R-h+r}{R})$ and $h = R(1 - cos\theta)$.

It is also possible to calculate what proportion of nodes on an arc will not interfere with the nodes belonging to another arc when PCR forms multiple trajectories. For example, in figure 2.4 there are two arcs that connect to a source and a destination. In this scenario we can assume that the nodes belonging to $\overset{\frown}{AB}$ will not interfere with any nodes belonging to $\overset{\frown}{CD}$. Let us assume that the arc on the top has a radius R_1 and for simplicity let us assume that its center is at C_1 whose cartesian coordinates are (0,0). The other arc has a radius of R_2, and its center is located at point C_2. Hence, the real solutions of equations $X^2 + Y^2 = (R_1 - r)^2$ and $(X - C_{2x})^2 + (Y - C_{2y})^2 = R_2^2$, will give the cartesian coordinates of $C(X_C, Y_C)$ and $D(X_D, Y_D)$. Using these coordinates we can calculate the length of $\overset{\frown}{CD}$, and hence we can also calculate the proportion of the first arc that will not interfere with the second arc. Similarly, it is also possible to calculate the length of $\overset{\frown}{AB}$ by calculating coordinates of A and B. $X_A = X_C \frac{R_1}{R1-r}$, $Y_A = Y_C \frac{R_1}{R1-r}$, $X_B = X_D \frac{R_1}{R1-r}$ and $Y_B = Y_D \frac{R_1}{R1-r}$.

Using this technique we can choose the curvature of an arc such that the resulting path will yield a low interference with the other paths.

2.4 PCR: Results

In this section we check the performance of PCR compared to BGR. Multipath routing schemes may be used for multiple objectives such as successful data delivery, congestion alleviation, data security etc. Instead of checking performance of the routing schemes individually for each application, we specify metrics that can compare two multipath routing

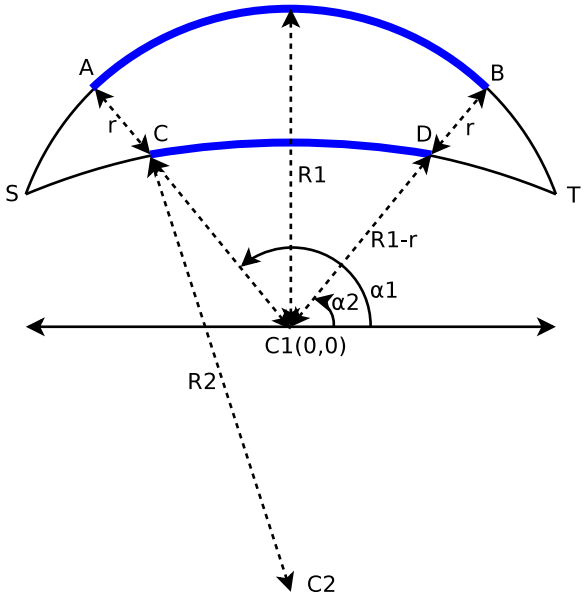


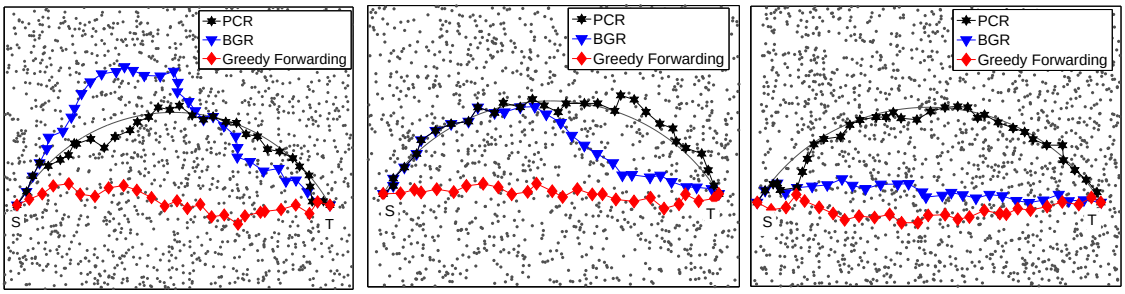
Figure 2.4: Estimation of nodes from an arc out of range of the nodes that belong to another arc

schemes, irrespective of the application they might be used for.

Our simulations show that the non−euclidian distance metric successfully allows network nodes to forward messages along an arc. Furthermore, if the messages have to deviate away from the arc due to the sparse node density, this distance metric provides them with a tendency to get back on the arc.

On the other hand, the path of the messages in BGR does not necessarily have a structure. Regardless of the initial θ, sometimes a path created by BGR can be very close to the greedy forwarding path, and sometimes it can be unnecessarily far away from the greedy path. Furthermore, the performance of BGR worsens if the node density is low.

Figure 2.5 compares the routing paths formed by PCR and BGR. For all three scenarios presented here, initial θ of BGR is set equal to the angle of the arc formed by PCR. In figure 2.5(a) messages of BGR deviate further away from the greedy forwarding path, hence it increases the hop count unexpectedly. On the other hand, in figure 2.5(b), path of



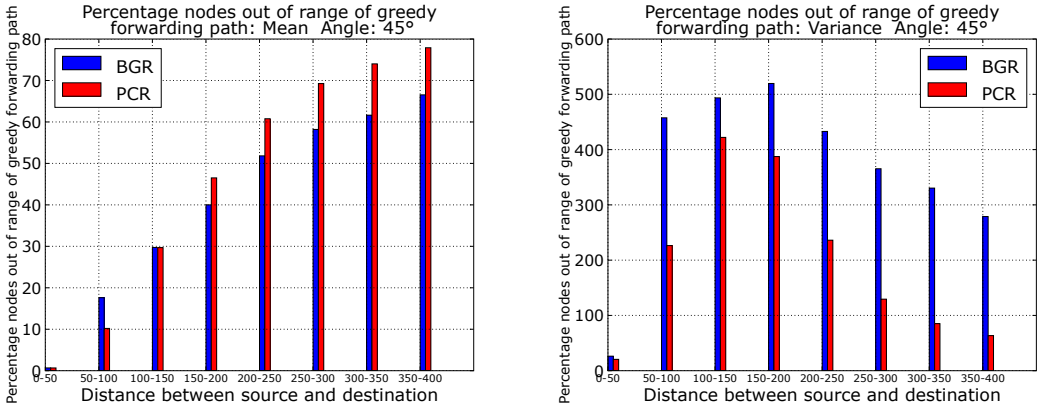
(a) (b) (c)

Figure 2.5: A Comparison of PCR and BGR

the BGR is closer to the greedy forwarding path, which can increase the interference. It should be noted that in both the figures 2.5(a) and 2.5(b) messages of PCR travel along the specified arc. Figure 2.5(b) is a quite common scenario for BGR. In BGR, nodes forward their messages at an angle θ according to the rule $\theta_{new} = \theta - \frac{K}{d^2}$, where d is the distance between the current node and the destination. Hence, at every hop the value of θ will decrease, which would cause the formation of an arc. However, eventually the θ will hit the value of zero and from that point the messages will be forwarded greedily to the destination. Thus, as a message in BGR gets close to the destination, its path becomes very close to the greedy forwarding path, increasing the probability of interference. Finally, figure 2.5(c) presents a scenario where BGR yields a path that is almost parallel to the greedy forwarding path. A careful inspection of the figure will indicate that the network density is quite sparse around the source, hence PCR and BGR both choose first couple of hops that are close to the greedy forwarding path. However, BGR fails to recover from this and the rest of the path also stays parallel to the greedy forwarding route. PCR on the other hand quickly recovers from this, and chooses its nodes wisely so that rest of the path progresses along the predefined arc.

It should be noted that the results presented in this section are based on a simulation where 2500 nodes are uniformly scattered in a $500 \times 500m^2$ area. Therefore, a node in this network is surrounded by approximately 12 other nodes on average. Each node in

the network has a transmission radius of $20m$. The results presented in the rest of the section are the averages of 1000 runs. Furthermore, in all our simulations, the constant K in $\theta_{new} = \theta - \frac{K}{d^2}$ is chosen in a manner such that the path formed by BGR will progress as close to the arc as possible given the distance between the source and the destination, where $K \in \mathbb{N}$.



(a) Percentage nodes out of range of greedy forwarding path: Mean

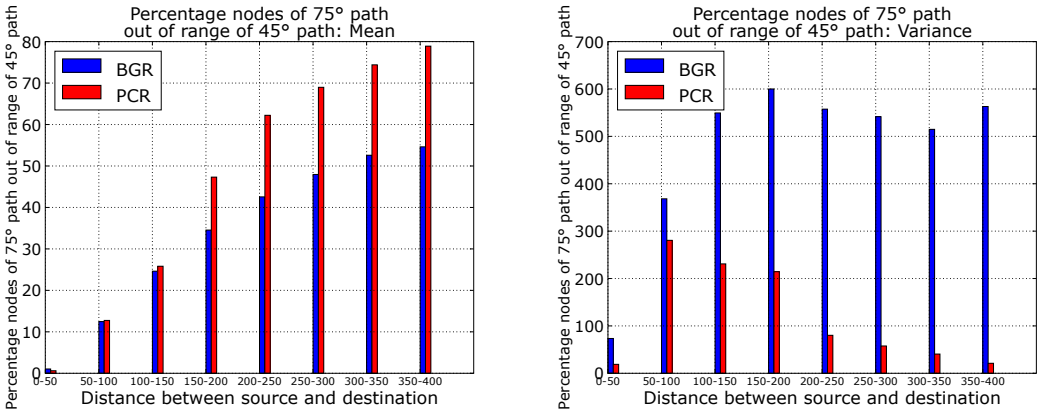
(b) Percentage nodes out of range of greedy forwarding path: Variance

Figure 2.6: Percentage nodes out of range of greedy forwarding path when multi-path routing schemes form paths at $\theta = 45°$

2.4.1 Path Separation

Figure 2.6 shows the means and variances of the percentage nodes of PCR and BGR that are completely out of range of the greedy forwarding nodes. It should be noticed that in most of the cases PCR yields a higher percentage of nodes that are out of range of greedy forwarding path compared to BGR. Therefore, paths created by PCR maintain a higher separation with the greedy forwarding route. Figure 2.6(b) shows that PCR yields very low variances compared to BGR. A low variance in separation indicates that unlike BGR, messages in PCR stick close to the predefined path. Therefore, PCR is much more stable compared to BGR in terms of forming paths.

Sometimes it is desirable to send messages along more than one arc. When we send messages across multiple arcs, it is necessary to maintain enough separation among these paths as well. We simulated the scenarios when the paths are formed at 45° and 75° angles, and calculated percentage of nodes on 75° path that are out of range of 45° path. These results are shown in figure 2.7. As the figure indicates, compared to BGR, PCR clearly generates better separated paths with low variances.



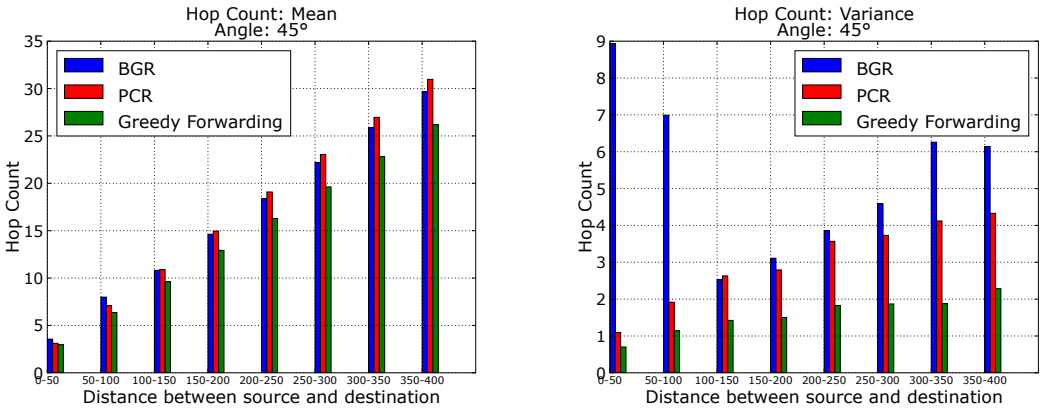
(a) Percentage nodes of 75° out of range of 45° path: Mean

(b) Percentage nodes of 75° out of range of 45° path: Variance

Figure 2.7: Comparison of BGR and PCR when paths are formed at 45° and 75°

2.4.2 Hop Count

Even though PCR outperforms BGR in terms of path separation, it does not create higher separations by forwarding messages through a large number of nodes. Figure 2.8 shows that regardless of the distance between the source and the destination, PCR's hop count would exceed BGR by no more than a couple of hops for $\theta = 45°$. It also shows that greedy forwarding routing will have the least number of hops, which is an expected result. It should be noticed that even in this case PCR yields low variances compared to BGR. Since in PCR messages do not digress away from the arc, it is natural that hop count in PCR will have a low variance.



(a) Hop count: Mean

(b) Hop count: Variance

Figure 2.8: Hop count when multi-path routing schemes form paths at $\theta = 45°$

We performed these simulations for the paths formed at $\theta = 30°, 60°, 75°$ and $90°$ as well. Due to space constraints we have not included results for all the values of θ. However, it should be noticed that the performance of PCR improves as θ increases.

2.5 PCR Integrated with Robotic Routing

While BGR does create multiple paths, it does not have any provisions to overcome obstacles in a network. TBF does try to overcome obstacles, but its performance is very poor. Since in a multipath routing technique messages travel across a larger portion of the network, the probability of encountering an obstacle is higher. Since obstacles are not uncommon and they can adversely affect the performance of a routing protocol, we find it necessary to come up with a scheme that does not fail in an environment with obstacles. We integrate PCR with robotic routing [Kim and Maxemchuk, 2005] protocol which is specially designed for routing messages in a network with obstacles.

Robotic routing requires a network to be divided into a zonal grid. A zone is basically a small square area with the length of its sides equal to $\frac{r}{\sqrt{2}}$, where r is the transmission radius of the nodes. Thus zones are designed in a manner so that a node in a zone can

hear every other node in the same zone. Each zone is given a unique ID and based on its coordinates it is possible for a network node to identify which zone it belongs too.

The idea of PCR's integration with Robotic routing is very simple. A message travels along an arc until it cannot find a next hop neighbor that is closer to the destination along the arc compared to the current node. Once a node cannot find a neighbor that is closer to the destination, it changes its routing scheme to robotic routing. Now the messages circumnavigate the obstacle using the rules defined by robotic routing, and switch back to PCR when it is possible to find a node that can lead closer to the destination using the non-euclidean distance metric.

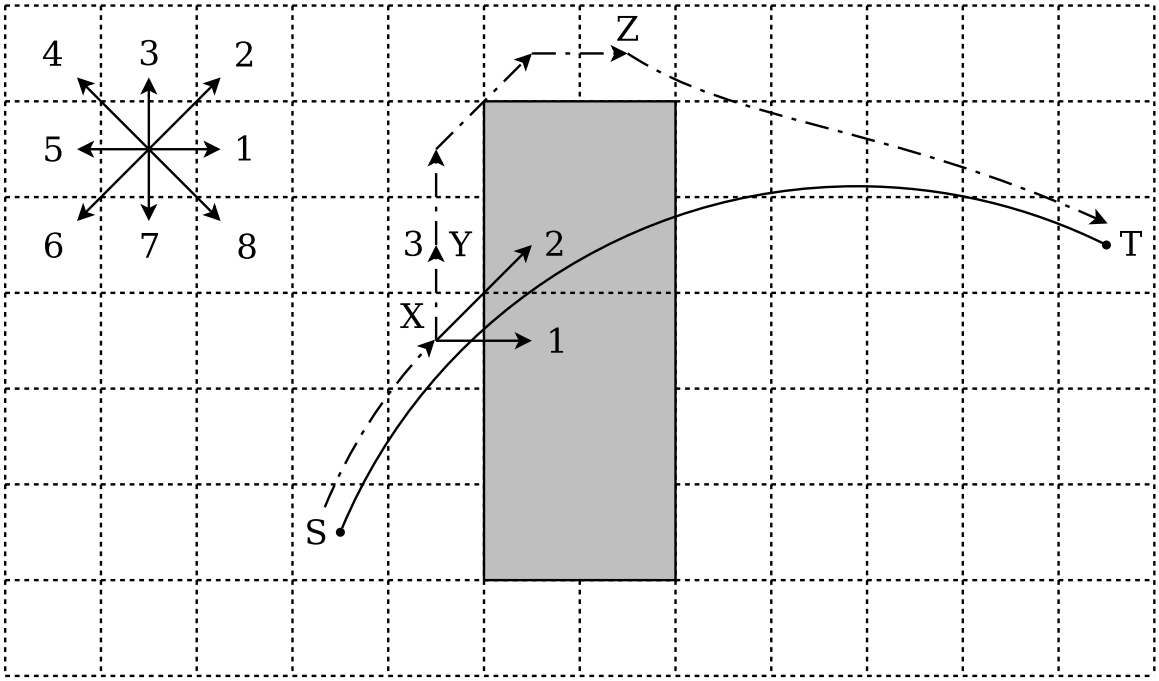
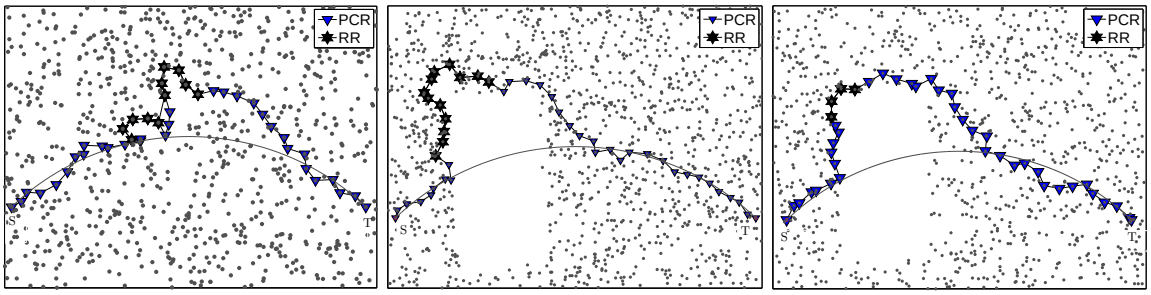


Figure 2.9: PCR integrated with Robotic Routing

While [Kim and Maxemchuk, 2005] thoroughly defines the rules of robotic routing, we present a simplified version here. In simple robotic routing a message can overcome obstacles using the right hand rule. We use the example presented in figure 2.9 to describe the rule.

- Each zone can have eight possible neighboring zones. These zones are labeled from 1 to 8 in the figure.
- At the source, the packet follows PCR until it reaches zone X.
- In zone X an obstacle prevents the packet from moving closer to the destination along the arc and the packet enters the robotic routing mode.
- In zone X we start with neighboring zone 1 and try to locate a node in that zone. If we do not find a node in zone 1, we select a zone counter clockwise, which would lead to neighboring zone 2. If we cannot find a node in this zone either, we continue this process until we find a zone with a node to which we can transmit the message.
- Say we find a node located in neighbor 3, hence we forward the message to a node in that zone. Let us call that zone Y. From zone Y, we continue using right hand rule until we reach zone Z.
- Say from zone Z we can find a node that is closer to the destination than the current node. Hence, we leave the robotic routing, and enter PCR.
- Since PCR has a quality to direct the messages along the arc, it will bring the messages along the arc and deliver it to the destination.

It is possible to route a message to the destination using robotic routing. However, our objective is to route the messages along multiple trajectories. If we continue to forward messages using robotic routing even after the obstacle is crossed, it is possible that different paths may overlap with each other. For example consider a scenario where we are sending messages at different angles, and all the paths run into the obstacle. Since all the nodes will use the right hand rule, the chances are the nodes will leave the obstacle in the same zone. Now if these messages are continued to be forwarded using robotic routing, all the messages will go through the same set of zones to reach the destination. To avoid this undesirable scenario it is important to leave robotic routing when an obstacle is crossed. We also define rules on when to quit robotic routing.



(a) Low node density, small obstacles

(b) High node density, large obstacle

(c) Low node density, small and large obstacles combined

Figure 2.10: PCR Integrated with Robotic routing. Nodes chosen according to PCR are shown in black, robotic routing nodes are shown in blue

- If we are routing using robotic routing and it brings the messages back to the same zone twice, and the only way out of this zone is the path that we took earlier, we identify a routing loop and stop forwarding the message.
- If the hop count goes beyond the maximum hop count we stop forwarding a message. In both these cases if we want to acknowledge the failure to the source depends on the nature of the application.
- If we find a node that is closer to the destination along the arc than the current node, we leave robotic routing and reenter PCR.

2.6 Results: PCR Integrated with Robotic Routing

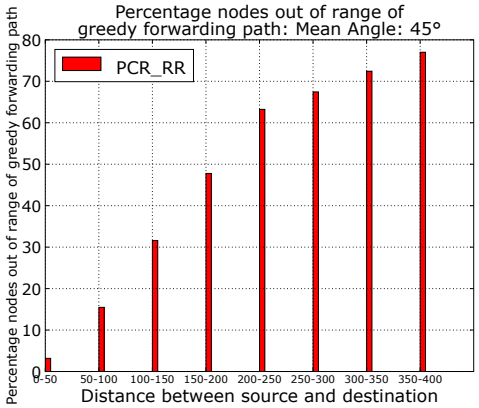
Figure 2.10 shows paths generated by PCR and robotic routing. All three arcs are generated at 45° angle. Figure 2.10(a) presents a scenario when the node density is low (1800 nodes in $500 \times 500m^2$ area). Note that in figure 2.10(a), while approaching the destination two transitions were made from PCR to robotic routing. In figure 2.10(b) a big obstacle is introduced and the node density is chosen to be 2500 nodes in $500 \times 500m^2$ area. Figure 2.10(c) presents a scenario where not only there is a big obstacle, but the node density

is also low. Usually it is difficult to route messages for long distances in such a scenario like figure 2.10(c). In all three figures it should be noticed that once the message leaves the robotic routing mode, the distance metric defined in section 2.3 starts to choose nodes that are closer to the arc, and the message indeed reaches the destination along the trajectory.

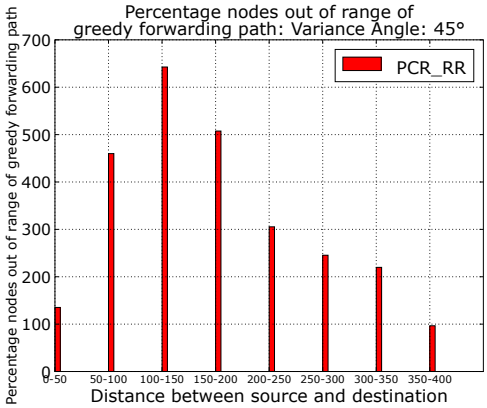
We use the metrics described in section 2.4 to evaluate the performance PCR's integration with robotic routing. For our evaluation instead of using a large obstacle in the network we use a network with a very low node density. A network with a sparse node density can represent a lot of small obstacles, since many parts of the network will have no nodes closer to the destination.

For our simulations we set up a scenario where 1800 nodes are uniformly scattered in a $500 \times 500m^2$ area, and each node has a transmission range of $20m$. We choose a node density of 1800 because it would cause a node to have 8 neighbors on average. Some research efforts show that a message will not be able to travel a large number of hops if the node density goes below this [L. Kleinrock, 1978]. The results presented in this section are the averages of 1000 runs. Since BGR does not have provisions to overcome obstacles, we only present the results of PCR. Moreover, since robotic routing is utilized along with PCR, it will be able to reach a destination more successfully compared to greedy forwarding. For figures 2.11 and 2.13 we have only considered the scenarios when not only PCR but greedy forwarding also reaches the destination.

