2009

Randomized Greedy Hot-Potato Routing on the Multi-Dimensional Torus

Raymond Y. Chi
San Jose State University

Recommended Citation

Chi, Raymond Y., "Randomized Greedy Hot-Potato Routing on the Multi-Dimensional Torus" (2009). *Master's Projects*. 43.
DOI: https://doi.org/10.31979/etd.tynk-zg4x
https://scholarworks.sjsu.edu/etd_projects/43

RANDOMIZED GREEDY HOT-POTATO ROUTING ON THE MULTI-DIMENSIONAL TORUS

A Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Computer Science

by

Raymond Y. Chi

November 2009

SAN JOSE STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Report Titled

RANDOMIZED GREEDY HOT-POTATO ROUTING
ON THE MULTI-DIMENSIONAL TORUS

by Raymond Y. Chi

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. David Taylor,	Department of Computer Science	Date
Dr. Robert Chun,	Department of Computer Science	Date
Dr. Teng Moh,	Department of Computer Science	Date

APPROVED FOR THE UNIVERSITY

ABSTRACT

RANDOMIZED GREEDY HOT-POTATO ROUTING
ON THE MULTI-DIMENSIONAL TORUS

by Raymond Y. Chi

We present extensive simulation and analysis on a traditional, simple, efficient dynamic hot potato routing algorithm on a multi-dimensional torus network. These simulations are performed under a more recent network model than previous, more limited studies, with dynamic (rather than batch) models, no flow-control, and extended high dimensional scenarios. We collect more comprehensive statistics on system performance, and empirically show that the system can recover from worst-case scenarios to quickly re-achieve its standard steady-state delivery rates, with expected delivery time for a packet of O(n), where n is the initial packet distance from its destination. Experiments also show that for our model, the constant multiplier hidden in the O() notation decreases with higher dimensions.

Table of Contents

1 Introduction

Routing is the process of moving packets across a network from a source to a destination. In this paper, we consider Hot Potato (or deflection) routing on a synchronous network. The important characteristic of hot potato routing is that packets are not buffered at the node. At every time step, all packets arriving at a node are routed, sometimes away from their destination (deflected), unless the current node is the destination of the packet.

A routing algorithm is considered greedy if it routes the packet in the direction of its destination, whenever such a link is available. The key advantages of greedy hot-potato routing algorithms are their simplicity. Routing choices are simple and calculated based on local states. This makes hardware implementation easier.

In this paper, we revisit a classical greedy hot-potato routing algorithm, and present extensive simulation results and analysis. The algorithm we consider is memory-less, it makes routing decisions based on the state of the current node without regard to what is happening at other nodes or in past routing history of the packets.

While this algorithm has been considered previously, we study it in more detail, and with more current models than were previously considered. Our system is dynamic, not a batch system. We study higher dimensions, not just 2 dimensional systems. This allows us to compare performance across dimensions and to see how the algorithm behaves in higher dimensions. No flow control is used, the system is always at the maximum capacity. We used two different methods of packet generation, random destination, and uniform distance destination (original here as far as the author knows). The latter is important, especially in high dimensional systems because the random destination model generates packets with a very narrow range of distances for high dimensional systems.

In previous studies, dynamic routing problems were frequently approximated by a series of static routing problems. In static routing problems, all packets enter the network at time zero, and the running time of the algorithm is the time it takes for all packets to reach their destinations. Modeling dynamic routing problems this way simplifies the analysis such that performance and stability bounds can be more easily proven. However, this model of analysis is dated and does not truly represent a dynamic system. We will analyze dynamic routing problems using the steady state model. In this model, packets are continuously delivered and injected into the system. The analysis of the system is based on its performance when it reaches the steady state.

2 Greedy Hot-Potato Routing

Here we present the model which we will use to analyze the routing algorithm. We then present the algorithm, and discuss it in more detail.

2.1 Network Model

We consider routing on a *d*-dimensional torus network. A torus is similar to a mesh, but 'wraps around'. A 1-dimensional torus is a ring. A 2-dimensional torus is like the surface of the donut, though in our case, the distance around the donut is always the same in each dimension.

The torus can be generalized to *d* dimensions. Let *d* be the dimension of the torus. Let *s* be the size of each dimension. The size of the torus for a given (d, s) is s^d. This is the number of nodes in the network.

Each node of the network is connected to its adjacent nodes via an undirected edge. Every node has $2d$ neighbors. Therefore, each node of the network has degree $2d$. Each edge represents a communication channel between the nodes that can be used to transmit packets.

The network operates synchronously. During each cycle, every node of the network will process all its input packets, and send all of them out. During each cycle, only one packet can travel along a particular edge in a given direction. Therefore, during each clock cycle, every node sends exactly one packet on each of its edges. Similar to other hot potato routing algorithms, packets are not buffered. Every node must send every packet it receives in each round.

If a packet has reached its destination, it is considered delivered. A new packet will be generated in its place. At every round, every node will receives $2d$ packets, and will send $2d$ packets. The system is always full with $2d * s^d$ packets.

2.2 Routing Algorithm

Our algorithm runs as follows. Each node will examine its input packets in some random order, and assign an out going edge for each packet as it is considered. It will calculate a packet's route preference based on the current node and the packet's destination, and assign available edges to packets based on its route preference, and out-going edge availability. Since only one packet can go out on each edge at each round, some packets may travel in the opposite direction of their destination.

Each node has $2d$ outgoing edges. Therefore, every packet has $2d$ edges it can use. We compute the packet's route preference based on the distance it needs to go in each dimension. The first *d* choices for the packet correspond to a direction in each dimension the packet needs to go, in decreasing dimensional distance. The remaining *d* choices are the opposite of the first d choices, but in increasing dimensional distance order. The algorithm tries to minimize the directional distance differences by routing packets toward the direction the packet needs to travel the most to reach its destination.

The first choice direction will always bring the packet closer to its destination. The 2nd choice direction is always perpendicular to the 1st choice direction. In a 3-d or higher system, the 3rd choice direction is perpendicular to the 1st and 2nd choice plane, etc. In a *d* dimensional torus with $2d$ edges, the first *d*-choices for the packet will generally bring the packet closer to its destination, while the remaining *d* choices will almost always route it further away from the destination (except in case of wrap-around).

For each time step, the first packet being processed will always get its first choice edge, since all the edges are available. Once that edge has been assigned, it is no longer available, and the 2nd packet will pick one of the remaining edges to travel based on its preferences. This may or may not correspond to its 1st choice direction. The last packet will always get whatever edge is remaining, regardless of its routing preferences.

To summarize, the input packets for a given node are processed in random order. Routing preferences are based on the packet's distance to destination for each dimension, in decreasing distance order. Routing preference is calculated entirely based on the packet's destination. All available edges are assigned a packet. Some packets will travel away from their destination. In our simulation, whenever choosing direction preferences, ties are broken randomly. This provides full symmetry to the system.

2.3 Related Work and Models

Hot-potato routing was first proposed by Baran [Bar64] as a viable packet routing algorithm. Later on, it was utilized in a number of notable applications, including the HEP multiprocessor computer system [Smi81], the Connection Machine [Hil85], and Caltech mosaic C [Sei92]. Hot-potato routing was used because it appears to work well in practice. As routing technology improved, greedy variants of hot-potato routing began to emerge. [BH85, Max89] confirmed that greedy hot-potato algorithms were very promising. These works led to development of better algorithms for addressing limitations encountered in previous work, such as packet collisions.

[BHS94] was the first paper to formalize the concept of greedy hot-potato routing. Using potential function analysis, they authors obtained upper bounds on the running time of a variety of class of greedy hot-potato algorithms. For the 2-dimensional *n* x *n* mesh, they obtained an upper bound of $O(n \sqrt{k})$, where *k* is the number of packets. Extending to higher dimensions, the author also provided an upper bound for the d-dimensional mesh. Their work is often referenced for its contribution to the analysis of greedy hot-potato routing algorithms.

[BU96] described a simple one-bend packet routing algorithm of a dynamic synchronous network on the *n* x *n* torus. The algorithm only knows how to route packets along a one-bend path. Each node is configured to randomly insert packets with some probability that is chosen such that the routing algorithm is stable (i.e., flow control). The algorithm achieved an expected delivery time per packet of O(n). It is simple and easy to analyze, however, it imposes a relatively high collision penalty.

[Fei99] showed that the algorithm by [BU96] is not monotonic. The routing is monotonic if decreased load results in better performance. The author noted that the algorithm by [BU96] does not exhibit this behavior. The author goes on to show "that non-monotonicity is a common property of routing algorithms and not just an artifact of their analysis". The author does present a monotonic, bufferless routing algorithm for the synchronous *n* x *n* mesh network. The algorithm prevents conflicts by having all packets in the system make their left turn at the same time.

[BHW00a] showed an improved version of randomized greedy hot-potato routing that runs in a 2 dimensional mesh and torus by utilizing “home-runs” (or one-bend paths). Packets are assigned different priorities (normal, turning, walking, excited, and running) during routing based on a set of rules. The priorities are used to resolve conflicts. The algorithm delivers packets in asymptotically optimal expected *O(n)* steps, and delivers all packets in *O(n ln n)* steps with high probability. The algorithm is local, the nodes are stateless, and each node makes routing decisions independent of the state of other nodes. However, it requires some state being stored on the packet itself. This may not be desirable or possible in some systems. The algorithm does not easily generalize to higher dimensions.

[BHW00b] improved their previous algorithm by utilizing mult-bend path instead of one-bend path. Packets attempts to take a multi-bend path “by choosing a logarithmic number of random intermediate destinations in a sequence of squares of decreasing size.” The author claims that it is the “first hot-potato routing algorithm [proven] for the *n* x *n* mesh whose running time on any ‘hard many-to-one’ batch routing problem is, with high probability, within a poly-logarithmic factor of optimal.”

[BCKV00] presented some experimental results on four hot potato routing algorithms using both the static and stochastic model for continuous packet generation on a 2 dimensional torus. The results showed that “although the running time for al the algorithm is close to the optimal for static routing problems, heavy traffic may influence differently the performance of each algorithm in the dynamic case”. The greedy algorithm referenced in the paper is the same as our algorithm. However results are limited to 2 dimensional tori, and only basic statistics are collected. The dynamic model uses an injection rate as flow control and buffer at nodes to store packets.

[BHW01] presented what according to the author is the “first dynamic [greedy] hot-potato routing algorithm that does not require any form of explicit flow control” and guarantees asymptotically optimal performance. The performance matches the algorithm presented in [BU96], though the constant factors are large. The large constant term in the running time of the algorithm might be attributed to the fact that packets that are one step away are just as likely to be deflected as packets that are further away.

3 The Simulation

We implemented a program to simulate our hot potato routing algorithm on the torus network. The simulation allows us to analyze the runtime behavior of the system, to obtain performance metrics and statistics, and to see how our algorithm performs under various conditions.

The simulation is implemented in C. It runs on Linux and can be easily ported to other platforms. The simulation supports an arbitrary number of dimensions, though in practice, this is limited by the amount of physical memory and the processing speed of the system.

We ran simulations on 1 to 6 dimensional tori. It is not practical to simulate higher dimensions using current hardware. The size of the torus grows exponentially with the number of dimensions. For lower dimensional runs, the system is CPU bound, and a moderately sized torus can be simulated. For higher dimensional runs, the system becomes more memory bound, only small torus sizes can be practically simulated. Packets and nodes take up most of the available memory.

The following section describes the various options and configuration settings that can be set in the simulation. The various options help us better understand the behavior of the system.

3.1 Basic Parameters

The basic input parameters to the program are:

- Torus dimension - *d*
- Size of each dimension - *s*
- Number of rounds to simulate - *n*

The program creates a torus of the specified size in memory, initializes the torus with packets (either random destination or uniform distance destination), and runs the simulation for the specified number of rounds, routing and delivering packets as needed. When a packet is delivered, a new random packet is generated in its place, and simulation continues.

The program tracks a number of statistics during the simulation. The user can specify which statistics are to be collected. The program will output the corresponding statistics after the simulation. The program makes an effort not to allocate memory for statistics not requested, to minimize the memory footprint and leave memory for the torus and packets.

3.2 Packet Generation

Packets with random destination are generated and placed on the torus network for routing. What is this “random destination”? We considered 2 generation models: equal probabilistic destination, and uniform distance to destination.

3.2.1 Equal Probabilistic Destination

In the equal probabilistic destination model, we consider each node with equal probability, and randomly pick a node in the torus. This node is the destination of the packet.

This is implemented using the following simple algorithm:

```
For each dimension d
     Generate a random integer r between 1 to torus size s
```

The packet’s destination is the vector <r_1, r_2, r_3, …> up to r_d dimensions.

This algorithm can be implemented very efficiently. The probability of a packet having a particular node as destination is uniformly distributed between all nodes, for all dimensions. Every node has an equal probability to become the destination of a packet. For example, in a 10x10 2D system, a packet is equally likely to go to any of the 100 target destinations.

The equal probabilistic destination model is the "random destination" model described in other studies. In the remaining sections, we'll continue to refer to it as equal probabilistic model to make it more obvious the way the destination are generated.

3.2.1.1 Zero Distance

Of all the possible destinations in the equal probabilistic destination model, one of them is a 0-distance destination, the current node. In our simulation, we treat the packet as instantly delivered, regenerate the packet until a non 0-distance packet is generated. The statistics calculation includes these 0-distance packets.

Alternatively, the simulation can be set to not generate 0-distance packets.

Since these packets are independent of one another, including them or not will not affect the other packets in any way. It will just affect the system throughput.

3.2.2 Uniform Distance to Destination

In equal probabilistic destination, packets are not uniformly distributed by destination distance. This is important for comparing simulations across dimensions.

Why is the distance not uniform? Let us illustrate this by using an example. If you roll a regular 6-faced die, the probability for getting any of the 6 values are the same. However, if you roll the die twice, the distribution of their sum is no longer uniform. The probability of getting a 7 is six times the probability of getting a 2, as there is only one way to roll a 2 (1, 1).

In equal probabilistic destination, the way we generate the destination is similar to throwing an *s*-faced die *d* times, where s is the size of torus in each dimension, and *d* is the number of dimensions. The distance to destination is the sum of distance in each dimension. As the analysis above shows, the distance to destination is not uniform. In a 2D torus of size 10 in each dimension, there will be more packets with distance 7 than distance 2.

This is not desirable when trying to compare results across dimensions. It skews the system by generating more packets having average distance and fewer packets with very short and long distances. The skew will be more extreme in higher dimensions. In a 6D system, most of the initial packet distances are concentrated on a narrow range of distances in the middle. To solve this problem, we came up with an alternative packet generation method.

Our method works by first picking a distance uniformly among all possible distances, then picking a node having that distance away.

Here's the algorithm:

1. Let *t* be the range of possible distances for the torus system (*d* * *s*/2).
2. Generate a random distance *x* between 1 to *t*.
3. Repeat following until all partitions are valid (less than *s*/2).
    - Partition *x* into *d* random segments by generating *d*-1 points between 1 to *x*.
    - The length of each segment, *p*, some of which may be 0, is the corresponding distance in each dimension.
4. The packet's distance to destination vector is <p_1, p_2, p_3, …> up to p_d.
5. Calculate the destination node from current node and <p_1, p_2, p_3, …>.
    - For each dimension *i*
        - Flip a coin, representing the direction, left or right, for dimension i from the current node.
        - Based on direction, calculate the destination address r_i based on current node address, and distance p_i.
    - The destination node is <r_1, r_2, r_3, …> up to r_d.

