



## Partisan Twist Exclusion in CB Algorithms

S.Kalyana kumar<sup>1</sup>,<sup>1</sup>Research Scholar/Mathematics,  
University of Allahabad,  
Allahabad, IndiaDr.Mata Ambar<sup>2</sup><sup>2</sup>Professor, Department of Mathematics,  
University of Allahabad,  
Allahabad, India

---

**Abstract:** To guarantee freedom from deadlock formation in computer communication networks, the Cycle Breaking, CB, algorithm is used to prohibit a minimum or near minimum number of turns using input/output pairs of ports at various nodes of the network. In our earlier approaches all turns of a graph had the same unity weight associated with them. A more general and practical case is when turns have non-unity weights associated with them. This however, is a more challenging case, because straightforward application of the CB algorithm would not satisfy the upper bound property of the CB algorithm. Authors in [1] introduced a new algorithm which addresses the problem of weighted turn prohibition or WTP. Specifically, for any arbitrary assignment of turn weights, the new algorithm breaks all cycles in the network, preserves the connectivity and guarantees that the sum of weights of prohibited turns does not exceed one third the sum of all turn weights in the topology. In this paper we use flow analysis based approach to prescribe meaningful weights to turns, and use the WTP algorithm to minimize the fraction of weighted turns. With the flow based turn weight determination approach, we improve the maximum sustained network throughput by up to 44% as compared with the non-weighted CB approach and introduce an average dilation of about 10%.

---

### I. Introduction

Cycle Breaking, CB, algorithm [2, 3] is used to assure deadlock-freedom in computer communication networks. This algorithm prohibits a minimum or near minimum number of turns using input/output pairs of ports at various nodes of the network. In these earlier approaches, all turns of a graph have unity weights associated with them. This means is that, no distinction is made between a turn in a communication network involving two communication links that are carrying say 50% of the total traffic and a turn carrying 0.1% of the total traffic. A more general and realistic case is when turns have non-unity weights associated with them. This is also a more challenging case, because a straightforward application of the CB algorithm would not satisfy the 1/3 upper bound property. Authors in [1] introduced a new algorithm which addresses the problem of weighted turn prohibition (WTP). Specifically, for any arbitrary assignment of turn weights, the new algorithm breaks all cycles in the network, preserves the connectivity and guarantees that the sum of weights of prohibited turns does not exceed one third the sum of all turn weights in the topology. This was in fact a challenging problem, in which a new approach to turn prohibition was invented.

A weighted graph  $G$  is conventionally defined [4, 5, 6] as an ordered pair  $(V, W)$ , where  $V$  is a set of  $N$  vertices in the graph, and  $W$  is a function giving a non-negative value  $w(v_i, v_j)$  to each pair of vertices. In undirected graphs without any self-loops, weights  $w(v_i, v_j) = w(v_j, v_i)$  and  $w(v_i, v_i) = 0$ . This notation very conveniently defines the weight or cost matrix of a weighted graph  $G(V, E)$  to be  $A(G) = (a_{i,j})$ , where

$$\begin{aligned} a_{i,j} &= 0 & i &= j \\ a_{i,j} &= w(v_i, v_j) & i &\neq j \end{aligned}$$

with  $i, j = 1, \dots, N$  for an  $N = |V|$  node graph,  $v_i \in V$  and  $(v_i, v_j) \in E$ . Depending on circumstances, these weights could represent distances between adjacent nodes, costs of traversing edges, bandwidths of the edges, or information carrying capacities of edges

In the CB approach [2, 3], a minimum degree node is selected and all turns at the selected node are prohibited. Since this node cannot participate in any cycle formation, it is subsequently deleted from the topology. The process continues by selecting the next minimum degree node from the smaller graph until all nodes are considered. Note that, when all turns at a node are prohibited, all edges of the node can participate in communication individually, but all incoming messages are blocked and not forwarded out onto other edges of the node. Therefore in this instance of the graph, the selected node becomes a dead-end node for messages. Special rules apply when prohibiting turns, if the selected node is a cut-node. This is done to preserve the connectivity in the graph after turn prohibitions are imposed. The new method does not necessarily prohibit all turns at the selected node. In the new approach, since turn weights may have no relationship to the degree of the nodes, in selecting a node for turn prohibition, WTP algorithm performs calculations on all nodes of the remaining graph, regardless of the degrees of nodes and determines the weight  $F$  of forbidden turns and weight  $A$  of

allowed turns. For each node, it determines the ratio  $F/A$  and then selects the node which has the smallest value for this fraction.