Figure 2.11 shows percentage nodes of PCR and robotic routing that are out of range of greedy forwarding path. Figure 2.12 shows percentage nodes of 75° path that are out of range of 45° path, and figure 2.13 shows the hop count. Comparing the variances presented in this section with the results shown in section 2.4 would indicate that if PCR has to switch to robotic routing due to obstacles, overall performance of this scheme worsens. The deterioration in the results is expected since the presence of obstacles can deviate the path of the messages away from the predefined arc.

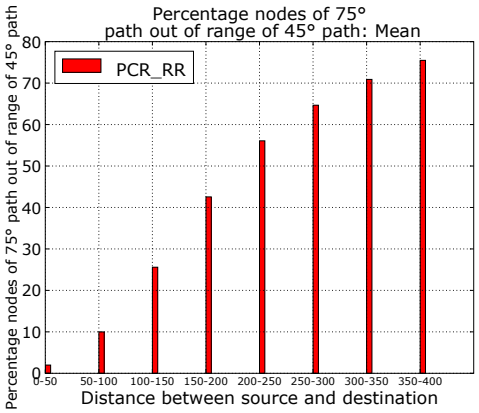


(a) Percentage nodes out of range of greedy forwarding path: Mean

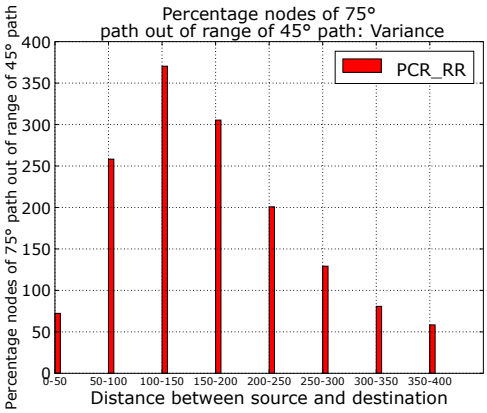


(b) Percentage nodes out of range of greedy forwarding path: Variance

Figure 2.11: Percentage nodes out of range of greedy forwarding path when PCR+RR form path at $\theta = 45°$

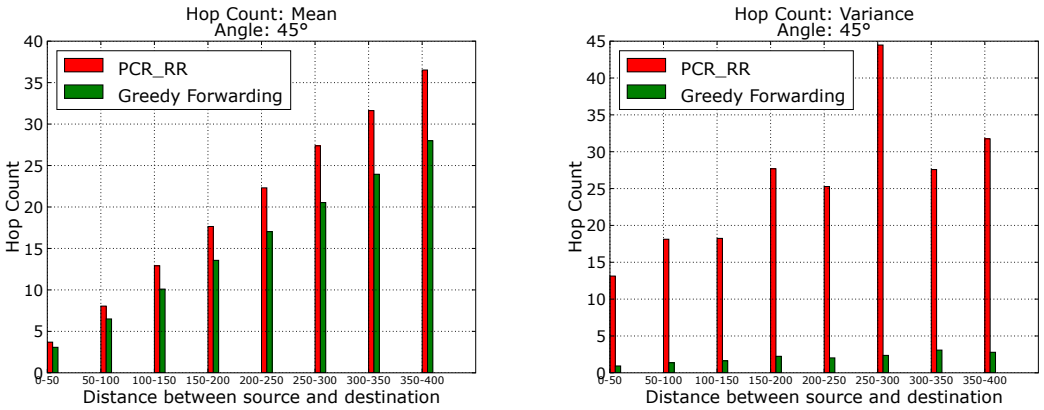


(a) Percentage nodes of 75° out of range of 45° path: Mean



(b) Percentage nodes of 75° out of range of 45° path: Variance

Figure 2.12: Performance of PCR+RR with obstacles in the network when paths are formed at 45° and 75°



(a) Hop count: Mean

(b) Hop count: Variance

Figure 2.13: Hop count when PCR+RR form path at $\theta = 45°$

2.7 Summary

We presented Polar Coordinate Routing, which creates multiple paths between a source and a destination. Since these paths are segments of circles with different radii, it is easy to control average separation between them. We also presented a non−euclidean distance metric that lets messages travel on these arc greedily, and does not increase overall hop count unnecessarily. We show that variances of hop count and average separation in PCR are too low compared to existing multipath routing scheme, hence PCR offers a much more reliable system. Furthermore, given a source and a destination a circular trajectory can be easily defined by including only its center in the message, which does not impose too much additional overhead of defining a trajectory on the system.

We also integrated PCR with robotic routing to overcome obstacles and areas with low node density. We also showed that with the help of our distance metric a message can reach its destination along the trajectory even after crossing the obstacle.

Chapter 3

A Packet Encoding Algorithm for Network Coding with Multiple Next Hop Neighbor Consideration and its Integration with a Routing Scheme

3.1 Introduction

In this chapter we present an efficient and exhaustive packet encoding algorithm that helps couple network coding with a routing scheme, irrespective of the routing scheme's objective.

Network coding is often employed to reduce the number of transmissions in wireless networks with omni-directional antennae, by using the packet information available at the nodes. Alice and Bob topology illustrated in figure 3.1 provides a classic example of the effectiveness of network coding. Suppose node A and B want to exchange m_a and m_b with each other through a node F. If F forwards these messages individually, four transmissions

will be required for A and B to exchange their messages. However, if F broadcasts $m_F = m_a \oplus m_b$ ($\oplus$ represents binary addition), A can retrieve $m_b = m_F \oplus m_a$, since it already has the knowledge of m_a. Node B can use the same transmission to retrieve $m_a = m_F \oplus m_b$. Thus network coding helps reduce the overall number of transmissions from four to three.

Due to its effectiveness in improving throughput, network coding is one of the most actively explored routing paradigms. Significant research efforts focus on finding best ways to code packets. Some schemes suggest routing flows through certain nodes in a network to improve the probability of finding packets that can be coded with each other [Sengupta *et al.*, 2007]. These techniques often rely on the global knowledge of the flows in a network, which may not always be available. Even if we are to propagate flow information throughout the network, it is unreasonable to assume that all the flows in the network will persist until a source makes its forwarding decisions. Therefore, in this chapter we focus on an encoding algorithm and a routing scheme that doesn't require the global knowledge of the network flows.

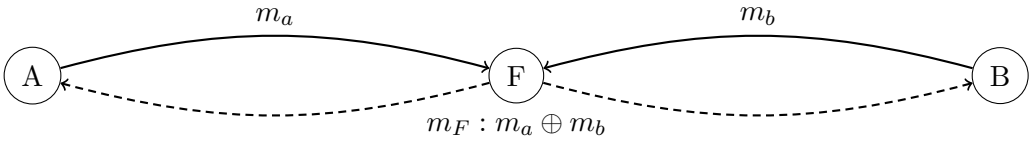


Figure 3.1: An example of network coding

Our packet encoding algorithm is not dependent on PCR. It can be easily integrated with a routing scheme, irrespective of its objective. This helps us leverage benefits offered by both an advanced routing scheme and an efficient packet encoding algorithm. To demonstrate this we integrate our packet encoding scheme with a routing mechanism named Delta routing where the next hop neighbors of a packet change dynamically, since it makes the job of encoding packets more difficult.

An important factor that can potentially limit the throughput gain due to coding is bot-

tleneck nodes in the network. In conventional shortest path routing, due to their geographic location or due to the flow directions, certain nodes receive packets at a rate higher than the rate at which they can transmit them. These bottleneck nodes throttle the throughput of the network. For example, if there are four flows (1) A to B, (2) B to A (3) C to D and (4) D to C passing through F in the network shown in figure 3.2, node F would become a bottleneck node. If such a bottleneck node manages to code one out of k packets on average, we can expect the throughput of the network to improve by a factor of $1/k$. However, it is possible to achieve a higher throughput gain if we can identify underutilized nodes in the network (such as E and G in figure 3.2) and route packets through these nodes, since it helps alleviate congestion (at node F). In this chapter we advocate routing packets according to a delta metric which helps packets circumnavigate congested parts of the network. Thus it not only alleviates congestion, but also helps improve throughput by allowing previously underutilized nodes to forward more packets. We use contour graphs to compare forwarding rates offered by conventional shortest path routing and delta routing in a randomly distributed network. Contour graphs help identify parts of the network that act as bottleneck while using conventional shortest path routing and demonstrate how delta routing improves the throughput by routing more packets through the underutilized parts of the network.

We integrate this routing scheme with network coding in order to gain from the benefits offered by both the schemes. In a scheme like delta routing a node considers multiple neighbors as next hops before choosing to forward the packet to the neighbor with the least value for delta metric. Depending on what neighbor is chosen as the next hop, a packet may or may not be combined with other packets. Therefore, we present an encoding algorithm that considers multiple neighbors as next hops (the ones with the smaller values for delta metric), and chooses to forward the packets to the neighbors that help combine maximum number of packets. Moreover prevailing encoding algorithm performs a sequential search to find the maximum packet combination, while our encoding scheme searches a node's queue exhaustively. We present this problem as an integer program and offer a bipartite graph

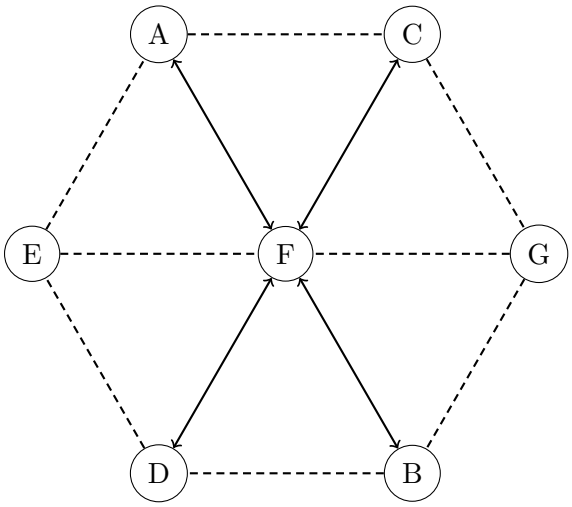


Figure 3.2: There are four flows in this network (i) A to B, (ii) B to A (iii) C to D (iv) D to A. In this case node F is a bottleneck node.

based algorithm to solve this problem which offers the following advantages.

- Although we only apply our encoding algorithm to delta routing, it can be easily integrated with any other routing scheme.
- Unlike a sequential search, our encoding algorithm searches a queue exhaustively.
- It considers multiple neighbors as next hop candidates for a packet and chooses to forward the packet to a node that yields the maximum packet combination.
- The algorithm is asymptotically faster than a naïve exhaustive search since it runs in polynomial time.

This chapter is organized as follows. Section 3.2 presents related works in the areas of network coding and delta routing. In section 3.3 we present our encoding algorithm to find the optimal packet combination. In section 3.4 we present the delta metric that helps packets circumnavigate congested parts of the network and integrate it with our encoding scheme. Section 3.5 demonstrates the results of our routing scheme that couples network coding with delta routing. We summarize our study in section 3.6.

3.2 Related Works

In this chapter our objective is to route packets using network coding while avoiding congested parts of the network by leveraging Delta routing. The concept of Delta routing was developed by Rudin in [Rudin, 1976], since then several routing mechanisms (such as [Porter and Ji, 2004]) have used the basic idea behind it. We make changes to Delta routing and make it more suitable for a wireless scenario with network coding as described in section 3.4. For any given packet if there are multiple nodes that can take the packet to the destination, Delta routing forwards the packet to a node that has the smallest value for the instantaneous queuing delay plus the average delay from that particular node to the final destination. Therefore, Delta routing helps packets circumnavigate congested parts of the network.

Network coding is a well studied subject. After Ahlswede et al.'s seminal work in [Ahlswede *et al.*, 2000], several research efforts have given more insight into how network coding can be utilized to improve a network's performance. In [Katti *et al.*, 2006] Katti et al. implement a routing scheme based on a simple form of network coding, which we use in this chapter. In this scheme a node can forward a combination of k packets (add k packets using exclusive-or:XOR) in a single transmission, if these k packets are going to k different neighbors, and each of these neighbors has the knowledge of the remaining $k-1$ packets. Upon receiving the combination of k packets, a neighbor adds $k-1$ packets that it already knows to this combination and retrieves the packet it is supposed to receive. Coding mechanism proposed in [Ho *et al.*, 2003] can be seen as a generalization of this basic coding scheme. Here, if all the neighbors have the knowledge of only l packets (instead of $k-1$), the forwarding node has to transmit $k-l$ linear combinations of these k packets. Upon receiving these $k-l$ linear combinations, a neighbor can decode its own packet provided these linear combinations were linearly independent. After [Katti *et al.*, 2006] several papers have proposed various ways to combine packets using "XOR-based" techniques to reduce the number of transmissions, notably [Li *et al.*,] and [Sengupta *et al.*, 2007]. In [Li *et al.*,] a transmitting node uses information from the nodes that are as far as two hops from it. On

the other hand [Sengupta *et al.*, 2007] uses global flow information from the entire network to route packets in order to achieve as much coding as possible. Our approach differs in three different ways from these previous approaches: (a) In our routing scheme a node only uses the information available from its immediate neighbors in order to make forwarding and coding decisions (b) While routing packets and looking for optimal packet combination, we also consider congestion in the network (c) We propose a novel algorithm that considers multiple next hop candidate nodes while searching for an optimal packet combination.

Our algorithm to find optimal packet combination relies on enumeration of cycles in a graph. There are several algorithms available in literature to precisely accomplish this task. Tarjan [Tarjan, 1972], Johnson [Johnson, 1975], Tiernan [Tiernan, 1970], Floyd [Floyd, 1967], Liu [Liu and Wang, 2006], Rao [Bapeswara Rao and Murti, 1969] are few of the algorithms that enumerate cycles in a graph. These algorithms differ from each other in terms of their asymptotic time complexity and difficulty of implementation. In our implementation we have used Johnson's algorithm to enumerate cycles, since a survey of literature suggests it to be the fastest among all the algorithms mentioned above.

3.3 Packet encoding algorithm

Our packet encoding algorithm is independent of PCR or Delta routing, and it can be integrated with any routing scheme. In this section we present the problem of finding the optimal packet combination as an integer program, and develop an algorithm to solve this problem.

Say a node has n packets in its queue. We want to find the largest set of packets that can be combined in a single transmission. In a queue of length n there can be $\sum_{k=2}^{n} \binom{n}{k} = 2^n - n - 1$ possible combinations of more than two packets. Therefore a naïve implementation of a thorough search could take at least $O(2^n)$ time. Note that if we are combining k packets, our algorithm has to verify that each next hop that receives this packet combination should have the knowledge of $k-1$ packets, other then the one it is supposed to receive. If a next hop doesn't know the other $k-1$ packets in the combination, it won't be able to decode its

own packet. A key question is how does a node learn which packets in its queue are known to which of its neighbors.

3.3.1 Acquiring the knowledge of neighbors' packets

Pairwise coding: If we are combining only two packets, this task is quite easy. If two flows are travelling in opposite directions as shown in figure 3.1, it is easy to combine packets from these flows. Here node F can transmit the combination $m_a \oplus m_b$ because it knows that A and B will have the knowledge of m_a and m_b respectively, since that's where those packets came from.

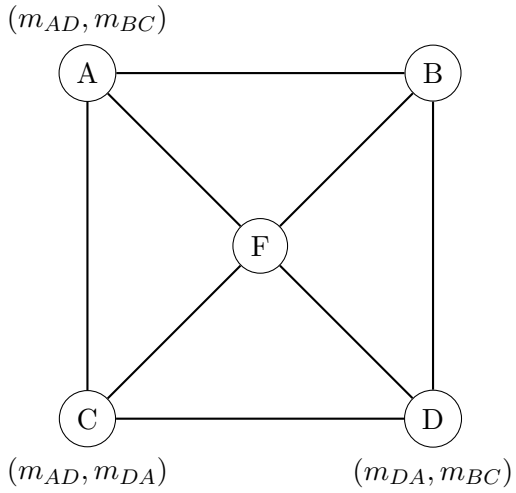


Figure 3.3: Opportunistic listening can be employed to combine more than two packets

Combining more than two packets: If routes are configured properly, we have to rely on opportunistic listening to combine more than two packets. In opportunistic listening a node listens to a transmission and stores a packet even if it is not supposed to receive this packet. Some other node can leverage this knowledge to combine more packets in a transmission. For example, in figure 3.3 say node F's queue has three messages $\{m_{AD}, m_{DA}, m_{BC}\}$ where message m_{ij} is going from node i to j. In this case it is safe for F to assume that nodes

A, C, D have the knowledge of $\{m_{AD}, m_{BC}\}$, $\{m_{AD}, m_{DA}\}$ and $\{m_{DA}, m_{BC}\}$ respectively. For example, in a wireless medium when B forwards m_{BC} to F, A and D cannot forward or receive some other packet at the same time. Hence, if F successfully receives m_{BC}, so do A and D. The same reasoning follows for the messages at node C. Therefore F can combine $m_F = m_{AD} \oplus m_{BC} \oplus m_{DA}$ in a single transmission such that A, C, D will be able to decode their respective messages. For example, at node C, $m_{BC} = m_F \oplus m_{AD} \oplus m_{DA}$. In [Katti *et al.*, 2006] a node maintains probability P_{mn} to indicate the certainty that neighbor n has the knowledge of packet m. For any combination of packets if the product of these probabilities (over all ms and ns) exceeds certain threshold, the node assumes that these packets can be combined together. However, while developing our encoding algorithm, we do not worry about the implementation details. For the sake of simplicity we assume that a node has accurate information regarding which neighbors have the knowledge of which packets.

3.3.2 Benefits of considering multiple next hop candidates

So far we have assumed that each packet in a node's queue is going to be forwarded to a designated next hop neighbor. Prevailing encoding algorithm [Katti *et al.*, 2006] also makes this assumption. However, considering multiple neighbors as next hop candidates for a packet may improve possibility of combining more packets. Consider the scenario presented in figure 3.4, where we have three flows, from C to A, D to A and from A to E. Clearly node B is the bottleneck node. Moreover, let's assume that flows $C \rightarrow A$ and $D \rightarrow A$ have a fixed rate, while flow $A \rightarrow E$ can transmit as many packets as possible as long as it doesn't reduce the rate of the other two flows. Say packets from $A \rightarrow E$ have a fixed path that goes through nodes B and C. In this case, B can code packets from $A \rightarrow E$ only with the packets from $C \rightarrow A$. If B doesn't have a packet from $C \rightarrow A$ in its queue, the packet from $A \rightarrow E$ will have to be transmitted individually, which wastes bandwidth. Note that nodes C and D are equidistant from E. If B considers multiple next hop candidates, i.e. if it also considers D as a next hop for $A \rightarrow E$ packets, we can combine two packets as long

as packets from either $C \rightarrow A$ or $D \rightarrow A$ are in B's queue. This would lead to a higher throughput for $A \rightarrow E$. In this case we have considered only two next hop candidates; the more next hop candidates we have, the higher the complexity of finding the optimal combination.

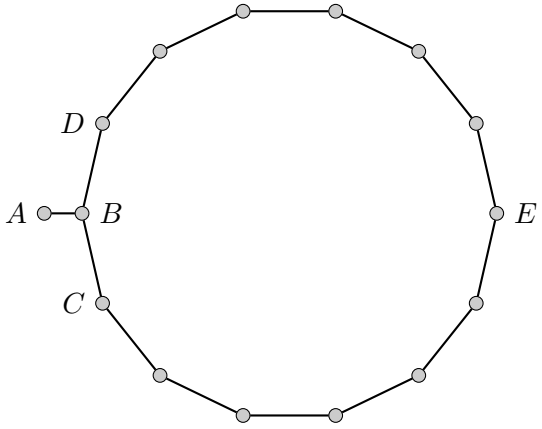


Figure 3.4: Considering multiple next hop candidates may improve coding opportunities

Moreover, certain routing schemes do consider multiple neighbors before forwarding a packet to one of them. For example, if we route packets on the path with the smallest delay or according to the delta metric, a node may consider multiple next hop candidates and choose to forward a packet to the neighbor that yields the smallest value of delay or delta metric. For a given packet, if the next hop chosen according to these metrics doesn't have the knowledge of any other packets from the current node's queue, this packet cannot be combined with any other packets. Therefore, ideally our encoding algorithm should consider multiple next hop candidates (neighbors with the smaller values of delay/delta metric) and select to forward packets to the neighbors that help yield maximum packet combination. By doing so we are meeting both the objectives, we are not only routing packets according to delta/delay metric but we are also improving throughput by combining more packets. Among $O(2^n)$ packet combinations, considering multiple next hop candidates for each packet and verifying that the next hop candidates have the knowledge of $k-1$ other packets

increases the complexity of the problem furthermore. We first represent this problem as an integer program. In subsection 3.3.4 we present a bipartite graph based construction that helps us exhaustively check a node's queue for optimal packet combination. In subsection 3.3.5 we extend our algorithm to consider multiple next hop neighbors.

3.3.3 Integer program to find the maximum possible packet combination

Figure 3.5 presents our integer program to exhaustively search a node's queue to look for an optimal packet combination, while considering multiple next hop neighbors for each packet.

3.3.3.1 Variable definition

Objective:

$$\max \sum_i I_i \tag{3.1}$$

Subject to:

$$\sum_j a_{ij} \leq 1 \qquad \forall i \tag{3.2}$$

$$\sum_i (I_i - b_{ij} I_i) \leq 1 \qquad \forall j \tag{3.3}$$

$$a_{ij} + b_{ij} \leq 1 \qquad \forall i, j \tag{3.4}$$

$$\sum_i \sum_j a_{ij} d_{ij} I_i \geq \sum_i I_i \qquad \forall i, j \tag{3.5}$$

$$a_{i,j}, b_{i,j}, d_{i,j}, I_i \in \{0, 1\} \qquad \forall i, j \tag{3.6}$$

Figure 3.5: Integer program to exhaustively search for an optimal packet combination while considering multiple next hop neighbors for a packet

1. In this integer program let N_i represent the set of next hop candidate neighbors for a packet i. In delta routing we choose a subset of neighbors with the smallest delta

metric as N_i. Similarly different criterion can be chosen for selecting N_i, for example in geographic routing [Kim *et al.*, 2005] N_i can be the set of neighbors that are geographically closer to the destination than the current node.

2. $I_i = 1$ if packet i is included in the combination, otherwise 0.

3. $a_{ij} = 1$ if node j is chosen as packet i's next hop, otherwise 0.

4. $b_{ij} = 1$ if node j has the knowledge of packet i, otherwise 0.

5. $d_{ij} = 1$ if node $j \in N_i$, otherwise 0. $i \in \{1, ..., p\}, j \in \{1, ..., n\}$

3.3.3.2 Explanation of constrains

- Objective function is straightforward.
- Equation 3.2 ensures that each packet has only one next hop.
- Equation 3.3 ensures that if we combine k packets, nodes who are supposed to receive this combination know at least $k - 1$ of them.
- Equation 3.4 checks that the node that is supposed to receive a packet, doesn't already have the knowledge of it. This also prevents a packet from going back to its previous forwarders.
- Equation 3.5 verifies that we have as many next hops as the packets that are combined in a transmission.

All four constraints work together to ensure that each packet has only one next hop, and the node that is supposed to receive a packet has the knowledge of the rest of the packets in that combination. Equation 3.6 makes sure that all the variables are binary.

