

Online Algorithms for Routing and Scheduling on Ring Networks

Rohit Bansal

Department of Mathematics and Computer Science
Denison University
Granville, OH 43023 USA
bansal_r@denison.edu

Abstract

The online packet routing problem is studied in the context of moving data packets efficiently among network nodes. This problem is a special case of the more general $k - k$ routing problem where it is assumed that all packets are known to an algorithm before any packet is sent and each node sends and receives exactly k packets. The online version relaxes these constraints and allows for packets to have arbitrary release times and for any number of packets to originate at or to be delivered to any node. The goal of an algorithm is to construct a schedule for any instance with small makespan (maximum completion time) relative to the optimal makespan for a worst case instance. Some algorithms are known for this problem on linear array networks and bidirectional rings. In this paper we investigate better algorithms for rings. First, we provide computer based simulation results in support of an earlier proof that FARTHEST FIRST (FF) scheduling is optimal with respect to makespan on linear arrays. Next we compare different routing algorithms [random routing (RR), probabilistic routing (PR), greedy routing (GR), maximum queue length (MQL) and maximum queue length 1 (MQL1)] to shortest path routing (SP) combined with FF scheduling. We then show that none of the above routing algorithms combined with FF is optimal with respect to makespan on bidirectional rings. Lastly, we show that routing affects makespan more than scheduling affects makespan on rings in the average case.

1 Introduction & Motivation

We study an online packet routing problem in which an arbitrary number of data packets having arbitrary release times are sent periodically from network nodes to other network nodes. An online algorithm must choose a route and a schedule for each packet as it arrives so that no two packets cross the same network link at the same time [5].

1.1 Problem Definition

In this problem, we consider full-duplex linear array and ring interconnection networks. A linear array network contains n nodes labeled $\{0, 1, 2, \dots, n - 1\}$ and $m = 2(n - 1)$ directed links $\{(i, i + 1), (i + 1, i) : i = 0, 1, \dots, n - 2\}$. A ring has n nodes and $m = 2n$ directed links $\{(i, (i + 1) \bmod n), ((i + 1) \bmod n, i) : i = 0, 1, \dots, n - 1\}$. The input to the problem is a sequence of packets $\sigma = p_1, p_2, \dots, p_k$, ordered by release time. Each $p_j = (s_j, t_j, a_j)$, where s_j is the packet's source node, t_j is the packet's destination node, and a_j is the packet's release time. Let P_j be the (monotonic) route assigned to packet p_j , and let $P_j(i)$ denote the i^{th} link in the route.

Each node on the network with incoming links has a queue for each of its outgoing links to temporarily store the packets being forwarded over that link. At each discrete time step $t \geq 1$, which represents a continuous time interval $(t - 1, t]$, each node decides whether each of the packets in its queues should remain in the queue or be the next packet to be forwarded over the queue's adjoining link during the next time step. If a packet leaves a node at time interval t then it would reach the next node on its path at time $t + 1$. In our problem we assume that no packets are delayed due to a full queue at its next link. Rather, packets are only delayed when the underlying capacity of the link is exceeded or the algorithm decides to hold the packets for some other reason. A schedule for the packet p_j is a function S_j where $S_j(i)$ is the time step during which packet p_j will cross link $P_j(i)$. Let C_j be the completion time of the packet p_j . The goal of the algorithm is to come up with a schedule to minimize the makespan or, in other words, minimize the maximum completion time of its schedule.

1.2 Online Algorithms & Competitive Ratio

An optimal offline algorithm receives the entire sequence of requests in advance and is required to take an action in response to each request. However the choice of each action can be based on the entire sequence of requests. An online counterpart on the other hand receives a sequence of requests and performs an immediate action in response to each request without prior knowledge of requests arriving in the future. Competitive ratio is a measure of performance for an online algorithm in which the online algorithm is evaluated by comparing its cost for a worst case request sequence with that of an optimal offline algorithm processing the same sequence of requests [4].

1.3 Packet Scheduling Algorithms

In this section we will study three different packet scheduling algorithms. First, LONGEST IN SYSTEM (LIS) [3] is an online scheduling algorithm that selects a packet from each queue in the order it appears in the sequence. The second scheduling algorithm, MOVING PRIORITY (MP) gives priority to packets that are passing through a node over the packets that originate at this node. Lastly, the third scheduling algorithm we study is FARTHEST FIRST (FF). This algorithm selects the packet which has the maximum distance left yet to travel to their destination to be forwarded next.

1.4 Past Research and Their Results

Havill [1] proved that FF scheduling is optimal with respect to makespan on linear arrays. He also showed that two other scheduling algorithms LIS and MP have competitive ratio 2 with respect to makespan on linear arrays. For bidirectional rings he proved that the competitive ratio of shortest path routing combined with FF scheduling is 2.