This approach essentially generates the destination in a top down approach. We first generate the desired distance, and then break the distance into its dimensional segments. It should be apparent from the algorithm that the total distance will be uniformly distributed.

In step 3, we partition the distance *x* into *d* segments by randomly generating *d*-1 points along the line 1 to *x*. However, this may produce a partition that's longer than the maximum possible packet distance for a dimension. Therefore, not all partitions are valid, and the steps must be repeated until they are.

For example, in a 3D system of size 40, the maximum distance in each dimension can be 20. Therefore, the range of possible distances are from 1 – 60. Let's say we picked distance 60. It should be obvious that the only target destination that will satisfy this distance is (20, 20, 20). If we partition the distance any other way, for example (1, 1, 58), it will not correspond to a valid torus address, as distance for each dimension can only be up to 20.

We solve this by discarding invalid partitions, and repeating step 3 again, until we get a valid partition. This will work, however performance may suffer for some of the corner cases. For example, it will likely take many tries to generate the 20, 20, 20 partition for distance 60.

We optimized this by noticing that partitions are symmetrical with respect to the middle distance. Using our example above, partitioning distance 60 is the inverse of partitioning 0. Partitioning distance 0 yields only 1 valid partition, (0, 0, 0). Simply flip the result by subtract max distance of each dimension (*s*/2) to this value, and we end up with the correct answer (20, 20, 20).

In other words, we restrict our partitioning up to *t*/2. For distance *x* above *t*/2, we simply partition the distance *t* - *x*, and set *s*/2 - *p* as the distance to destination vector. The chances of producing an invalid partition of distance 30, where each partition can be up to 20, is significantly less than trying to partition distance 60. In fact, 30 becomes our worst case. Distance 60 will be partitioned in one round.

Using this method, uniform distance to destination can be generated with just a very slight performance hit compared to equal probabilistic destination.

3.3 Simulation Cutoff

The simulation typically runs for *n* iterations. At the end of the iteration, we calculate statistics such as delivery rate, delivery time, etc. However, at the end of the simulation, the system is still full of packets that need to be delivered, and they are not random packets. If these packets are ignored in some of the calculations, it could affect the result (since they are not delivered, they don’t contribute to the delivered statistics).

This could be important because packets that are left over are packets that didn’t get delivered. So the statistics we collect may be skewed by packets that got delivered, but not taking into consideration packets that are not yet delivered. This may make our result look better than what it really is. We could improve the calculation if we can take these packets into consideration.

3.3.1 Approximation

The initial change we made was to estimate the delivery distance for these pending packets by using data from already delivered packets. If during the simulation packets that are *x* away got delivered in *y* steps on average, then we can estimate the delivery time for packets in the system at the end of the simulation by using this information. When the estimates are calculated, we update our average using the new estimate.

3.3.2 Run Until Deliver

While the approximation method is an improvement, it is still just an approximation. To make the result more precise, the simulation can be set to run until all packets at the ending iterations are delivered. Packets introduced after the ending iteration are routed accordingly, but will not be included in the statistics calculation. When all packets that were present at the ending iteration got delivered, the simulation ends.

This way, we are essentially running the system until all useful data are gathered. We are no longer estimating the delivery time of pending packets, we are running the simulation until they are all delivered. This ensures the system is running for long enough that the simulation results are somewhat meaningful.

3.4 Statistics Start

By default, statistics are collected from round 1. An option exist to ignore statistics for the first n rounds, useful for collecting statistics after the system has reached steady state.

3.5 Initial Network State

The torus is initialized with random packets using one of the packet generation models. This represents the average/normal state, i.e. all packets have some destination to go that have certain distribution (equal destination probability for every node, or uniform distance destination), i.e. a truly random system.

We can also start with a different state, the "bad state". What is a bad state? A bad state is where packet interference keeps packets from making good progress toward their destinations.

This is very useful because knowing how the system behaves in the bad state would allow us to simulate what happens if the system got into the bad state by chance, however unlikely. What happens afterwards? Does it recover or stay in the bad state? Knowing that it recovers from the bad state will give us some confidence that even if it goes into some bad state, it won't stay there forever.

3.5.1 Starting from a Bad State

Let's first decide on what is a bad state. Let's analyze the 1D case. What is the worst possible configuration? The worst possible configuration occurs when every packet wants to go in the same direction. However, it doesn't quite work if we simply start by setting every packet to go to the same direction for the maximum possible distance. It may seem bad, but actually is not. During the 1st round, half of them will get their first choice and go to the left, the other half will get their 2nd (wrong) choice, and go to the right. However, at the 2nd round, because all packets start out with the maximum possible distance and the torus wraps around, all those packets that got their 2nd choice, will "wrap-around", having a new 1st choice opposite of the packets that got their first choice in the 1st round. In other words, all packets have non-conflicting 1st choice. The system is in the best possible configuration, in that all packets will move closer to their destination (even if all packets are still far from their destinations).

To ensure that the system stays in a bad state as long as possible, we cannot use the maximum possible distance. What we are really trying to do is, after this round, ensure that those packets that did not get their first choice, still have the same first choice. We want to maximize the number of rounds that this property is true.

The distance that maximizes this property is $s/4$ for 1D torus. For example, if the torus size s is 20, then every packet wants to go to the left, and is 5 away from its destination. This ensures that, for the first 5 steps, every packet still has the same first choice direction.

This can be generalized to higher dimensions. In the 2D bad state, every packet wants to go in the exact same direction (having the same 1st and 2nd choice direction), and retains that property for as long as possible. For example, not only do all packets wants to go up and left, they all want to go up first, then left, for as many iterations as possible.

Let *d* be the torus dimension, *x* be the radius of the torus (*s*/2, the maximum distance a packet can go in each dimension). The distance vector p_i is simply

```
For $i$ = 1 to $d$
    $p_i = i / (d + 1) * x$
```

Essentially, we place *d*+1 equal distance markers on *x*. The distance vector is simply the starting position to each of these markers. For 1D, this is <1/2> * *x*. For 2D, this is <1/3, 2/3> * *x*. For 3D, this is <1/4, 2/4, 3/4> * *x*. This ensures that for the first 1/*d*+1 rounds, no matter what choice a packet travels, its first *d* choices are exactly the same.

Once the distance vector is calculated, we generate the destination address using the distance and a fixed random direction (same for all packets) for each dimension.

This forces packets into a random walk for the first *x*/(*d*+1) steps, during which no packets can be delivered, and no packet choices can change. A packet gets its *i*-th choice if it is the *i*-th packet chosen at a node, completely random. Hence, the random walk.

3.6 Directional Dependency

The simulation starts with a random state. All packets have some destination to go that is independent of one another. As the simulation progresses, packets are no longer independent. The route a packet takes depends on its history.

It would be interesting to know how much of the routing performance is changed by directional dependencies between neighboring nodes. Does it help, or make things worst? That is, do dependant packets interfere with each other's progress more constructively, or destructively, compared to independent packets?

Beyond just looking at how many packets get their *i*-th choice directions, we also consider a system just for comparison, with "reset packet direction".

How do we simulate a system with no directional dependencies? We do so by forgetting our destination at each step, while retaining the distance to destination vector, and instead using a random permutation of this vector. At each step, we abandon our current destination and randomly pick another destination with a random permutation of the abandoned distance to destination vector.

This ensures that for this round, the routing choices for this packet are not affected by the previous routes it has taken, or previous collisions with other packets. For example, if a packet needs to go <5, 3> to reach its destination, after the reset, it may end up with <-5, 3>, <-3, 5>, etc. The distance to destination and the distance vector is still the same, but the direction it needs to go to reach destination is completely random (independent) now.

Comparing this simulation result (clearly, this is not true routing, and is only for information collection comparison) to the original allows us to see how directional dependency affects the routing.

3.7 Program Usage

There is one executable for each dimension. The default compile produces executables for 1-8D. Higher dimensions can easily be compiled by simple makefile changes.

Here's the output of the help screen of the 2D executable (showing an extra 2d only option, packet configuration count statistics). The program supports more options than those discussed above. The extra ones should be self explanatory.

```
Random Torus 3.6 [Sep 27 2009 14:46:37] (X-DIM edition)

Usage: bin/2d_torus [options] <torus_size> <iterations>

Options:
  -z            do not generate 0 distance packets
  -d <dist>     only generate <d> distance packets [0: 1 to s/2 * dim]
  -i            worst possible initial packet dependencies
  -r            reset packet direction at each round
  -m            do NOT randomly pick equal distance direction
  -c            simulate until all packets at round MAX are delivered
  -a <n>        only collect statistics starting with round n [default: 1]
  -p <second>   progress thread sleep interval [default: 2]
  -s <options>  statistics options
      1         packets delivered at each round
      2         packets with initial distance [X] delivered at each round
      3         average delivery time for packets with initial distance [X]
      4         % of packets getting their n-th choice at each round
      5         % of packets moving closer to destination at each round
      6         % of packets having only [X] direction to go at each round
      7         packets that are [X] away from their destination at each round
      8         packets that started [X] away from their destination at each round
      9         average delivery time for packets with initial distance <x1,x2..>
      a         packet configuration count
      -         all statistics
  -l <options>  debug output options [default: 0]
      1         output delivered packets at each step
      2         output torus configuration at each step
      3         output direction reset information for -r
      4         enable and output tracer packet
  -x <n>        generate [n] random numbers (0-255)
  -y <n>        generate a random permutation of 0-[n]
  -w <n>        print [n] numbers per line for -x/-y [8, 5]

Routing on 2-dimensional torus with random packets.
```

3.7.1 Tracer Packets

We used tracer packets to validate the correctness of the algorithm. We output the progress of the tracker packet to make sure the routing and delivery is performing correctly.

Here's the output of a 2D simulation with tracer packet enabled:

```
Initializing torus (2D x 10) ... (100 nodes) done.
Initializing statistics ... done.
Generating 400 packets ... done.
```

```
Torus size: 10, dim: 2, rounds: 10 (100 nodes, 4 packets each, 400 packets total)

[1] (0, 0) tracer: [(0, 0) => (3, 1), dist=4, hops=0], id=1 ==> choice #2 (dim=2, dir=1) *
[1] (0, 1) tracer: [(0, 0) => (3, 1), dist=3, hops=1], id=1 ==> choice #3 (dim=2, dir=0)
[1] (0, 0) tracer: [(0, 0) => (3, 1), dist=4, hops=2], id=1 ==> choice #1 (dim=1, dir=1) *
[1] (1, 0) tracer: [(0, 0) => (3, 1), dist=3, hops=3], id=1 ==> choice #2 (dim=2, dir=1) *
[1] (1, 1) tracer: [(0, 0) => (3, 1), dist=2, hops=4], id=1 ==> choice #2 (dim=2, dir=1)
[1] (1, 2) tracer: [(0, 0) => (3, 1), dist=3, hops=5], id=1 ==> choice #2 (dim=2, dir=0) *
[1] (1, 1) tracer: [(0, 0) => (3, 1), dist=2, hops=6], id=1 ==> choice #1 (dim=1, dir=1) *
[1] (2, 1) tracer: [(0, 0) => (3, 1), dist=1, hops=7], id=1 ==> choice #1 (dim=1, dir=1) *
[1] (3, 1) tracer delivered: [(0, 0) => (3, 1), dist=4, hops=8] id=1
[2] (3, 1) tracer: [(3, 1) => (4, 3), dist=3, hops=0], id=613 ==> choice #1 (dim=2, dir=1) *
[2] (3, 2) tracer: [(3, 1) => (4, 3), dist=2, hops=1], id=613 ==> choice #4 (dim=1, dir=0)

start time: 2009-09-28 13:09:05
end time:   2009-09-28 13:09:05
duration:   00:00:00 (0s)

+ packet initial distance: average: 5.066079, min: 0, max: 10, n: 681
  packet delivery time   : average: 5.345196, min: 0, max: 9, n: 281

Freeing statistics ... done.
Freeing torus ... done
```

Each line of the tracer output contains:

1. tracer packet # in [], incremented when delivered.
2. current node in ()
3. packet info in [] (start node, destination, distance to destination, hops traveled)
4. packet id
5. choices taken (dimensions taken, directions taken)
6. whether it got closer or not (* if it got closer).

3.7.2 Simulation Time Estimate

When simulating a large torus for extended number of rounds, the simulation time is often in the hours. It is useful to know how long the simulation will take under those circumstances.

To better assist the user in setting up simulation runs and get an estimate of how long a particular simulation will take, the program uses a monitor thread to report progress on the simulation, and calculate an estimated simulation time. The program attempts to compute an accurate time estimate by timing existing simulation speed, and project simulation time based on this data.

The time projection calculation is not entirely straightforward. In each round, the simulation engines goes through 2 stages of processing. In the 1st stage it moves all packets from input buffer (packets received from previous round) to output buffer (to be processed/routed). In the 2nd stage, it processes all the packets on the output buffer, delivering packets that have arrived at destination, and routing packets as needed based on our routing algorithm, moving packets from the output buffer of this node to the input buffer of the node it got routed to.

Intuitively we can simply record the starting time, the current round and the time, and based on this, calculate time needed for simulation. This works for small dimensions and small size tori, where every second we go through many rounds. It doesn't quite work for higher dimensional tori, where each round can take some time, and each stage within a round can take quite a while. We don't want to wait for a few rounds to get an estimate.

To improve the estimate, we first figure out the ratio between 1st stage and 2nd stage through trial runs. This is approximately 1/3 vs. 2/3, i.e. the 2nd stage is about twice as long. This initial ratio is hard coded into the simulation, and gets updated when real data is available. With the ratio, we can get a better sense of where we are in the simulation even when the 1st stage within the 1st round is not finished. For example, if we are 2% into the 1st stage after 2 seconds (based on number of nodes processed, and total number of nodes), we can estimate how long the 2nd stage will take, and estimate how long each round will take. Based on this, we can estimate how long the simulation will take. Once more data are available, a updated estimate is displayed.

With this method, the program can compute a fairly accurate time projection just a few seconds after the simulation has started. The estimate improves slightly as the simulation progresses, but the initial estimate generally is well within the ballpark figure.

3.7.2.1 Time Estimate Output

Here's the output from the monitor thread, updated every 2 seconds (configurable):

```
68.73% [=================>        ]  660/960 [399966/512000]  14470 D/s  1199896 R/s  00:29:28 [00:13:24]
```

1. percentage completed
2. progress
3. current round / set round: which round is the simulation on, how many round total.
4. current node / total node: The current node being processed, total node in the system.
5. deliveries / second: # of packets delivered per second
6. routing / second : # of packets routed per second
7. time elapsed
8. estimated remaining time

If the remaining time is more than 24 hours, instead of simply output a large `H:M:S`, it will reformat it to day, month, and year if necessary, as in "`[1y 128d 13:39:12]`".