Hence, for each packet this integer program will take a set of next hop neighbor candidates, and it will figure out which neighbors should the packets go to in order to combine the maximum number of packets in a single transmission.

3.3.4 Exhaustive search: single next hop candidate

Prevailing encoding algorithm assumes that each packet in a queue has a designated next hop neighbor, and it searches for a packet combination sequentially. A sequential search can yield suboptimal results. Consider the scenario presented in table 3.1. First column represents packets in a node's queue. Second column indicates their next hops, and the third column indicates packets known to these next hops. Iterating through this node's queue sequentially suggests that the best packet combination is $P_1 \oplus P_2$, however an exhaustive search would indicate that the optimal packet combination is in fact $P_1 \oplus P_3 \oplus P_4$. Therefore, first we develop an efficient algorithm that searches a node's queue thoroughly for the optimal packet combination assuming each node is assigned a designated next hop neighbor (i.e. $\mid N_i \mid = 1$), and then we extend it to accommodate for multiple neighbors as next hop candidates.

Packets	Next hop	P_i known to next hop
P_1	n_1	P_2, P_3, P_4
P_2	n_2	P_1
P_3	n_3	P_1, P_4
P_4	n_4	P_1, P_2, P_3

Table 3.1: Sequential search:$P_1 \oplus P_2$. Optimal combination: $P_1 \oplus P_3 \oplus P4$.

Construction 1 Let us consider a directed bipartite graph $G(V, E)$. Here $V = V_p \cup V_n$ and $V_p \cap V_n = \emptyset$. V_p represents the set of packets P_i in the queue. V_n is the set of next hop neighbors n_j. A directed edge from P_i to n_j exists in E, if n_j is chosen as P_i's next hop neighbor. Also a directed edge from n_j to P_i exists if neighbor n_j has the knowledge of P_i.

Before we show how this construction can be helpful in finding the best packet combination, let's introduce the definition for *combination rate*. If a node wants to forward k packets to k different neighbors, and if it has to transmit this combination a minimum of m times, we say that the rate for this combination is $\frac{k}{m}$. For example if each of the neighbors have the knowledge of a minimum of l packets out of k, we only have to transmit $m = k - l$

linear combinations of type $c_m = \sum_k \mu_{m,k} P_k$ as suggested in [Ho *et al.*, 2003]. Here, $\mu_{m,k}$ is some scalar from a Galois field. As long as these $m = k - l$ combinations are linearly independent, a neighbor can use the knowledge of its l packets and solve a system of m equations with m unknowns to retrieve the packet it desires (here it is implicitly assumed that the scalars $\mu_{m,k}$ are previously agreed upon by the transmitter and receivers).

Theorem 1. *In the graph construction presented above, if we find a cycle of length* $2k$*, it is possible to achieve a combination rate of* $\frac{k}{k-1}$*.*

Proof. Since it is a bipartite graph, all the cycles will have even lengths. The cycle of length $2k$ will consist of k packets and their k next hops. If a neighbor n_j is supposed to receive a packet P_i, there is no edge from n_j to P_i, since n_j doesn't have the knowledge of P_i. Therefore, the existence of the cycle means that each of k next hops have the knowledge of at least one packet other than the packet it is supposed to receive. Hence the combination rate $\frac{k}{k-1}$ is achievable. □

However, we do not concern ourselves with the packet combinations that require multiple transmissions in order for the next hops to decode their packets. Instead we focus only on the packet combinations where all the packets can be delivered in a single transmission. Therefore, if we are combining k packets in a transmission, each of the next hop neighbors should have the knowledge of $k - 1$ packets other than the one it is supposed to receive. By enumerating cycles in the graph, we have narrowed down our search to the packet combinations that can potentially be delivered in one transmission. Now given a cycle $C \subseteq G$, let us separate its vertices into packets P_i and their next hops n_j. All that remains for us is to ensure that each $P_i \in C$ has an incoming edge from each $n_j \in C$ other than its own next hop neighbor. If we find a cycle that meets this condition, all the packets that are part of this cycle can be delivered in a single transmission. From this observation we can state the following theorem.

Theorem 2. *Given the graph construction above, if we can find* $(k-1)!$ *cycles of length* $2k$*, such that each of these cycles contain the same* k *packets, then these* k *packets can be*

combined in a single transmission.

Proof. From the graph construction presented above it is immediate that if each of these cycles has the same k packets, then they also have the same k next hop neighbors. Also, being able to find $(k-1)!$ such cycles means that each packet has incoming links from $k-1$ next hop neighbors other than its own next hop. In other words each next hop neighbor knows $k-1$ packets other then the one it is supposed to receive. Therefore, these k packets can be combined in a single transmission. □

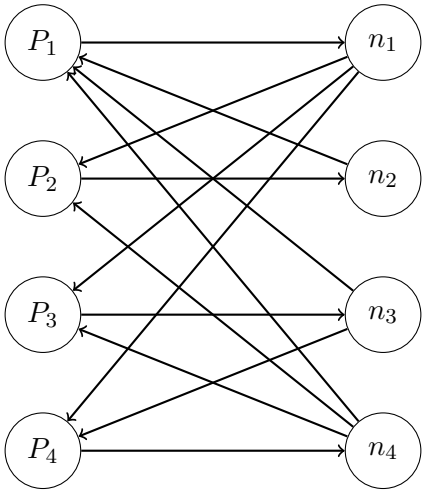


Figure 3.6: Graph construction based on the example given in table 3.1.

Based on the construction discussed above, we can generate a graph corresponding to the example presented in table 3.1. This graph is shown in figure 3.6. Table 3.2 enumerates all the cycles present in this graph. From theorem 2 it becomes obvious that given this queue we can find five different combinations $P_1 \oplus P_2, P_1 \oplus P_3, P_1 \oplus P_4, P_3 \oplus P_4$ and $P_1 \oplus P_3 \oplus P_4$. Clearly $P_1 \oplus P_3 \oplus P_4$ is the best combination as mentioned earlier, since it achieves the highest combination rate.

Packets	Cycles
P_1, P_2	$P_1 \rightarrow n_1 \rightarrow P_2 \rightarrow n_2 \rightarrow P_1$
P_1, P_3	$P_1 \rightarrow n_1 \rightarrow P_3 \rightarrow n_3 \rightarrow P_1$
P_1, P_4	$P_1 \rightarrow n_1 \rightarrow P_4 \rightarrow n_4 \rightarrow P_1$
P_1, P_2, P_4	$P_1 \rightarrow n_1 \rightarrow P_4 \rightarrow n_4 \rightarrow P_2 \rightarrow n_2 \rightarrow P_1$
P_1, P_3, P_4	$P_1 \rightarrow n_1 \rightarrow P_3 \rightarrow n_3 \rightarrow P_4 \rightarrow n_4 \rightarrow P_1$ $P_1 \rightarrow n_1 \rightarrow P_4 \rightarrow n_4 \rightarrow P_3 \rightarrow n_3 \rightarrow P_1$
P_1, P_2, P_3, P_4	$P_1 \rightarrow n_1 \rightarrow P_3 \rightarrow n_3 \rightarrow P_4 \rightarrow n_4 \rightarrow P_2 \rightarrow n_2 \rightarrow P_1$
P_3, P_4	$P_3 \rightarrow n_3 \rightarrow P_4 \rightarrow n_4 \rightarrow P_3$

Table 3.2: Enumeration of cycles in figure 3.6.

3.3.5 Exhaustive search: multiple next hop candidates

Since a scheme like delta routing considers multiple next hop candidates (N_{P_i}), if we are to integrate network coding with it, our encoding algorithm should also consider multiple neighbors as next hop candidates.

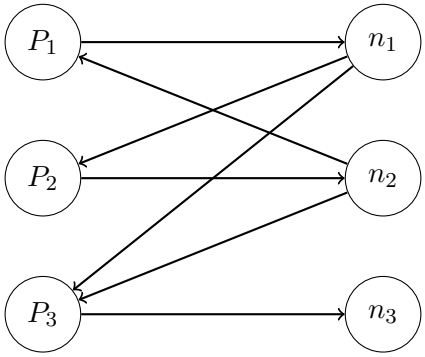


Figure 3.7: Original graph

Considering multiple next hop candidates also increases likelihood of combining more packets. While constructing the bipartite graph according to Construction 1, we assumed that each packet has only one neighbor as the candidate for the next hop. However, if this particular neighbor does not have the knowledge of other packets, it might be difficult to

obtain a higher combination rate. For example, let us consider the graph given in figure 3.7. A quick application of theorem 2 would suggest that the best packet combination for this queue is $P_1 \oplus P_2$. However, let us assume that there exists a neighbor $n_4 \in N_{P_3}$, and it also happens to have the knowledge of P_1 and P_2. Therefore, if we choose to deflect P_3 to n_4 instead of sending it to n_3, we can obtain a better combination in the form of $P_1 \oplus P_2 \oplus P_3$. Hence, considering multiple neighbors as the candidates for the next hop increases the chances of finding a better combination.

Construction 2: We can extend the existing construction of the bipartite graph $G(V, E)$ in the following manner. There exists an edge in E from P_i to n_j if $n_j \in N_{P_i}$. There exists an edge from n_j to P_i, if n_j has the knowledge of P_i. $V_n = \bigcup_{P_i} N_{P_i}$, $V = V_p \cup V_n$ and $V_p \cap V_n = \emptyset$.

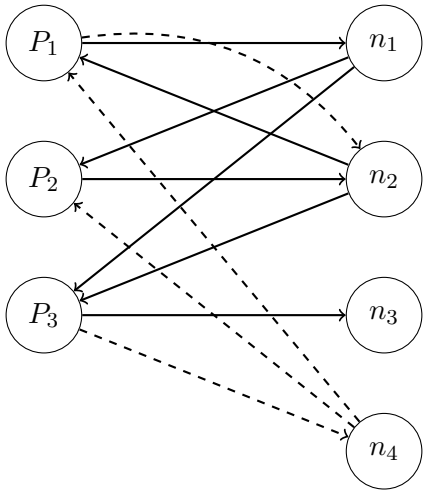


Figure 3.8: Extension graph

Figure 3.8 is an extended version of graph 3.7, assuming $N_{P_1} = \{n_1, n_2\}, N_{P_2} = \{n_2\}, N_{P_3} = \{n_3, n_4\}$. Once we have extended a bipartite graph in this manner, we can use theorem 3 to find the maximum packet combination, which is $P_1 \oplus P_2 \oplus P_3$ in this case.

Theorem 3. *Consider a bipartite graph that accepts multiple neighbors as a packet's next hop candidates. If we can find* $(k-1)!$ *cycles of length* $2k$ *in this graph, such that each of these cycles contain the same* k *packets and each of these* k *packets have the same neighbors*

as the next hops in all $(k-1)!$ *cycles, then these* k *packets can be combined in a single transmission.*

Proof. This theorem is a direct consequence of theorem 2. □

Note that the next hop neighbors for these packets will be the nodes chosen by the cycles. While it is possible that a node can be a *next hop candidate* for multiple packets, it will be the *next hop* for at most one packet, since a node can appear in a cycle only once. Figure 3.9 provides a quick summary of our packet encoding algorithm.

$C : \{\}$ = Best cycle so far

$Q : \{P_i\}$ = Current node's queue

N_{P_i} = Set of next hop candidates for P_i

$\mathbb{P}_{n_i}$ = Set of packets known to node n_i

Directed graph $G = (V, E)$

$V = Q \cup \{\bigcup_{P_i \in Q} N_{P_i}\}$

$E = \{(P_i, n_j) \mid n_j \in N_{P_i}\} \cup \{(n_i, P_j) \mid P_j \in \mathbb{P}_{n_i}\}$

Enumerate cycles in G

If we find $(k-1)!$ cycles $\{c_1, ...c_{(k-1)!}\}$ of length $2k$
such that each of these cycles contain the same k
packets and if each of the k packets have the
same next hop in all $(k-1)!$ cycles and if $2k > \mid C \mid$

$\Rightarrow C = c_1$

Separate P_is and n_js from C and return

Figure 3.9: Summary of packet encoding algorithm

3.3.6 Benefit of considering multiple next hop candidates

To see the throughput benefit of considering multiple next hop candidates, let's once again consider the simple scenario presented in figure 3.4. The rates of flows $C \rightarrow A$ and $D \rightarrow A$

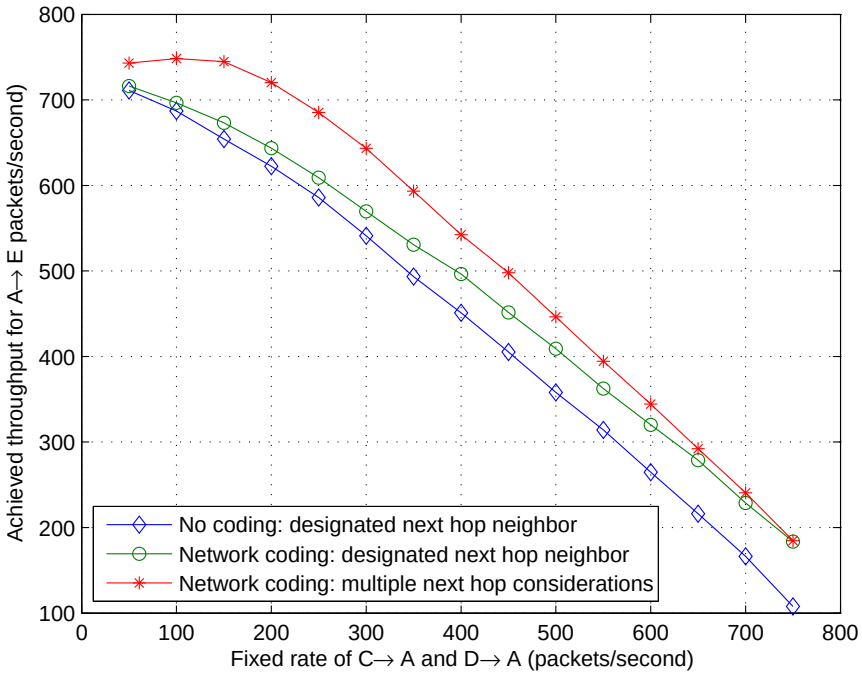


Figure 3.10: Improvement in the throughput of flow $A \rightarrow E$ by considering multiple neighbors as next hop candidates

are fixed, and $A \rightarrow E$ is allowed to transmit as many packets as possible as long as it doesn't hurt the rates of the other two flows. We run a simulation on our custom built discrete event simulator that allows for coding packets. The channel access scheme used is 802.11b with RTS/CTS disabled. A slot length is $20\mu S$. Each packet is $100\mu S$ long ($\approx 1.2KB$ on $90Mbps$ transmission rate). A node tries to retransmit a packet if a transmission fails. Each node's transmission and interference range is $250m$. Each simulation is run for 110 seconds. Throughput of a flow is measured by calculating the number of packets that leave the network during the last 100 seconds. We compare three routing schemes here (1) conventional shortest path routing with a designated next hop neighbor for each packet at any node (shortest path is chosen according to Dijkstra's algorithm, when node B has a packet of type $A \rightarrow E$ it is forwarded to C), (2) network coding applied to conventional

shortest path routing (3) network coding with multiple next hop considerations (i.e. node B considers C and D both as next hop candidates for the packets of type $A \rightarrow E$). In figure 3.10 x-axis shows the fixed Poisson arrival rate for packets of flows $C \rightarrow A$ and $D \rightarrow A$, and y-axis shows the maximum throughput achieved by flow $A \rightarrow E$, without reducing the throughput of any other flows. As the figure indicates considering multiple neighbors always achieves a higher throughput for $A \rightarrow E$ compared to the other two schemes. It should be noted that this particular scenario will only allow combining at most two packets, hence the results of a greedy search and an exhaustive search are not too different. Therefore, the throughput improvement is obtained mainly by considering multiple next hop candidates.

3.3.7 A note on the complexity of the algorithm

Based on the discussion so far, the complexity of the algorithm to find the best possible packet combination depends on the algorithm we use to enumerate cycles in a graph. Fortunately enumeration of cycles in a directed graph is a well studied problem. Algorithms presented by Tarjan [Tarjan, 1972], Johnson [Johnson, 1975], Tiernan [Tiernan, 1970], Floyd [Floyd, 1967], Liu [Liu and Wang, 2006], Rao [Bapeswara Rao and Murti, 1969] are few of the algorithms among many that can accomplish this task.

In our simulations we have utilized Johnson's algorithm [Johnson, 1975] to enumerate cycles in a graph, since it performs better than other known algorithms. Its time complexity is $O((|V| + |E|)(c + 1))$. Here c is the number of cycles in the graph. Since our algorithm runs in polynomial time it is significantly faster than a naïve exhaustive search which runs in $O(2^n)$ time.

Note that even if we extend the graph according to Construction 2, the asymptotic complexity of the algorithm still remains $O((|V| + |E|)(c + 1))$. However, since extending a graph adds more edges to it, complexity of the algorithm also increases. Hence one may have to consider the trade off between the speed of the algorithm on an extended graph and the ability to find a better packet combination. If a new next hop candidate already belongs to the original V_n, we are adding only an edge to the graph ($P_1 \rightarrow n_2$ in graph

3.7 (b)). On the other hand, if we introduce a new next hop candidate that is not part of the original V_n, we will most likely end up adding multiple edges to the graph (three edges introduced by adding n_4 to graph 3.7 (b)).

3.4 Delta Routing and its Integration with Network Coding

As mentioned earlier, this packet encoding algorithm doesn't depend on a particular routing scheme such as PCR. It can be integrated with any routing mechanism such that we can reap the benefits offered by the routing scheme and network coding both. To demonstrate this, we integrate our packet encoding algorithm with Delta routing. In Delta routing a packet's next hop neighbor changes dynamically according to a delta metric. Hence it is difficult to identify packets that can be combined together. We demonstrate that using our encoding algorithm we can manage to route packets using the delta metric and yet achieve a higher coding gain.

As mentioned in section 3.1, like any other scheme, throughput benefit offered due to network coding can also be hampered by bottleneck nodes. Delta routing helps packets circumnavigate congested parts of the network. By routing packets through underutilized nodes and thus promoting spatial reuse, it improves throughput of the network. In this section we first present the concept of Delta routing and then integrate it with network coding.

The idea of Delta routing can be traced back to [Rudin, 1976]. It can be seen as a derivation of the concept of routing packets on the smallest delay path. [Rudin, 1976] explains Delta routing as follows. In a wireline network consider a node s that has two links going to nodes n_1 and n_2. s maintains two separate queues for both the outgoing links, let q_{s,n_1} and q_{s,n_2} be the length of these queues. Moreover let $D'_{n_1,d}$ and $D'_{n_2,d}$ be the delay from n_1, n_2 to the destination, measured in terms of packet lengths. Without loss of generality assume that $D'_{n_2,d} > D'_{n_1,d}$. Let's define $\delta = D'_{n_2,d} - D'_{n_1,d}$. When s receives a

new packet, its next hop is assigned according to equation 3.7.

$$\text{Next hop} = \begin{cases} n_2 & \text{if } q_{s,n_2} - q_{s,n_1} < \delta \\ n_1 & \text{otherwise} \end{cases} \tag{3.7}$$

The idea here is to use local information to compare two outgoing links' queues with respect to delay to the destination. If the local queue for one link is too long, conclude that the node connected to this link is congested and send the packet elsewhere.

We make modification to this idea in order to make it more suitable for a wireless network with network coding. In a wireless network there are no separate links, a node transmits all its packets on the same channel. Hence a node maintains just one queue. Say a new packet arrives at s, and there are two nodes n_1 and n_2 that can take the packet closer to its destination. Once a packet joins s's queue, time to transmit this packet to n_1 or n_2 is the same, since there is just one queue. Hence we may not have enough local information to estimate congestion at the neighboring nodes. Fortunately in a wireless medium a node can judge congestion at its neighboring nodes if they broadcast their queue length periodically or attach them with their data packets. Once a packet is transmitted its delay to the destination depends on the queuing delay at the next hop, delay from the next hop to the destination, channel access delay per packet (which are greater at a congested node) and the transmission rate. Therefore in the modified version of Delta routing, if a node has multiple neighbors, it chooses a next hop according to equation 3.8.

$$\text{Next hop} = \underset{n_i}{\text{argmin}}\{D_{n_i,d} + \frac{Q_{n_i}}{T_{n_i} P_{n_i}}\} \tag{3.8}$$

In equation 3.8, $D_{n_i,d}$ is the average time between n_i transmitting a packet until it reaches d. This delay is measured in time units. Q_{n_i} is the current queue length at neighbor n_i. T_{n_i} is the average transmission rate and P_{n_i} is the probability of a successful transmission. The packet is forwarded to a neighbor that has the smallest value for the current queuing delay plus the average delay to the destination. In a wireless scenario different nodes can experience different probability for a successful transmission. Clearly higher the

probability of a successful transmission and higher the transmission rate, smaller the queuing delay. In equation 3.8 we only consider neighbors that are geographically closer to the destination than the current node. This rule helps prevent routing loops while forwarding the packets.

Average delay ($D_{i,j}$) between two nodes can be easily propagated through the network using Bellman-Ford algorithm. In order to judge the quality of the channel a lot of times wireless nodes maintain ETX metric [De Couto *et al.*, 2003] which periodically calculates the probability of a successful transmission to individual nodes. Hence one way to calculate the probability of a successful transmission (P) is to take the weighted average of delivery probabilities calculated by ETX, where the weights are the fraction of packets that travel to a particular neighbor. A straightforward way to calculate P is to observe a window of last k transmissions, and then take the ratio of successful transmissions over total transmissions (k). We use the latter method to calculate P in our simulations. The values for transmission rate T and the queue length Q are readily available at each node. As the values of D, Q, T, P change, a node can either periodically broadcast them to its neighbors or attach these values along with data packets. Therefore in this routing mechanism a node only requires information available from its immediate neighbors to make forwarding decisions.

3.4.1 Throughput Comparison: Delta Routing Vs. Conventional Shortest Path Routing

In this section we compare the throughput offered by conventional shortest path routing and Delta routing. For both the routing schemes we insert packets into the network, and observe if the packets can successfully leave the network without building up queues at the nodes.

If $Q_i(t)$ represents node i's queue length at time t, we define that a network is in stable state as long as the following condition is satisfied: $\limsup_{t\to\infty} Q_i(t) < \infty, \forall t$. For any arrival rate of new packets, if this condition is violated, the queues start to build up at the network nodes, packets leave the network at a rate slower than the arrival rate and

the delay approaches infinity. In our simulation packets enter the network according to Poisson process with arrival rate λ. Once a packet enters the network, it chooses its source and destination uniformly. (We choose this traffic pattern instead of inserting flows in the network, since it creates diversity of packets in terms of next hops in a node's queue. This makes the task of identifying optimal packet combinations computationally challenging. Therefore this traffic pattern will help us test our packet encoding algorithm thoroughly.) We gradually keep increasing λ until the queues start building towards infinity. Network throughput is defined as the maximum arrival rate λ for which the network remains in the stable state (i.e. maximum packet arrival rate for which departure rate equals the arrival rate).

Using this method we test the throughput offered by both the routing schemes on the network shown in figure 3.11. In this network 75 nodes are scattered uniformly on a $1000 \times 1000 m^2$ field. We run the simulation for 110 seconds, network throughput is measured using the number of packets that leave the network in last 100 seconds. It is assumed that each node can sustain a very large queue. Information regarding channel access scheme, packet lengths, transmission/interference range are the same as provided in section 3.3.6.