1.4.1 Scheduling with FF

In FF scheduling the algorithm gives priority to packets that have the farthest distance yet to travel to their destinations. To provide empirical evidence to the claim that FF scheduling is optimal on linear arrays, we compare it with an optimal offline backtrack algorithm (*OPT*) that generates all possible schedules and then selects the one with the least makespan. The pseudocode for *OPT* is given below.

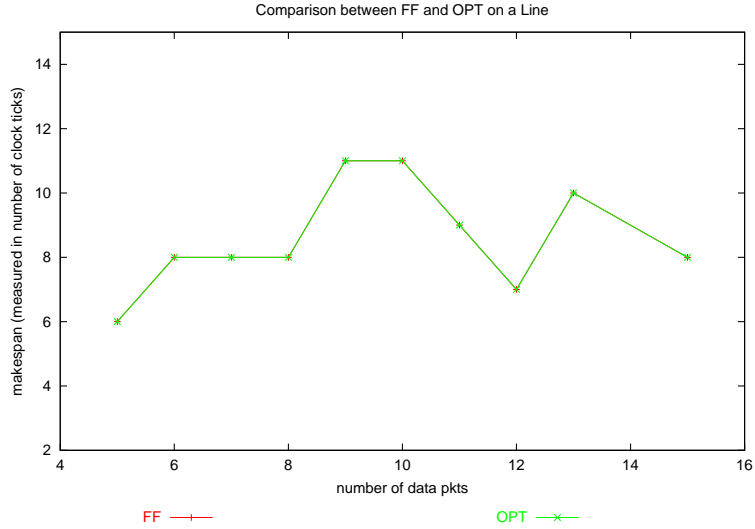


Figure 1: Comparison of FF scheduling and OPT on a Linear Array

Backtrack(timeStep k , network N)

```

for each packet  $p_i \in$  each node  $\in$  network  $N$ 
  select a packet  $p_i$  from each of  $k$  nodes  $\in$  network  $N$ 
    if (Ok( $p_i, p_{i+1}, \dots, p_k, N$ )) // returns true if its ok to move each of these pkts
      Move( $p_i, p_{i+1}, \dots, p_k, N$ )
    else
      continue
  if all queues in network  $N$  are  $\emptyset$  // we have a solution
    if (bestMakespan >  $k + 1$ )
      bestMakespan =  $k + 1$ 
  else
    Backtrack( $k + 1, N$ ) // recursive call with the updated network
    Restore( $N$ ) // restore to the original state

```

The graph in figure 1 compares OPT with FF scheduling. As we can see, FF computes the optimal makespan for each of these tested cases. We were only able to provide results for small numbers of data packets because the optimal algorithm takes too long to run.

1.4.2 Shortest Path Routing combined with FF

Shortest Path routing routes all the packets on their shortest path and because of this performs worse than an algorithm that also takes into account for congestion at the queues.

Theorem 1 *Shortest Path (SP) routing combined with FF scheduling is not optimal with respect to makespan on a ring network.*

Proof Let us assume that SP is optimal. Then for any input sequence σ , SP should create a schedule with the least makespan. Let us consider the following input sequence on a full-duplex ring with five nodes: $\sigma = (0, 2, 0), (0, 2, 0), (0, 2, 0), (0, 2, 0)$. Routing with SP results in a makespan

of five, but there is a better schedule that results in a lower makespan. Please refer to Table 1. SP routes all the packets on their shortest path. Because of this it performs worse compared to an algorithm which chooses to route some of the packets on their longer path. \square

	(0,1)	(1,2)	(2,3)	(3,4)	(4,0)	(0,4)	(4,3)	(3,2)	(2,1)	(1,0)
1	(0,2,0)									
2	(0,2,0)	(0,2,0)								
3	(0,2,0)	(0,2,0)								
4	(0,2,0)	(0,2,0)								
5		(0,2,0)								

	(0,1)	(1,2)	(2,3)	(3,4)	(4,0)	(0,4)	(4,3)	(3,2)	(2,1)	(1,0)
1	(0,2,0)					(0,2,0)				
2	(0,2,0)	(0,2,0)					(0,2,0)			
3	(0,2,0)	(0,2,0)						(0,2,0)		
4		(0,2,0)								

Table 1: FF schedule on a ring (top) and a better schedule (bottom)

One of the primary goals of this research is to find algorithms which perform better than Shortest Path routing. We summarize our results below.