Given a portion of a graph in Figure 1, we illustrate the selection rule and the definition of various types of turns in WTP algorithm. In this discussion we assume that node  $x$  is being considered for selection. For each edge of node  $x$ ,  $(x,y)$ ,  $(x,u)$ , and  $(x,v)$  we perform the following operations. In the figure, we show the case where edge  $(x,y)$  is being considered. For the edge  $(x,y)$  we identify turns that would be prohibited at both ends of the edge. This is the principal difference between WTP and CB or MTP [2]. WTP algorithm considers prohibitions not only at the selected node but also at its adjacent node. To simplify our discussion, prohibited turns at the neighbor node are called *PType1* turns and involve the edge  $(x,y)$  and the neighbor nodes of node  $y$ . As shown in the figure, at this instance there are three *PType1* turns,  $(x,y,a)$ ,  $(x,y,b)$ , and  $(x,y,c)$ . Note that all of these turns start with the node being considered for selection. *PType2* turns are at the node that is under consideration that exclude those involving the edge  $(x,y)$ , such as the turns  $(y,x,u)$  and  $(y,x,v)$ . At the instance of the algorithm depicted in the figure, the only *PType2* is  $(u,x,v)$ . Once these turns are identified, their weights are added up which makes up the weight of forbidden turns in this iteration at node  $x$ . We then identify the permitted turns. Again we consider two types of turns, *AType1* and *AType2* turns. *AType1* turns are those that start with the neighbor node  $y$  of the edge  $(x,y)$  and involve node  $x$  as the middle node in the turn. *AType1* turns at this instance for allowed set are  $(y,x,u)$  and  $(y,x,v)$ . *AType2* turns are those that start with the node  $x$  and do not involve neighbor node  $y$  of the edge  $(x,y)$ . For the case in the figure, these are  $(x,u,u_1)$ ,  $(x,u,u_2)$ ,  $(x,v,v_1)$ ,  $(x,v,v_2)$  and  $(x,v,v_3)$ . After identifying these turns that would contribute to the weight of the allowed turns, we add their weights and determine  $A$  and calculate the ratio  $F/A$ . If this ratio is the minimum value we save it with the corresponding node, and the associated neighbor node. After all nodes and all of their edges are considered, we select the node with the minimum fraction, delete node and its edges and repeat the process for the remaining graph. Algorithm terminates when no edges are left.

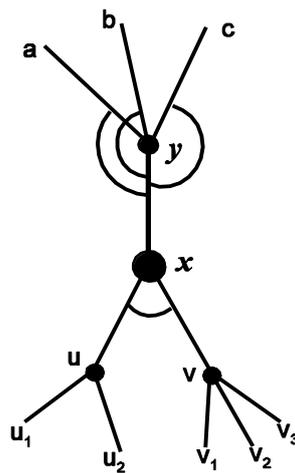


Figure 1. WTP at node  $x$  showing different turn types

There are some details of the WTP algorithm which will now be discussed in reference to Figure 2. This procedure is invoked for every node of a topology. With every invocation of this procedure we initialize the *CurMin* variable which will hold the current minimum value for the fraction. We then call the function *DELETEDEGREEONENODES* which returns 1 when it deletes at least one node of degree one or 0 otherwise. This function is called as many times as long as it returns 1. Note that, if the graph is a tree, the algorithm will remain in this loop until all nodes are consumed with no weighted turns being prohibited. When this function returns a value of 0, we know that all degree one nodes have been deleted from the graph. Algorithm then identifies all cut-nodes of the graph by means of the *FINDCUTNODES* procedure. Then, as shown in the figure, we start the selection process with the for loop. Function called *GETNCNODE* returns a candidate node  $x$  that is not a cut-node. With the next statement the number of neighbors of node  $x$  is determined. Subsequent for loop involves all edges incident on node  $x$ . For each such edge we identify the neighbor of node  $x$  and calculate the forbidden and the permitted turn weights, and determine the fraction,  $F/A$  as discussed. At the end of the outermost for loop, the node with minimum fraction is in the variable called *SelectedNode*. Algorithm then prohibits both *PType1* and *PType2* turns, deletes the node and its edges and returns.

A closer examination of Figure 1 reveals that similar to the CB algorithm [3], node  $y$  is a dead-end node for messages coming in from node  $x$ . Also note that no message can proceed beyond node  $y$  towards node  $x$ . Significance of this is that similar to original CB, when a node is selected is rendered a dead-end node by prohibiting all turns at the selected node. In the WTP algorithm, this is accomplished by prohibiting some nodes at the selected node and some at one of its neighbor. In other words, with the prohibitions of *PType1* involving the edge  $(x,y)$  of the selected node, the edge cannot be part of any cycle. Similarly, because of the prohibited *PType2* turns at node  $x$ , the latter also cannot be