For Delta routing $D_{s,d}$ is measured by averaging the travel times of last 25 packets between s and d. Similarly, probability of successful transmission P is measured by observing the success rate of last 25 transmissions. In order to measure the maximum possible throughput gain, we assume the ideal scenario where the values of D, P, Q, T are made readily available at each node during the course of the simulation.

Figure 3.12 plots departure rate vs. arrival rate for both the routing schemes. The figure indicates that the conventional routing offers a throughput of 2800 packets/second, whereas the modified Delta routing offers a throughput of 4000 packets/second. Thus, Delta routing improves the throughput of this particular network by almost 43%.

Contour graphs are helpful in demonstrating the distribution of a variable over a field, for example mountain elevations over an area. We can use contour maps to show the distribution of forwarding rates in a network as well (i.e. forwarding rates are analogous to

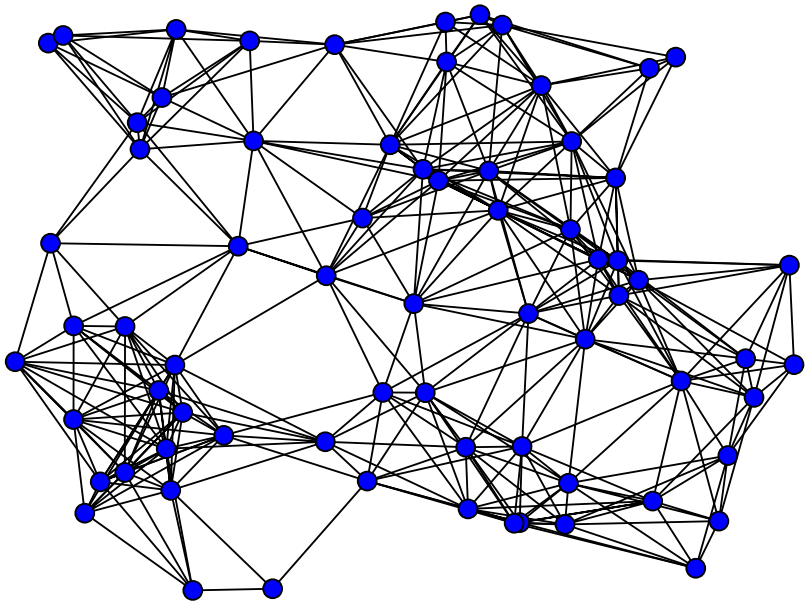


Figure 3.11: Network specification: 75 nodes scattered uniformly in an area of $1000 \times 1000 m^2$

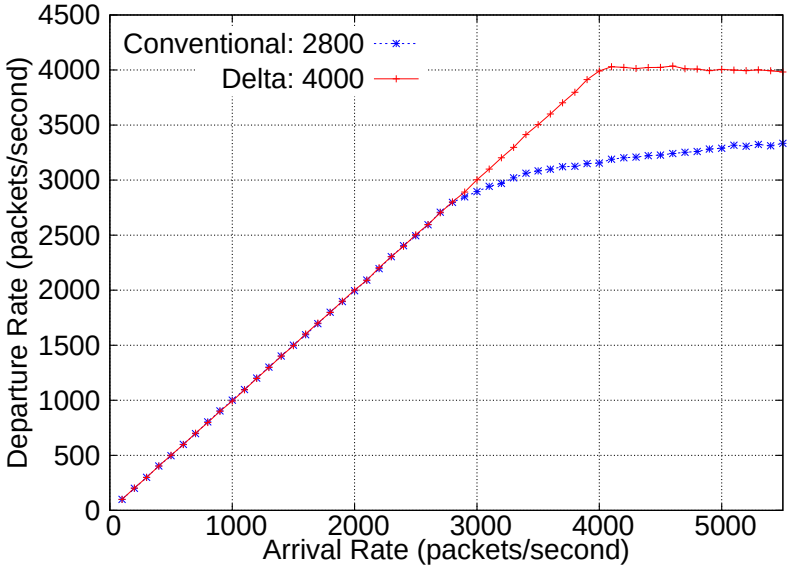


Figure 3.12: Throughput: Conventional shortest path routing: 2800 packets/sec, Delta routing: 4000 packets/sec

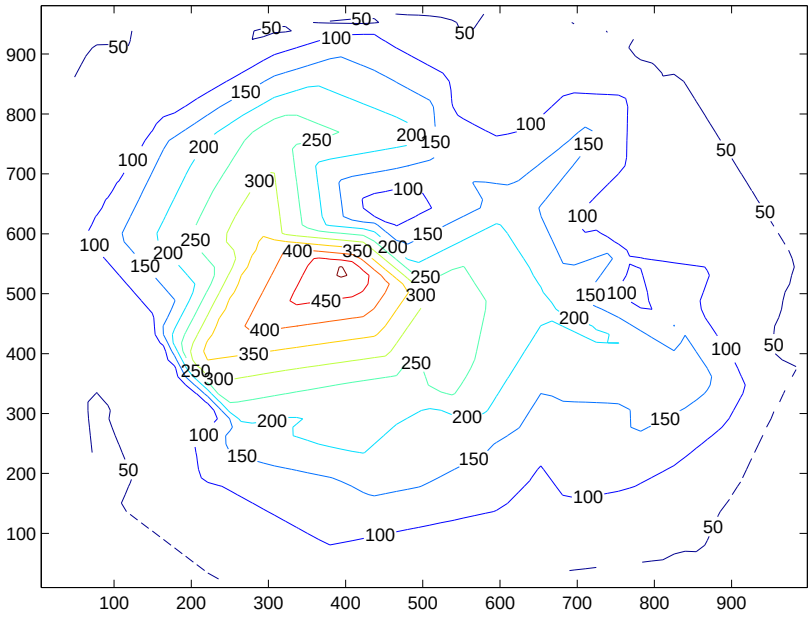


Figure 3.13: Contour graph of forwarding rates for conventional routing, arrival rate = 2800 packets/sec

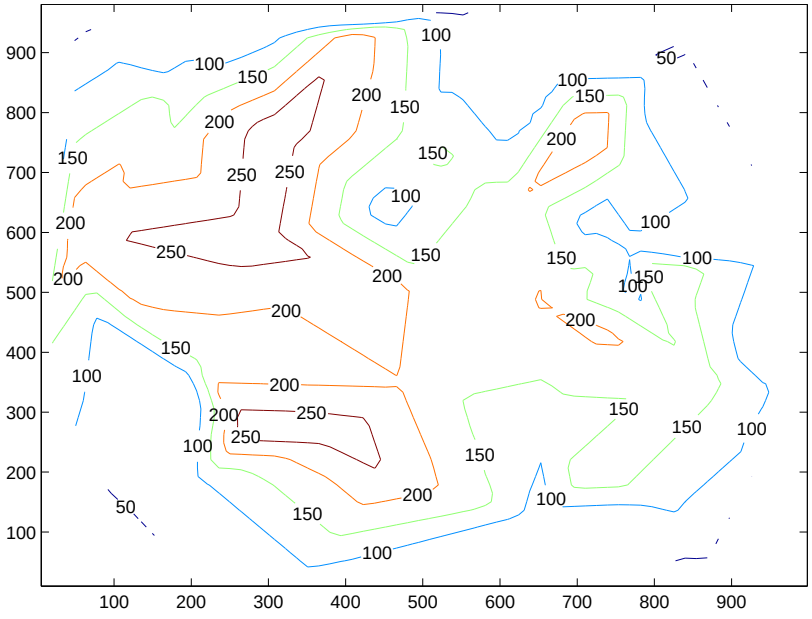


Figure 3.14: Contour graph of forwarding rates for Delta routing, arrival rate = 2800 packets/sec

elevations). When the contours are closer to each other, the magnitude of the gradient is high, hence the variation in forwarding rates is steep. Therefore that part of the network is congested. On the other hand, if the contours are spread far apart, it implies that the traffic is distributed evenly in that part of the network. Figures 3.13 and 3.14 plot the forwarding rates (packets/second) at various nodes in the contour representation for both the routing schemes when arrival rate = 2800 packets/second. From figure 3.13 it is obvious that in conventional routing majority of the traffic is forwarded through the center of the network, while the remaining parts of the network are largely underutilized. Hence, the congestion at the center is throttling the throughput of the network. Figure 3.14 demonstrates that Delta routing has alleviated this congestion at the center by directing the traffic towards the underutilized nodes. These results demonstrate that the network throughput can be substantially improved by forwarding packets according to Delta routing.

3.4.2 Integrating network coding with Delta routing

In this subsection we couple network coding with Delta routing. As mentioned in sections 3.1 and 3.3, for a given packet, if the next hop chosen according to equation 3.8 doesn't have the knowledge of any other packets from the current node's queue, that packet cannot be combined with any other packet. We can improve the possibility of combining more packets by choosing a few next hop candidates with the lower values for delta metric, instead of choosing just a single next hop with the least delta metric value. By doing so we are leveraging benefits of both the schemes, we are not only routing packets towards underutilize nodes, we are improving the possibility of combining more packets as well.

To apply network coding to Delta routing we make a small change to delta metric. We replace the probability of successful transmission in equation 3.7 with C_{n_i}. C_{n_i} is the rate with which n_i successfully delivers packets to its neighbors. In order to measure C_{n_i} we observe a window of last k transmissions. If a node combines $2, 1, 3, 2, 1$ packets in its last five transmissions, but if only $1, 0, 2, 2, 1$ packets are delivered successfully, C_{n_i} for this node

is 1.2. The new delta metric can be written as in equation 3.9.

$$\text{Next hop} = \underset{n_i}{\operatorname{argmin}}\{D_{n_i,d} + \frac{Q_{n_i}}{T_{n_i}C_{n_i}}\} \tag{3.9}$$

In order to couple Delta routing with network coding, we directly build a bipartite graph according to Construction 2. A packet P_i's next hop candidate set (N_{P_i}) is chosen as neighbors with the lower values for the delta metric (equation 3.9). How many neighbors should be included in N_{P_i} is left to user's discretion and it would depend on how much computational power does the user have. Larger the size of N_{P_i}, denser the graph and longer it takes to enumerate cycles.

Should we find two different packet combinations that have the same number of packets in it, we would want to give priority to the combination that will be forwarded to the underutilize nodes. This can be done by adding weights to the edges in the graph. An edge that goes from a packet to a node is assigned a weight $w = D_{n_i,d_{p_i}} + \frac{Q_{n_i}}{T_{n_i}C_{n_i}}$. Here, d_{p_i} is the final destination for the packet P_i. Also, every edge that goes from a node to a packet is assigned a weight of zero. If we happen to find two different combinations that include the same number of packets, we sum the edge weights of the cycles for these combinations. Note that all $(k-1)!$ cycles for a combination of k packets will have the same total edge weight. We choose to transmit the combination that has the lower total edge weight, since the packets in this combination will be forwarded to the underutilized nodes of the network.

In case a node's queue doesn't have any packets that can be combined together, the node would forward the packet from the top of its queue to the neighbor chosen by equation 3.9.

Even though the algorithm we presented is asymptotically faster, if a node's queue is too large, it can still take significant amount of time to find the optimal packet combination. Therefore it is not ideal to use the entire queue to find the best packet combination. We list few methods that can help improve the speed of the algorithm.

• If a node has a very large queue, the node can use only first k of its packets to build the bipartite graph $G(V_p \cup V_n, E)$. In our simulations we have constructed G by using only first ten packets from a node's queue.

• Even if we build a graph G using a partial queue, if the cardinality of N_{P_i} is too high, the graph can still be very dense and depending on the speed of the processor enumerating cycles can still be time consuming. We can reduce the number of edges in a graph by controlling the cardinality of N_{P_i}. In our simulations we have chosen $|N_{P_i}| \leq 3$.

Advantages of this encoding algorithm are not limited to Delta routing. By replacing the edge weights in our algorithm with the metric a routing scheme uses, this algorithm can be applied to any routing scheme. For example, if we choose to route packets on the smallest delay path, we can replace the edge weights with delays to the destination, and find the optimal packet combination.

3.5 Simulation Results

In this section we check the performance of our routing scheme where we couple delta routing with network coding. We compare its performance with network coding on the conventional shortest path routing (XOR) as presented in [Katti *et al.*, 2006]. Note that this scheme also uses our improved search procedure for combining packets. The only difference is that every packet in this routing scheme will have a designated next hop neighbor chosen according to Dijkstra's algorithm that selects the shortest path to the destination in terms of hop count. The details of the simulation are provided in section 3.4.1. In order to evaluate the best case performance for both the routing schemes, we have assumed that the information needed from the neighbors to make coding and forwarding decisions are readily available at every node.

Figure 3.15 indicates that combining delta routing and network coding offers a throughput of 5100 packets/second on the network shown in figure 3.11. On the other hand, network coding on conventional shortest path routing offers a throughput of only 3600 packets/second. Hence our routing mechanism outperforms pure coding by 41.67%. This throughput benefit is 82.14% improvement over conventional shortest path routing (shown in figure 3.12).

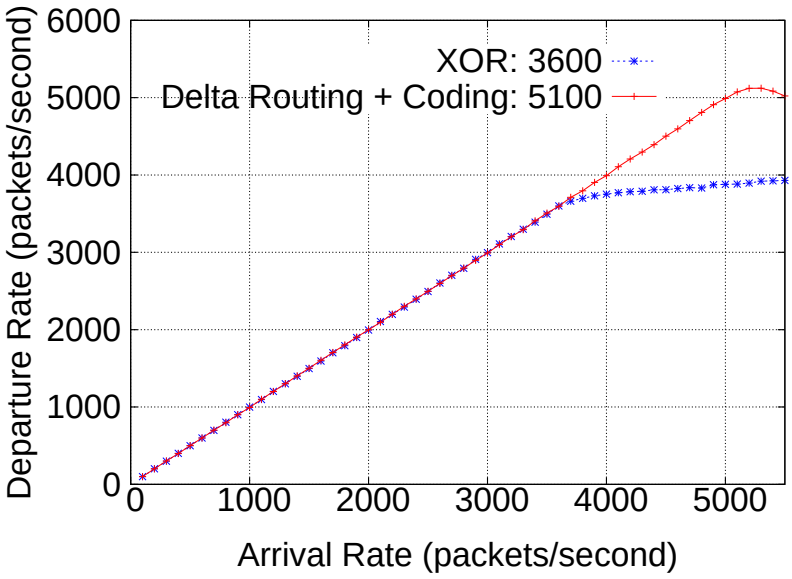


Figure 3.15: Network coding + Conventional routing: 3600 packets/sec, Delta routing + Network Coding: 5100 packets/sec

We run these routing schemes on a total of 40 random networks (according to the specifications provided in subsection 3.4.1). Throughput achieved by various routing schemes on these networks are demonstrated in figure 3.16. For all the topologies the combination of delta routing with network coding significantly outperforms all the other routing schemes.

Table 3.3 summaries the throughput improvement achieved by the combination of network coding and delta routing over other schemes, based on figure 3.16. The combination of delta routing and network coding improves the throughput of a conventional shortest path routing scheme by as much as 92.59%. While pure network coding (XOR) offers better throughput than conventional shortest path routing (figure 3.16), the combination of delta routing and network coding outperforms pure coding by an average of 33.34%.

As the arrival rate of new packets increases, the queues start to build up at the network nodes. As mentioned in section 3.4.2, when the queues are very large, we cannot use the entire queue to search for optimal packet combination without slowing the algorithm down. However, the throughput can also suffer if we limit our search to only few packets. Figure

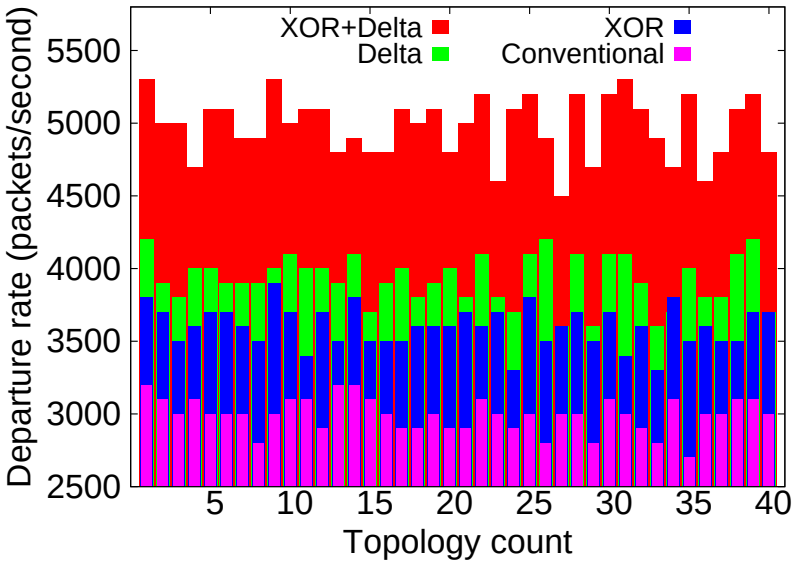


Figure 3.16: Throughput achieved by various routing schemes on different topologies

Routing Scheme	Maximum improvement	Average improvement	Minimum improvement
Conventional Shortest Path	92.59	66.44	50.00
XOR	55.88	33.34	23.68
Delta	37.83	26.91	16.67

Table 3.3: Throughput improvement (in %) achieved by the combination of network coding and delta routing over other schemes.

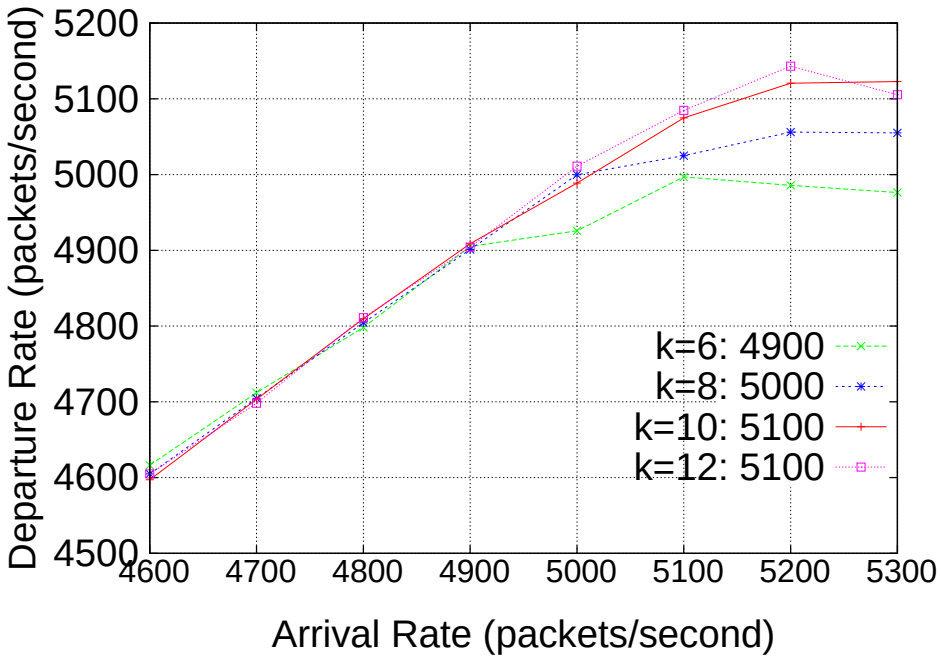


Figure 3.17: Trade off between the throughput and how deep we search our queue

3.17 studies the trade off between the throughput and how deep we search the queues to look for a packet combination. Figure 3.17 shows network throughput when we include 6, 8, 10 and 12 packets respectively in our algorithm. Naturally, the more packets we include in our search, the higher the coding gain, and higher the throughput. However, as this result indicates, after having included sufficient number of packets ($k = 10$ in this case), the coding benefit of adding further more packets starts to diminish. Therefore the routers lacking enough computational resources can also obtain a higher throughput without including all the packets in the search algorithm for the maximum packet combination.

3.6 Summary

In this chapter we proposed a novel packet encoding algorithm for network coding. This algorithm runs in polynomial time and searches a node's queue exhaustively for an optimal packet combination. This algorithm also considers multiple next hop candidates for a

packet, which increases the likelihood of combining more packets. This ability also makes it easy to integrate this algorithm with routing schemes that consider several next hop candidate nodes before forwarding a packet to one of them. We coupled this encoding algorithm with delta routing. Delta routing helps packets in a combination circumnavigate congested nodes, and improves the throughput of the network by forwarding them through the underutilized parts of the network. Our simulations showed that integration of our encoding algorithm with delta routing outperformed conventional shortest path routing by 50.00% to 92.59%. This throughput benefit was 23.68% to 55.88% improvement over utilizing coding on the shortest path routing.

Chapter 4

Max-min Fair Rate Allocation in Multihop Wireless Networks with Intersession Network Coding

4.1 Introduction

In this chapter we present an algorithm to assign max-min throughput fair rates to the flows in a wireless network that uses coding.

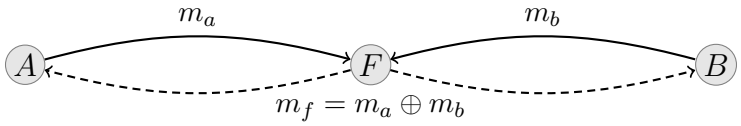


Figure 4.1: An example of network coding

Our focus is on a coding mechanism presented in [Katti *et al.*, 2006]. Consider the topology shown in figure 4.1. Say nodes A and B want to exchange messages m_a and m_b with each other. Moreover, since these two nodes are situated far apart from each other, the messages have to be forwarded through a node F. If F were to relay these

messages separately, it will require a total of four transmissions for A and B to exchange these messages. However, instead of forwarding these messages individually, say F combines them using bitwise exclusive-or and transmits the combination $m_f = m_a \oplus m_b$. In a wireless medium both A and B will receive this combination. Since A already has the knowledge of m_a, it will retrieve its desired message m_b as $m_b = m_f \oplus m_a$. Similarly, B will recover m_a as $m_a = m_f \oplus m_b$. Therefore, the coding scheme improves the throughput by reducing the number of transmissions from four to three. Similarly, in a large network say a node n has k messages going to k different neighbors. If each of these neighbors have the knowledge of $k-1$ messages other than the message they are supposed to receive, n can forward these k messages in a single transmission. Due to its simplicity, this coding scheme has garnered significant research attention. We present an algorithm to assign max-min throughput fair rates to the flows when a network utilizes this coding scheme.

In a network, fairness can be defined over different commodities such as time, throughput etc. In throughput fairness, the objective is to assign various network flows a rate according to some fairness criterion. Fairness can be defined in several different ways.

- **Proportional Fairness:** In proportional fairness a flow is assigned a rate that is generally inversely proportional to the amount of network resources it consumes. Naturally in proportional fairness a flow that travels large number of hops, suffers in throughput.

- **Max-min Fairness:** Let F be the set of flows in a network. Let R be the set of all the achievable rate vectors $\{r_f\}$. A feasible rate vector $r^* \in R$ is max-min fair if for each $f \in F$, r_f^* cannot be increased while maintaining the feasibility without decreasing $r_{f'}^*$ for some flow f' for which $r_{f'}^* < r_f^*$ [Bertsekas and Gallager, 1992]. Max-min fairness is of interest in the objectives such as rate allocation and quality of service maintenance, since it attempts to improve the rates of the flows who are achieving the least rates. In max-min fairness a flow that travels more hops doesn't necessarily suffer in throughput.