2 Routing vs. Scheduling

For any non trivial network like a ring, mesh, or a torus, we can divide the online packet scheduling problem into two distinct yet connected problems. One is the packet routing problem, in which we decide a route at the source for each packet to be routed. The other is the scheduling problem which selects a packet from each queue at each time step to be forwarded next on its link.

In the literature, the packet routing problem [2] on rings has been studied extensively with respect to different scheduling algorithms (FF, MP, LIS). However earlier research on rings has been limited to a single routing algorithm (SP). Studying this problem in the light of the current simulation results has shown two interesting results. First, there is little difference in average cases between different scheduling algorithms. This is irrespective of the routing discipline chosen. Second, the makespan varies greatly amongst different routing algorithms. Refer to figures 2 and 3. The first of the two graphs shows a comparison between SP and Random Routing (RR) in combination with three different scheduling algorithms Longest in System (LIS), Moving Priority (MP), and Farthest First (FF). The second graph similarly compares SP and Probabilistic Routing. A first observation is that SP does far better in both the cases. And on a more detailed examination we observe that there is very little difference among the three scheduling algorithms for either of the routing algorithms.

3 Routing algorithms on Rings

For bidirectional rings we compare five different routing algorithms to SP routing. Our earlier results have shown that there is little difference amongst the three different scheduling algorithms (MP, LIS, FF) and FF is optimal amongst them. Therefore from now on, we always choose FF as

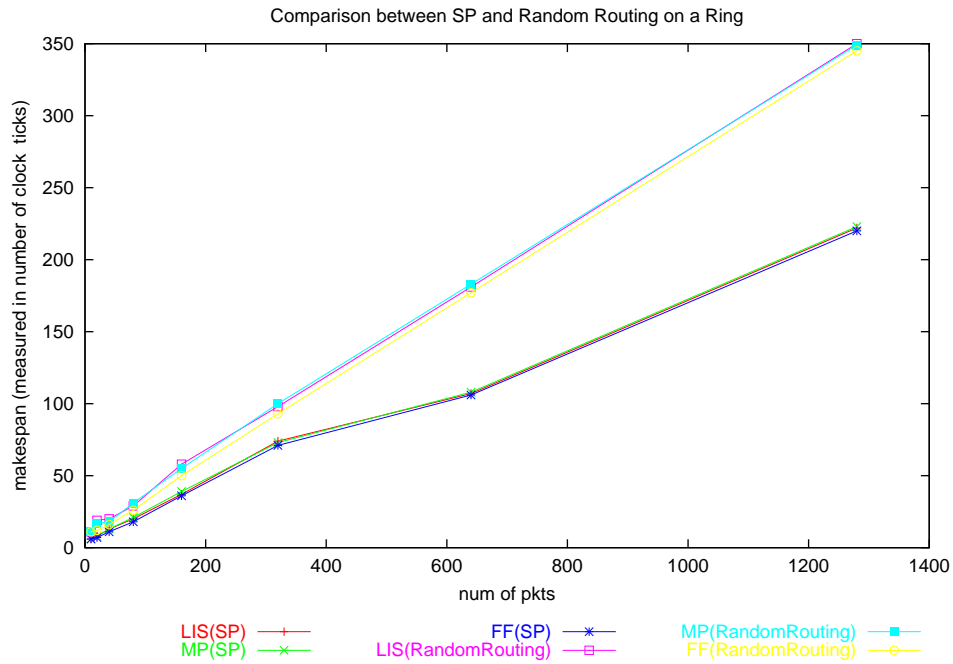


Figure 2: Shortest Path routing compared to Random Routing with several scheduling algorithms.