part of any cycle. We therefore initiated investigation to determine if for some nodes, this approach of prohibitions would result in better performance.

```

procedure PERFORMWTP()
  CurMin ← 32000.0
  DelDegOne ← DELETEDEGREEONENODES()
  while DelDegOne = 1
    do DelDegOne ← DELETEDEGREEONENODES()
  FINDCUTNODES()
  for i ← 0 to NumOfNodes - 1
    {
      x ← GETNCNODE()
      NumOfNeighbors ← Gnode[x].degree
      for edge ← 0 to NumOfNeighbors - 1
        {
          nbor ← Gnode[x].Neighbors[edge]
          F ← CALCULATEFORBIDDENWEIGHTS(x, nbor)
          A ← CALCULATEPERMITTEDWEIGHTS(x, nbor)
          do {
            Ratio ← F/A
            if Ratio < CurMin
              then {
                CurMin ← Ratio
                SelectedNode ← x
              }
          }
        }
      PROHIBITURNS(SelectedNode)
      DELETENODEANDEDGES(SelectedNode)
    }

```

Figure 2. WTP Algorithm

To illustrate the operation of the WTP algorithm consider the simple graph in Figure 3 in which the weights of the turns  $(W, C, N)$ ,  $(E, C, N)$ , and  $(S, C, N)$  are equal to 1 and weights of all the other turns are 10. Lets perform the calculations for node  $x = N$ .. Since there are three edges incident on node  $N$  we calculate the forbidden and permitted turns for node  $N$  three times as shown in Figure 4. In (i) we show three prohibited turns for a total weight of prohibited turns of  $F_N(NW) = 30$  and a total weight of permitted turns of  $A_N(NW) = 43$ . Permitted turns for the case (i) are,  $(W, N, C)$ ,  $(W, N, E)$ ,  $(N, C, W)$ ,  $(N, C, S)$ ,  $(N, C, E)$ ,  $(N, E, C)$ , and  $(N, E, S)$ . Due to symmetry, case (ii) is identical to case (i). For the case (iii) we have  $F_N(NC) = 13$  and  $A_N(NC) = 60$ . With similar calculations performed for the remaining nodes we get  $F_W(WN) = 30, A_W(WN) = 61, F_W(WS) = 30, A_W(WS) = 61, F_S(SW) = 30, F_S(SC) = 31, F_S(SE) = 30, A_S(SW) = 61, A_S(SC) = 60, A_S(SE) = 61, F_E(EN) = 30, F_E(EC) = 31, F_e(ES) = 30, A_e(EN) = 61, A_e(EC) = 60, A_e(ES) = 61,$  and finally for the node  $C$  we have,  $F_C(CN) = 50, F_C(CW) = 32, F_C(CS) = 32, F_C(CE) = 32, A_C(CN) = 63, A_C(CW) = 81, A_C(CS) = 81, A_C(CE) = 81$ . From these calculations we see that the smallest fraction  $F/A$  occurs when node  $N$  is selected with the edge  $NC$  for a fraction of  $\frac{F_N(NC)}{A_N(NC)} = 0.2167$ . Once the node  $N$  is selected, turns shown in (iii) of Figure 4 are prohibited, node  $N$  is deleted and the process continues for the smaller graph of four nodes.

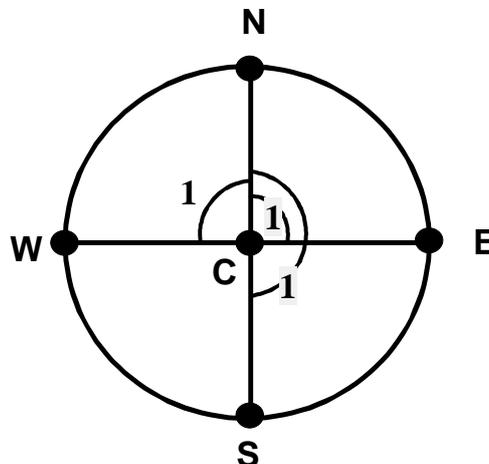


Figure 3 A Simple Topology for Applying the WTP Algorithm. As shown, three turns have unity weights and all others have weight of 10.

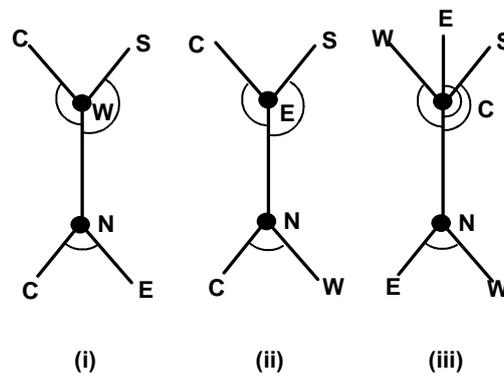


Figure 4 Prohibited turns when node  $N$  is being considered for selection

In [1] authors have shown that for **any** weight assignment to the turns in a topology, WTP algorithm will find a set of prohibited turns that satisfy following three properties:

**Property 1.** Any cycle in  $G$  contains at least one turn from  $W(G)$ , the set of prohibited turns.

**Property 2.** For any two nodes  $a$  and  $b$ , if there exists a path between  $a$  and  $b$  in  $G$ , then there exists a path between them with no turns from  $W(G)$  along the path, after the WTP algorithm is applied.