In figure 4.1 since the flows are symmetrical, both proportional and max-min fairness

criterion would yield the same rates for both the flows. If the network is using a CSMA based channel access scheme such as 802.11, in figure 4.1 only one transmission can be scheduled at a time. Therefore, if there was no coding in the network, a fair rate for the flows would be 0.25 packets per unit transmission time. On the other hand, if the network was utilizing coding, a fair rate would be 0.33 packets per unit transmission time. Clearly network coding changes the fair rates of the flows in a network. Hence we must make enhancement to the existing algorithms for fair rate allocation in order to accommodate for network coding.

[Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009] extend the seminal work of Kelly et al. [Kelly *et al.*, 1998] and provide a rate control algorithm for the flows in a wireless network with coding. The rates assigned using this algorithm are proportionally fair. To the best of our knowledge no algorithm has been proposed that assigns max-min fair rates to flows in a wireless network with coding.

We propose an algorithm that emulates progressive filling [Bertsekas and Gallager, 1992], [Jaffe, 1981] to achieve max-min fairness. From a global perspective, in progressive filling rates of all the flows are increased gradually by an equal amount, until we identify a flow whose rate cannot be increased further. We fix the rate of this bottleneck flow, allocate the remaining channel capacity to the rest of the unconstrained flows and repeat the procedure until all the flows are assigned a fixed rate. The resulting rates for the flows are max-min throughput fair rates.

However, application of such an algorithm in a wireless network with coding is not straightforward. We often run into scenarios where the direct application of this algorithm may yield incorrect or suboptimal results. In order to use progressive filling, we couple a conflict graph based framework with a linear program to directly identify the bottleneck flows in the network. We demonstrate the caveats in setting up the constraints of the linear program such that the resulting rates will be feasible. We also present conditions for selecting bottleneck flows such that the flows are not assigned suboptimal rates.

Our fairness algorithm can be easily implemented in a decentralized manner. We provide

details of a distributed algorithm that helps flows achieve the same rates as the centralized algorithm. Using various comparison metrics (e.g. Jain's index), we also compare the rates achieved by our algorithm with the existing rate control algorithm.

We provide a scheduling mechanism, that helps flows realize rates allocated according to our fairness mechanism. We avoid introducing new scheduling protocols by implementing our scheduling mechanism using existing 802.11 standards. We simulate the behaviour of this scheduling mechanism on different topologies and present the throughput achieved by various flows in the network.

4.2 Related Work

The seminal work presented in [Ahlswede *et al.*, 2000] showed that the capacity of a network can be improved by combining packets. Since then there was a large body of theoretical work that looked into different aspects of network coding. Among some of the notable works is [yen Robert Li *et al.*, 2003], which showed that in a multicast network linear codes can achieve the capacity defined by the max-flow bound. In [Koetter *et al.*, 2003] extends the work of [yen Robert Li *et al.*, 2003] to any arbitrary network, they also present a polynomial time packet encoding and decoding algorithm for linear codes. [Ho *et al.*, 2003] proposes a robust approach to coding called random coding, where the probability failing to decode the packet decreases exponentially with the codeword length. [Li and Li, 2004] investigates how network coding can lead to throughput improvement in a network with unicast sessions. [Chou *et al.*, 2003] presents a distributed coding mechanism that works with random delays, packet losses and varying channel capacities. [Wu *et al.*, 2005] discusses simple network coding for wireless networks with the scenarios similar to figure 4.1. [Katti *et al.*, 2006] presents an implementation (COPE) of such a coding scheme for a wireless network. Since then the interest in network coding and routing has been reinvigorated. [Sengupta *et al.*, 2007] presents a method to calculate the maximum throughput that can be achieved in a network that uses a method like COPE. [Zhao and Medard, 2010] shows that the local fairness enforced by the MAC scheme plays an important role in improving

the performance of COPE. [Le and Lui, 2008] gives an upper bound on the number of packets that can be coded using such a coding scheme. [Seferoglu *et al.*, 2011] couples inter-session network coding with intra-session coding to make the scheme more resilient to packet losses.

[Bertsekas and Gallager, 1992] offers a progressive filling algorithm to assign max-min fair rates to the flows in a network. However, their solution is limited to the wireline networks. We extend this progressive filling algorithm to wireless networks that use network coding. [Nandagopal *et al.*, 2000], [Huang and Bensaou, 2001] offer a conflict graph based framework to achieve max-min fairness for the flows in wireless networks. However, their work is limited to the networks with single hop flows. [Gambiroza *et al.*, 2004] also offers a conflict graph based solution to assign max-min fair rates to the multihop flows in backhaul networks. However, their solution is limited to smaller networks where only one transmission can be scheduled at a time. Apart from these [Sridharan and Krishnamachari, 2007], [Rangwala *et al.*, 2006] and [Dong *et al.*, 2006] work on a routing tree based framework to assign max-min fair rates to the flows in a network. However, since it is relatively easier to adapt a conflict graph based framework for a wireless network with coding, we work with such a framework instead of working with a routing tree based model.

There has been significant work done on assigning proportional fair rates to the network flows as well. The most prominent of them is [Kelly *et al.*, 1998]. They discuss assigning network flows rates that maximize a concave utility function of flow rates. It has been shown that rates assigned using this scheme are proportionally fair rates. [Lin and Shroff, 2004] present a rate control algorithm for multihop wireless networks. [Ronasi *et al.*, 2009], [Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009] extend these algorithms to assign proportionally fair rates to the flows in a wireless network with coding. However, to the best of our knowledge there hasn't been any work that discusses max-min fair rates for the flows in a wireless network with a coding scheme such as COPE.

4.3 Fairness algorithm from a global perspective

In this section we present our fairness algorithm from a global perspective. As mentioned in section 4.1, our algorithm attempts to emulate progressive filling in order to achieve max-min fairness. In progressive filling one gradually increases rates of all the flows by an equal amount until a flow is identified whose rate cannot be increased any more. This flow is the bottleneck flow. We fix the rate of this flow, allocate the remaining channel capacity to the rest of the unconstrained flows and repeat the procedure until all the flows have a fixed rate. Since progressive filling maximizes the minimum rate a flow can achieve, it achieves max-min fairness. However in a wireless network that utilizes network coding, the task of identifying a bottleneck flow is not straightforward. A direct application of such an algorithm may yield incorrect or suboptimal results. In this section we first define a network model that helps identify the sets of conflicting transmissions in a network. We also present a linear program that helps us calculate fair rates for the flows that are part of each set of conflicting transmissions. We list the caveats in setting up the constraints for this linear program such that the flows will be assigned feasible rates. The fair rates assigned according to this linear program will help us identify the bottleneck flows. We will also present conditions for selecting bottleneck flows such that their rates are not suboptimal.

4.3.1 Network Model

We represent a network by a graph $G(V, E)$. Here V is the set of nodes/routers $\{n_i\}$ in the network. An edge $(n_i, n_j) \in E$ if nodes n_i and n_j can forward messages to each other. Each edge (n_i, n_j) is assigned a capacity c_{ij}. This capacity is usually inversely proportional to a power of the distance between two nodes.

Let r_i^j represent a transmission by some node n_i. Index j in r_i^j is the set of flows whose packets the transmission is forwarding. For example if node n_i is combining packets from flows f_k and f_l in a single transmission, this transmission would be represented by $r_i^{\{k,l\}}$. In figure 4.1, let's represent the flow from A to B as f_a and the flow from B to A as f_b. In this case the network has five transmissions $r_A^{\{a\}}, r_F^{\{a\}}, r_B^{\{b\}}, r_F^{\{b\}}$ and $r_F^{\{a,b\}}$. Each transmission

is assigned a corresponding capacity C_i^j. If a transmission is combining multiple packets, this capacity is the minimum capacity of the edges on which this packet combination is forwarded. For example in figure 4.1, $C_F^{\{a\}} = c_{FA}$, while $C_F^{\{a,b\}} = min(c_{FA}, c_{FB})$. Also let t_i^j be the channel access time allocated to the transmission r_i^j.

From these definitions we can construct a conflict graph $G_c(V_c, E_c)$. Here V_c is the set of all the possible transmissions in the network. An edge $(r_i^j, r_k^l) \in E_c$ if r_i^j and r_k^l cannot be scheduled at the same time. A clique in a graph is a subset of vertices where every two vertices are connected to each other. A maximal clique is a clique that cannot be extended by including any more vertices to it. Therefore a maximal clique in the conflict graph G_c is the set of transmissions that cannot be scheduled at the same time. Since only one transmission in a maximal clique can be scheduled at a time, in order to meet the objective of rate fairness, we must allocate channel access times to the transmissions in a clique such that their flows will have fair rates. Next we define a linear program that helps determine the fair rates for the flows in a maximal clique.

4.3.2 Linear Program to Calculate Fair Rates in a Maximal Clique

In this subsection we present a linear program to assign fair rates to the flows that are part of a maximal clique. This linear program is run on all the maximal cliques in G_c. Naturally, the clique that assigns the smallest rate to its flows is identified as the bottleneck clique. A bottleneck clique can have multiple flows going through it. We also demonstrate which flow in the bottleneck clique serves as the bottleneck flow and we fix its rate.

4.3.2.1 Variable Definition

Let $\mathbb{F} = \{f_l\}$ be the set of all the flows in the network. Furthermore, we split $\mathbb{F}$ into two disjoint sets $\mathbb{F}^c$ and $\mathbb{F}^u$. Here $\mathbb{F}^c$ is the set of constrained flows, whose rates have been fixed by our algorithm. $\mathbb{F}^u$ is the set of unconstrained flows in the network. Let $\mathbb{S}$ be the set of all the maximal cliques in G_c. Notice that a clique is a collection of transmissions r_i^j. For a maximal clique $S_k \in \mathbb{S}$ let F_k be the set of flows that are part of S_k, i.e. F_k

Objective function

$$\text{maximize} \quad \lambda_k \tag{4.1}$$

Subject to

$$\sum_{\{(i,j)|r_i^j \in S_k\}} t_i^j \leq 1 - \epsilon \tag{4.2}$$

$$\lambda_k = \sum_{\{(i,j)|r_i^j \in R_i^l\}} C_i^j t_i^j, \forall R_i^l \in \{R_p^q | f_q \in F_k \cap \mathbb{F}^u, n_p \in N_k\} \tag{4.3}$$

$$\rho_l = \sum_{\{(i,j)|r_i^j \in R_i^l\}} C_i^j t_i^j, \forall R_i^l \in \{R_p^q | f_q \in F_k \cap \mathbb{F}^c, n_p \in N_k\} \tag{4.4}$$

$$t_i^j \geq 0, \forall t_i^j, \quad \text{i.e.} \{(i,j)|r_i^j \in \cup_k S_k\} \tag{4.5}$$

Figure 4.2: Linear program to identify the maximum rate all the unconstrained flows can achieve in a maximal clique

$= \{f_l | l \in j, r_i^j \in S_k\}$. Let N_k be the set of nodes that yield the transmissions in S_k, i.e. $N_k = \{n_i | r_i^j \in S_k\}$.

Let $R_i^l = \{r_i^j | l \in j\}$ be the set of transmissions that forward packets from flow f_l at node n_i. For example in figure 4.1, $R_F^a = \{r_F^{\{a\}}, r_F^{\{a,b\}}\}$. Let λ_k represent the fair rate allocated according to the linear program for maximal clique S_k. Our algorithm iterates through all the cliques, identifies the bottleneck flow that is assigned the smallest rate and fixes its rate. Once a flow's rate is fixed we begin the new iteration to fix the rates of the other unconstrained flows. If a flow f_l's rate has been fixed in previous iterations, in the linear program we represent this rate by ρ_l.

4.3.2.2 Constraint explanation

The objective function is straightforward. Although we want to assign all the unconstrained flows in a clique the same rate, the objective function dictates that we want to maximize

this rate.

Equation 4.2 is the channel access constraint. It ensures that all the transmission times in a clique are constrained to a unit time, hence only one transmission can be scheduled at a time in the clique. ϵ is the time lost due to channel access scheme. In reality ϵ is difficult to measure. In our simulations we have assumed perfect scheduling, hence $\epsilon = 0$.

The constraint presented in equation 4.3 applies to unconstrained flows in the network and it serves several purposes. It indicates that the rate achieved by a flow f_l at node n_i comprises of the rates achieved by all the transmissions that forward the packets of f_l at n_i. For example, in figure 4.1 rate achieved by flow f_a at node F equals the rate achieved by transmissions $r_F^{\{a\}}$ and $r_F^{\{a,b\}}$ both. Hence, we will have $\lambda_k = C_F^{\{a\}} t_F^{\{a\}} + C_F^{\{a,b\}} t_F^{\{a,b\}}$. For a flow f_l this constraint is applied to all the nodes through which f_l travels in the current clique. Since each of these nodes is assigning f_l a rate of λ_k, this serves as a flow conservation constraint as well. Furthermore this constraint is applied to all the flows that are part of the clique, which means that the channel access times will be chosen such that all the flows in the clique will have an equal rate λ_k.

Constraint in equation 4.4 is applied to the constrained flows whose rates have already been fixed, and it is similar to constraint 4.3. Constraint 4.4 determines the channel access times of the transmissions whose flow's rate has already been fixed to ρ_l. Hence this constraint determines what portion of the channel is already utilized. The remaining channel access time is used by constraint 4.3 to determine fair rate λ_k for the unconstrained flows.

Finally, constraint in equation 4.5 states that all the channel access times should be nonnegative.

Topology in figure 4.1 has only one clique. If we assume that all the links have unit capacity, the linear program in figure 4.2 would yield $t_A^{\{a\}} = t_B^{\{b\}} = t_F^{\{a,b\}} = \frac{1}{3}$ and $t_F^{\{a\}} = t_F^{\{b\}} = 0$. Hence both the flows in figure 4.1 will achieve a rate of $\frac{1}{3}$.

4.3.2.3 A note on constraints 4.3 and 4.4

It should be noted that constraint in equation 4.2 is applied to the transmissions that are only part of a maximal clique S_k, since these are the transmissions that cannot be scheduled simultaneously. However a careful observation of constraints in equations 4.3 and 4.4 will indicate that these constraints may contain some transmissions that are not part of S_k. For example, if figure 4.1 were a subgraph of a larger topology, it is possible that $r_F^{\{a\}}$ may not be part of some clique that has $r_F^{\{a,b\}}$ as a vertex. However, these constraints have to be applied to every flow at every node that is part of a clique. Therefore in order to maintain feasibility in the linear program, for a given flow f_l at any node n_i, all the transmissions that carry the packets of f_l must be included in these two constraints. Therefore in our example both $r_F^{\{a\}}$ and $r_F^{\{a,b\}}$ must be included in these constraints, even though only $r_F^{\{a,b\}}$ may be part of some maximal clique S_k and not $r_F^{\{a\}}$ (i.e. $\lambda_k = C_F^{\{a\}} t_F^{\{a\}} + C_F^{\{a,b\}} t_F^{\{a,b\}}$).

4.3.2.4 Selecting the bottleneck flow

If a topology has multiple maximal cliques, this linear program is applied to every maximal clique S_k in the conflict graph and the corresponding rate λ_k is recorded. All the flows in S_k should be able to achieve the rate λ_k. Clearly the bottleneck flow will be part of the clique that has the least value of λ_k.

Let's call the maximal clique that assigns the least rate to its flows S_{min}, and call this rate λ_{min}. If there was no coding in the network, all the unconstrained flows that are part of S_{min} can be assigned a rate of λ_{min}, since it is impossible for these flows to achieve any higher rate. However, when there is coding in the network, λ_{min} has to be looked at as the minimum of maximum rate flows in S_{min} can achieve. One can always generate scenarios where it is possible for some flows in a clique to achieve a rate higher than λ_{min}. We demonstrate this by an example.

Let's consider a subgraph of a path graph topology shown in figure 4.3. While we may have other flows going through different parts of the network as well as through nodes A to D, let's only consider three unconstrained flows f_1, f_2 and f_3. Under standard CSMA

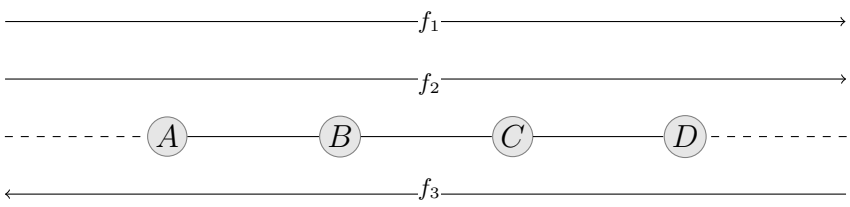


Figure 4.3: Bottleneck flow in this scenario would be either f_1 or f_2

model, following transmissions of flows f_1, f_2, f_3 at nodes A, B and C will be part of a clique S_k: $r_A^{\{1\}}$, $r_B^{\{1\}}$, $r_C^{\{1\}}$, $r_A^{\{2\}}$, $r_B^{\{2\}}$, $r_C^{\{2\}}$, $r_B^{\{3\}}$, $r_C^{\{3\}}$, $r_A^{\{1,3\}}$, $r_B^{\{1,3\}}$, $r_C^{\{1,3\}}$, $r_A^{\{2,3\}}$, $r_B^{\{2,3\}}$ and $r_C^{\{2,3\}}$.

Say during the i^{th} iteration of the algorithm, using the linear program in figure 4.2 we identify this clique (S_k) as the bottleneck clique with a rate λ_{min}. We have three choices f_1, f_2 and f_3 for choosing the bottleneck flow. Notice that for the purpose of this clique, f_1 and f_2 are identical flows. Hence if this clique is the bottleneck clique, both these flows will end up achieving the same rate $\rho_1 = \rho_2 = \lambda_{min}$. Say during the iteration number i and $i+1$ we choose f_1 and f_2 as the bottleneck flows. During the next iteration when the linear program is applied to S_k, we will have only one unconstrained flow f_3 and the linear program will calculate its rate as $\lambda_k = \rho_1 + \rho_2$. This happens because at every node in the clique, packets from f_3 can be combined with packets from f_1 and f_2. Hence once the rates of f_1 and f_2 are fixed, f_3 will be able to achieve the rate of f_1 plus the rate of f_2. The same results would not follow if f_3 was chosen as the bottleneck flow before f_1 or f_2. Since f_1 and f_2 cannot achieve any higher rate, in such a case all three flows will achieve a rate of λ_{min}.

For such a scenario to take place there must be a flow in a maximal clique that can be combined with two or more flows at every node it travels through in the clique. Let $N_{k,l} \subset N_k$ be the set of nodes that make transmissions $r_i^j \in S_k$, carrying packets of flow f_l ($l \in j$). For each $n_i \in N_{k,l}$ we count how many other flows packets from f_l can be combined with. We define *coding opportunity* $\gamma_{k,l}$ as the minimum of these counts. For example, in figure 4.3 flow f_3 can be combined with two other flows at each node A, B and C, hence

$\gamma_{k,3} = 2$, while $\gamma_{k,1} = \gamma_{k,2} = 1$. When a clique S_k is identified as the bottleneck clique in an iteration, we identify the unconstrained flow with the least value of $\gamma_{k,l}$. This flow is chosen as the bottleneck flow. By doing so we ensure that the rates of the flows are fixed in ascending order of their coding opportunities. Hence the flow with a higher coding opportunity can achieve a higher rate by leveraging the fixed rates of the constrained flows. Therefore, in figure 4.3 flows f_1 and f_2 will be chosen as the bottleneck flows before f_3.

4.3.3 Update $\mathbb{F}^u$ and $\mathbb{F}^c$

We use the linear program in figure 4.2 to identify the bottleneck clique S_{min} and corresponding rate λ_{min}. Using the variable $\gamma_{k,l}$ we identify the flow f_l in S_{min} that is the bottleneck flow. We fix the rate of this flow $\rho_l = \lambda_{min}$. For the future iterations of our algorithm, this flow is considered a constrained flow. Hence we update $\mathbb{F}^u$ and $\mathbb{F}^c$ as $\mathbb{F}^u = \mathbb{F}^u - \{f_l\}$ and $\mathbb{F}^c = \mathbb{F}^c \cup \{f_l\}$.

Algorithm 1 provides pseudo code for our algorithm to assign max-min fair rates to the flows in a wireless network with network coding.

4.3.4 Algorithm Complexity

Our algorithm to assign max-min fair rates runs in polynomial time. From algorithm 1 we can deduce that its complexity is $O\left(|\mathbb{F}| \left(|\mathbb{S}| O(LP) + |\mathbb{F}| \right) \right)$. Here $|\mathbb{F}|$ is the total number of flows in the network. $|\mathbb{S}|$ is the number of maximal cliques in the conflict graph of the network. $O(LP)$ is the complexity to solve a linear program. $O(LP)$ is generally a function of the variables it contains. As figure 4.2 shows, the only variables in our linear programs are transmission times t_i^j and λ_k. It has been shown that such a simple linear program can be solved in polynomial time, yielding the complexity of our fairness algorithm to be polynomial time as well.

Algorithm 1 Algorithm to assign max-min fair rates to flows in a wireless network with coding. Procedure "calculate-rate" calculates rate λ_k for a maximal clique S_k according to linear program in figure 4.2.

$\mathbb{F}^u \leftarrow \mathbb{F}$
$\mathbb{F}^c \leftarrow \emptyset$
while $|\mathbb{F}^u|$ **do**
    $bc \leftarrow 0$ ▷ Holds index of bottleneck clique
    $\lambda_{min} \leftarrow \infty$
    for $k = 1$ to $|\mathbb{S}|$ **do**
        if $F_k \cap \mathbb{F}^c \neq \emptyset$ **then**
            $\lambda_k \leftarrow$ calculate-rate(S_k)
            if $\lambda_k < \lambda_{min}$ **then**
                $bc \leftarrow k$
                $\lambda_{min} \leftarrow \lambda_k$
            end if
        end if
    end for
    $bf \leftarrow 0$ ▷ Holds index of bottleneck flow
    $\gamma^{min} \leftarrow \infty$ ▷ Holds minimum $\gamma_{bc,l}$
    for all $f_l \in F_{bc} \cap \mathbb{F}^u$ **do**
        if $\gamma_{bc,l} < \gamma^{min}$ **then**
            $bf \leftarrow l$
            $\gamma^{min} = \gamma_{bc,l}$
        end if
    end for
    $\rho_{bf} \leftarrow \lambda_{min}$
    $\mathbb{F}^u \leftarrow \mathbb{F}^u - \{f_{bf}\}$
    $\mathbb{F}^c \leftarrow \mathbb{F}^c \cup \{f_{bf}\}$
end while
return ρ

4.3.5 Max-min Fair Rate Allocation: Results

In this section we apply our rate allocation algorithm to a few flow configurations in different topologies.

We compare rates assigned using our algorithm with the rate control algorithm presented in [Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009]. This algorithm extends the work of Kelly et al. [Kelly *et al.*, 1998] for the flows in a wireless network with coding. Their rate control algorithm is represented by a linear program. Here, the objective is to assign flows rates that maximize a concave utility function, subject to channel access and flow conservation constraints. For more details we refer the reader to [Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009]. In our simulations we have chosen summation of logarithms of the flow rates as the concave utility function, which is a standard choice for such an objective function. Notice that this type of an objective function generally yields proportionally fair rates. We compare our scheme with this algorithm, since to the best of our knowledge this is the only existing rate control algorithm for flows in a wireless network with coding. We use *fmincon* function from Matlab [MATLAB, 2010], which uses interior point method to solve the optimization problem. All the variables are initialized to 10^{-8}. For both the schemes, we have assumed perfect scheduling (i.e. $\epsilon = 0$ in equation 4.2). Finally, for this part of the results we assume that each edge connecting two nodes have unit capacity.