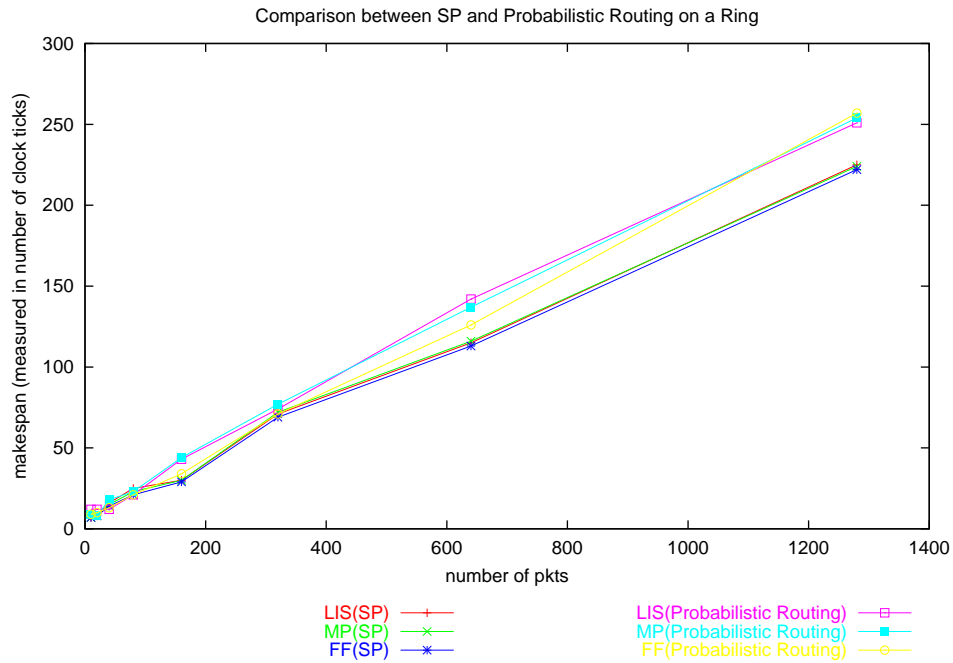


Figure 3: Shortest Path routing compared to Probabilistic Routing with several scheduling algorithms.

the scheduling algorithm. A generic outline for an online algorithm is described below, which we will make use of in the rest of this section.

OnlineAlgorithm(Algorithm Alg)

```

ReadPktsFromFile()
while (!StableState() and !AllPktsArrived)
  Timer→OneTick(); // advance a clocktick
  if (!AllPktArrived) // set true when all pkts have arrived
    do
      newPkt=PktAvail(Timer→t) // get a pkt released at time t
      Routing Discipline(newPkt) // make the routing decision
      Update Queues(newPkt) // add newPkt to appropriate queue
    while(newPkt != NULL)
  MovePkts(Alg) // choose a pkt from each queue at every node and move it
  // to the next node on its path
bestMakespan = timer → t + 1

```

We now introduce the 5 different routing algorithms for bidirectional rings.

3.1 Random Routing (RR)

This algorithm selects the short path with probability 3/4 and the long path with probability 1/4. The motivation here is that the nondeterministic factor will help to route some of the packets on the longer path and in turn help to minimize the makespan.

3.2 Probabilistic Routing (PR)

This algorithm computes a ratio β between the shortest path and the total ring length for each packet, and then routes this packet on its shortest path with probability $(1 - \beta)$ or on its longer path with probability β . The intuition here is that the shorter the shortest path the higher is the probability that it should be to sent on its shortest path.

3.3 Max Queue Length (MQL)

This algorithm computes an estimated completion time for each packet by summing the path length (distance from source to destination) and the maximum queue length on each of its two paths. The algorithm routes the packet in the direction which has the least estimated completion time for that packet. The intuition here is to take into account both the delay due to congestion in the queues and the actual distance traveled by the packet.

3.4 Max Queue Length 1 (MQL1)

This algorithm also computes an estimated completion time for each packet. However, unlike the previous algorithm, which simply takes into account the maximum queue length on each path, this algorithm only takes into consideration the maximum number of packets which would delay the current packet in a Farthest First schedule.

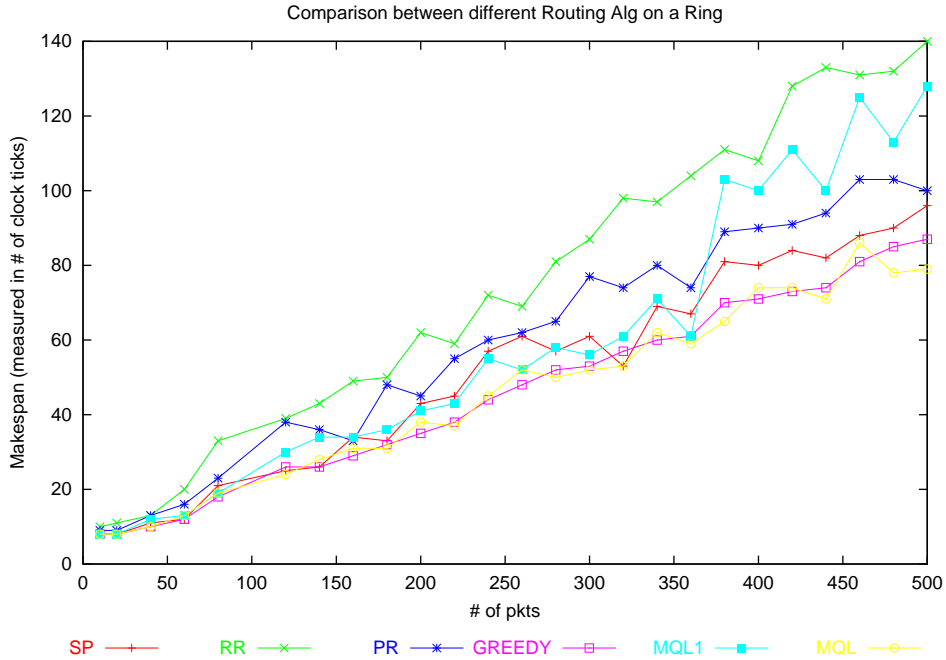


Figure 4: Comparison between different Routing algorithms all using FF scheduling on a Ring when the number of packets ranges from 10 - 500