**Property 3.** For any graph  $G$  and any weight assignment to turns, weight of prohibited turns is no more than 1/3 of weights of all turns of the graph.

However no algorithm or process for prescribing weights to turns was identified in [1]. In this paper we investigate the significance of nodes and links in an undirected graph representing a network of communicating nodes and investigate one approach for weight assignment based on flow analysis. The rest of the paper is organized as follows. In the next section we provide some general insights into the flow analysis. This is followed by the algorithm for determining weights of edges and turns in the subsequent section. We show high-level simulation results, followed by an Opnet model and low level simulations using Opnet DES environment. We then conclude by summarizing our approach and results.

## II. Flow Analysis

In this section we investigate the significance of nodes and links in an undirected graph representing a network of communicating nodes and investigate one approach for weight assignment based on flow analysis.

For flow analysis we assume that the traffic model for the interconnection network is uniform. This also means that edges are full duplex communication lines with symmetric attributes. Given such a network, in conventional flow analysis, edges already have weights assigned to them, as maximum link capacities, and then problem becomes that of identifying the maximum flow between any given source and destination nodes [4]. Here, our motivation is different. We can state our motivation in the form of a question as follows. *Given a graph representing an interconnection network with uniform traffic model, how can we prescribe weights to the edges of the graph that can be used for capacity planning?* Same question could be posed involving the turns as well. However, in this investigation we assume that once the weights of the edges are determined, it would be a simple matter of assigning weights to turns.

Let us briefly consider a scenario, where most of the nodes use a specific section of the network in their communications with other nodes. It stands to reason that the set of links in this section of the network have a *high* value and that it behooves the network designer to treat these links more favorably than others with a *lower* value. Flow analysis undertaken here, is one in which, we characterize all edges of the network graph with the number of concurrent message flows that, given the appropriate channel or link capacity, they would be able to carry. In our analysis, we assume that messages flow through shortest paths between the source and destination. If there is a path that splits into multiple parallel paths, a message flow is divided equally among all such paths. When multiple parallel paths join into one along the shortest path to the destination, their flows are added to be the capacity of the single path. In Figure 5 we show the split and join rules for flows. Since traffic model is uniform, flows from  $a \rightarrow b$  is the same as flows from  $b \rightarrow a$ .

We shall now analyze the graph in Figure 6 for flows. In the figure we labeled edges for enumeration purposes and therefore the edge labels in the figure should not be interpreted as edge weights. Table 1 shows partial results of the analysis. Full table has  $N(N-1)/2$  rows, one for each pair of source and destination nodes. In the first row, we show the flows  $0 \rightarrow 1$ ; from node 0 to node 1. What starts out as 1 flow at node 0, splits into two on edges  $m=(0,2)$  and  $p=(0,6)$ . We therefore show a flow value of 0.5 for each edge  $m$  and  $p$  in the table. Both of these flows independently arrive at node 2 via edges  $i=(8,2)$  and  $j=(6,2)$ , each with the full value of their incoming flow. These flows join at node 2 to produce a net flow of 1 for the edge  $h=(2,5)$ , which subsequently splits into two at node 5, progress independently to nodes 7 and 9, and finally join at the destination node 1. Similarly, flow  $0 \rightarrow 2$  splits at the source and join at the destination, with each edge of the two parallel paths having a flow value of 0.5. When all calculations for the graph are completed, we determine the number of flows for each edge by adding all weights for each

edge, which we incorporated into the graph in Figure 7. Not surprisingly, we see from this analysis that with a flow count or potential flow capacity of 25, edge  $h = (2,5)$  is the most valuable link in the graph.