When the simulation is set to continue until all packets at the ending round are delivered, the progress bar resets back to 0% after reaching the ending round, and remaining progress is calculated based on a count of the remaining number of packets that needs to be delivered.

4 Statistics

In this section, we discuss the various statistics our simulation collects. Statistics can be turned on/off as desired, maximizing available memory for the simulation itself.

4.1 Packets delivered at each round

This is the delivery rate. This allows us to track the system throughput. How does the system perform? How does it change over time? How does it compare across dimensions? Does the delivery rate stabilize? We consider a stabilized delivery rate the steady state, where the number of deliveries per round is consistent from round to round. How long does it take to reach the steady state?

4.2 Packets with initial-distance [X] delivered at each round

Delivery rate but by initial distance. Are packets being delivered for all initial distances? How does this distribution compare with the packet generation model? At steady state, this should be the same as the packet generation distance distribution.

4.3 Average delivery time for packets with initial-distance [X]

This is our main delivery time metric. It allows us to answer questions such as how long does it take to deliver a packet? Is the delivery time linear with respect to initial distance?

This calculation is affected by how the simulation is ended, whether approximation or run-until-delivery methods are used.

4.4 Average delivery time for packets with initial distance (x,y,z,..)

Delivery time but by initial distance vector instead of initial distance. This allows us to see for same distance packets, whether more directions to destination translate to faster delivery time, i.e. packets with distance <5, 5> gets delivered faster than packets with distance <0, 10>.

4.5 % of packets getting 1st, 2nd, etc choice direction at each round

This will provide some insight into our routing. If few packets are getting their 1st choice, then the routing is obviously bad. If most packets are getting their 1st choice, then we have a good indication that the routing is performing well. How does 2nd choice number compare to 1st choice? Are they about the same, or different?

We only consider the first *d* choices, as the remaining choices will generally move the packet further from its destination. Note, it is possible that some of the first *d* choices will move a packet further from its destination (if a packet location already matches its destination in some dimension), and also that if a packet is at the maximum distance in lower dimension, movement in one of the last d choices will actually move the packet closer (the latter happens infrequently).

4.6 % of packets moving closer to destination at each round

This metric is related to, but different from the previous one. If a packet does not get its first choice, it may still move closer to its destination if it gets its 2nd, or 3rd, etc, hence making progress toward its destination. Sometimes the 2nd choice will not move the packet closer, if the packet only has 1 dimension to go to reach its destination. Therefore a different indication of routing performance is to see what percentage of packets are moving closer. If this number is high then routing is working well. If this number is low then routing is performing poorly.

4.7 % of packets having only 1, 2, 3... directions to go at each round

When a packet starts out, it may have several directions to go to get closer to its destination. However, when it gets closer, it may have fewer dimensions to go, and hence its route flexibility decreases. This metric allows us to get a sense of the overall packet route flexibility of the system, and how it changes over time. Do packets retain their flexibility as long as possible? Or does flexibility run out quickly and a lot of packets end up having only few directions to go after a short period of time?

This is an important metric since one of the key points of our algorithm is that it retains packet routing flexibility as long as possible, unlike the one-bend path algorithms.

4.8 Packets that are [X] away from their destination at each round

If routing is going well, then most packets will move closer to their destination. Therefore, there should be more packets closer to their destination than packets that are further away. If routing is not working well, this may not be true. This metric allows us to see if this is the case, and gain a better understanding of the system and our routing.

4.9 Packets that started [X] away from their destination at each round

This is similar to the previous metric, but uses the packet's initial distance rather than current distance. If routing is going well, there should be fewer packets with short initial distance than long initial distances, since those packets should be delivered already. This basically brings us another view of steady state and how we get to it.

4.10 Packets configuration at each round (2D-Only)

The configuration of a node is the direction of the 1st and 2nd choice for all the packets in the node. That is, how many nodes have 4 packets with the same first and 2nd choices, vs. those with 4 different choices?

This allows us to compare our system with that of a completely independent system. If we generate the packet directions randomly, we can calculate the number of configurations we have, and the probability of a given node to be in a specific configuration. We can see how our system compares with the independent system. Do more nodes end up in a particular configuration than it otherwise should? Or does it closely resemble the random independent system?

5 Results

In this section we discuss the simulation results and our analysis on these results.

For our analysis, we performed various simulations on 1 - 6D torus of various sizes. We hope to answer the following questions based on the simulation:

- What happens to the system as time goes by? Are packets being delivered? Are they going where they needed to go?
- Does the system reach a steady state? What % of packets are going in the right direction? As a packet gets closer to its destination, does it become more difficult for it to make progress toward its destination?
- How long does it take for packets to be delivered? What is the expected delivery time of a packet with distance x to its destination?
- How do torus size and the number of dimensions affect packet delivery time? Do higher dimensions imply longer delivery time? Or shorter delivery time?
- What is the system throughput? How many packets are being delivered? How close is the system to the theoretical optimal?

In the following data, we used abbreviations for the packet generation model. EP refers to equal probabilistic destination. UD refers to uniform distance to destination.

5.1 Resource Utilization

Here is some information on the resource utilization of our simulation. This will provide some insight into how torus dimension and size translates into memory and cpu usage, and what size torus can be practically simulated on current systems.

Our simulation system is a 2-core hyper-threading P4 system with 2GB of RAM. Although the size of each dimension can be up to 65536, and the total number of nodes in the system can be up to 2^32, in practice it is much smaller than this limit because the packets themselves take up a large portion of the memory in higher dimensions, and higher dimensions do have lots of packets.

- A 1-dimensional torus of size 65536 has 131072 packets and uses about 7MB of ram. Simulating 1000 rounds takes about a minute.
- A 3-dimensional torus of size 128 has 12582912 packets and uses about 456MB of ram. Simulating 1000 rounds takes about 5 hours.
- A 6-dimensional torus of size 11 has 21258732 packets, and uses about 1034MB of ram. Simulating 1000 rounds takes about 17 hours.
- An 8-dimensional torus of size 6 has 26873856 packets, and uses about 1500MB of ram. Simulating 1000 rounds takes about 30 hours.

Each packet is 4d+7 bytes (4d bytes for the start and destination location, 7 bytes for tracking data such as id, hops traveled, etc). Each node is 16d + 8 bytes, essentially 2 arrays of pointers representing the receiving and sending buffer.

5.2 Delivery Time

The first metric we will examine is delivery time. This is the measure of how long it takes to deliver a packet from the source to its destination. This important metric allows us to see how fast packets are being delivered, what the expected packet delivery time is, and how delivery time changes with respect to change in initial packet distance.

5.2.1 Simulation Setup

We ran the following simulations on 1-6D tori. Both equal probabilistic destination and uniform distance to destination packet generation models were used. We started collecting statistics after 120 rounds, to wait for the system to reach a stable state get a better average of the delivery time. The simulation ends when all the packets at the ending round are delivered. Therefore the actual number of rounds simulated is larger than the set round.

The different torus sizes are picked so the average packet distance is the same across dimensions. This allows us to compare routing performance across dimensions.

Here are the torus sizes we used and the simulation setup:

Dimension	***Torus Size***	***Average Distance***	***Rounds***
1	60	15	360
2	30	15	360
3	20	15	360
4	15	14 (UD), 14 14/15 (EP)	360
5	12	15	360
6	10	15	360

For 4D, the torus size is an odd number. The average distance is not 15 as with the other dimensions where the torus size is an even number. (We can make all the torus sizes larger, however the larger dimensions would take too long to simulate).

In 4D, the maximum distance of each dimension is 7. The maximum total distance is 28. Therefore, the average distance is 14, not 15, in the uniform distance to destination model.

In equal probabilistic model, the average distance is 14 14/15. We pick each of the 15 nodes for each dimension with equal probability. This results in an average distance of

4 * (0/15 + 2 * (1/15 + 2/15 + … 7/15)) = 4 * (56 / 15) = 14 14/15 = 14.933333.

Because the 4D average distance is not the same as the rest of the dimensions, it will affect some of our results, as we will note later.

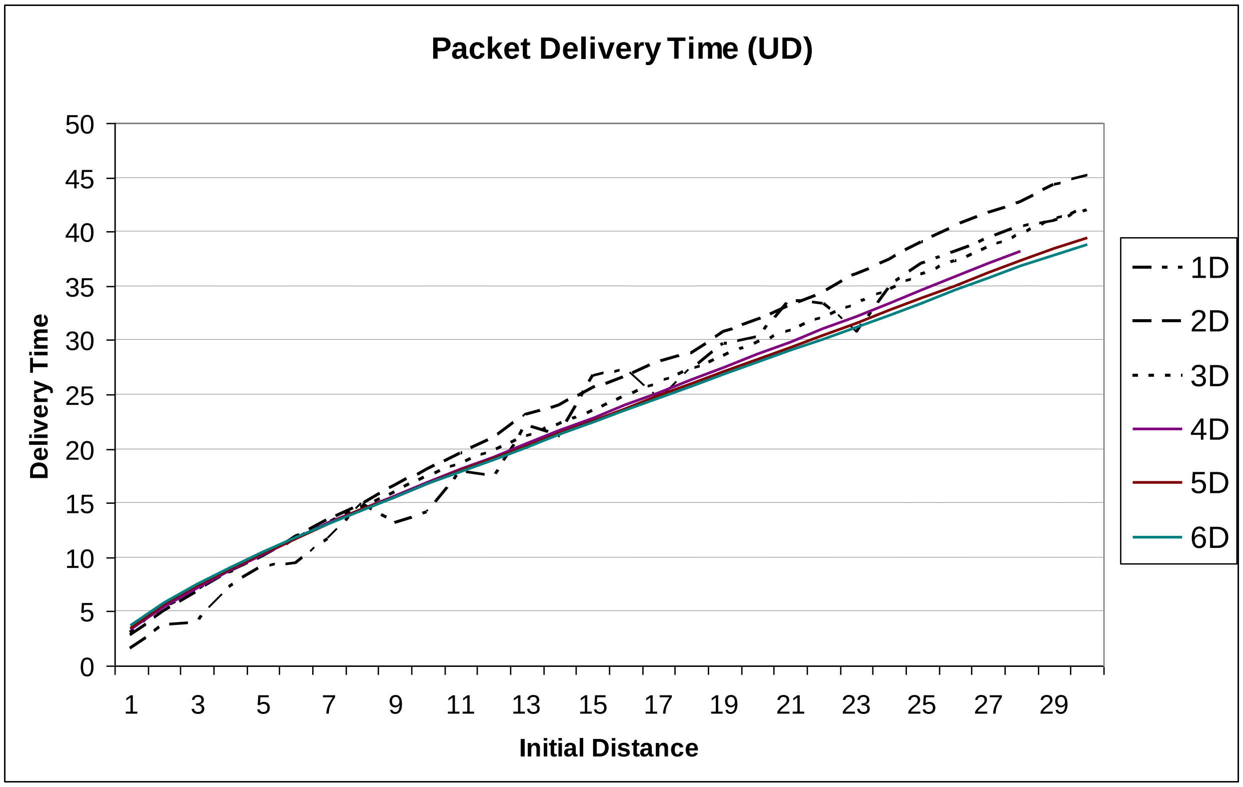
5.2.2 Delivery Time (Uniform Distance to Destination)

We ran the simulation for delivery time using uniform distance to destination packet generation model. The set round is 360.

This is the number of rounds the simulation actually ran to deliver all the packets in the system at round 360. Approximately 100 extra rounds were needed to delivery all the packets. Note the 4D case with average initial distance 14 instead of 15.

	1D	***2D***	***3D***	***4D***	***5D***	***6D***
Actual Round	456	456	440	439	472	464
Average Initial Distance	15.045958	14.947924	14.992438	*14.000414*	14.997775	14.999822
Average Delivery Time	23.551943	25.160409	23.611064	21.555919	22.455879	22.269574

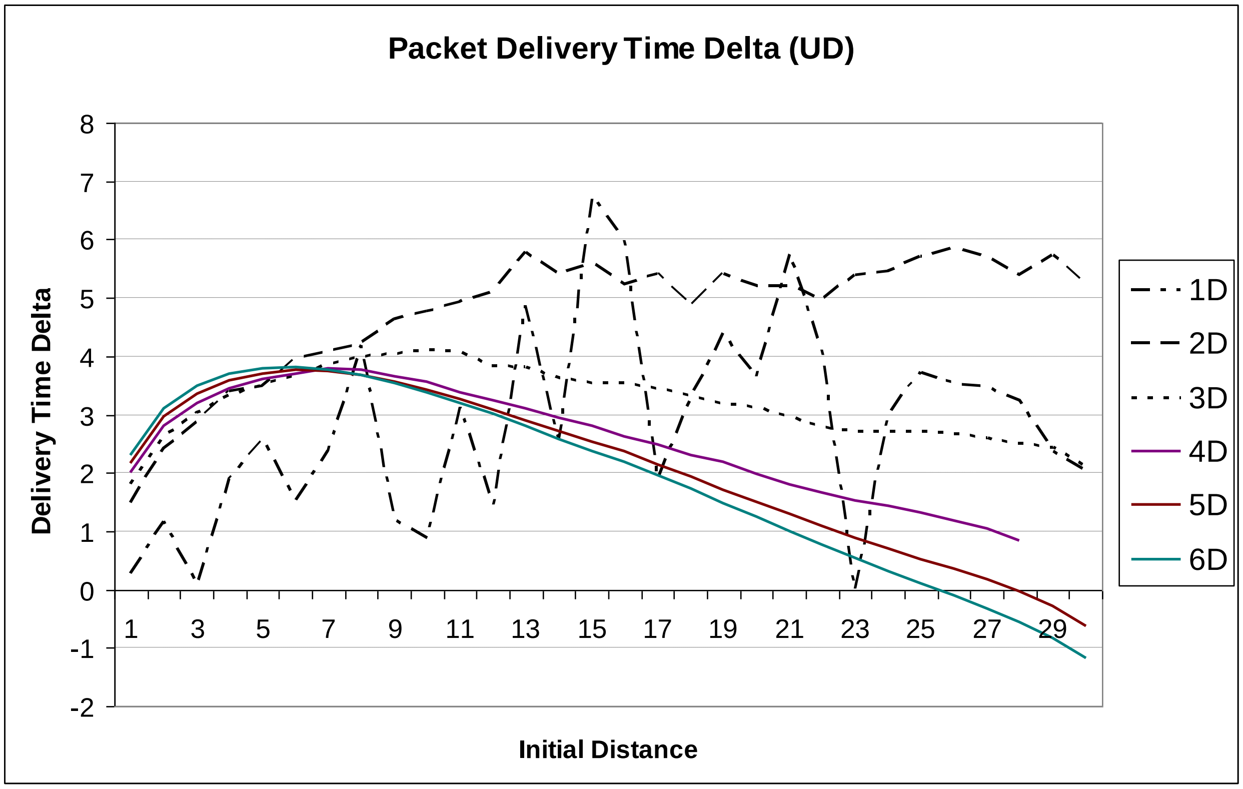
Here is the plotting of packet delivery time against its initial starting distance:



The chart starts at initial distance 1, since 0 distance packets take 0 time to get delivered, so we do not include them in our chart. In addition, for 4D, the initial distance only goes up to 28 (4 * 7). Therefore the line does not cover the entire range.