3.5 GREEDY (GR)

Havill [1] showed that Farthest First constructs an optimal schedule on a linear array with makespan $\max_{v,t \leq T_v} \{N_{\geq}(v,t) + t\} - 1$, where $N_{\geq}(v,t)$ is the number of packets that would arrive at node v at time t or later if they were not delayed and T_v is the latest time a packet would arrive at node v if it were not delayed. The GREEDY algorithm chooses the path that minimizes this quantity. In this process the algorithm fills in a table for each direction to compute the above maximum. The packet to be routed is tentatively scheduled on both the long and the short path. The tables for both directions are updated. The packet is then routed on the path with the least maximum completion time while the table in the other direction is restored. The idea here is to see how the current packet affects the overall makespan when routed in either direction and then make the greedy choice.

4 Comparison between different Routing Algorithms on a Ring

We now compare the above five algorithms with shortest path routing. For all our simulations we assume $n = 10$, the release time for all the packets is in $\{0, 1, 2, 3, 4\}$ and all packets are uniformly randomly generated. The graphs in figures 4, 5, and 6 all compare different routing algorithms on a ring for an average case request sequence. The graph in figure 4 compares the six different routing algorithms on instances between ten and five hundred packets. As we can see from the graph, RR performs the worst, and both MQL and GREEDY defeat SP routing. If we quantify this difference we find that, on average, SP routing is 8% - 8.5% worse than MQL and 8.5% - 9% worse than GREEDY. The graphs in figure 5 and 6 show the comparison for larger numbers of packets. There

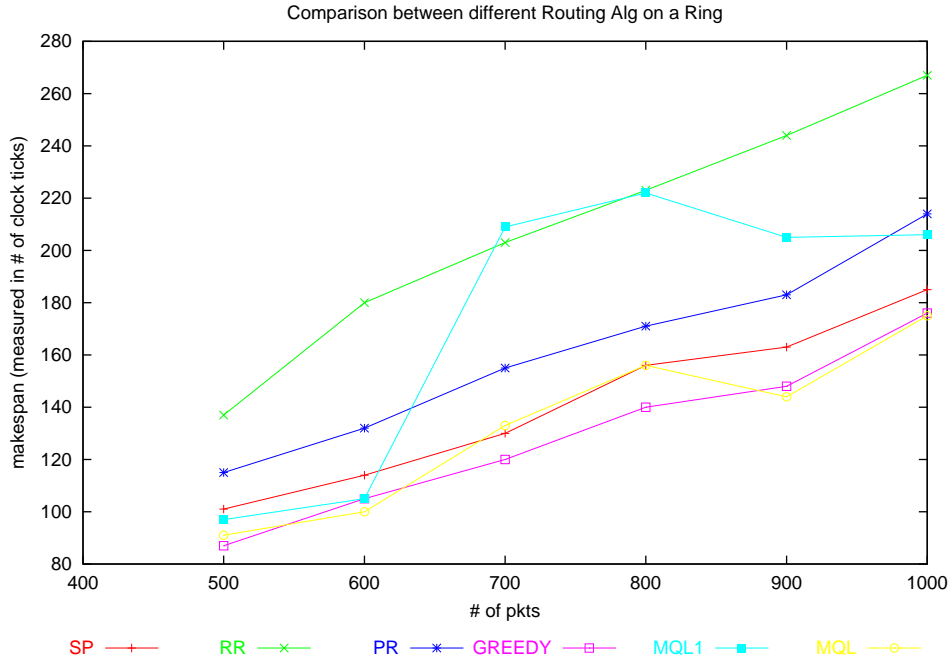


Figure 5: Comparison between different Routing algorithms all using FF scheduling on a Ring when the number of packets ranges from 400 - 1000

are three interesting observations we make from these two latter graphs:

1. As the number of data packets increases, MQL1 performs the worst amongst all routing algorithms, which differs from the results we had on a small range.
2. MQL and GREEDY continue to defeat SP routing (6%–6.5% worse than MQL and 7%–7.5% worse than GREEDY).
3. After about eight thousand packets, the difference between GREEDY and MQL starts to widen.