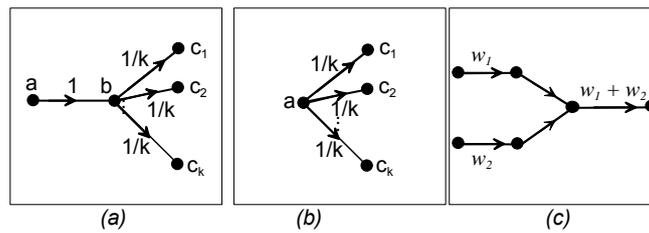


Figure 5. Split and join rules for flows

With given weights  $w(a,b)$  and  $w(a,c)$  of two edges  $(a,b)$  and  $(a,c)$ , there are a number of ways that one could prescribe a weight to the turn  $(b,a,c)$  by using the weights of the two edges of the turn. Whichever assignment option is chosen it is necessary that  $w(b,a,c) = w(c,a,b)$  due to symmetric aspect of turns in undirected graphs. Of the many available options for this assignment, only the additive and multiplicative approaches were considered. Specifically we considered the three Pythagorean means or averages, namely the arithmetic, the geometric, and the harmonic means of the contributing edge weights. Note that if we rank nodes with respect to their total turn weights, all three assignments will result in the same node order. For the arithmetic mean assignment, Figure 8, node rankings from most valuable to least valuable is  $\{2,5\}$ ,  $\{6, 7, 8, 9\}$ ,  $\{0, 1, 3, 4\}$  where nodes of equal value are grouped in the same set. This ordering is the same for geometric and harmonic assignments as well. Note that if we designate three approaches by  $w_A(a,b,c)$  for the arithmetic,  $w_G(a,b,c)$  for the geometric and  $w_H(a,b,c)$  for the harmonic assignments involving edges  $(a,b)$  and  $(b,c)$ , we get  $w_A(a,b,c) \geq w_G(a,b,c) \geq w_H(a,b,c)$ . If we further normalize the weights so that the largest turn weight is 1, by dividing each turn weight by the largest turn weight, the ordering remains the same. Because of these observations it was decided to use the arithmetic mean assignment in our simulation experiments.

### III. Algorithm for flow analysis and link weight determination

We modified Floyd Warshall [5] algorithm to implement the link weight determination. The modified Floyd Warshall algorithm generates all shortest paths for all node pairs and all predecessors of all nodes in the shortest paths. In the original Floyd Warshall algorithm, a distance matrix, e.g. one two-dimensional array,  $DM[i][j]$ , with only one shortest path, and the corresponding predecessor matrix  $PredM[i][j]$  are generated. In addition to these two matrices, in the modified version we added a new array called  $PredList$  in which we maintain the list of all predecessors. For example if in the shortest path from node  $v_x$  to  $v_y$  we determine that node has three predecessors, then the  $PredList[v_x][v_y]$  will point to a list of these three predecessors of node  $v_y$ . This would correspond to the case in the graph of a three-way join into node  $v_y$ . In the modified algorithm when a shorter path is identified from node  $v_i$  to  $v_j$  we update the current predecessor in  $PredM[i][j]$  with the new predecessor and we update the  $PredList$  by replacing the old predecessor with the new predecessor. If the new path is determined to be of the same length as the old entry in the distance matrix  $DM[i][j]$ , we add the new predecessor to the list  $PredList$ . Finally we replace the distance matrix entry, with the new shortest distance.

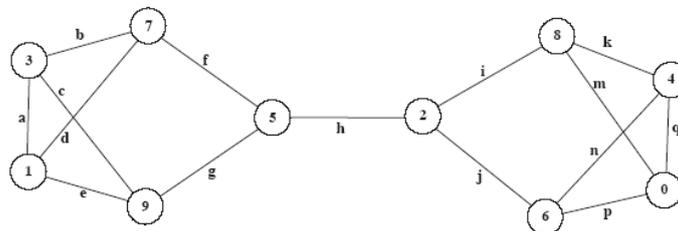


Figure 6. An example graph for flow analysis

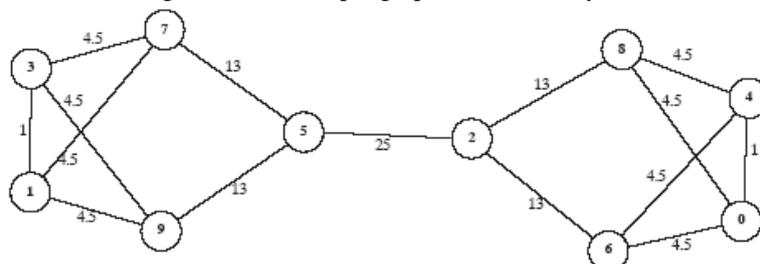


Figure 7. Edge weighted graph showing the number of concurrent flows as edge weights

After the algorithm completes, we calculate the number of times that each node has been a predecessor along all of the shortest paths. Complexity of this algorithm is  $O(N^3)$ , where  $N$  is the order of the graph. Then, another procedure is called which determines the link weights in  $O(N^2)$  time. With the edge weights calculated, the weights of turns of the graph are calculated using the additive approach with a complexity of  $O(N)$ .