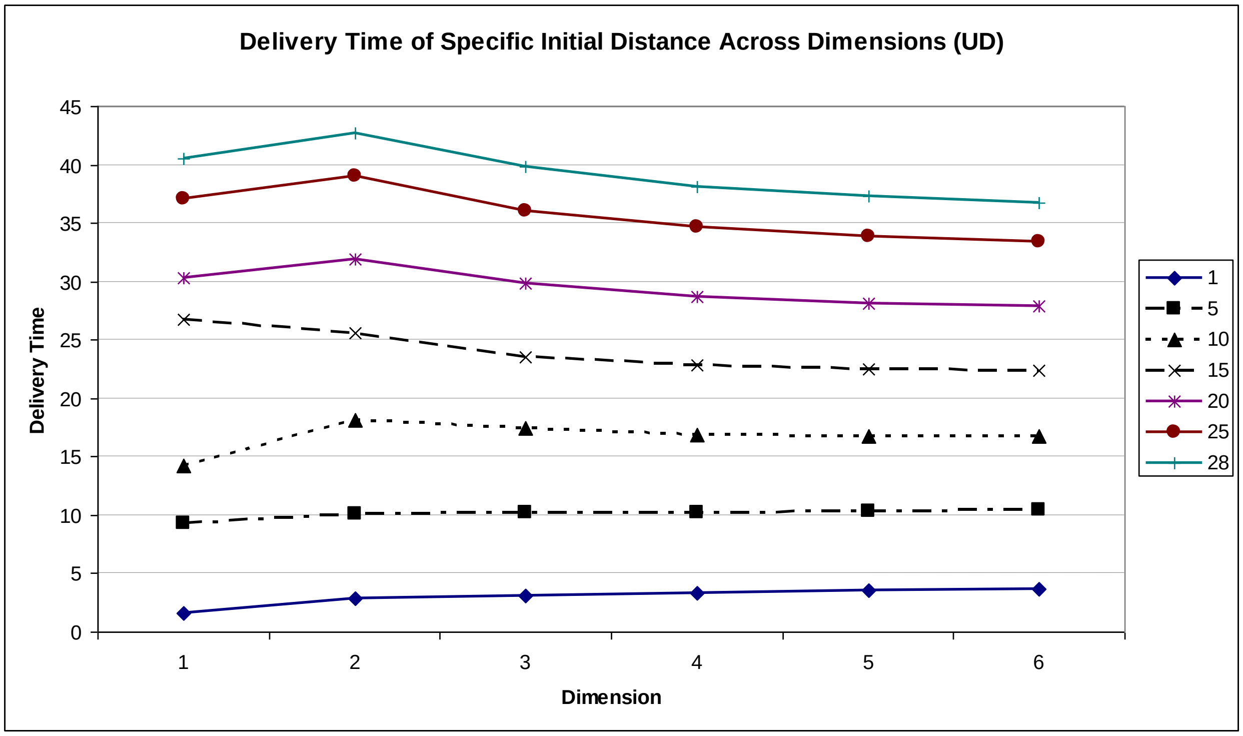
From the data, we can see that delivery time grows linearly with respect to initial distance, for all dimensions. This indicates delivery time is consistent across the entire range of packet distances for all dimensions without any major anomalies in its delivery time. It's our first indication that the system is behaving well. Linear delivery time is very good and the constant does not appear to be large either, roughly about 1.3. There does appear to be a bit of an overhead, noted by the non-zero intersect of about 2.

For 2D to 6D, the delivery time is slightly smaller for higher dimensions. 1D does not follow this trend, even if we increase the number of rounds to smooth out the average. To more easily see the trend difference between the various dimensions, we plot the delivery time delta against a standard reference approximation of 4/3 initial distance. Here's the result:



The chart shows the delivery time difference from that of a reference 4/3 initial distance plot, which is the x-axis. From the data, we see that with the exception of 1D, higher dimensions contribute to lower delivery time, for packets with initial distance 7 or larger.

Let us examine the same data in yet another way. We plot the delivery time of a specific distance, 1, 5, 10, etc, for the different dimensions. (Remember the torus sizes are chosen such that the average and maximum distance for packets are the same). This allows us to more easily how dimensions affect delivery time.



The results are encouraging. We see that dimensions contribute positively to the delivery time of a packet, especially for longer distance packets. This makes sense if we think about it. In higher dimensions, there are more ways for the packet to be traveling in the 'right direction' than it is in lower directions. As a result, the packet is more likely to be traveling in the right direction than in lower dimensions. This should result in decreased delivery time.

For example, in 1-dimension, a packet is either traveling the right direction, or in the wrong direction, it's one way or the other and nothing in between. But in 2-dimension, there could be 2 'right' directions, and only if both of these directions are not available does the packet go in the 'wrong' direction.

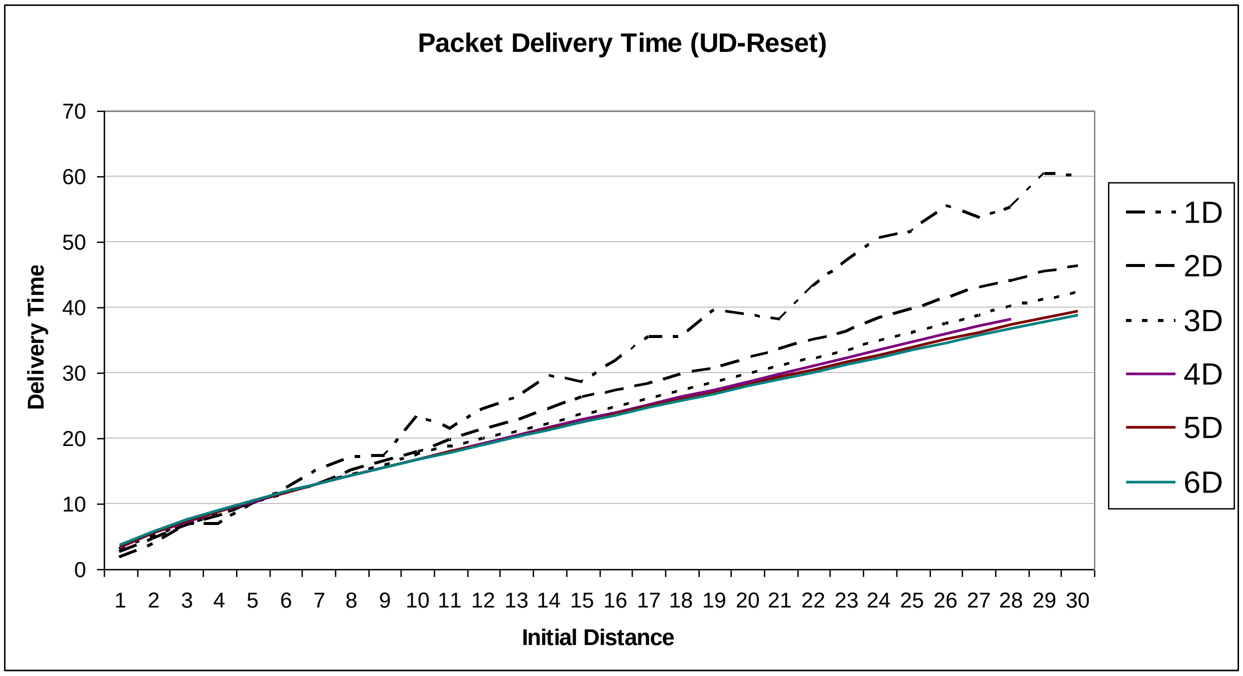
Note again that 1D result does not quite match the trend, with better results than the 2D system.

5.2.3 Delivery Time (Uniform Distance to Destination, Reset Direction)

To see how directional dependencies affect the delivery time, we re-ran the above simulation, but using the reset direction option.

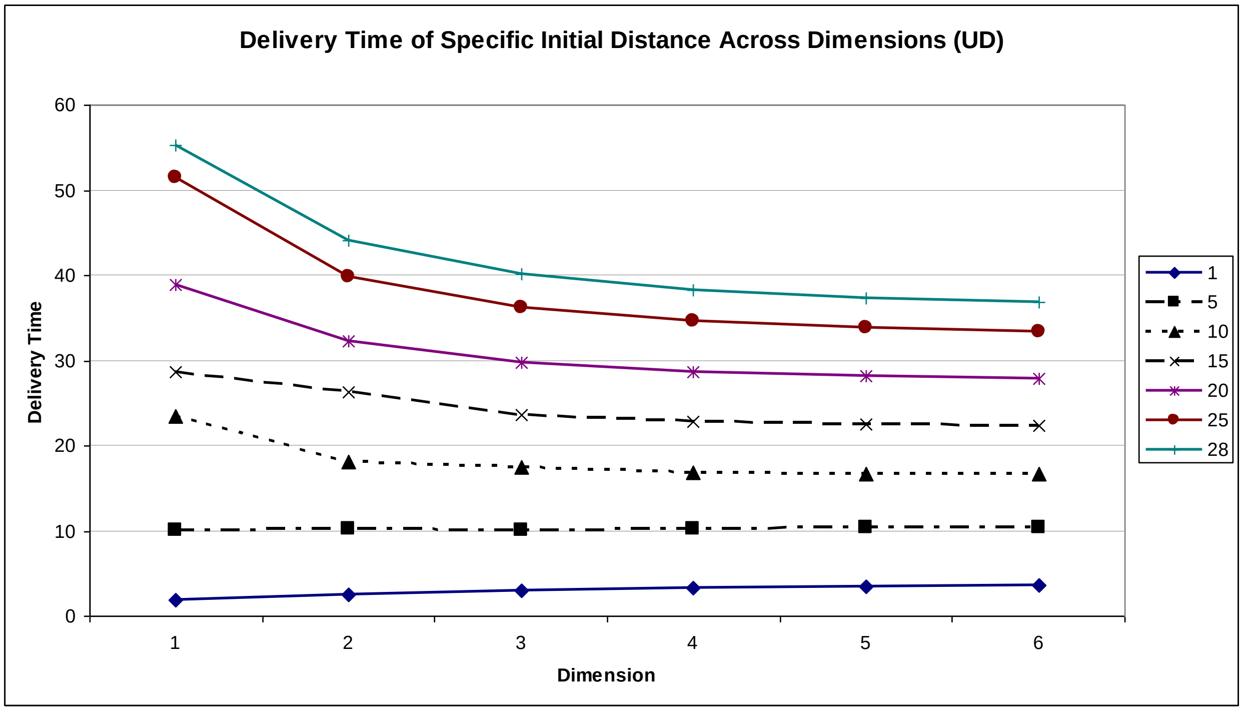
	1D	***2D***	***3D***	***4D***	***5D***	***6D***
Actual Round	432	428	441	449	449	457
Average Initial Distance	14.921472	15.022666	14.982343	14.003154	15.002155	15.000401
Average Delivery Time	31.470046	25.631999	23.64371	21.529618	22.449131	22.255259

Here are the results:



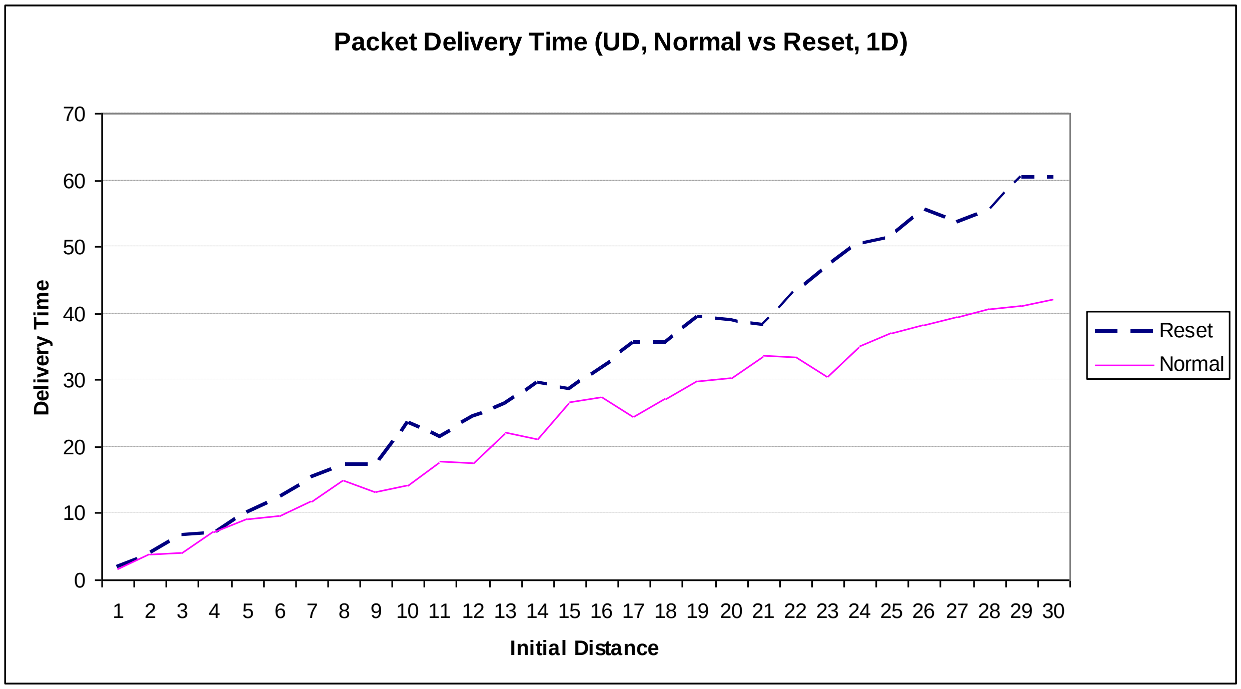
Results are similar to the regular UD simulation, with 1D being higher than the other dimensions this time. The growth rate is linear with similar constant.