In order to make some sense of these results we need to closely examine each of these algorithms. Here are some possible explanations for these results:

- **Random Routing** – The probability is not based upon any heuristics or on actual requests. This algorithm naively ends up routing far too many packets incorrectly on the longer path, and because of this flaw it performs the worst amongst all the routing algorithms most of the time.
- **Probabilistic Routing** – This algorithm fails to take into account the delay due to any congestion at the queues. Some of the packets which may have ideally arrived earlier if they were routed on the longer path end up getting delayed at the queues on the short path. Because of this flaw this algorithm ends up performing worse than SP routing.
- **Max Queue Length** – Although this algorithm performs the best amongst all the other routing algorithms, it is pessimistic. By taking the maximum queue length into account, it

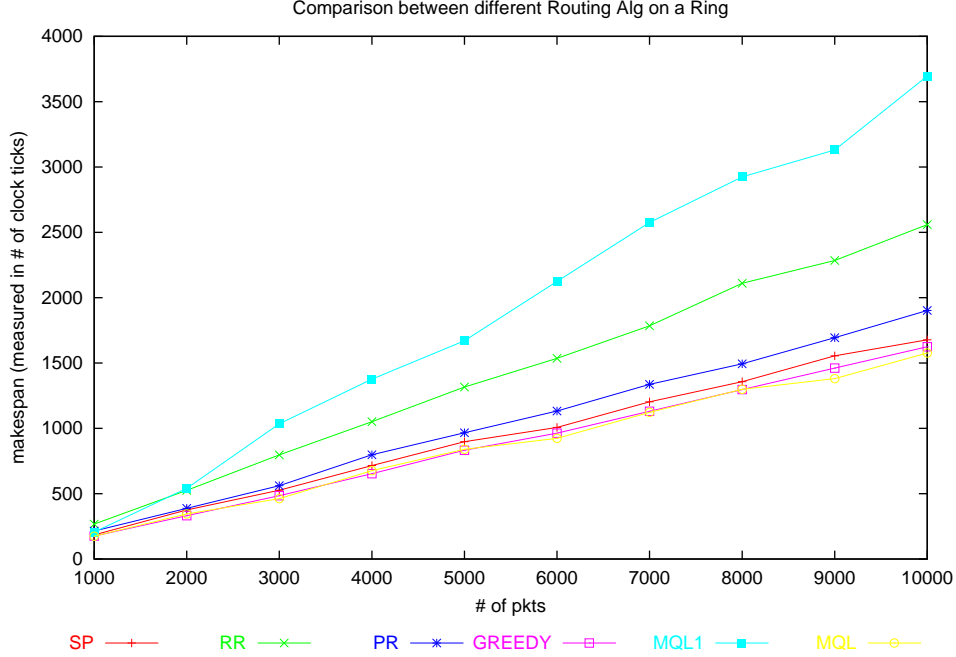


Figure 6: Comparison between different Routing algorithms all using FF scheduling on a Ring when the number of packets ranges from 1000 - 10000

computes the worst case scenario by assuming that all the previous packets in the queue will delay the current packet, which may not always be true. This flaw in the algorithm prevents it from scheduling many packets on the longer path, and hence is defeated by GREEDY for small instances. However, for large numbers of packets (eight thousand packets and above), the average queue length in both directions is pretty close, so most of the packets are routed on their shorter path. Other routing algorithms that decide to schedule these later packets on their longer path end up delaying the packets that have been scheduled before. Because the later packets have a distance further than half way around to travel.

We will now prove that max queue length routing algorithm is not optimal with respect to makespan on a ring network.

Theorem 2 : *The Max Queue Length routing algorithm is not optimal with respect to makespan on a ring network.*

Proof Let us assume that MQL is optimal. Then for any input sequence σ , MQL should create a schedule with the least makespan. Let us consider the following input sequence on a full-duplex ring with four nodes, $\sigma = (2,2,0), (2,2,0), (2,1,1), (1,0,1), (3,2,1), (2,1,1), (0,2,1), (0,2,1), (3,0,1), (1,2,2), (1,2,2), (2,1,2), (3,1,2), (0,2,2), (2,0,3), (0,1,3), (1,0,3), (3,0,3), (0,3,3), (1,0,3)$. Routing with MQL results in a makespan of six, but schedule constructed by GREEDY results in a makespan of five. There by contradicting our earlier assumption that MQL was optimal. Please refer to Table 2. \square

- **Max Queue Length 1** – This algorithm is too optimistic because it only counts packets that will ultimately delay the current packet. By doing this the current packet delays packets