Table 1. Table showing flow calculations for nodes 0, 1, 7 and the sum of all edge weights for the topology in Figure 6.

	a	b	c	d	e	f	g	h	i	j	k	m	n	p	q
0 → 1				0.5	0.5	0.5	0.5	1	0.5	0.5		0.5		0.5	
0 → 2									0.5	0.5		0.5		0.5	
0 → 3		0.5	0.5			0.5	0.5	1	0.5	0.5		0.5		0.5	
0 → 4															1
0 → 5								1	0.5	0.5		0.5		0.5	
0 → 6														1	
0 → 7						1		1	0.5	0.5		0.5		0.5	
0 → 8												1			
0 → 9							1	1	0.5	0.5		0.5		0.5	
1 → 2				0.5	0.5	0.5	0.5	1							
1 → 3	1														
1 → 4				0.5	0.5	0.5	0.5	1	0.5	0.5	0.5		0.5		
1 → 5				0.5	0.5	0.5	0.5								
1 → 6				0.5	0.5	0.5	0.5	1		1					
1 → 7				1											
1 → 8				0.5	0.5	0.5	0.5	1	1						
1 → 9					1										
...															
6 → 8								1	1	0.5	0.5	0.5	0.5		
6 → 9							1	1		1					
7 → 8						1		1	1						
7 → 9		0.5	0.5	0.5	0.5	1	1								
8 → 9							1	1	1						
SUMS	1	4.5	4.5	4.5	4.5	13	13	25	13	13	4.5	4.5	4.5	4.5	1

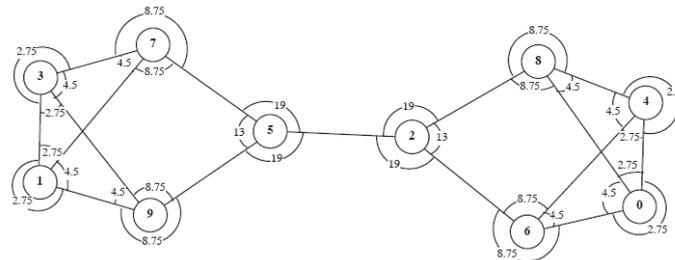


Figure 8. Weight assignment to turns by arithmetic mean

#### IV. High level simulation results

In this section we review our results comparing three approaches to prescribing weights to turns. First approach assumes all turn weights are equal to 1. We can now compare WTP with the CB approach. This approach of WTP in which all turns of a graph have the same unity weight, is referred to as the WTPS (S for same) approach. We see in Figure 9 that two results are different only in the third decimal place and that given the standard error bars for the experimental results as shown and hence we conclude that the performances of the two algorithms are indistinguishable from one another.

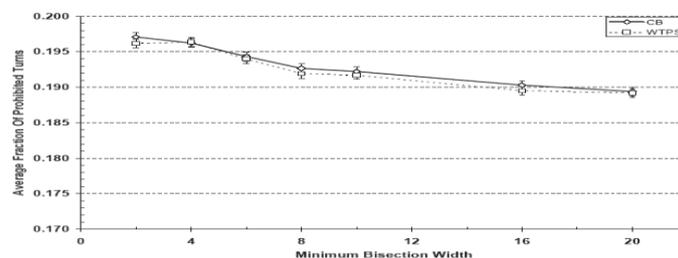


Figure 9. Experimental results comparing CB and the WTPS, where all turn weights are equal to unity. Two approaches yield indistinguishable results.

Next set of experiments compared the results for the fraction of weighted turn prohibition called WTPR (R for random) in which weights for turns are random integers between one and five inclusive. These randomly generated weights were randomly distributed among the turns of the graphs. The second approach called the WTPF (F for Flow), weights are prescribed to turns of a graph based on flow calculations as outlined earlier. We used the same family of a large number of topologies and calculated the fractions of weighted turn prohibitions for these two approaches and plotted them in Figure 10. We used  $\pm 1$  standard deviation bars in the figure.

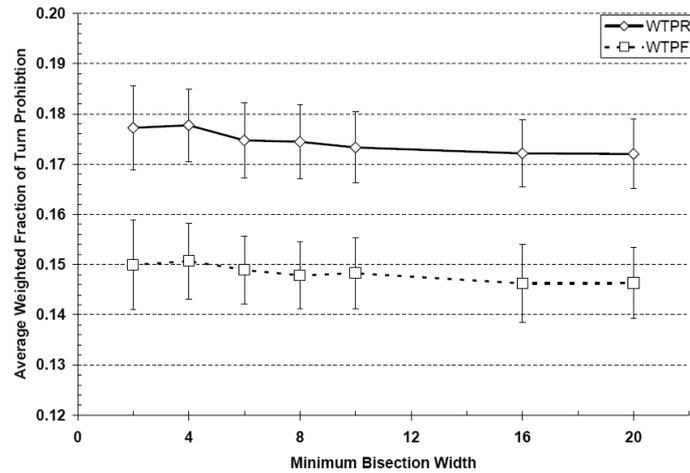


Figure 10. Fraction of weights of prohibited turns due to WTPR and WTPF algorithms compared.