Here's the 2nd plot comparing delivery time across dimensions:

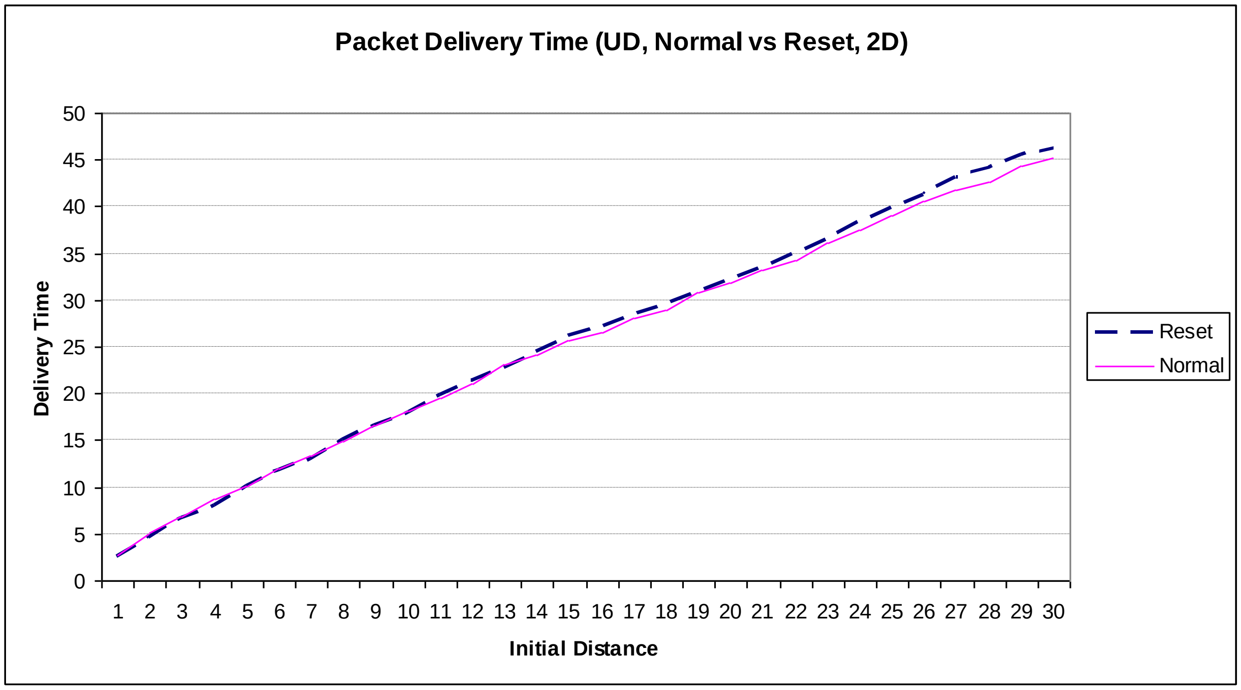


Unlike the UD simulation, 1D does follows the trend of the remaining dimensions. The advantage in delivery time is still there for higher dimensions for long distance packets.

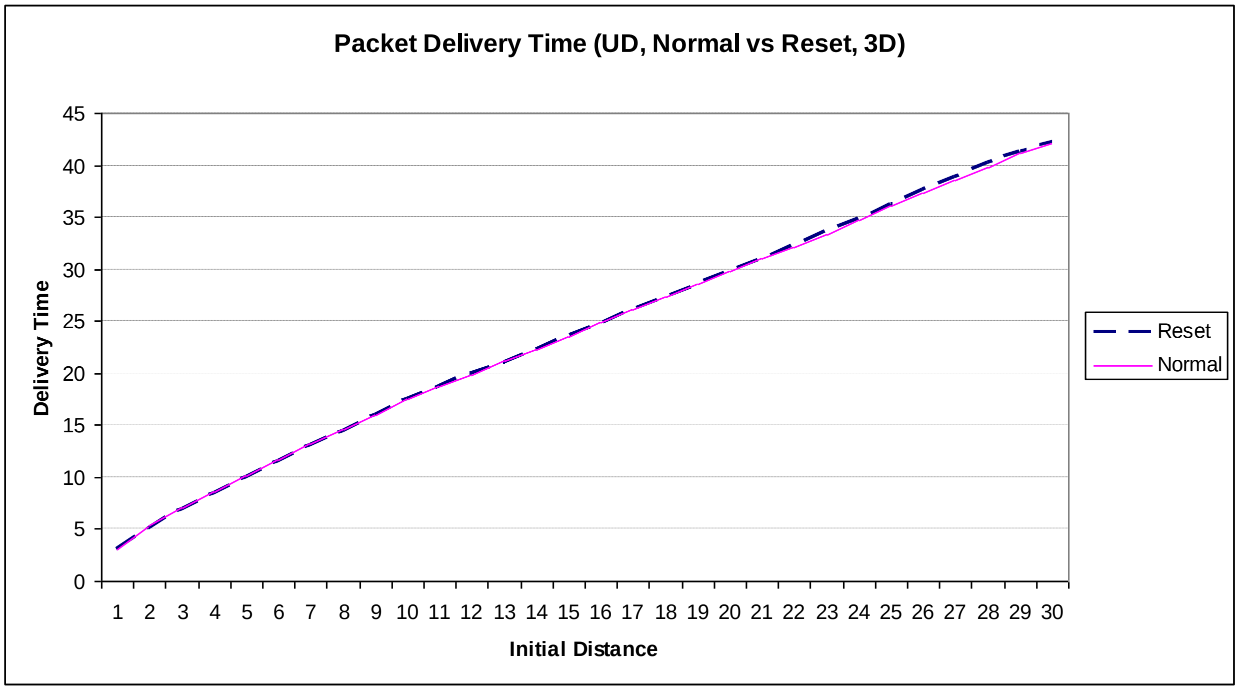
Let’s now take a look at how the reset option affects delivery time, for each dimension:



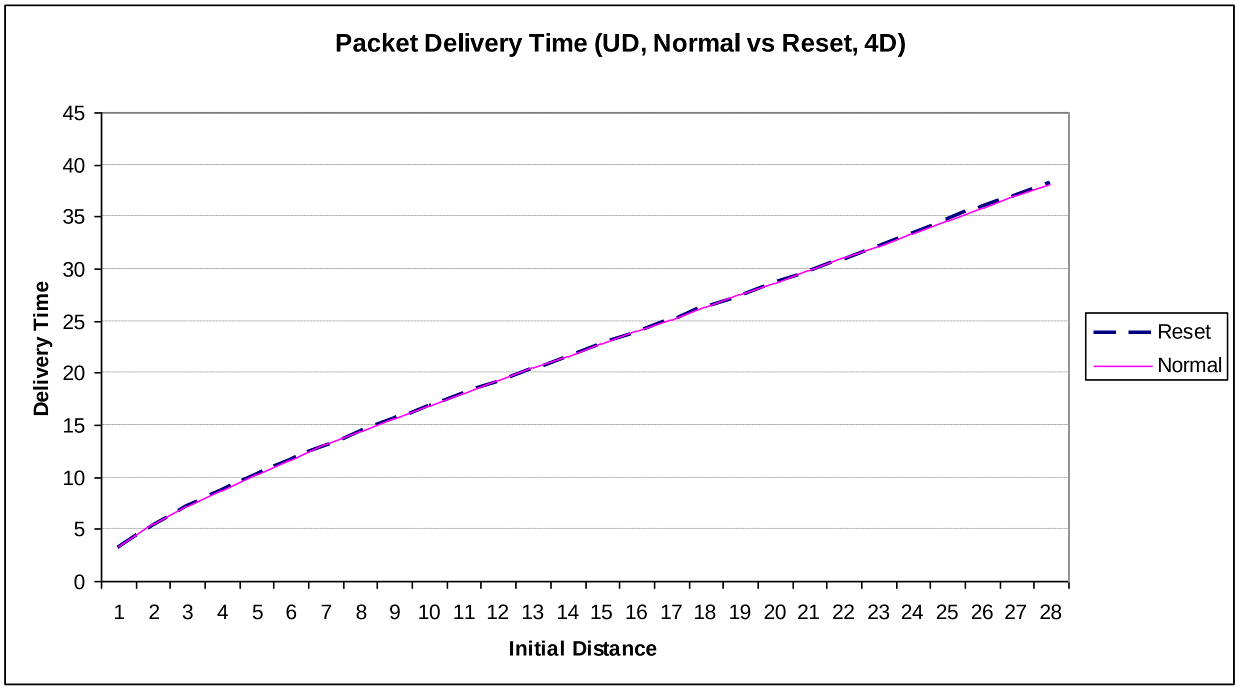
In 1D, delivery time is longer with reset direction option. This indicates directional dependency improves delivery time, as packet momentum helps cutting down delivery time. Reset direction removes any momentum from routing, therefore the delivery time is longer.



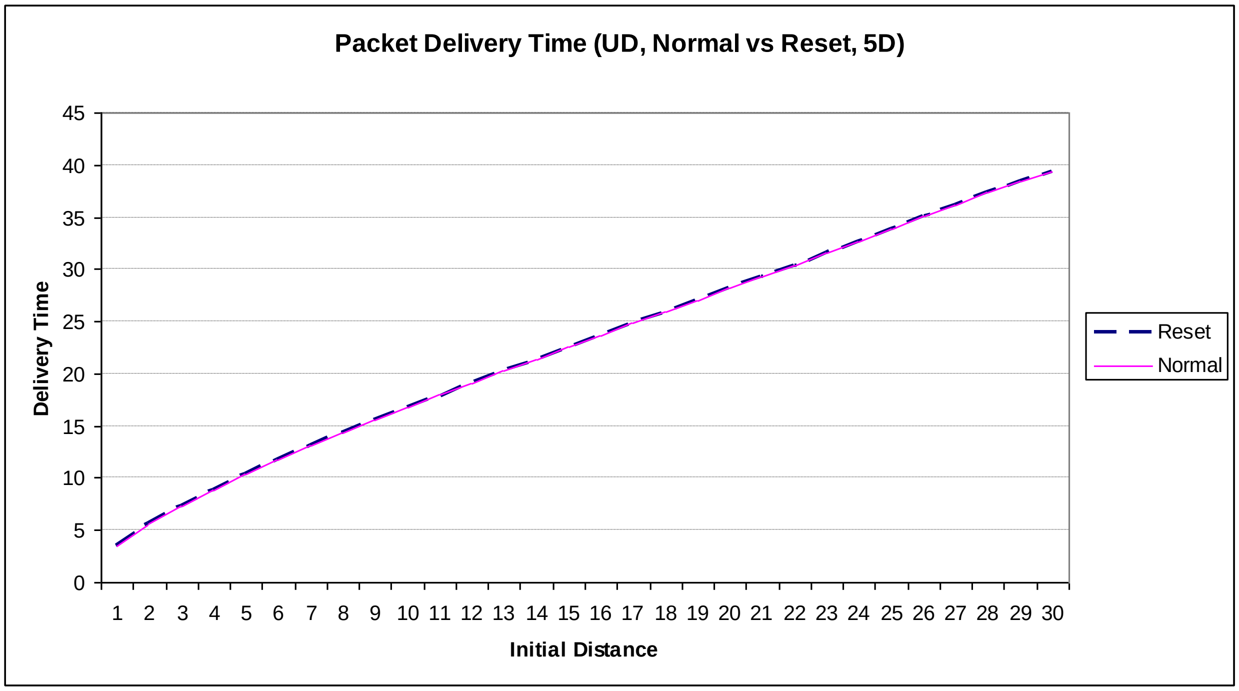
In 2D, reset direction also negatively affects delivery time, however the differences are much less than those in 1D.

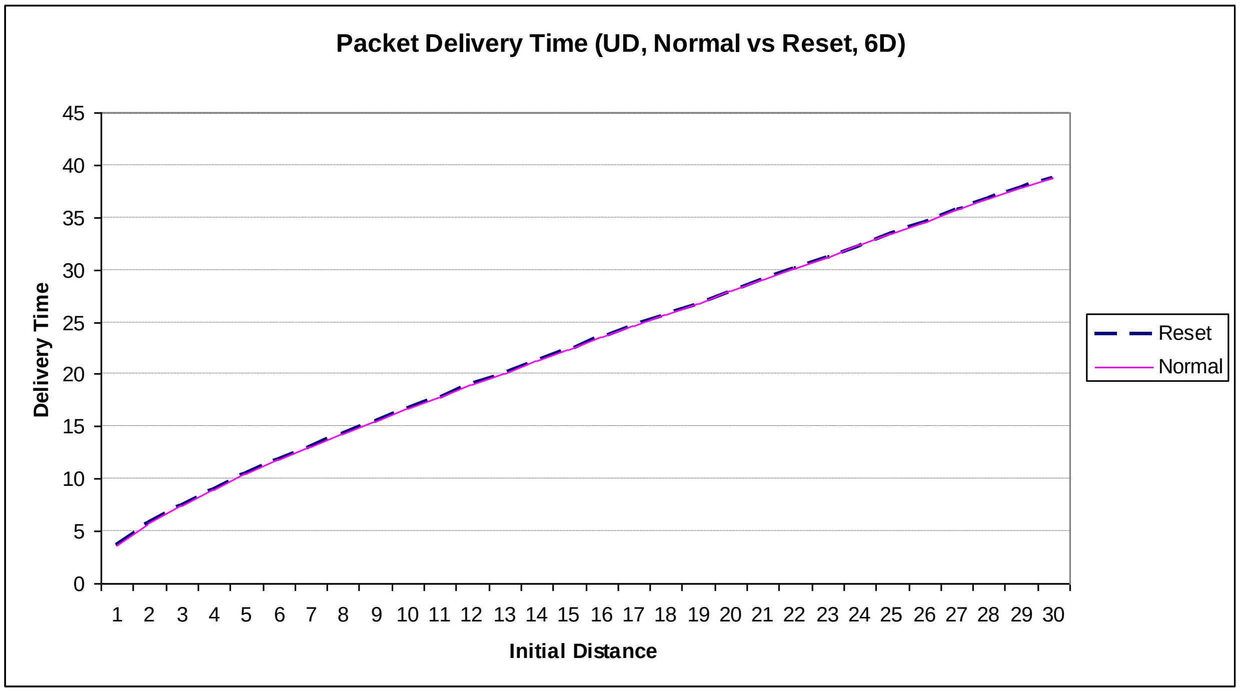


In 3D, the differences are minimal.



4D and higher there are no difference as the 2 line tracks each other precisely.





As we can see directional dependency improves delivery time most notably in 1D and 2D. In higher dimensions, there isn't really a difference. Looking at the raw data, the reset direction delivery time is slightly larger, but the difference really is minute.

It is important to note that in all cases, directional dependency helps us, i.e. the system is slightly better than that of a completely independent system.