Figure 4.4 shows a network where twenty nodes are scattered uniformly in a $1000 \times 1000m^2$ area. We have inserted four flows in the network. Figure 4.5 shows the rates of these flows assigned using two different algorithms. The rates with label 'Max-Util' are assigned using the rate control algorithm of [Seferoglu *et al.*, 2009] and [Seferoglu and Markopoulou, 2009]. While the rates with label 'Max-Min' are the rates allocated using our algorithm.

Flows f_1 and f_2 travel longer distances. Moreover they do not have much coding opportunities. Since these two flows use more network resources, as figure 4.5 indicates, proportional fairness assigns these two flows lesser rates compared to f_3 and f_4. However,

when we create a conflict graph of this network, we realize that there exists a maximal clique which contains the transmissions belonging to all four flows. Instead of giving more rates to f_3 and f_4, our algorithm will try to be fair to all four flows. It identifies the bottleneck clique, and allocates some of the channel capacity from f_3 and f_4 to f_1 and f_2. As the figure indicates our rate allocation is fairer, since all four flows end up achieving the same rates.

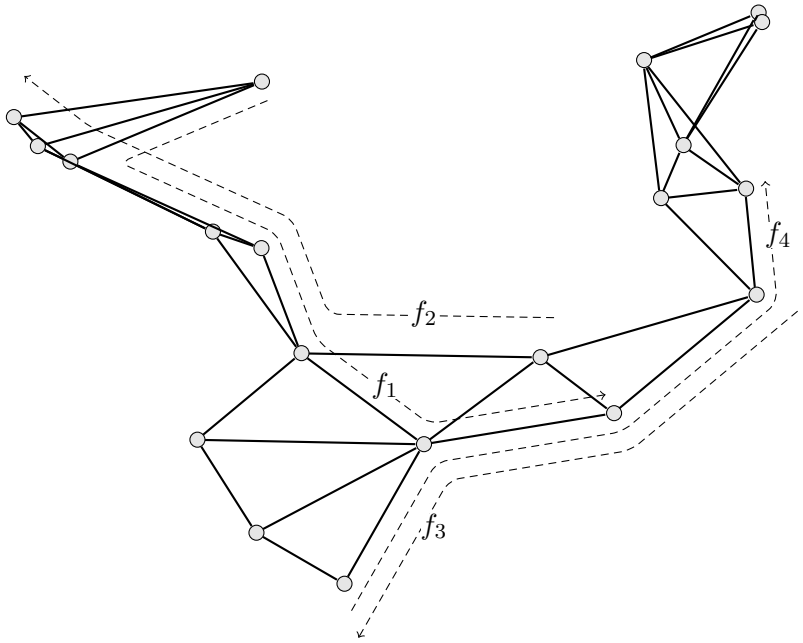


Figure 4.4: Twenty nodes scattered randomly in a $1000 \times 1000m^2$ area

Figure 4.6 gives an example of a structured topology, where we have inserted six flows in the network. Figure 4.7 shows the rates allocated to these flows. Once again, rate assignment by maximizing a utility function is quite unfair to some of the flows. Flows f_5 and f_6 end up achieving quite a high rate, while the rates of the flows f_1 and f_2 is completely suffocated. On the other hand, as the figure indicates, rates assigned using our algorithm are much fairer.

Figure 4.8 is an example of a network with heavy traffic. Flows in this network have

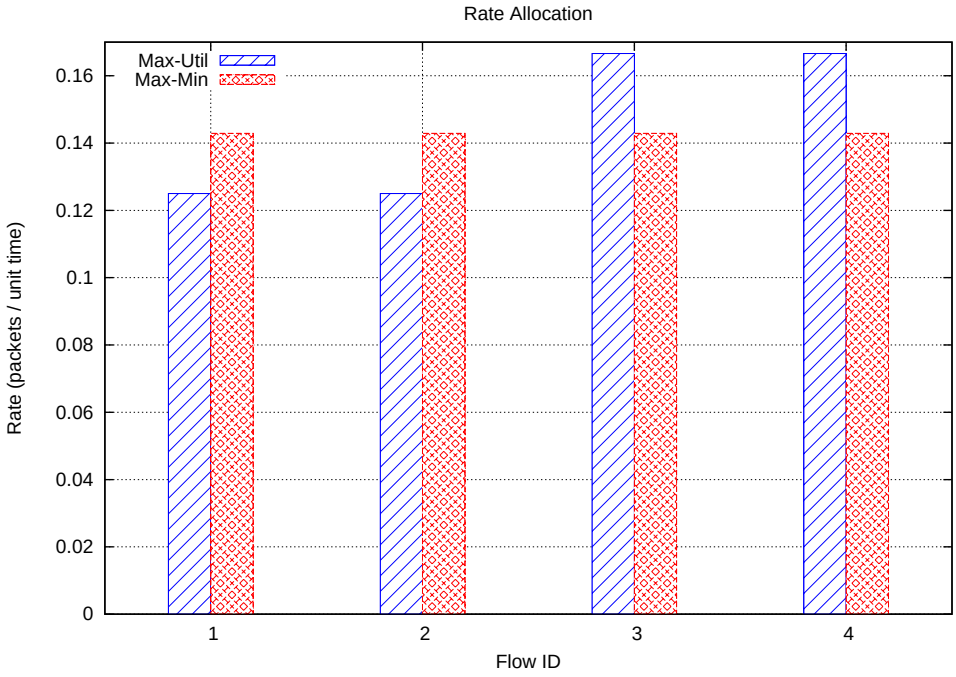


Figure 4.5: Rates assigned to the flows in figure 4.4

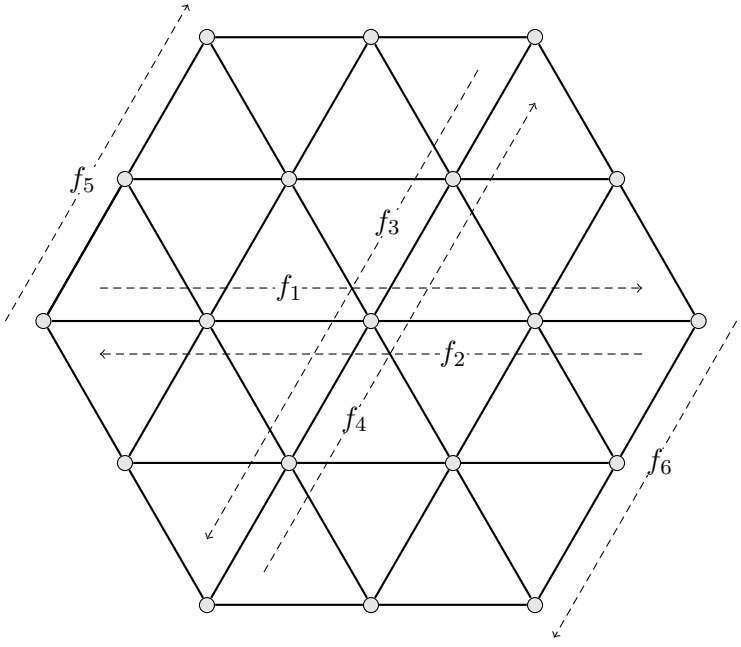


Figure 4.6: Nodes in a structured topology

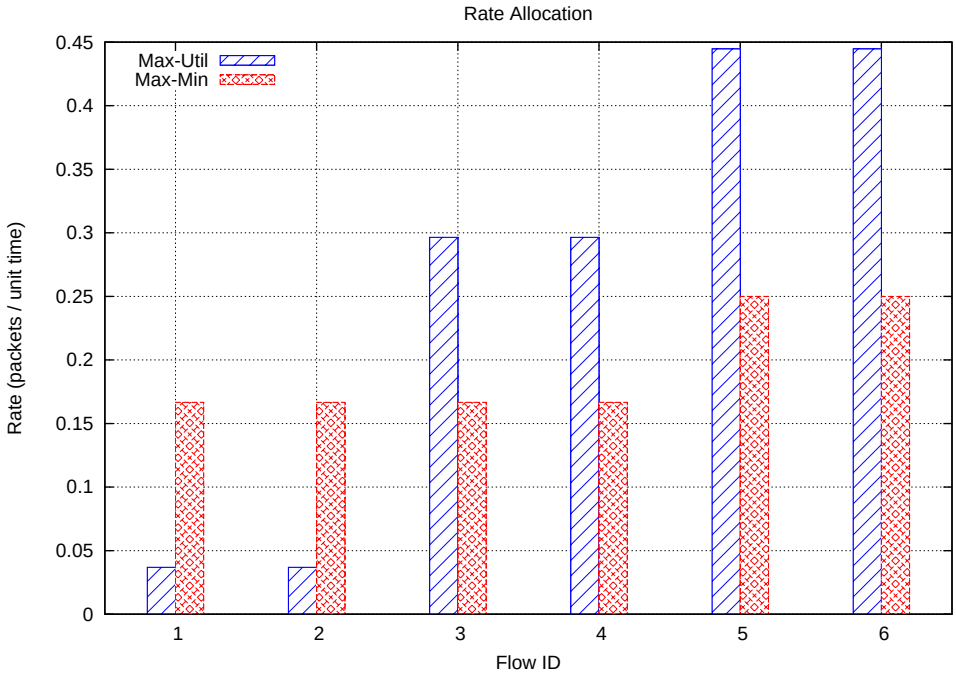


Figure 4.7: Rates assigned to the flows in figure 4.6

several coding opportunities. This network is also an example of the scenario where if one overlooks the caveats presented in sections 4.3.2.3 and 4.3.2.4, the algorithm may give suboptimal results or the linear program of figure 4.2 may yield incorrect results. When a fairness mechanism tries to maximize a utility function, some flows (f_5 in this example) may end up getting significantly lower rates. Our scheme tries to maximize the minimum rate a flow is getting. Hence the rate of a flow such as f_5 is improved significantly, by taking away the channel capacity from the other flows.

When we run our algorithm on this network, during the first iteration we identify a bottleneck clique that has transmissions from flows f_1, f_2, f_3, f_4 and f_5. Since the coding opportunity γ is zero for all five flows in this clique, they all end up achieving the same rate. As mentioned in section 4.3.2.4, if coding opportunity $\gamma \geq 2$ for a flow, it may end up achieving a higher rate. During subsequent iterations we do identify a bottleneck clique for which coding opportunity $\gamma = 2$ for flow f_6, hence it ends up achieving a higher rate. Flow f_7 achieves a higher rate as well since it conflicts with fewer flows, and it has a higher

channel capacity available since f_6 is using up less channel capacity by combining its packets with f_1 and f_5.

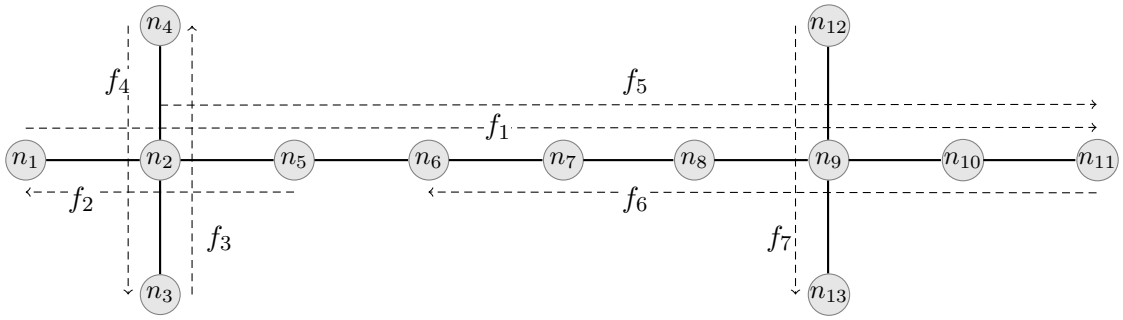


Figure 4.8: A network with dense flows

So far all the edges in the networks had a unit capacity. We modify the capacities of some of the links in figure 4.8 to create a network with heterogeneous capacities. In the new configuration, capacity of edges $(n_2, n_5), (n_7, n_8), (n_8, n_9), (n_9, n_{12})$ and (n_9, n_{13}) are changed to 2, while the remaining edges still have unit capacities. Figure 4.10 shows the rates assigned to various flows using two algorithms. As the figure indicates, the rates assigned using our algorithm are more fair.

We also compare the fairness of these rates using fairness indices. One of the popular fairness index is Jain's index [Jain *et al.*, 1984]. For a rate vector $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$, Jain's index is defined in equation 4.6.

$$\mathcal{J}(\rho) = \frac{(\sum_i \rho_i)^2}{n \sum_i {\rho_i}^2} \tag{4.6}$$

Under the best case scenario Jain's index would be 1, when all the flows are assigned the equal rate. Under the worst case scenario, it will be $\frac{1}{n}$, when only one flow is assigned a rate, and the rest of the flows are assigned a rate of zero. Closer the index to 1, more fair the rate allocation. Note that it makes sense to use Jain's index only after the channel capacity is completely utilized. Otherwise one may assign the same low rates to all the flows in the network without completely using the channel and still have Jain's index as 1. The

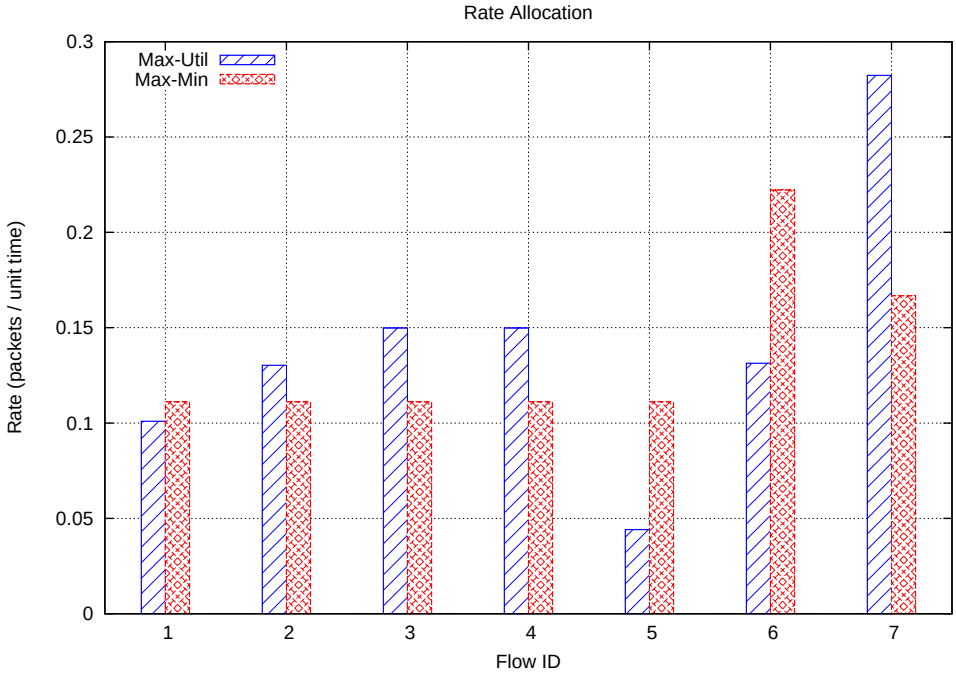


Figure 4.9: Rates assigned to the flows in figure 4.8

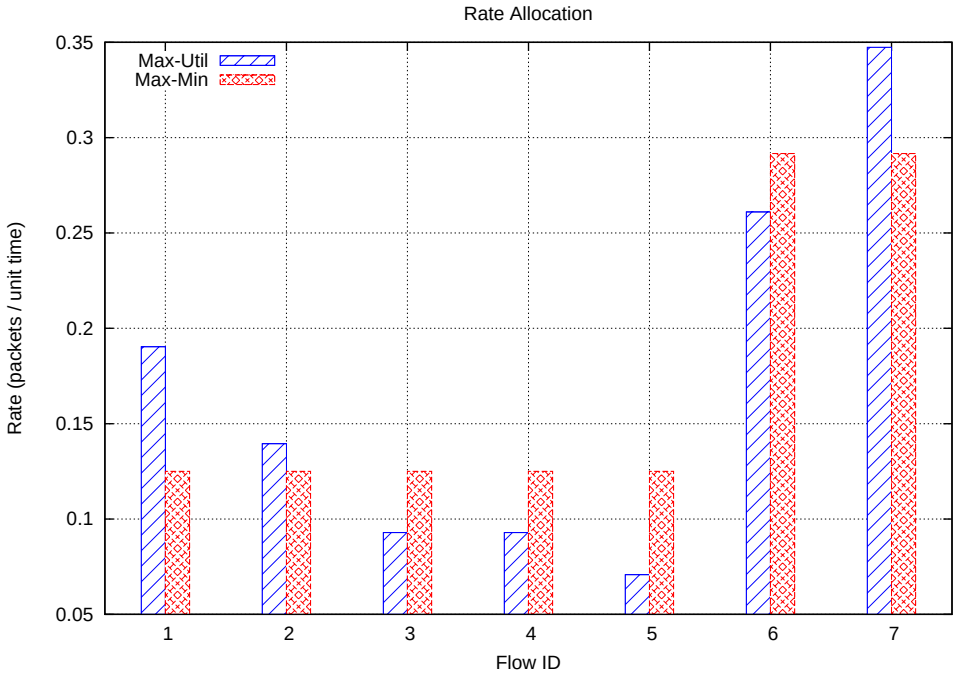


Figure 4.10: Rates assigned to the flows in figure 4.8, with the heterogeneous capacities

	Jain's Index		Max/Min	
Network	Max-Util	Max-Min	Max-Util	Max-Min
Figure 4.4	0.98001	1.0	1.3328	1.0
Figure 4.6	0.7031	0.9608	12.0515	1.4997
Figure 4.8 Unit capacities	0.8167	0.9174	6.4036	2.0
Figure 4.8 Heterogeneous capacities	0.7645	0.8401	4.9068	2.3326

Table 4.1: Comparison of rate fairness using fairness indices

second index we try is the ratio of maximum and minimum flow rates: $Max(\rho)/Min(\rho)$. This index is quite intuitive. Closer the ratio to one, fairer the rate allocation. Table 4.1 compares these two indices for the two rate allocation schemes. As the results indicate, according to both the indices, our scheme offers a more fair rate assignment.

While proportional fairness attempts to allocate fair rates based on the network resources, max-min fairness ensures that the flows that end up using more of a network resource (such as channel capacity) do not suffer in throughput. Since a relatively higher rate is assigned to the flows that consume more resources, max-min fairness generally results in a lower aggregate throughput for the network. Table 4.2 compares the aggregate throughput of a network when two different fairness criterion are used.

Network	Max-Util	Max-Min
Figure 4.4	0.5832	0.5716
Figure 4.6	1.5562	1.1668
Figure 4.8: Unit capacities	0.9888	0.9444
Figure 4.8: Heterogeneous capacities	1.1948	1.2084

Table 4.2: Comparison of aggregate network throughput

4.4 Scheduling algorithm from a global perspective

In this section we present our scheduling algorithm which helps flows achieve rates that are assigned using the fair rate allocation algorithm in section 4.3.

For a small enough network whose conflict graph has only one clique, rate allocation and scheduling algorithms both can be represented by the linear program shown in figure 4.2. Once the optimization problem is solved, we can obtain values for transmission times t_i^j. A node n_i can transmit packets from flow f_l for $t_i^j (l \in j)$ amount of time.

However, when we have multiple maximal cliques in the network, we cannot always use the transmission times assigned using the same linear program. The objective of the linear program in figure 4.2 is to identify the maximum rate λ_k that all the unconstrained flows in a bottleneck clique S_k can achieve. There may be multiple transmission time allocations that can achieve this rate. For example consider channel access times t_i^j and $t_{i'}^{j'}$ that are allocated to two different transmissions. Say the bottleneck flow in S_k will achieve a rate λ_k irrespective of whether $t_i^j = \tau^{hi}, t_{i'}^{j'} = \tau^{lo}$ or $t_i^j = \tau^{lo}, t_{i'}^{j'} = \tau^{hi}$, where $\tau^{hi} > \tau^{lo}$. Say transmission $r_{i'}^{j'}$ is part of some other clique $S_{k'}$ as well, i.e. $r_{i'}^{j'} \in S_k \cap S_{k'}$, while $r_i^j \notin S_k \cap S_{k'}$. Say the optimizer chooses to fix the transmission times to $t_i^j = \tau^{lo}$ and $t_{i'}^{j'} = \tau^{hi}$. In this case during the subsequent iteration, when the linear program is run on $S_{k'}$, unconstrained flows in this clique will have less channel access time available to them, hence these flows may end up achieving suboptimal rates. Therefore, when we have multiple maximal cliques in the conflict graph, we are better off running a scheduling algorithm only *after* we have fixed the rates of all the flows using the algorithm in section 4.3.

Definitions of all the variables in the linear program of figure 4.11 are identical to the variables in figure 4.2.

Equation 4.8 is the channel access constraint. It constrains the transmission times in a maximal clique to a unit time, hence only one transmission can be scheduled at a time. This constraint is applied to all the maximal cliques in the conflict graph. Equations 4.9 indicates that for a node n_i, rates achieved by all the transmissions that forward packets of a flow f_l should add up to the rate assigned to f_l. Finally, constraint 4.10 forces all the

Objective function

$$\text{minimize} \sum_{\{(i,j)|r_i^j \in \bigcup_k S_k\}} t_i^j \tag{4.7}$$

Subject to

$$\sum_{\{(i,j)|r_i^j \in S_k\}} t_i^j \leq 1 - \epsilon, \forall S_k \in \mathbb{S} \tag{4.8}$$

$$\rho_l = \sum_{\{j|l \in j\}} C_i^j t_i^j, \forall n_i \in V, \forall, f_l \in \mathbb{F} \tag{4.9}$$

$$t_i^j \geq 0, \forall t_i^j, \quad \text{i.e.} \{(i,j)|r_i^j \in \cup_k S_k\} \tag{4.10}$$

Figure 4.11: Linear program to identify the maximum rate all the unconstrained flows can achieve in a maximal clique

transmission times to be nonnegative.

Note that the objective function of equation 4.7 says that we want to minimize the total transmission time in the network. However, this objective function isn't necessary. Any allocation of transmission times that meets the constraints 4.8, 4.9 and 4.10 can sustain the rates assigned using our fair rate allocation algorithm.

Once we have determined the rates for the flows in a network, we can run this linear program to assign channel access times to each transmission. Note that unlike algorithm 1, we have to run this linear program only once. We do not have to run multiple iterations of it. After figuring out the channel access times, one can run a graph coloring algorithm [Gross and Yellen, 2005] on the conflict graph. The nodes in the network can take turns as suggested by the colors and forward packets for the alloted time.

4.5 Distributed algorithms

After having presented centralized fair rate allocation and scheduling algorithms, in this section we present distributed versions of these mechanisms.

4.5.1 Distributed fair rate allocation algorithm

The objective in this subsection is to assign the same rates to the flows as shown in section 4.3, however in a decentralized fashion.

Our centralized rate allocation algorithm depends on the maximal cliques in the conflict graph of the network. In a CSMA based channel access scheme only the transmissions that are at most two hops away can conflict with each other. Therefore, if a node can exchange information with its neighbors that are at most two hops away, it can identify a subset of maximal cliques that are part of the global conflict graph. [Huang and Bensaou, 2001] provides an algorithm to achieve that, hence to avoid redundancy we won't repeat such an algorithm here. When each node in the network runs this algorithm, each maximal clique in the conflict graph will be identified by at least one node in the network.

Once the network nodes identify what maximal cliques their transmissions are part of, they can run algorithm 1 to determine the rates of the flows that are part of the cliques they have identified. After that a node can communicate the rates it calculates with relevant nodes in the network. However, two key questions are (a) When we are running a distributed algorithm, how do we determine which flows' rates have been fixed (constrained) and which flows are still unconstrained? (b) Different nodes may calculate different rates for the same flow, how do they agree on a rate for a flow?