### An OPNET Model

In order to investigate message delivery latencies we used OPNET to develop a simulation model for the wormhole routing protocol [2]. As shown in Figure 11, the node model contains the message generation (PGQueue) and consumption (PSQueue) modules and the router module (router). Inter-node communication handshaking is accomplished by (a) stream interrupts, and (b) remote interrupts. A stream interrupt is used to indicate the arrival of a Flit (smallest unit of information that is exchanged atomically, known as FLOW control digITs) and a corresponding remote interrupt sent back to the sender is used to indicate that the Flit has been received and that the receiving node is ready for the next flit of the message.

The process model for the router is shown in Figure 12. Router Module has two blocking states and a forcing state. In *Init* state, router identifies itself and the attached PGQueue and PSQueue objects and prepares the node for simulation by reading the run-time attribute values, initializes variables, allocates memory for routing tables, and reads in the node specific routing table. In the *Identify* blocking state, each node identifies its neighbors by sending just a Source Node Address Flit to each of its active ports. When the router receives responses from all of its active ports, it schedules a self interrupt with no delay and transitions to the next blocking state called *Listening*.

In the *Listening* state, arrival of the flits is identified by the stream interrupts. All flit types are processed by a function represented by the HANDLE\_FLIT macro. This function processes all incoming flits. For example, when a tail flit is transmitted out on any busy output port, the list of all waiting headers is examined. If there is a header flit waiting for the output port that has just been freed up, then the input port that it came from is associated with the, now free, output port and binding takes place. If the HANDLE\_FLIT macro identifies that the destination node address is equal to its own node address, then the local output port to the PSQueue is bound to the port delivering the flit. From now on, all incoming flits and the tail flit from the associated input port are sent out to this output port. With each transmitted flit, router process will send a stat wire to the PGQueue object and will be awaiting a remote interrupt from the adjacent router.

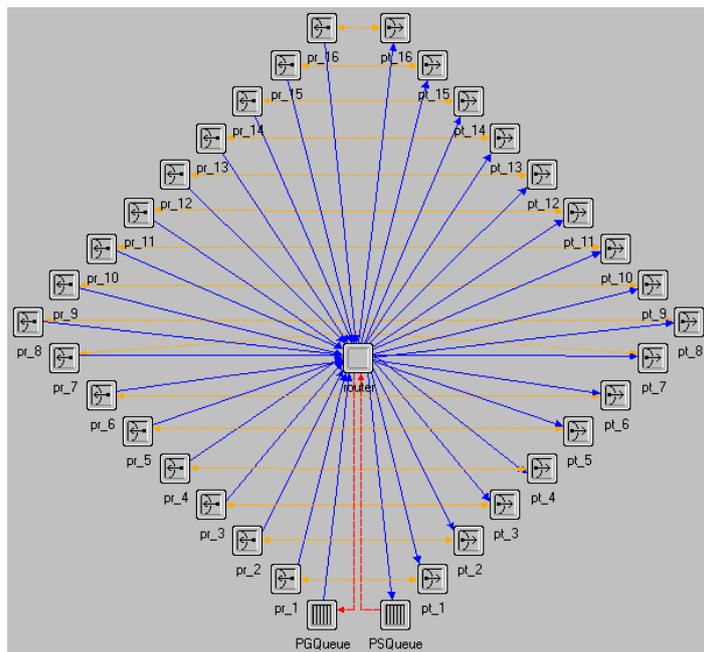


Figure 11. Sixteen-port wormhole node model that was used in our simulation experiments.

The transition event ACK is a remote interrupt from an adjacent node. Router process identifies the adjacent node by the interrupt code and interprets it to mean that remote node is ready for the next flit. All remote interrupts are processed by the HANDLE\_ACK macro.

PSQUEUE\_READY transition event triggers when a statwire interrupt is received from the PSQueue module. This indicates that the PSQueue is ready for the next flit from the router. This interrupt is managed by the HANDLE\_SWIRE macro.

The TIME\_OUT macro is for debugging purposes and is not used at this time.

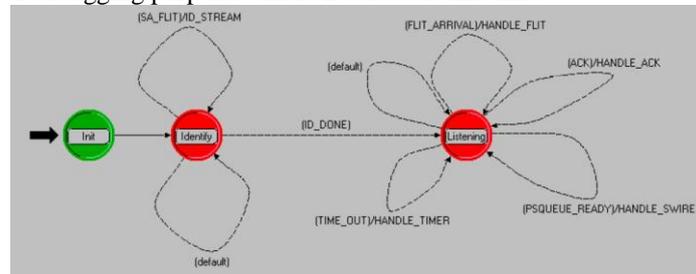


Figure 12. Process model of the router in a wormhole node.