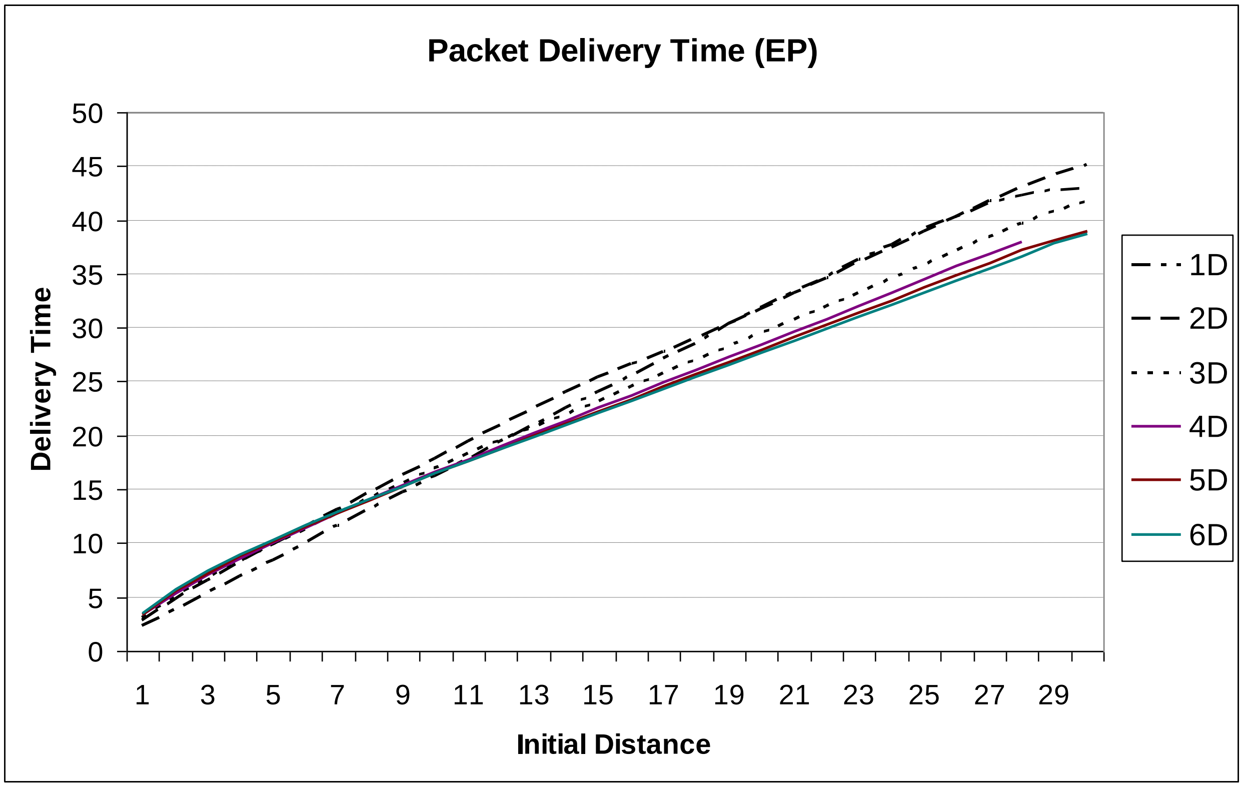
5.2.4 Delivery Time (Equal Probabilistic Destination)

Let's take a look at delivery time using equal probabilistic destination packet generation. We want to see if there are any routing performance differences using the other model.

For this simulation, we set the number of rounds to be significantly higher for lower dimensions, to get a better average of the delivery time.

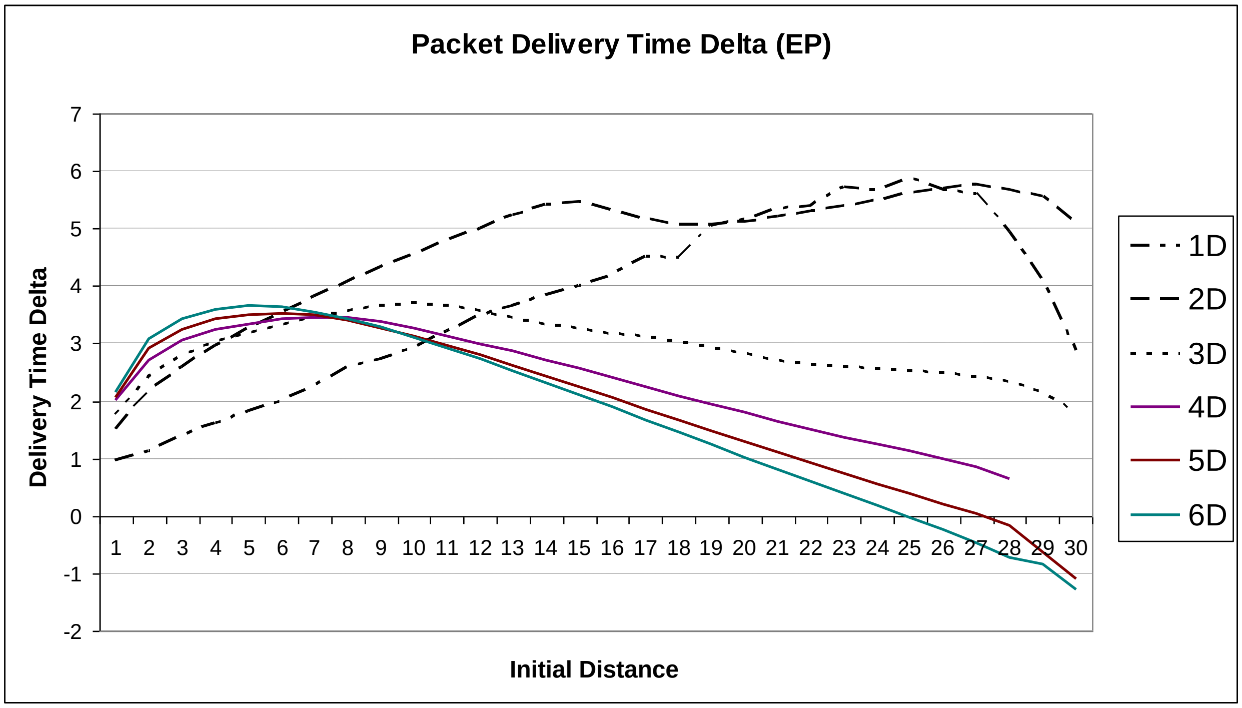
	1D	*2D*	*3D*	*4D*	*5D*	*6D*
Actual Round	100000	100000	100000	10000	1120	600
Average Initial Distance	14.998545	14.999843	15.000145	**14.933647**	15.000228	14.999806
Average Delivery Time	23.695799	24.884063	23.20561	22.440261	22.214286	22.083824

Note for 4D, the average initial distance is not 14 as with UD, but 14 14/15. In equal probabilistic destination, the simulation result confirms this.



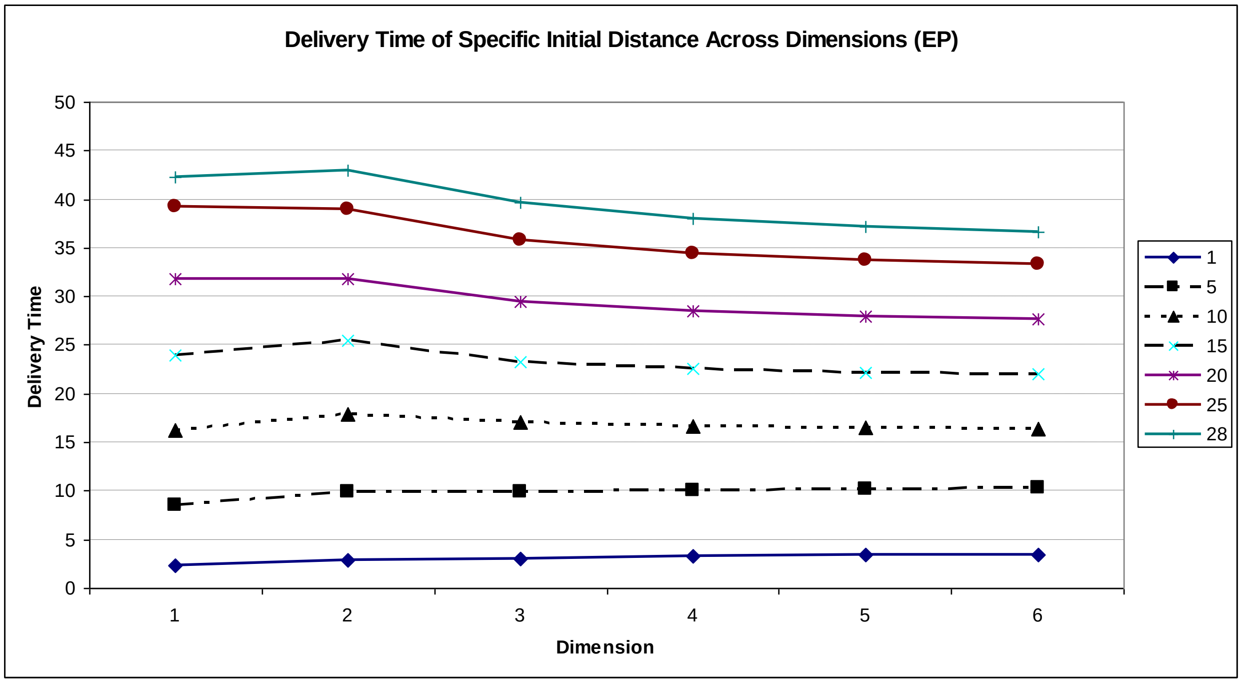
The data looks very similar to uniform distance to destination model. The 1D data is a lot smoother now, thanks to the larger number of packets used to do the average due to the larger number of rounds.

Here's the delivery time delta plot comparing to 4/3 initial distance:



Similar to the Packet Delivery Time Delta (UD) plot, from 2D to 6D, delivery time is lower for higher dimensions, for initial distance 8 and higher. The 1D data is smoother because the EP simulation used a significantly higher number of rounds for lower dimensions.

The delivery time is linear with respect to initial distance, and higher dimensions have lower delivery time, as the next chart shows:



The above data shows routing is consistent for both of our packet generation models. This is good, as it says our routing is not impacted by the packet distance distribution, as in equal probabilistic destination model, there are significantly more packets having average distance in higher dimensions.

5.2.5 2D Delivery Analysis

In our analysis of delivery time above, we grouped packets by initial distance. A packet that is x away is expected to be delivered faster than a packet that is x+1 away. It would be interesting to know how change in the initial distance vector for the same packet distance affects delivery time. For example, is delivery time faster for packets having distance <5, 5> vs. packets of distance <10, 0>?

The <5, 5> packets should have a more flexible route than the <10, 0> packets, since they has 2 directions they can travel that will bring it closer to destination instead of 1 for the <10, 0> packets. We want to know if this analysis holds for all the combinations of distance vectors in general.

We ran a more detailed simulation on a 2D torus of size 30 for 100,000 rounds, group packets by their initial distance vector, folding symmetry between x/y coordinates (for example, <2, 3> is considered the same as <3, 2> in our grouping). Statistics are collected after 120 rounds, to allow the system to reach a stable state.

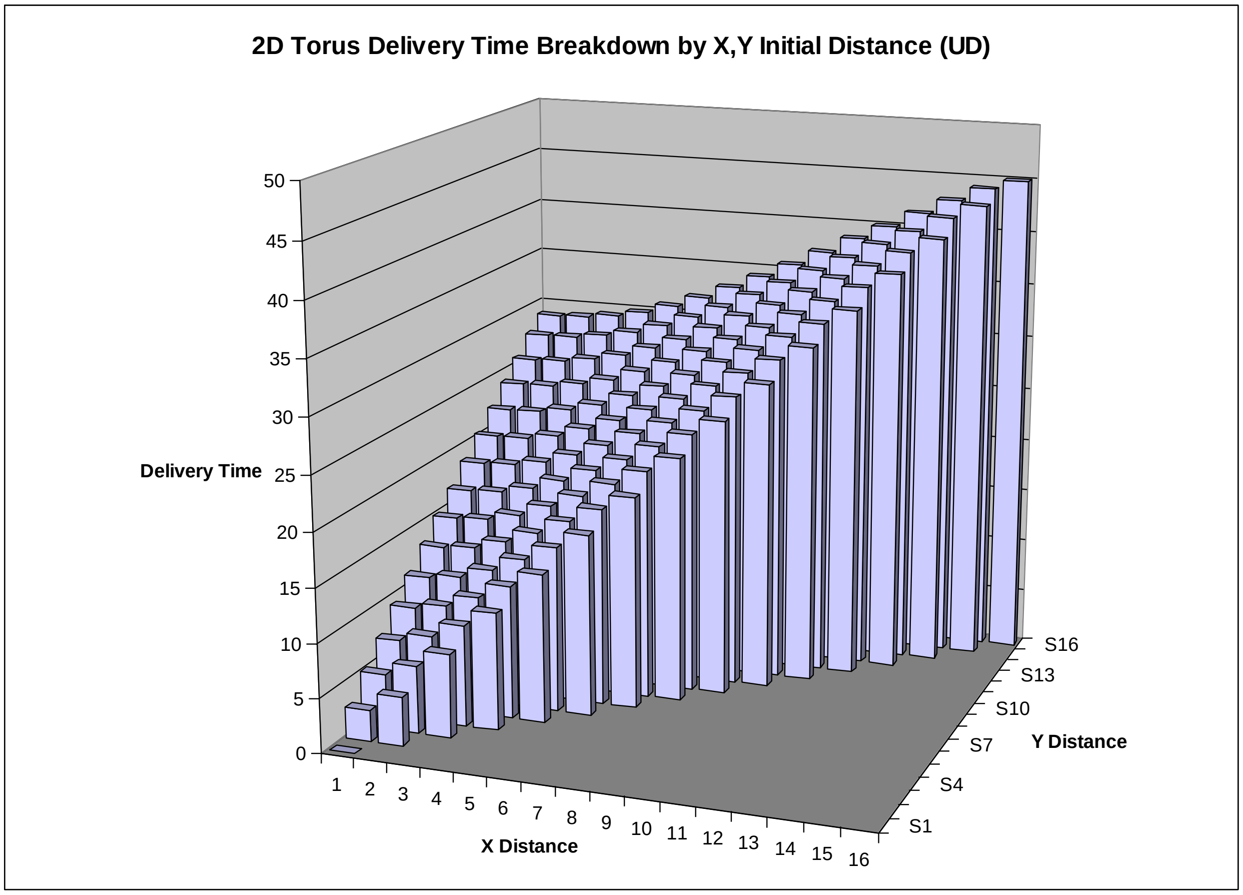
Here are the results in a tabular format:

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	0.00	2.83	5.11	7.21	9.24	11.18	13.10	14.98	16.81	18.63	20.42	22.24	24.05	25.75	27.51	28.83
1		4.48	6.21	7.99	9.77	11.56	13.39	15.19	17.01	18.78	20.57	22.35	24.18	25.88	27.59	28.96
2			7.74	9.31	10.92	12.55	14.27	15.93	17.64	19.35	21.08	22.85	24.60	26.31	27.99	29.28
3				10.83	12.34	13.89	15.45	17.06	18.68	20.34	22.00	23.59	25.30	26.99	28.56	29.84
4					13.79	15.30	16.82	18.38	19.91	21.52	23.11	24.75	26.36	27.96	29.45	30.74
5						16.81	18.32	19.81	21.31	22.89	24.39	25.89	27.50	29.04	30.54	31.80
6							19.73	21.29	22.77	24.27	25.75	27.28	28.81	30.36	31.75	33.02
7								22.75	24.23	25.66	27.18	28.72	30.20	31.76	33.16	34.31
8									25.66	27.17	28.66	30.17	31.58	33.06	34.58	35.70
9										28.64	30.14	31.61	33.08	34.59	35.99	37.10
10											31.57	33.09	34.53	36.05	37.39	38.61
11												34.54	36.04	37.44	38.89	40.02
12													37.46	38.93	40.33	41.47
13														40.37	41.79	42.88
14															43.09	44.14
15																45.03

The row and column heading represents the initial distance vector. The table is half empty because we group distance vector <x, y> and <y, x> together. The diagonal of the table, represented by the direction of the 2 lines of shaded cells, are packets with the same initial distance, but with different initial distance vectors.

From the shaded diagonal lines, we see that delivery time is smaller as you closer to the center, represents packets with more even distance distribution. Delivery time for <15, 0> is 28.83, delivery time for <8, 7> is 24.23, noticeably less.

Here is the same data in a 3d plot:



The chart plots delivery time (z-axis) against the packet's initial distance vector (x/y-axis). Half of the x/y values have no data because we fold symmetrical vectors.