	(0,1)	(1,2)	(2,3)	(3,0)	(0,3)	(3,2)	(2,1)	(1,0)
1			(2,0,0)				(2,0,0)	
2	(0,2,1)			(2,0,0)	(0,2,1)	(3,2,1)	(2,1,1)	(2,0,0)
3		(0,2,1)		(3,1,2)	(0,2,2)	(0,2,1)	(2,1,1)	(1,0,1)
4	(3,1,2)	(1,2,2)		(3,0,1)	(0,3,3)	(0,2,2)	(2,0,3)	(1,0,3)
5	(0,1,3)	(1,2,2)		(3,0,3)			(2,1,2)	(2,0,3)
6								(1,0,3)

	(0,1)	(1,2)	(2,3)	(3,0)	(0,3)	(3,2)	(2,1)	(1,0)
1			(2,0,0)				(2,0,0)	
2	(0,2,1)			(2,0,0)	(0,2,1)	(3,2,1)	(2,1,1)	(2,0,0)
3		(0,2,1)		(3,0,1)	(0,2,2)	(3,1,2)	(2,1,1)	(1,0,1)
4	(0,1,3)	(1,2,2)	(2,0,3)	(3,0,3)	(0,3,3)	(0,2,2)	(3,1,2)	(1,0,3)
5		(1,2,2)		(2,0,3)		(0,2,1)	(2,1,2)	(1,0,3)

Table 2: MQL schedule on a ring (top) and a better GREEDY schedule (bottom)

that have already been scheduled before. For a large number of packets this negatively affects the schedule making it perform the worst amongst all.

- **GREEDY** – This algorithm performs better than SP routing. However at times it is defeated by MQL. The GREEDY algorithm makes a locally optimal routing decision. However because of the way FF scheduling selects packets from the queue, current packet ends up delaying other packets that have already been scheduled. We try and illustrate this more explicitly by the following proof.

Theorem 3 :*The GREEDY routing algorithm is not optimal with respect to makespan on a ring network.*

Proof Lets assume that GREEDY is optimal. Then for any input sequence σ , GREEDY should create a schedule with the least makespan. Let us consider the following input sequence on a full-duplex ring with four nodes,: $\sigma = (1,3,0), (1,3,1), (0,2,2), (0,2,2), (2,1,2), (0,2,2), (0,3,3)$. Routing with GREEDY results in a makespan of six, but schedule constructed by MQL results in a makespan of five. Therefore contradicting our earlier assumption that GREEDY was optimal. Please refer to Table 3. \square

On average case input sequences we have seen that some of the more adaptive algorithms perform worse than SP routing. However, in a worst case instance for SP in which all packets have the same destination, source and arrival time, all of these adaptive algorithms perform better than SP. The graph in figure 7 shows the result of a simulation where all the packets destinating are half way around the ring while the graph in figure 8 shows results all packets' destinations are one fifth of the way around. In both these graphs MQL, MQL1 and GREEDY perform the same and are better than all the other algorithms.

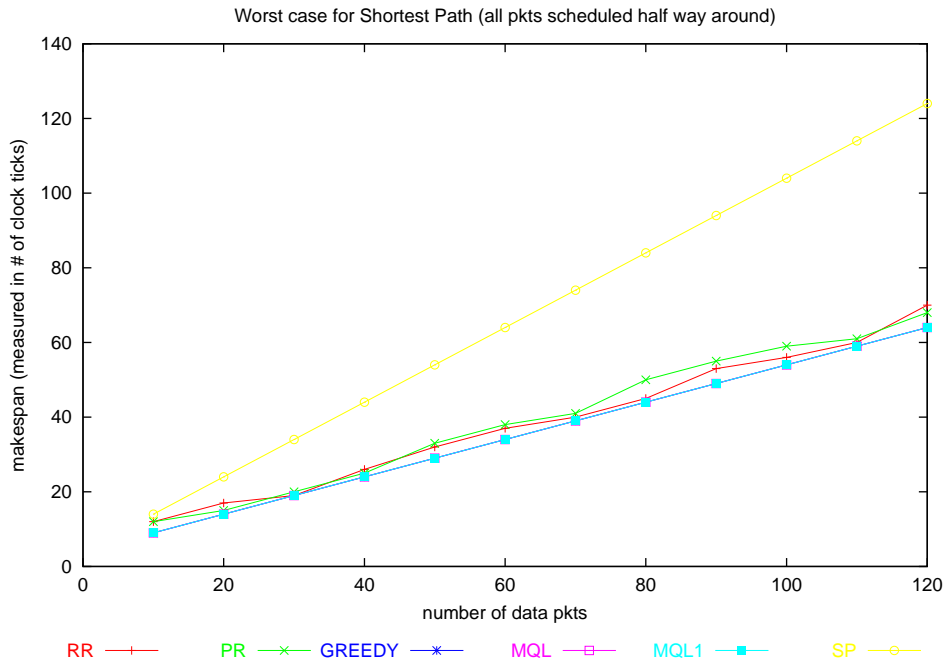


Figure 7: Comparison between different Routing algorithms all using FF scheduling on a Ring when all the packets are scheduled half way around.