### V. Simulation Results

We used techniques described in [2] to generate different family graphs and their Opnet subnet models using the EMA. We then used the WTPF algorithm to generate a set of prohibited turns and routing tables. Routing tables are constructed based on shortest paths that do not include any prohibited turns. We simulated the network using the *op\_runsim\_opt* in Opnet. In our simulations, we used 200 flits long messages and uniform traffic model. We identified the saturation point as the message generation rate, at which the average latency is  $2 \times 10^2$  times the low injection latency. To do this, we parsed the GDF files to identify the saturation points for each topology. We calculated the distribution of the saturation points for each algorithm and plotted the results. In Figure 13 we show the saturation points for the CB, Up/Down (UD) and the WTPF algorithms versus the bisection width. Each experimental point in the figure is the mean of a family of 100 randomly generated topologies. Each topology has 64 nodes of average degree four and the given bisection width. It can be seen that the WTPF approach is at least 24% and as much as 43% better than the CB and this improvement is even larger when comparing the WTPF and the Up/Down approaches at large bisection widths.

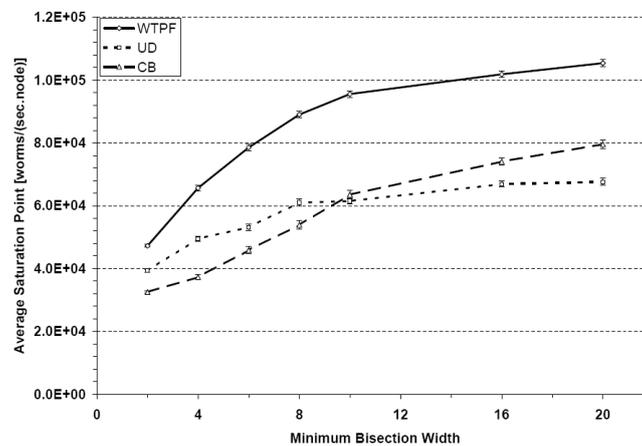


Figure 13. Comparison of the saturation points for CB, Up/Down (UD), and WTPF algorithms

### VI. Conclusion

We investigated a technique for weight assignment to turns in a graph using flows. In this method we assumed that flows will take the shortest paths between any source and destination pair of nodes. Paths of equal lengths between a source-destination pair share the flow equally which corresponds to load balancing in the network. Under these circumstances, we prescribed the number of concurrent flows as weights to edges. We then used the weights of edges to calculate turn weights by taking the average of the weights of two edges defining the turn. Turn weights calculated this way are favorable attributes and any turn prohibition approach should minimize the total weight of prohibited turns. To minimize the weight of prohibited turns, authors used the technique in [1], where some turns are prohibited at the selected node and some are prohibited at its neighbor. Using Opnet DES environment, we demonstrated by simulation experiments that this approach provided performances better than the Up/Down and CB techniques for message delivery latencies and maximum network throughputs.

In Table 2 we show our normalized results comparing these three techniques. We normalized the results such that those due to CB are unity. We also show the average distance dilation introduced by the three techniques. Dilation is the

amount by which the average distance is increased in the network from the average distance with no turn prohibitions. In the table we also show the run-time complexities of three competing algorithms.

Table 2 Performance (average) and complexity of algorithms investigated. Dilations are with respect to average distance in topologies with no prohibition. Fraction of prohibited turns for WTPS where all turn weights are equal to one is 0.995-1.001 as normalized to TP. For the WTPF case the fraction of weights of the prohibited turns was used. The maximum node degree of a graph is denoted by  $\Delta$ .

Algorithm	Frac. Proh. Turns	Dilation	Saturation Point	Complexity
Up/Down	1.195-1.241	10%-19%	0.85-1.20	$O(N\Delta)$
CB	1	16%-19%	1	$O(N^2\Delta)$
WTPF	-	9%-11%	1.32-1.45	$O(N^3)$

### References

- [1] L. Levitin and M. Karpovsky "Deadlock Prevention in Networks Modeled as Weighted Graphs," *Proceedings of the 8th International Conference on Information Networks, Systems and Technologies (ICINSAT-2002)* pp. 42-47, 2002.
- [2] M. Mustafa, M. Karpovsky and L. Levitin "Cycle Breaking in Wormhole routed Computer Communication Networks," *OpnetWork 2005* 2005.
- [3] L. Levitin, Karpovsky, M., M. Mustafa and L. Zakrevski "A New Algorithm for Finding Minimal Cycle Breaking Sets of Turns in a Graph," to appear in *Journal of Graph Algorithms and Applications (JGAA)* 2006.
- [4] N. Christofides "Graph Theory An Algorithmic Approach," 1975.
- [5] T.H. Cormen and R.L. Rivest "Introduction To Algorithms," 1989.
- [6] N. Deo "Graph Theory With Applications to Engineering and Computer Science," *Prentice-Hall Series in Automatic Computation* 1974.