Similar to the analysis of the table, if you visualize a line from <0, 15> to <7, 8>, you see the bar decreases gradually from the edge to the center.

Apply this analysis to the rest of the data, we see that for the same distance, the curve slopes down towards the center. Packets with more evenly distributed distance vectors have a smaller delivery time. This trend can be observed for all distance vectors, confirming our theory that flexibility improves routing performance, as packets have more correct paths to travel than less flexible packets.

5.2.6 Delivery Time Trend

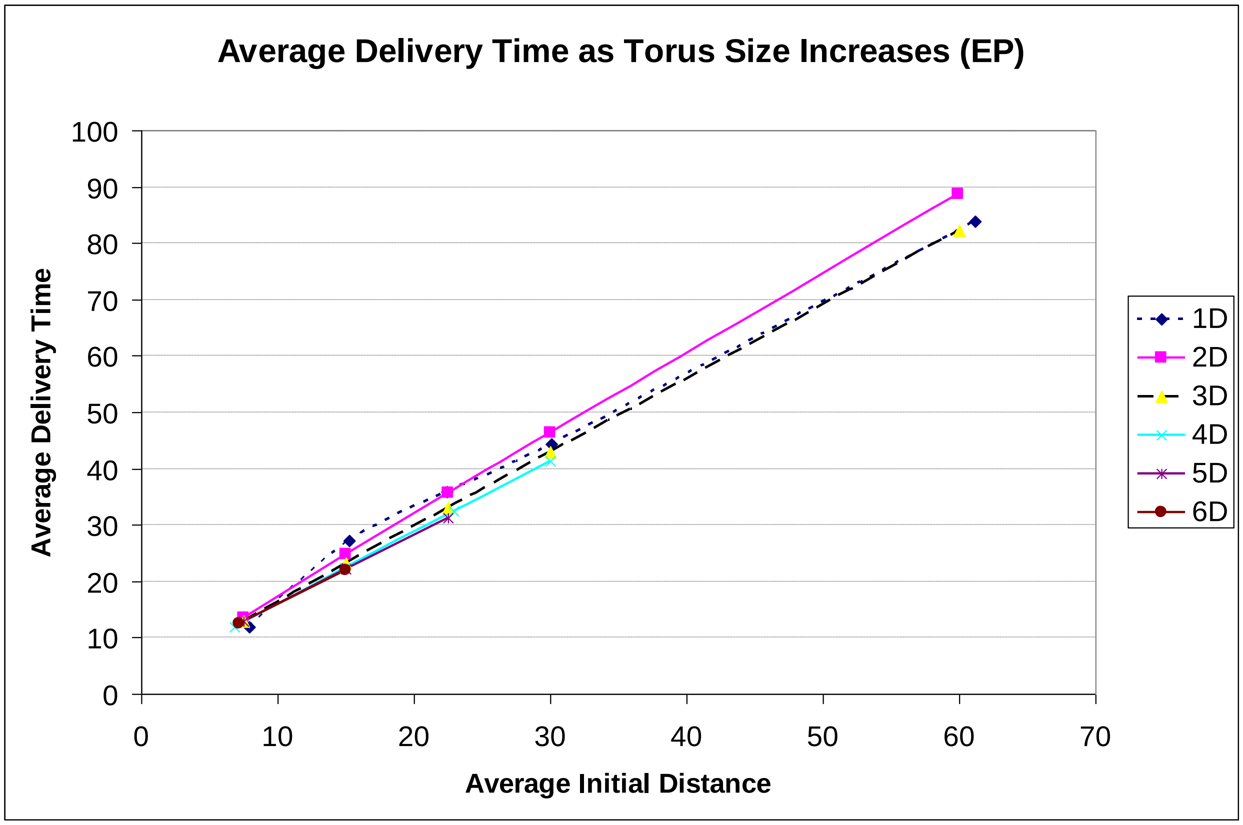
Let's now take a look at the delivery time for the various dimensions while changing the torus size. For these simulations, we computed the overall average packet delivery time vs. the initial packet distance across dimensions.

Here are the torus sizes we used. For most of the runs, we set the # of rounds to be 8x average initial distance, and run the simulation until all packets at round 8x average are delivered.

1D Size	*2D Size*	*3D Size*	*4D Size*	*5D Size*	*6D Size*
30	15	10	7	6	5
60	30	20	15	12	10
90	45	30	23	18	
120	60	40	30		
240	120	80			

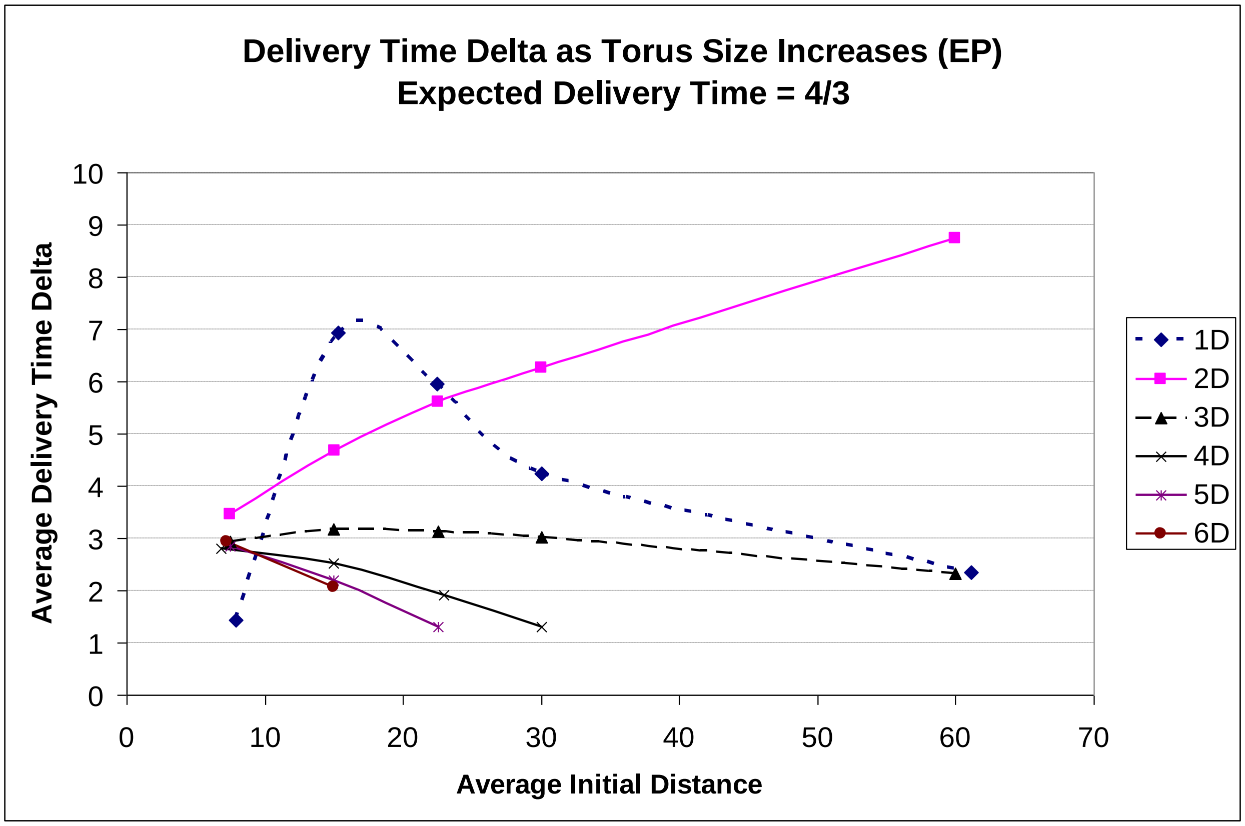
We were not able to completely simulate some of the higher dimension sizes due to cpu/memory constraints. There should be enough data to see the trend.

Here are the results of the simulations:



We see that as torus size increases, the average delivery time increases linearly as well. The data is very consistent and no anomalies observed, except for the 1D data, which we represented using dotted lines. This tells us that our routing algorithm and packet delivery seems stable, independent of the torus size.

To better understand the difference between the various dimensions, we plot the data in a different format. The delivery time is linear, and the value is approximately 4/3 of initial distance. We plot the difference of the delivery time from this approximation. Here's the result:



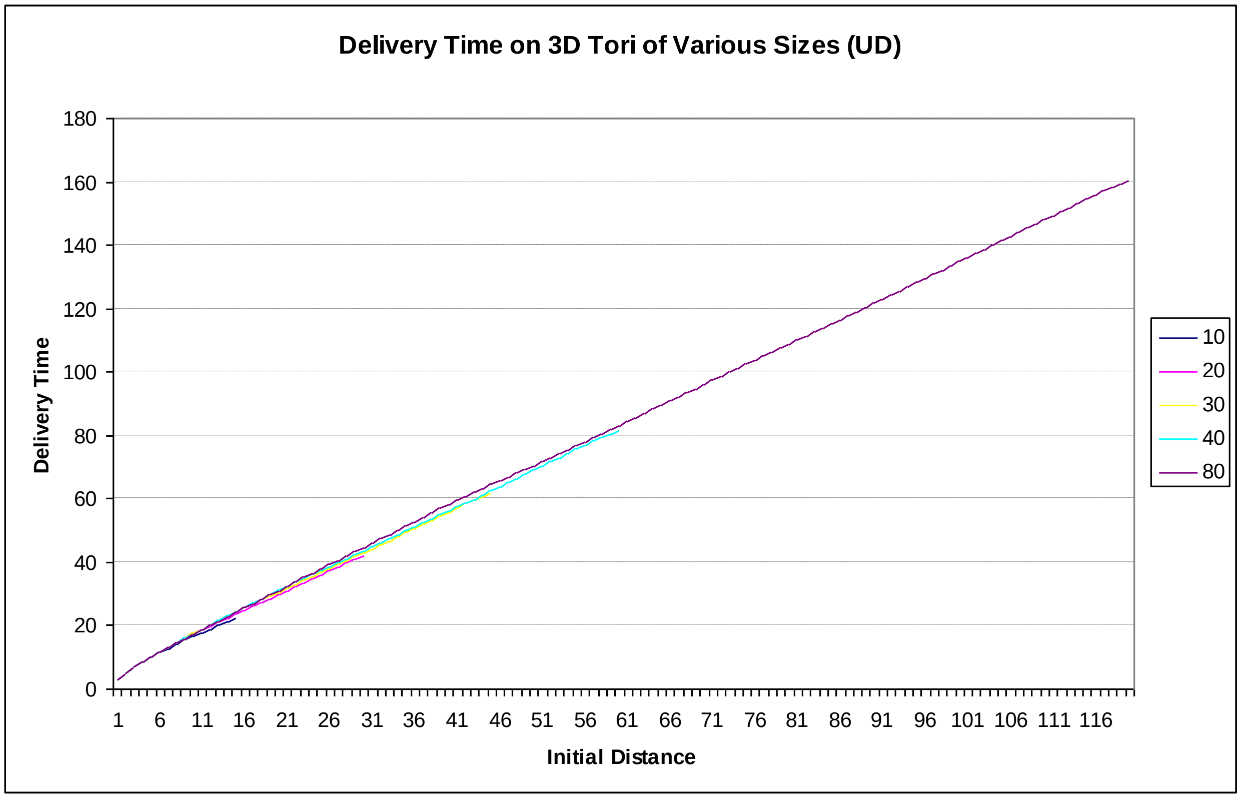
Ignoring the 1D data, we can clearly see a trend where higher dimensions contribute to faster delivery time, consistently from 2D to 6D. The 1D data difference is probably due to more helpful dependence between packets, as seen in section 5.2.3.

5.2.6.1 Delivery Time Trend on 3D Tori

We saw the average delivery time of the various tori in the previous section. Let's now take a look at the 3D system in more detail, tracking delivery time of each initial distance as the torus size increases.

We ran simulation on the 3D torus system using the uniform distance to destination model. The number of rounds are set to 12x average distance, and continue until all packets at the ending round are delivered. Statistics are collected after 3x average distance rounds, to allow steady state to be reached for a better average.

Here are the simulation results, plotting the delivery time against the packet's initial distance for the various tori sizes.

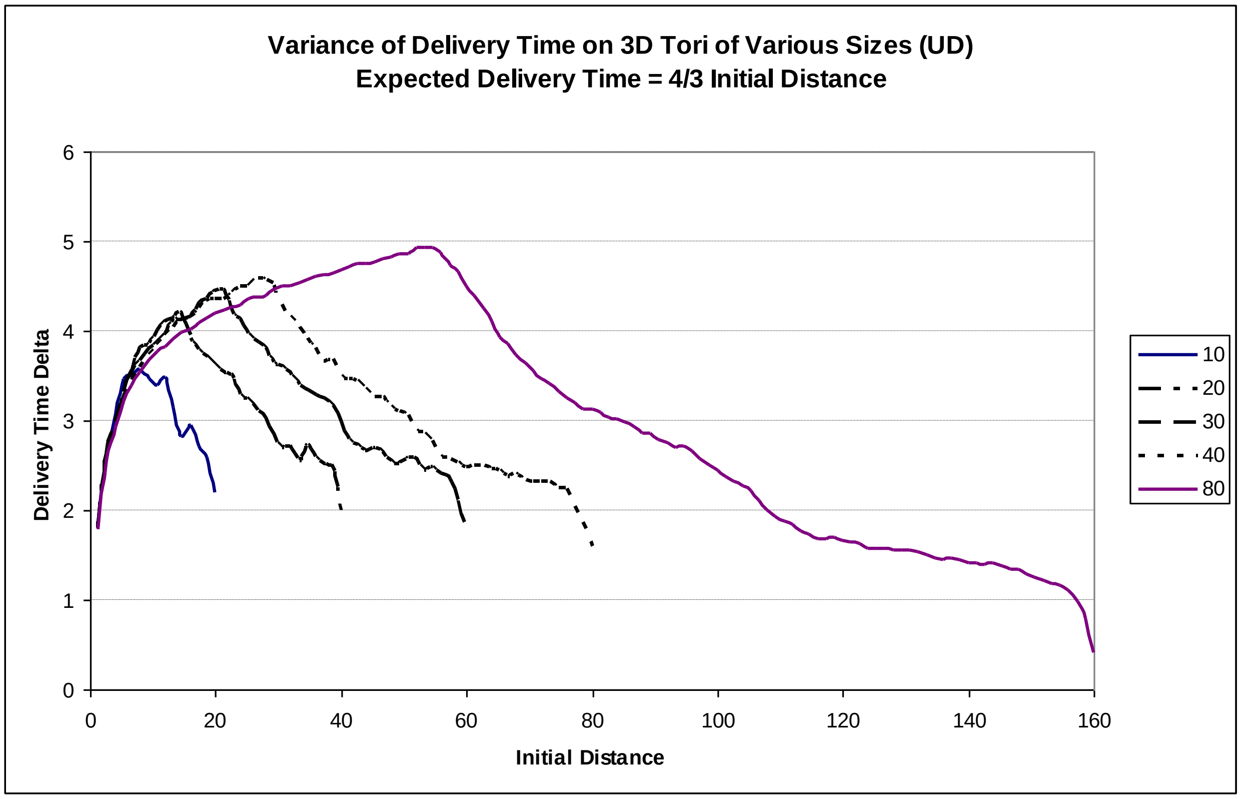


The maximum initial distance for the various torus is 15, 30, 45, 60, and 120 respectively, 3*s/2 in a 3D system. Therefore not all the lines are of same length.