To address both these questions we devise a novel yet simple mechanism. Each node contains a map indicating the flows that are part of its locally identified maximal cliques and the rates assigned to them. Note that a flow may not travel through a node, but it can still be part of the maximal clique that the node identifies. Moreover, a node may identify multiple maximal cliques that are part of the global conflict graph. Initially each flow is assigned a rate zero. In this mechanism the source of each flow sends a control packet to the flow's destination, and receives the packet back from it. The control packet contains following fields *(flow, currentRate, minRate, complete, direction)*. *flow* is a tuple containing the source and destination of the flow. *currentRate* indicates the flow's currently assigned rate, initially it is set to zero. Initial values for *minRate*, *complete* and *direction* are ∞,

True and "toDst" respectively. All the information contained in a control packet may also be embedded in the data packets, but for the sake of simplicity, we stick to control packets while describing this scheme. While the packet is travelling towards the destination, it asks each node in its path (including the source) to run algorithm 1 on its local snapshot of the global conflict graph. While running this algorithm, the node treats each flow in its map whose rate is less than *currentRate* as constrained flow. The flows that have rates greater than or equal *currentRate* are considered unconstrained flows. By doing so each control packet is trying to see if its flow can achieve a rate higher than its currently assigned rate. After running algorithm 1 the node informs the control packet what rate it (or its maximal cliques) is willing to allocate to its flow. If this newly computed rate is less than *minRate*, the *minRate* is updated to the new value. Therefore, when the control packet reaches flow's destination, it knows what minimum rate (*minRate*) different nodes (maximal cliques) are willing to allocate to its flow. The destination node sets *direction* to "toSrc", and forwards the packet back towards the source of the flow. On its way back towards the source, the control packet asks each node on its path to update its map and set the new flow rate to *minRate*. The node not only updates its local map, but it also informs its two hop neighboring nodes to update their maps to *minRate*. Once the control packet returns to the flow's source, *currentRate* is assigned the value of *minRate* and *minRate* is set to ∞ again. This process is repeated until the convergence is achieved.

It is easy to show that this process converges to the same rates as assigned by the centralized algorithm. Say all the flows in the network are unconstrained, and the source of each flow sends a control packet to the flow's destination and receives it back. Let's consider what happens to the flow that gets the least rate after every control packet has come back to its source for the first time. Since all the flows were unconstrained, this control packet must have gone through the clique that gets identified during the first iteration of the centralized algorithm as the bottleneck clique. Hence the flow will get the same rate as the flow that was constrained after the first iteration of the centralized algorithm. Without loss of generality let's call this flow f_1, its rate ρ_1. The source of this flow may not know

that its flow has achieved the minimum rate. When the sources send out control packets again towards the destinations, there are two scenarios that will take place.

1. **f_1's control packet goes through the same nodes (maximal cliques):** Since every other flow has been allocated a rate higher than f_1, when f_1's control packet reaches a node, the node considers all other flows as unconstrained flows. However, f_1 achieved its current rate only after considering all the flows in the network as unconstrained flows. Therefore, when the control packet comes back to its source, f_1 still will be assigned a rate that it originally had.

2. **Some other flow's control packet goes through some of the nodes (maximal cliques) that f_1 has gone through:** During its journey back towards its source, f_1's control packet asks all the nodes on its path to update its rate to ρ_1. Hence some of the maximal cliques on its path other than the bottleneck clique will have some additional capacity that can be allocated towards the flows other than f_1. **(a)** If the flow goes through f_1's bottleneck clique as well, it may get the same rate as f_1. In this case this flow could have been chosen as the bottleneck flow as well, and in this regard it is no different than f_1. Hence it will also keep getting the same rate as f_1 during subsequent message passing. **(b)** Now consider a flow that goes through some of the maximal cliques of f_1, but not through its bottleneck clique. Since f_1 had the least rate, when some other flow asks a node to recalculate the fair rates, it will consider f_1 as a constrained flow. After updating f_1's rate to ρ_1, the cliques other than the bottleneck clique will have some additional capacity that was previously assigned to f_1. But now this capacity will be allocated to the remaining flows, and hence such a flow will always get a rate higher than f_1. Hence f_1 will always be considered a constrained flow. Therefore essentially its rate is fixed and the problem reduces to settling the rates for the remaining flows.

During subsequent iterations of the message passing, the flow with the second least rate will ask its nodes to consider all the flows other than f_1 as unconstrained flow, and the process

will continue until all the flows have been assigned a fixed rate.

Once a node realizes that all of the flows in its locally identified maximal cliques are requesting to update to the same rates, it sets *complete* flag in a control packet to True, otherwise the *complete* will be set to False. When all the nodes have set the *complete* flag to True, upon reception of such a control packet the source will realize that all the flows have converged to their fair rates, and it will stop sending more control packets.

Algorithm 2 describes the procedure a node follows upon receiving a packet *pkt*. In this algorithm, functions *sendPacketTowardsDestination* and *sendPacketTowardsSource* forward the control packet towards the destination and the source of the flow respectively. After updating the rates map to *minRate* a node uses *updateTwoHopNeighbors* function to inform its two hop neighbors to update their rates maps. *checkForConvergence* is a function that checks if all the flows in a node's locally identified cliques have started to request the same rates. One way to implement this function is to, keep track of all the control packets that are travelling towards the source, as well as neighbors' update requests (*updateTwoHop-Neighbors*). When all the update requests and control packets have requested the same rates at least twice, *checkForConvergence* would return True, otherwise, it would return False.

Note that ideally every time all the sources in the network send control packets to the destination and receive it back, at least one flow must get its rate fixed. Maximum distance a control packet has to travel is $2D$ where D is the diameter of the network (diameter is defined as the maximum distance between two vertices in a graph [Gross and Yellen, 2005]). Say U is the maximum number of packets a node has to forward to update the rate maps of its two hop neighbors. In that case we can conclude that the network may have to transmit $O(|\mathbb{F}|DU)$ packets until the distributed rate allocation algorithm converges. In reality it is difficult to determine the worst case number of packets that are transmitted in the network to determine the fair rates. Since the packets are transmitted in asynchronous manner, it is not necessarily true that when one source receives its control packet back, all other sources will have received their control packets back as well.

Algorithm 2 Algorithm a node follows upon receiving a control packet

```
1: if pkt.direction = "toDst" then
2:     newRatesMap = ratesMap
3:     for flow ∈ newRatesMap do
4:         if newRatesMap[flow] ≥ pkt.currentRate then
5:             newRatesMap[flow] = "unconstrained"
6:         end if
7:     end for
8:     rates = Algorithm-1(newRatesMap)
9:     if rates[pkt.flow] < pkt.minRate then
10:        pkt.minRate = rates[pkt.flow]
11:    end if
12:    sendPacketTowardsDestination(pkt)
13: end if
14: if node = pkt.flow.destination then
15:    pkt.direction = "toSrc"
16: end if
17: if pkt.direction = "toSrc" then
18:    if ratesMap[pkt.flow] ≠ pkt.minRate then
19:        ratesMap[pkt.flow] = pkt.minRate
20:        updateTwoHopNeighbors(pkt)
21:    end if
22:    flag = checkForConvergence()
23:    pkt.complete = pkt.complete and flag
24:    sendPacketTowardsSource(pkt)
25: end if
26: if node = pkt.flow.source then
27:    if pkt.complete then
28:        terminateProcess()
29:    else
30:        pkt.direction = "toDst"
31:        pkt.currentRate = pkt.minRate
32:        pkt.minRate = ∞
33:        pkt.complete = True
34:    end if
35:    newRatesMap = ratesMap
36:    rates = Algorithm-1(newRatesMap)
37:    if rates[pkt.flow] < pkt.minRate then
38:        pkt.minRate = rates[pkt.flow]
39:    end if
40:    sendPacketTowardsDestination(pkt)
41: end if
```

Figure 4.12 shows the fraction of flows that are converged to their max-min rates vs. number of control messages transmitted in the network. As the figure indicates it takes only 44 control messages for all four flows in network 4.4 to converge to their max-min fair rates. Flows in figure 4.6 take 69 control messages to converge to their fair rates. It takes 160 messages for the flows in figure 4.8 to converge to their fair rates. Network 4.8 is much smaller in size compared to the networks in figures 4.4 and 4.6. However, this network transmits significantly more number of control messages to achieve max-min fair rates. This happens because it has more number of flows and they travel longer distances. This verifies that the number of messages transmitted in the network are a function of total number of flows in the network and how many hops they travel. It should also be noted that the number of control messages transmitted are also a function of how many flows travel through a bottleneck clique. For example, if all the flows in a network pass through a clique, this clique may serve as a bottleneck clique and the control messages have to travel to the destination and come back only once.

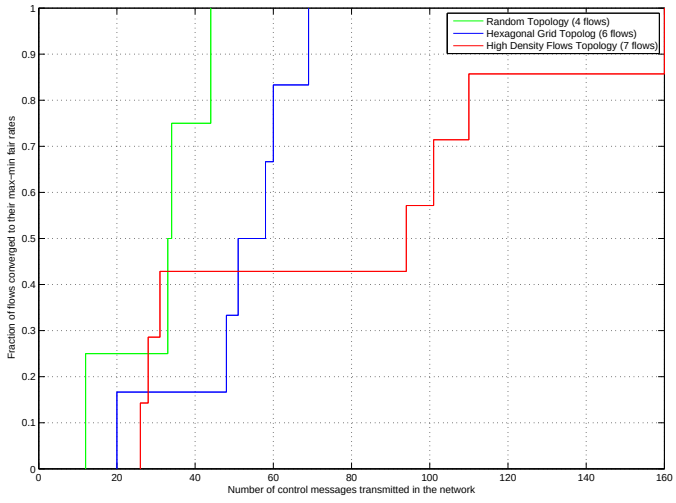


Figure 4.12: Fraction of flows converged to their max-min fair rates vs. Number of control messages transmitted

4.5.2 Distributed scheduling algorithm

Our distributed scheduling algorithm is similar to the one presented in [Akyol *et al.*, 2008], [Gupta and Stolyar, 2006], [Gupta *et al.*, 2005], [Huang and Bensaou, 2001] and [Nandagopal *et al.*, 2000]. This scheme operates on a random channel access mechanism such as CSMA. Therefore we can implement our scheduling scheme on widely used 802.11 standard without adding an additional layer of protocol.

In 802.11 before transmitting a new packet a node waits for a random period of time. This random waiting period is chosen uniformly from $(0, 2^{CW} - 1)$. Every time a node senses that the channel is busy or if its packet experiences collision, the value of CW is incremented by one, until it reaches some value CW_{max}. After a successful transmission CW reduces back to CW_{min}. Commonly accepted values for CW_{min} and CW_{max} are 5 and 10 respectively. After agreeing upon the rates for the flows in a network, nodes can calculate the cumulative rate at which they are supposed to forward the messages. For example if a node n_i is supposed to forward packets from the flows $f_1, \ldots, f_k$ with rates $\rho_1, \ldots, \rho_k$, its cumulative rate would be $\xi_i = \sum_{j=1}^{k} \rho_j$. The idea here is to keep track of neighbor's cumulative rates, and adjust the contention window based on what rate the neighbors have achieved. If a node has transmitted more packets than its neighbors with respect to their cumulative rates, it chooses a higher contention window and waits for a longer period before transmitting a new packet. On the other hand, if a node has forwarded less number of packets (after normalizing with cumulative rates) compared to all its neighbors, a smaller contention window is chosen, giving the node a higher priority in sending its packets.

Once the convergence for the rate allocation is achieved, a node keeps track of the rate at which its neighbors are transmitting data packets. To achieve this a node keeps track of how many data packets as well as acknowledgements its neighbors are transmitting. By doing so a node is able to determine the rate with which some of its two hops neighbors are transmitting packets. Note that a node may not be able to hear packets from all the relevant flows from its two hop neighbors. In this case it has to estimate a node's total packet transmission from the flows it is able to observe. For example consider the

scenario presented in figure 4.13. Note that f_1 conflicts with the packets from f_2 and f_3 both. Upon reception of an f_2 packet whenever n_2 sends an acknowledgement back to n_3, n_1 can estimate the rate achieved by f_2. However, n_1 has no way of knowing what rate f_3 is achieving. Let's denote the number of packets from flow f_l transmitted by a node n_i as x_i^l. If n_1 observes that n_3 has forwarded x_3^2 packets of f_2, it can estimate that n_3 may have forwarded $x_3^3 = x_3^2 \frac{\rho_3}{\rho_2}$ packets of the flow f_3. Let χ_i denote the total packets successfully forwarded by a node n_i. From this information node n_1 can estimate the total packets transmitted by n_3 as $\chi_3 = x_3^2 + x_3^3$. In general if a node can only partially observe the packets from a two hop neighbor, it can calculate the unobserved packets as follows: total unobserved packets = (total observed packets) $\times$ $\frac{\text{total rate of unobserved flows}}{\text{total rate of observed flows}}$. Note that if a node n_i forwards k packets in a single transmission it adds k to the tally of χ_i not 1. It should also be noted that a node n_i does not have to exchange additional information with its neighbors to convey flow rates or ξ_i. During the distributed rate allocation algorithm a node requests its two hop neighbors to update their flow rate maps, every time a control packet requests a new rate. ξ_j for a neighbor n_j can be calculated using these updates.

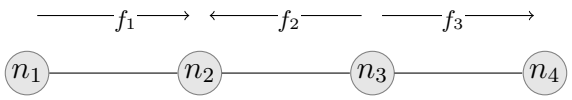


Figure 4.13: An example of calculating total number of successful transmissions

4.5.2.1 Contention Window Adjustment

Once a node n_i has calculated the number of packets χ_j its neighbor n_j has forwarded, it standardizes χ_j with respect to cumulative rates to obtain weights $w_j = \frac{\chi_j}{\xi_j}$. A node n_i compares its weight w_i with the neighbors' weights. Contention window of a node n_i is adjusted by comparing its weight w_i with its neighbors' weights. Scheduling schemes of [Akyol *et al.*, 2008], [Huang and Bensaou, 2001] also calculate some form of weights in their mechanism. If a node does not have the smallest weight among all its neighbors,

it chooses a contention window higher than CW_{min} before transmitting a new packet. Otherwise, the contention window is chosen to be CW_{min}. These schemes use RTS/CTS to estimate the number of packets a node's neighbors are transmitting. Unfortunately, in their implementation of the coding scheme COPE, [Katti *et al.*, 2006] does not have RTS/CTS enabled. Therefore, we have to rely on the data packets and acknowledgements to count a neighbor's successful transmissions. Consider the scenario presented in figure 4.14. This scenario is similar to figure 4.13, except there is no flow f_2. In the absence of RTS/CTS, if we only observe data packets and acknowledgements n_3 can estimate how many packets n_1 is forwarding by observing the acknowledgements from n_2. However, n_1 has no way of knowing how many packets n_3 is forwarding. Therefore if we were to follow the schemes in [Akyol *et al.*, 2008], [Huang and Bensaou, 2001], in this case only n_3 will adjust its contention window to a higher value. n_1 will never have to adjust its contention window, resulting in a higher throughput for f_1. To avoid such a scenario wherever a node identifies that it has the smallest weight w_i, it chooses a smaller contention window than CW_{min}. Therefore in our scheme, before transmitting a new packet, a node adjusts it CW according to the following rules.

- $CW = CW_{min} + 2$ if a node n_i does not have the smallest weight among its neighbors ($CW = 7$ in our experiments).

- $CW = CW_{min} - 2$ if a node n_i has the smallest weight among its neighbors ($CW = 3$ in our experiments).

- $CW = CW_{min}$ if a node n_i does not know the weight of its neighbors (for example node n_1 in figure 4.14).

Whenever a node gets access to the channel it has to determine which flow's packet it should transmit. In order to decide this a node n_i once again calculates the weight for the flow f_l as $w_i^l = \frac{x_i^l}{\rho_l}$ The node n_i decides on the packet from the flow f_l that has the smallest

Figure 4.14: A case for using a smaller contention window

weight w_i^l. It looks for the other packets in its queue that can be combined with this packet and transmits the combination.

4.5.2.2 Performance of the distributed scheduling algorithm

As mentioned earlier our distributed fair rate allocation algorithm achieves the same rate as the centralized algorithm. In this section we check the performance of our distributed scheduling mechanism on our custom built discrete event simulator that allows network nodes to combine packets. The simulator implements 802.11 with RTS/CTS disabled and slot length = $9\mu Sec$. Each link between two nodes in the network has the same capacity, according to which each packet is 0.0001 seconds long. It is assumed that each flow always has a packet to transmit. The simulation is run for 110 seconds, and the rate of the flows is measured by calculating the number of packets that leave the network in last 100 seconds. Each node in the network is assigned a buffer length of 100 packets. The buffer is divided into several virtual buffers, where each virtual buffer stores a packet belonging to a particular flow that the node transmits. If the virtual buffer for a flow is full, the node drops its packets. The size of the virtual buffers are proportional to the rate of the flows the node is transmitting.

We run our distributed rate allocation algorithm on the networks shown in figures 4.4, 4.6 and 4.8. Once the algorithm converges, we observe what rates the network flows are able to achieve using our distributed scheduling scheme. Generally it is difficult to calculate the channel access time lost due to the MAC scheme. This time is represented by ϵ in the linear program of figure 4.2. In our simulation we assume that $\epsilon = 0$. Since each link in the network has the same capacity (1 packet per 0.0001 seconds), distributed rate allocation

algorithm assigns rates assuming each link has a unit capacity. This in turn achieves the same results as shown in figures 4.5, 4.7 and 4.9 respectively. The rates assigned using our scheduling algorithm should be proportional to the rates presented in figures 4.5, 4.7 and 4.9.

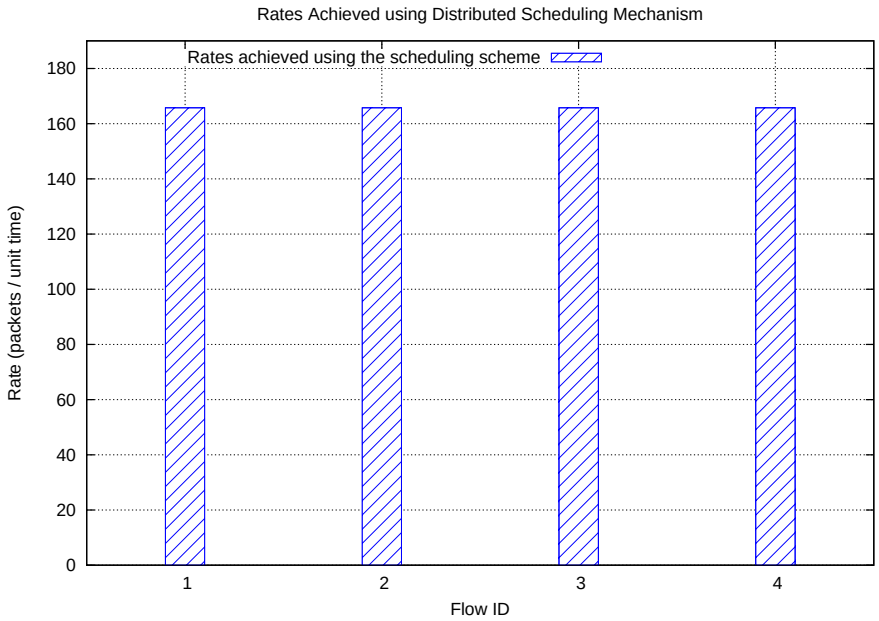


Figure 4.15: Flow rates achieved using distributed scheduling scheme for the network shown in figure 4.4

Figure 4.15 shows the flow rates achieved using distributed scheduling scheme for the network shown in figure 4.4. By comparing figures 4.5 and 4.15, it is easy to observe that after having compensated for a different channel capacity and the lost channel access time, distributed scheduling mechanism achieves the same rates as calculated according to our fair rate allocation algorithm. To further assess the accuracy of our scheduling mechanism we devise the following scheme. For each flow in the network, we calculate what fraction of total network throughput is contributed by a flow using two methods: (1) fair rate allocation of algorithm 1 (2) distributed scheduling scheme. For each flow we calculate the relative difference between these two values, and report the error. For example according to figure 4.5, all the flows in network 4.4 are assigned a max-min fair rate $\frac{1}{7}$. Hence flow

f_1 contributes 25% to the total throughput. Now, according to figure 4.15 distributed scheduling scheme achieves rates of 165.72, 165.73, 165.72 and 165.72 packets/second for flows $f_1, \ldots, f_4$ respectively. Hence using this scheduling scheme f_1 contributes 24.9996% to the overall network throughput. In other words, the scheduling scheme makes $\frac{24.9996-25}{25} \times 100 = -0.0015\%$ error in assigning f_1 its max-min fair rate. Figure 4.16 shows error in assigning max-min fair rates for all four flows in the network shown in figure 4.4. The fact that these errors are close to zero, demonstrate the accuracy of the distributed scheduling mechanism.

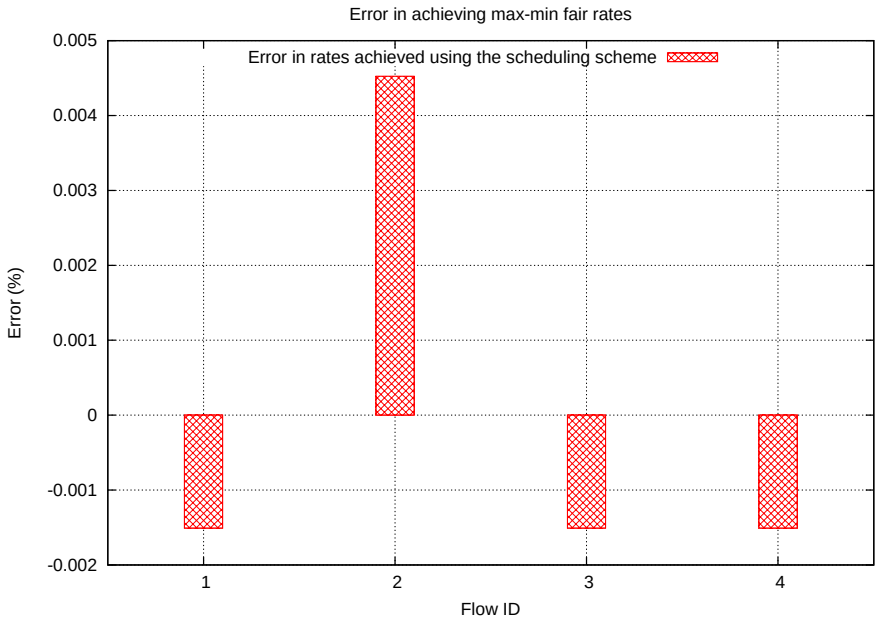


Figure 4.16: Error made by scheduling scheme in assigning max-min fair rates for the flows in figure 4.4

Figure 4.17 shows the rates achieved by the flows in network 4.6 using this scheduling scheme. Figure 4.18 shows the error in achieving the true max-min rates for the same flows.

Figure 4.19 shows the rates achieved by the flows in network 4.8 using this scheduling scheme. Figure 4.20 shows the error in achieving the true max-min rates for the same flows.

For all three scenarios, small errors in achieving the true max-min fair rates indicate the accuracy of the scheduling scheme.