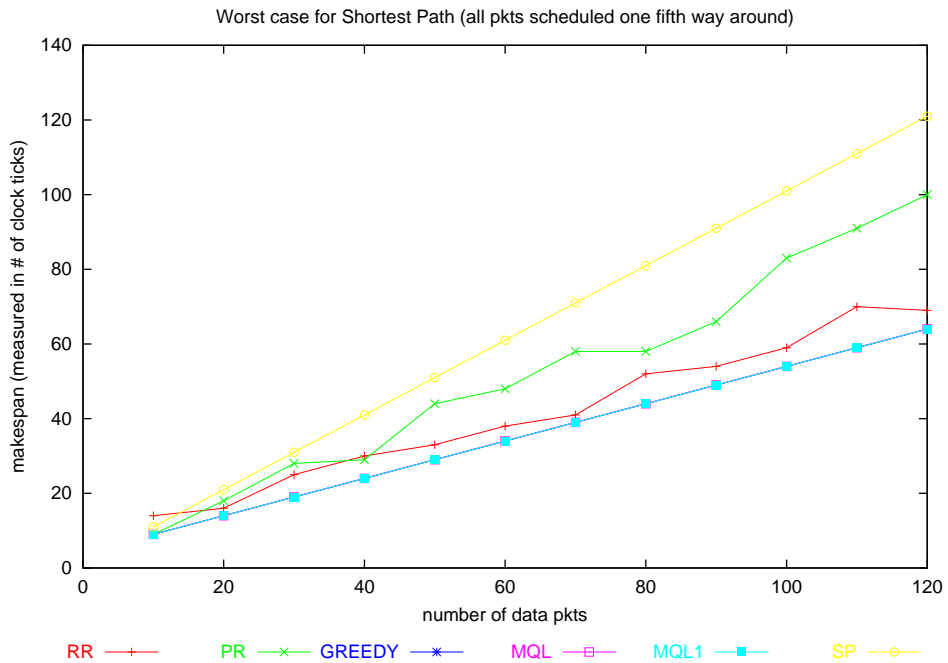


Figure 8: Comparison between different Routing algorithms all using FF scheduling on a Ring when all the packets are scheduled 1/5 way around

	(0,1)	(1,2)	(2,3)	(3,0)	(0,3)	(3,2)	(2,1)	(1,0)
1								(1,3,0)
2					(1,3,0)			(1,3,1)
3	(0,2,2)				(0,2,2)		(2,1,2)	
4		(0,2,2)			(0,2,2)			
5					(1,3,1)	(0,2,2)		
6					(0,3,3)			

	(0,1)	(1,2)	(2,3)	(3,0)	(0,3)	(3,2)	(2,1)	(1,0)
1								(1,3,0)
2		(1,3,1)			(1,3,0)			
3	(0,2,2)		(1,3,1)		(0,2,2)		(2,1,2)	
4		(0,2,2)			(0,2,2)	(0,2,2)		
5					(0,3,3)	(0,2,2)		

Table 3: GREEDY schedule on a ring (top) and a better MQL schedule (bottom)

5 Conclusions

We have studied five different algorithms to schedule online packets instances on a ring. We have shown that MQL and GREEDY both achieve a lower makespan than SP routing on a average case instance. We have also proved that neither SP, MQL nor GREEDY is optimal for all input sequences. In reality, an optimal online algorithm which performs well on all input sequences may not exist.

Our results so far have been based only on simulations. These results give us some insight into the problem, however we cannot prove anything simply based on these results. In order to show that MQL and GREEDY achieve lower makespan than SP in the worst case, we would need to do an analysis to find the competitive ratio for both MQL and GREEDY and compare it to that of SP.

References

- [1] Jessen T. Havill. Online Packet Routing on Linear Arrays and Rings. *In Proc. 28th ICALP, LNCS 2076, pp. 773-784, 2001.*
- [2] Fillia Makedon and Antonios Symvonis. Optimal Algorithms for Multipacket Routing Problems on Rings. *Journal of Parallel and Distributed Computing, 22(1): 37-43, 1994.*
- [3] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling networks. *Journal of Computing and Information, 1(1): 559-574, 1994.*
- [4] Amos Fiat and Gerhard J. Woeginger. Online Algorithms *LNCS State of the Art Survey, 1998.*
- [5] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis, 1998.*