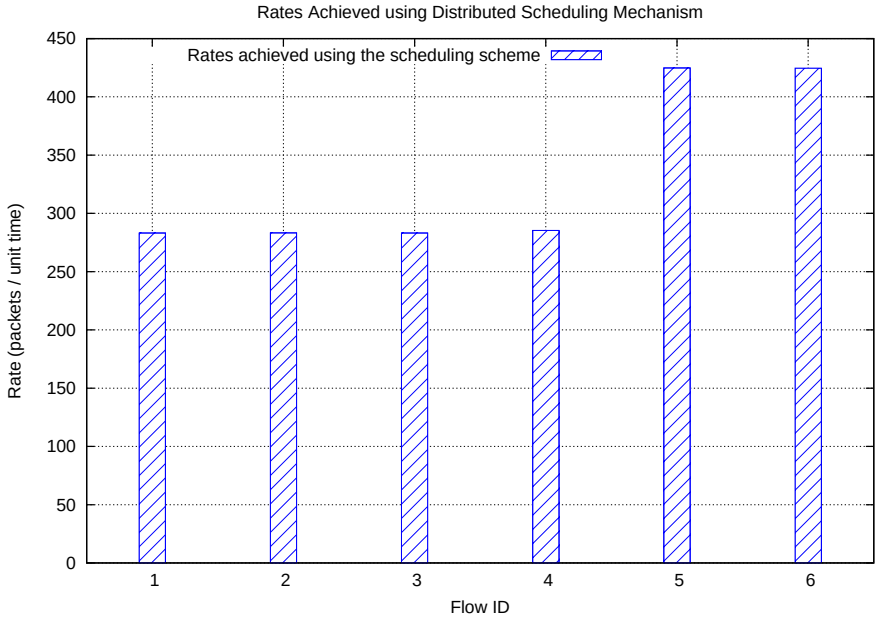


Figure 4.17: Flow rates achieved using distributed scheduling scheme for the network shown in figure 4.6

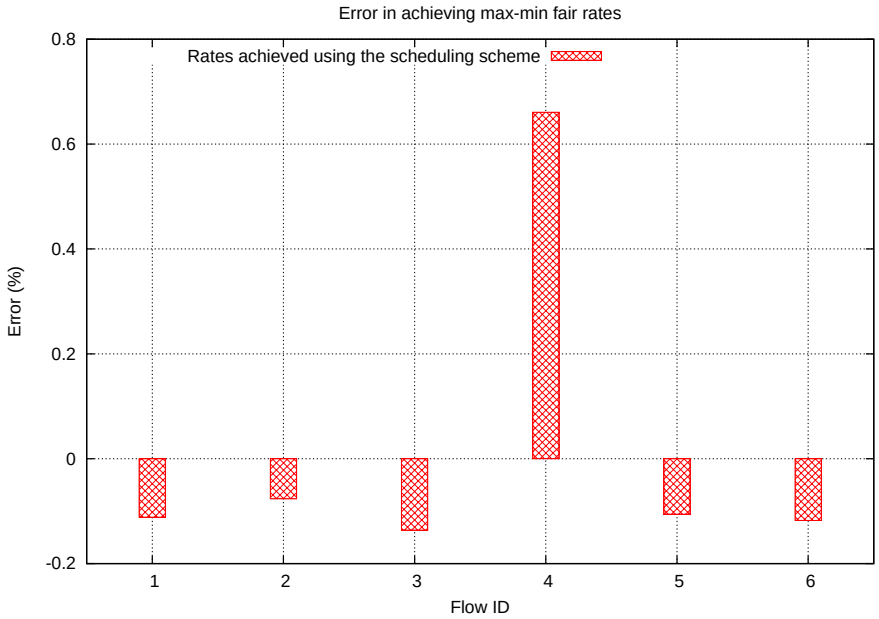


Figure 4.18: Error made by scheduling scheme in assigning max-min fair rates for the flows in figure 4.6

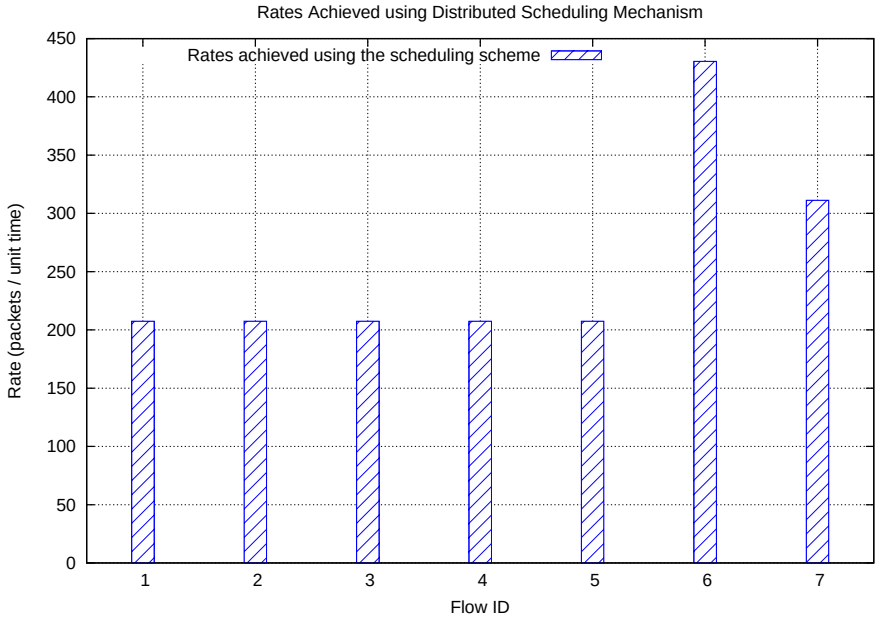


Figure 4.19: Flow rates achieved using distributed scheduling scheme for the network shown in figure 4.8

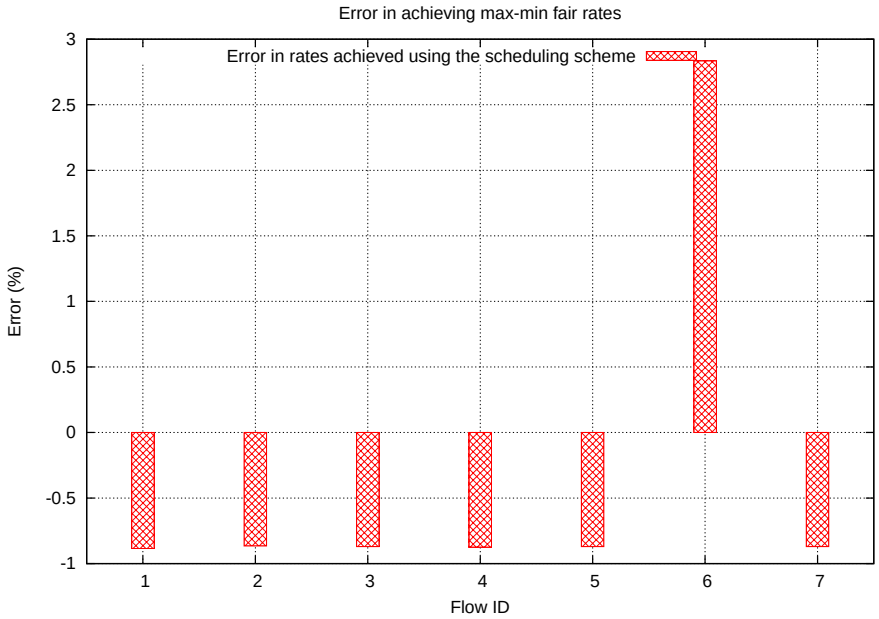


Figure 4.20: Error made by scheduling scheme in assigning max-min fair rates for the flows in figure 4.8

4.6 Summary

In this chapter we presented an algorithm to assign max-min fair rates to the flows in a network with coding. We emulated progressive filling on a wireless network with coding by coupling a conflict graph based framework with a linear program. We used various fairness metrics to compare our rate allocation algorithm with existing schemes, and showed that we our mechanism outperforms exiting algorithms in terms of fairness. We demonstrated that our fair rate allocation algorithm runs in polynomial time. We also presented a novel yet simple distributed fair rate assignment algorithm that achieves the same rates as the centralized version. We also presented centralized as well as distributed versions of a scheduling scheme that helps flows achieve max-min fair rates assigned using our rate control algorithm. We demonstrate that the error in achieving the max-min fair rates is significantly small, which indicates the accuracy of our distributed scheduling scheme.

Chapter 5

Conclusion

In this dissertation we presented several topics in the areas of routing and network coding. We presented a multipurpose multipath routing scheme. We proposed an efficient and exhaustive packet encoding algorithm that can integrate network coding with any routing scheme, irrespective of the routing mechanism's objective. We also presented max-min fair rate allocation and scheduling algorithms for the flows in a wireless network that use coding.

5.1 Polar Coordinate Routing

We presented a multipath routing mechanism called Polar Coordinate Routing. It can be used to serve multiple purposes, such as (a) congestion alleviation (b) reliable data delivery (c) security against an intercepting adversary etc. PCR sends packets on multiple different trajectories that are segments of circles with different radii. It also helps maintain a known separation between the paths. By varying the radii of these circles we can change the separation between different trajectories. This allows us to control the interference the nodes in these two trajectories will have. We demonstrated that even though PCR helps maintain a known separation between paths, it doesn't increase the hop count by too much. We presented a non-euclidean distance metric that helped packets travel in a fashion similar to the geographic routing. Our distance metric serves two purposes. It not only ensures

that the packet is travelling towards its destination, but it also makes sure that this forward progress is closer to the trajectory. We also presented rules to avoid routing loops while forwarding packets. We compared the performance of PCR with existing multipath schemes using the metrics such as average separation between trajectories, fraction of nodes in two trajectories that are out of each other's range, hop count etc. Our extensive simulations showed that PCR not only outperformed existing scheme, but it did so with a low variance, indicating the stability of our mechanism.

One of the issues that prevent successful data delivery is the presence of obstacles in the network. We integrated PCR with simple robotic routing. Robotic routing helps packets circumnavigate obstacles much like the robots in a maze. While travelling towards a destination using PCR's non-euclidean distance metric, if a packet encounters an obstacle, it switches its routing rule to robotic routing. The packet circumnavigates the obstacle using robotic routing, and after that switches its routing mode back to PCR in order to follow the path towards the destination on the predefined trajectory. We presented concrete rules to help packets switch their routing modes back and forth between PCR and robotic routing in order to overcome obstacles.

5.2 Packet Encoding for Network Coding

Next we presented our novel packet encoding algorithm to integrate a routing scheme with network coding. Our packet encoding algorithm is not dependent on PCR. In fact it can be coupled with any routing scheme. This helps us leverage the benefits offered by both an advanced routing scheme and an efficient packet encoding algorithm. Our packet encoding algorithm offers several benefits. (a) It searches a node's queue exhaustively to look for maximum number of packets that can be combined in a single transmission. (b) It can consider multiple next hop neighbor candidates for a packet. Hence it improves the probability of combining more packets in a single transmission. (c) The algorithm is asymptotically faster than a naïve exhaustive search. (d) It can be easily integrated with a routing scheme. We first presented our packet encoding algorithm as a binary integer program. We observed

that such a mathematical program falls under NP Complete complexity class. Hence we offered a novel bipartite graph based algorithm to look for an optimal packet combination. We demonstrated that looking for an optimal packet combination is analogous to enumerating cycles in a bipartite graph. We extended this algorithm to consider multiple next hop neighbor candidates for a packet. We gave examples to show that considering multiple next hop neighbor candidates may improve possibility of combining more packets. We also showed its throughput benefits using a simple simulation scenario.

We also integrated our packet encoding algorithm with a routing scheme. If we force packets to go to the neighbors that help yield a better packet combination, we may lose the advantages offered by the routing scheme. Therefore we presented rules to combine a routing scheme with our encoding algorithm such that we can reap the benefits offered by the routing scheme and enhanced packet encoding both. We coupled our packet encoding algorithm with a routing scheme where a packet's next hop neighbor is changed frequently in a dynamic manner. We also presented the throughput benefit offered by this combination of routing and packet encoding schemes.

5.3 Max-Min Fair Rate Allocation Algorithm

While calculating the throughput for different scenarios using our packet encoding algorithm, we assumed that each flow in the network is assigned the same rate. We observed that if we fluctuate the rates of some of the flows, the total throughput of the network would change disproportionately. Therefore we worked on the resource allocation problem as well. Namely, we presented an algorithm to calculate max-min fair rates for the flows in the network. We combined a conflict graph based framework with a simple linear program to allocate max-min fair rates to the flows in a wireless network with coding. While our algorithm emulated progressive filling to achieve max-min fairness, we demonstrated that this task wasn't straightforward. We pointed out caveats in setting up constraints of the linear program and selecting the bottleneck flows, such that the resulting rates wouldn't be incorrect or suboptimal. We first presented our algorithm in a centralized manner.

We demonstrated that its complexity is polynomial time. We ran our algorithm on a few topologies. We compared the rates allocated using our algorithm with prevailing rate control mechanism that rely on maximizing a utility function. We used metrics such as Jain's index and Max/Min rates to demonstrate that the rates assigned using our algorithm were fairer than the existing algorithms. We also presented a distributed rate allocation algorithm that helps achieve the same rates as the centralized algorithm. We also simulated how many total number of messages are transmitted network wide before the distributed algorithm achieves the same rates as the centralized algorithm.

Coming up with a rate control algorithm is not enough. Ideally we should also have a scheduling scheme that helps flows achieve their max-min fair rates. We presented centralized as well as distributed scheduling mechanisms that help flows achieve rates that are proportional to their max-min fair rates. We simulated the throughput that can be achieved using our scheduling scheme. We also calculated the error in achieving the max-min fair rates using this scheduling method. We demonstrated that the errors were significantly low, indicating the accuracy of our scheduling mechanism.

Bibliography

[Ahlswede *et al.*, 2000] Rudolf Ahlswede, Ning Cai, Shuo yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 46(4):1204–1216, 2000.

[Akyol *et al.*, 2008] Umut Akyol, Matthew Andrews, Piyush Gupta, John D. Hobby, Iraj Saniee, and Alexander L. Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. In *INFOCOM*, pages 619–627. IEEE, 2008.

[Bapeswara Rao and Murti, 1969] V.V. Bapeswara Rao and V.G.K. Murti. Enumeration of all circuits of a graph. *Proceedings of the IEEE*, 57(4):700 – 701, april 1969.

[Bertsekas and Gallager, 1992] Dimitri P. Bertsekas and Gallager. *Data Networks (2nd Edition)*. Prentice Hall, 2 edition, January 1992.

[Bose *et al.*, 1999] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, DIALM '99, pages 48–55, New York, NY, USA, 1999. ACM.

[Chou *et al.*, 2003] Philip A. Chou, Yunnan Wu, and Kamal Jain. Practical network coding, 2003.

[De Couto *et al.*, 2003] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. MobiCom '03, New York, NY, USA, 2003. ACM.

[Desai and Maxemchuk, 2010] Maulik Desai and Nicholas Maxemchuk. Polar coordinate routing for multiple paths in wireless networks. In *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, pages 1 –9, June 2010.

[Dong *et al.*, 2006] Q. Dong, S. Banerjee, and B. Liu. Throughput optimization and fair bandwidth allocation in multi-hop wireless lans. In *INFOCOM*, 2006.

[Dong *et al.*, 2007] Qunfeng Dong, Jianming Wu, Wenjun Hu, and Jon Crowcroft. Practical network coding in wireless networks. In Evangelos Kranakis, Jennifer C. Hou, and Ram Ramanathan, editors, *MOBICOM*, pages 306–309. ACM, 2007.

[Finn, 1987] G. G. Finn. Routing and Addressing Problems in Large Metropolitan-Scale Internetworks, 1987.

[Floyd, 1967] Robert W. Floyd. Nondeterministic algorithms. *J. ACM*, 14(4):636–644, October 1967.

[Gambiroza *et al.*, 2004] Violeta Gambiroza, Bahareh Sadeghi, and Edward W. Knightly. End-to-end performance and fairness in multihop wireless backhaul networks. In *In Proceedings of ACM MOBICOM*, pages 287–301, 2004.

[Gross and Yellen, 2005] Jonathan L. Gross and Jay Yellen. *Graph Theory and Its Applications*. 2005.

[Gupta and Stolyar, 2006] P. Gupta and A. L. Stolyar. Optimal Throughput Allocation in General Random-Access Networks. In *Proc. ˜of CISS*, Princeton, NJ, March 2006.

[Gupta *et al.*, 2005] P. Gupta, Yogesh Sankarasubramaniam, and Alexander L. Stolyar. Random-access scheduling with service differentiation in wireless networks. In *INFOCOM*, pages 1815–1825. IEEE, 2005.

[Ho *et al.*, 2003] T. Ho, B. Leong, M. Medard, R. Koetter, Y. Chang, and M. Effros. The benefits of coding over routing in a randomized setting. In *Proc. IEEE ISIT'03*, june 2003.

[Huang and Bensaou, 2001] Xiao Long Huang and Brahim Bensaou. On max-min fairness and scheduling in wireless ad-hoc networks: analytical framework and implementation. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '01, pages 221–231, New York, NY, USA, 2001. ACM.

[Jaffe, 1981] J. Jaffe. Bottleneck Flow Control. *Communications, IEEE Transactions on*, 29(7), July 1981.

[Jain *et al.*, 1984] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical report, Digital Equipment Corporation, September 1984.

[Johnson, 1975] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.

[Karp and Kung, 2000] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[Katti *et al.*, 2006] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, 2006.

[Kelly *et al.*, 1998] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *The Journal of the Operational Research Society*, 49(3):237–252, 1998.

[Kim and Maxemchuk, 2005] Daejoong Kim and Nick Maxemchuk. Simple robotic routing in ad hoc networks. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, pages 159–168, Washington, DC, USA, 2005. IEEE Computer Society.

[Kim *et al.*, 2005] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 217–230, Berkeley, CA, USA, 2005. USENIX Association.

[Koetter *et al.*, 2003] Ralf Koetter, Muriel Mdard, and Senior Member. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782–795, 2003.

[Kranakis *et al.*, 1999] Evangelos Kranakis, Harvinder Singh, and Jorge Urrutia. Compass routing on geometric networks. In *IN PROC. 11 TH CANADIAN CONFERENCE ON COMPUTATIONAL GEOMETRY*, pages 51–54, 1999.

[Kuhn *et al.*, 2003] Fabian Kuhn, Rogert Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pages 63–72, New York, NY, USA, 2003. ACM.

[L. Kleinrock, 1978] J.A. Silvester L. Kleinrock. Optimum transmission radii for packet radio networks or why six is a magic number. In *IEEE National Conference on Telecommunication*, 1978.

[Le and Lui, 2008] Jilin Le and John C. S. Lui. How many packets can we encode? - an analysis of practical wireless network coding. In *Proceedings of IEEE INFOCOM*, 2008.

[Li and Li, 2004] Z. Li and B. Li. Network coding: The case for multiple unicast sessions. In *Proc. of Allerton*, sep 2004.

[Li *et al.*,] Li Li, R. Ramjee, M. Buddhikot, and S. Miller. Network coding-based broadcast in mobile ad-hoc networks. In *INFOCOM 2007*.

[Lin and Shroff, 2004] Xiaojun Lin and Ness B. Shroff. Joint rate control and scheduling in multihop wireless networks. In *in Proceedings of IEEE Conference on Decision and Control*, pages 1484–1489, 2004.

[Liu and Wang, 2006] Hongbo Liu and Jiaxin Wang. A new way to enumerate cycles in graph. In *Telecommunications, 2006. AICT-ICIW '06.*, page 57, feb. 2006.

[MATLAB, 2010] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[Nandagopal *et al.*, 2000] Thyagarajan Nandagopal, Tae-Eun Kim, Xia Gao, and Vaduvur Bharghavan. Achieving mac layer fairness in wireless packet networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, pages 87–98, New York, NY, USA, 2000. ACM.

[Niculescu and Nath, 2003] Dragos Niculescu and Badri Nath. Trajectory based forwarding and its applications. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 260–272, New York, NY, USA, 2003. ACM.

[Omiwade *et al.*, 2008] Soji Omiwade, Rong Zheng, and Cunqing Hua. Butterflies in the mesh: lightweight localized wireless network coding. 2008.

[Popa *et al.*, 2006] Lucian Popa, Costin Raiciu, Ion Stoica, and David Rosenblum. Reducing congestion effects in wireless networks by multipath routing. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 96–105, Washington, DC, USA, 2006. IEEE Computer Society.

[Porter and Ji, 2004] George Porter and Minwen Ji. Delta routing: improving the price-performance of hybrid private networks. In *NOMS (1)*, 2004.

[Rangwala *et al.*, 2006] Sumit Rangwala, Ramakrishna Gummadi, Ramesh Govindan, and Konstantinos Psounis. Interference-aware fair rate control in wireless sensor networks. In *In Proceedings of the ACM SIGCOMM*, pages 63–74, 2006.

[Ronasi *et al.*, 2009] Keivan Ronasi, Amir Hamed Mohsenian Rad, Vincent W. S. Wong, Sathish Gopalakrishnan, and Robert Schober. Reliability-based rate allocation in wireless inter-session network coding systems. In *GLOBECOM*, pages 1–6. IEEE, 2009.

[Rudin, 1976] H. Rudin. On routing and "delta routing": A taxonomy and performance comparison of techniques for packet-switched networks. *Communications, IEEE Transactions on*, 24(1):43 – 59, jan 1976.

[Seferoglu and Markopoulou, 2009] Hulya Seferoglu and Athina Markopoulou. Distributed rate control for video streaming over wireless networks with intersession network coding. In *In Packet Video*, 2009.

[Seferoglu *et al.*, 2009] Hulya Seferoglu, Athina Markopoulou, and Ulas Kozat. Network coding-aware rate control and scheduling in wireless networks. In *Proceedings of the 2009 IEEE international conference on Multimedia and Expo*, ICME'09, pages 1496–1499, Piscataway, NJ, USA, 2009. IEEE Press.

[Seferoglu *et al.*, 2011] Hulya Seferoglu, Athina Markopoulou, and K. K. Ramakrishnan. I2nc: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM*, pages 1035–1043. IEEE, 2011.

[Sengupta *et al.*, 2007] Sudipta Sengupta, Shravan Rayanchu, and Suman Banerjee. An analysis of wireless network coding for unicast sessions: The case for coding-aware routing. In *in Proc. of IEEE INFOCOM*, pages 1028–1036, 2007.

[Sridharan and Krishnamachari, 2007] Avinash Sridharan and Bhaskar Krishnamachari. Maximizing network utilization with max-min fairness in wireless sensor networks. In *WiOpt*, pages 1–9. IEEE, 2007.

[Sundararajan *et al.*, 2008] Jay Kumar Sundararajan, Devavrat Shah, Muriel Médard, Michael Mitzenmacher, and João Barros. Network coding meets tcp. *CoRR*, abs/0809.5022, 2008.

[Tarjan, 1972] Robert Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, page 211, 1972.

[Tiernan, 1970] James C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Commun. ACM*, 13(12):722–726, December 1970.

[Wu *et al.*, 2005] Y. Wu, P. A. Chou, and S. Y. Kung. Information exchange in wireless networks with network coding and physical-layer broadcast. In *Proc. 39th Annual Conf. Inform. Sci. and Systems (CISS)*, 2005.

[yen Robert Li *et al.*, 2003] Shuo yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49:371–381, 2003.

[Zhao and Medard, 2010] Fang Zhao and Muriel Medard. On analyzing and improving cope performance. In *ITA*, pages 317–322. IEEE, 2010.