From the chart, we see the delivery rate is consistently linear for all the torus sizes. Increasing torus size does not really affect the packet's delivery time. This is good, indicating that packet's delivery time is independent of the torus size.

There is one interesting observation to be made the chart. For each torus size, the delivery time for the upper half of the initial distance range is smaller than the delivery time in the next larger torus. In other words, packets that are in the upper half of the distance range, have a slightly shorter delivery time than they do in a larger sized torus.

It is not easy to see this from the previous chart, since the lines are very close to one another. Here we present the data in a different way, plotting the difference between the expected delivery time (which is approximately 4/3 of initial distance) and the actual delivery time.



The x-axis represents the difference from a delivery time of 4/3 initial distance. From the chart, we observe 2 interesting points. First, for each torus, there is a peak in delivery time delta from our reference line, at approximately 1/3 initial distance. The higher the initial distance, the larger the delta. After the peak, the delivery time delta drops steadily for larger initial distance. Second, for the same initial distance that are in the upper 2/3 section, delivery time in the next larger torus is longer (larger delta value). For example, the delivery time of initial distance 26 is fastest in size 20 torus, followed by 30, 40, and finally size 80 torus.

There are two contributing factors to this. First, the torus wraps around, therefore the delivery time for high distance packets would be slightly better than if the torus size is infinite. In an infinite torus, if higher distance packets go in the wrong direction, they will need to 'undo' these steps before they can reach the destination. In our system, since the torus wraps around, these packets may suddenly be traveling in the right direction after a few wrong directions, reducing its needed delivery time. Second, for the same distance packets, they will likely have more flexible routes in smaller torus than they do in larger torus. For example, for distance 6, it is possible to have no flexibility in routing for a size 80 torus, but must have some flexibility for size 10 torus. More flexibility implies better delivery time.

Therefore, the delivery curve for all tori have a slight bias toward faster delivery time for the 2nd half of the initial distance range. If we want to accurately simulate delivery time for distance x, we should simulate it on a larger size torus where x is well within the first half of the distance range. (In practice, this doesn't really make too much of a difference, since the differences are quite small.)

5.3 Delivery Rate

We've analyzed delivery time in detail. Now let's take a look at another important metric, packet delivery rate. This is simply a measurement of how many packets are being delivered. Of course, for steady state, this can be calculated from the delivery time, but it is interesting to observe this from initial state to steady state to see how it changes.

5.3.1 Simulation Setup

For this simulation, we used the same torus sizes as we used in the previous section, keeping the average initial distance same for all the dimensions.

We ran simulation on 3 different scenarios:

- Normal starting state (uniform distance to destination)
- Starting from the bad state
- Resetting packet directions at each round.

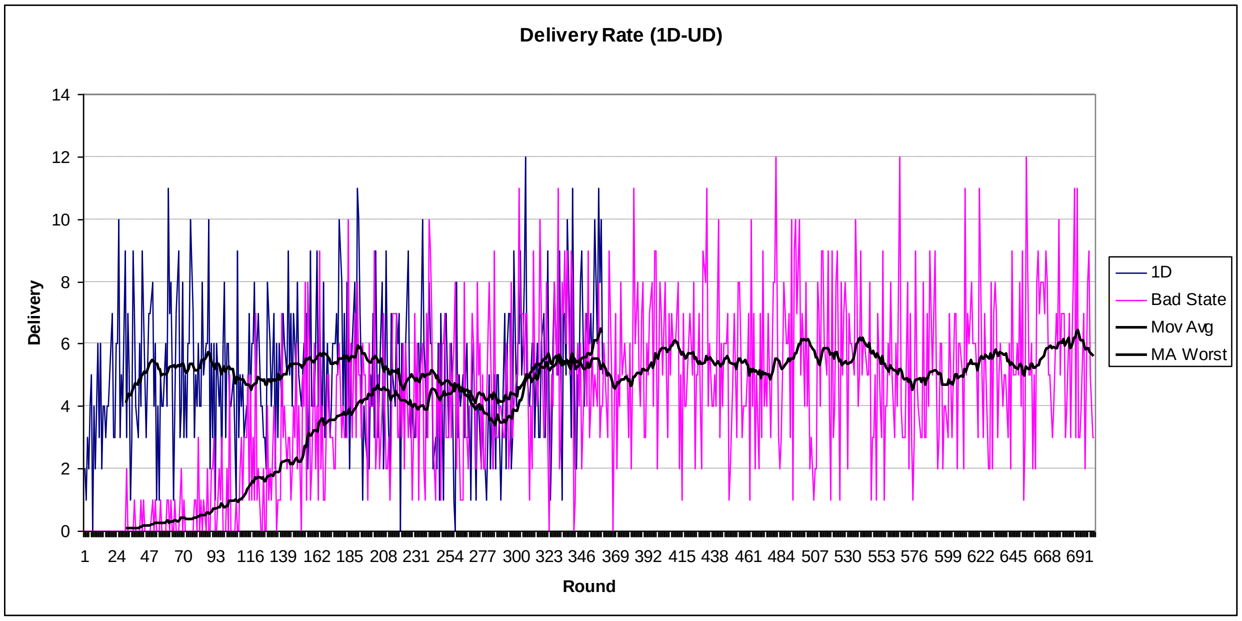
The number of rounds is set to 360 for all dimensions for the normal starting state and the reset direction option. Larger rounds are used for starting from the bad state to better observe the delivery rate.

Dimension	*Torus Size*	*Average Distance*	*Rounds*	*Rounds (Bad State)*
1	60	15	360	960
2	30	15	360	960
3	20	15	360	960
4	15	14	360	1440
5	12	15	360	1440
6	10	15	360	1440

5.3.2 Delivery Rate (Normal & Starting from Bad State)

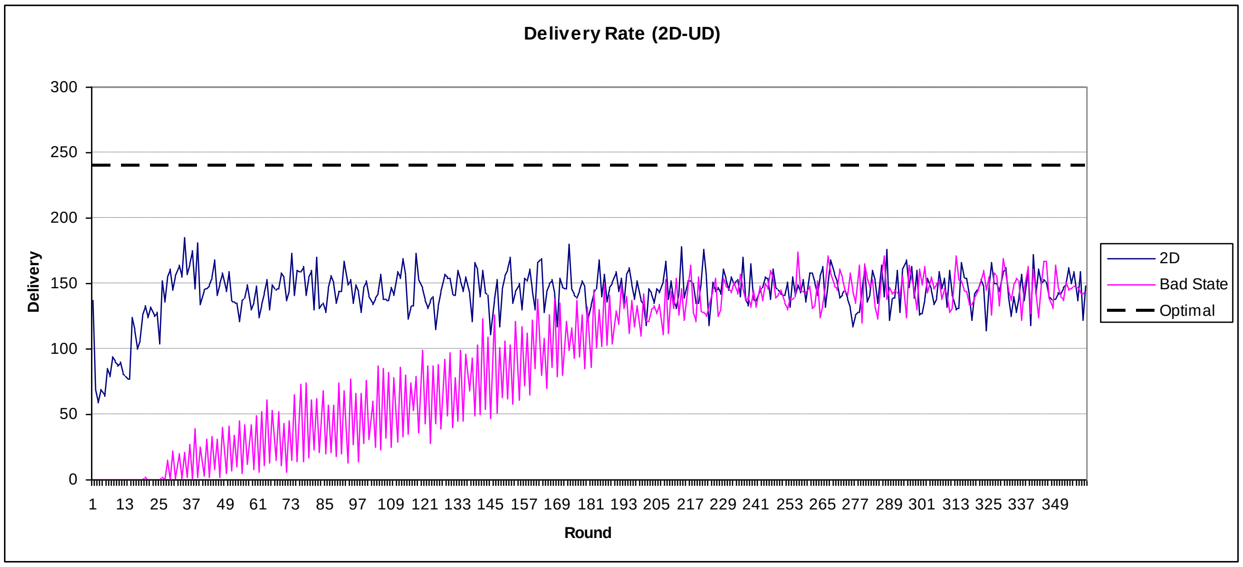
Here are the results of the simulation comparing the delivery rate for 1-6D for normal and bad initial starting state. Although the simulation for starting with bad state contains more rounds than the chart below shows, the delivery rate not shown is very stable. The chart range is selected to maximize the transition data.

The "optimal" delivery rate in the chart represents the impossible to achieve theoretical optimal where every packet moves closer at every time step. This is simply not possible to achieve due to packet route conflicts. We include this to show how the routing algorithm performs with respect to this theoretical optimal.



The 1D data is noisy because the number of packets we are averaging each round is small, around 5 for 1D torus. This number will not increase even if we uses a larger size torus. Therefore a moving average is provided for the data.

From the chart, we see the delivery rate for the normal case is sort of stable. Delivery rate for starting with bad state rises up to match the normal starting state after about 250 rounds.



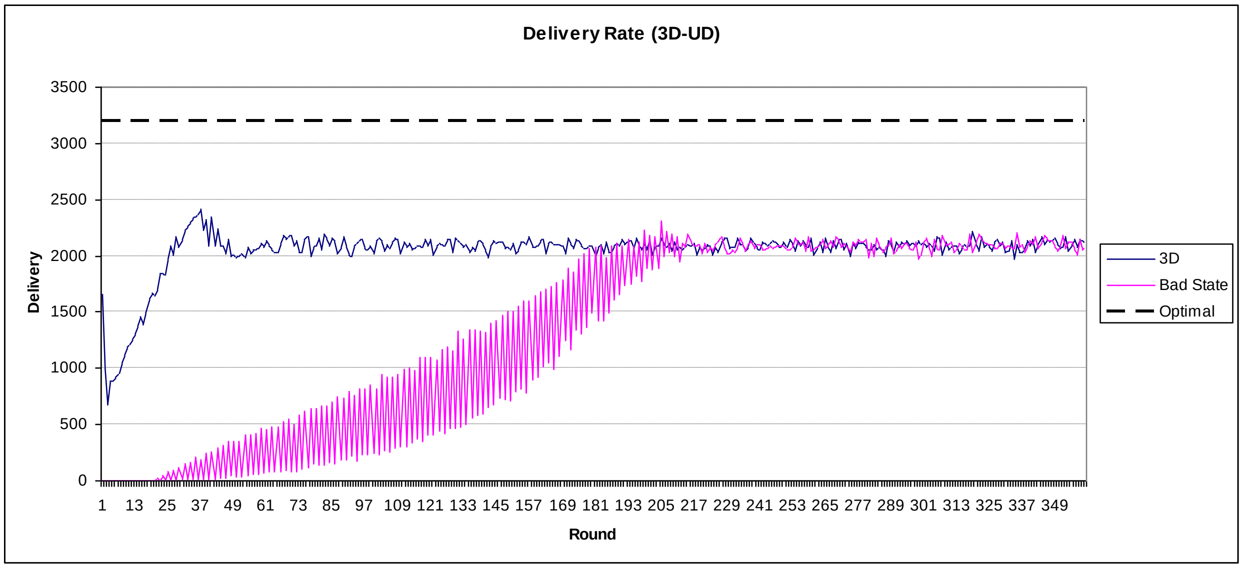
Here are the results for 2D. The data is smoother compared to 1D because we are averaging a larger number of packets, around 140 per round.

The plot shows packet delivery rate stabilizes very quickly for the normal starting state, and remains consistent. The bad state approaches the same steady state after about 220 rounds.

For the initial *x* rounds in the bad state, no packets are being delivered, as all packets have the same distance *x*. After *x* rounds, there's an interesting zigzagging pattern to the delivery rate. In one round, the delivery rate is large, in the next round, the delivery rate is significantly smaller (0 initially). This pattern eventually smoothes out as the delivery rate matches the normal case.

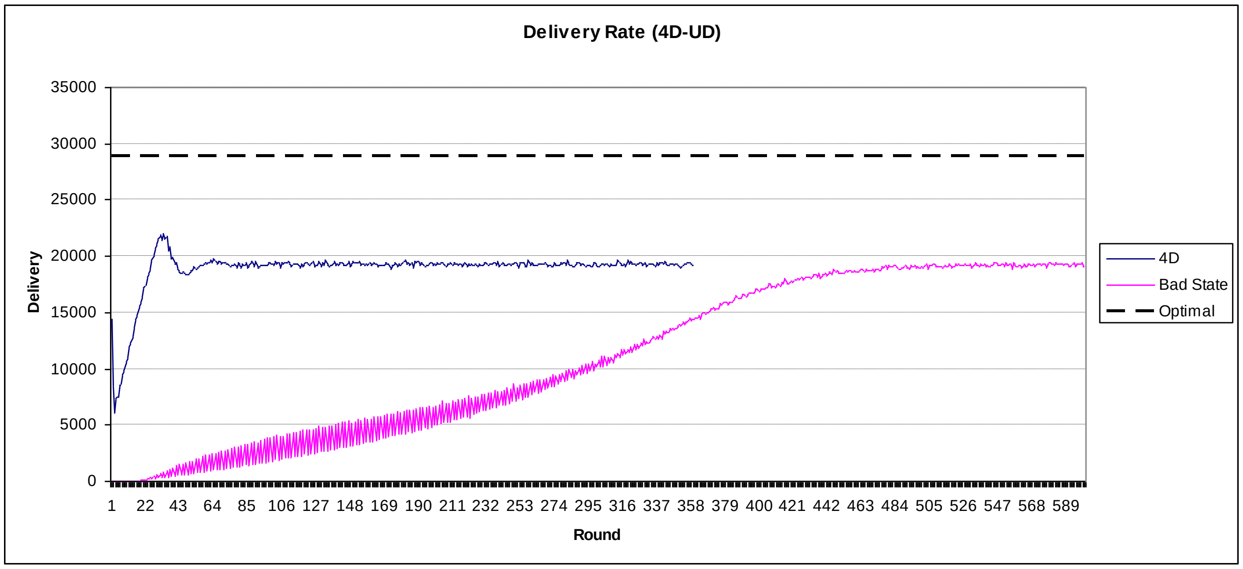
The reason this occurs is because initially all the packets start out with the same distance, 15 in our 2D case. Assuming none of the initial packets can ever "wrap around", then every one of these packets must be delivered at an odd time step. (On all even time steps, they must be an odd number of steps from their destination, so they cannot have arrived on that even numbered timestamp). If they do wrap around, then for even size torus, wrapping around doesn't change this. However, on odd sized torus, wrapping around does change this, as the packet can now be delivered in even time step. We will see the effect of this in 4D.

Once the packet is delivered, it is replaced with a regular packet, using uniform distance to destination model. Therefore, some of those packets will be at even distance from its destination, and could be delivered in even time steps. Therefore, the top of the zigzag represents the delivery of initial bad packets. The bottom of zigzag represents delivery of 2nd generation and later normal packets. Eventually all the 1st generation packets are delivered, and the system is left with normal packets, and as we can see, this is where it matches the normal case.

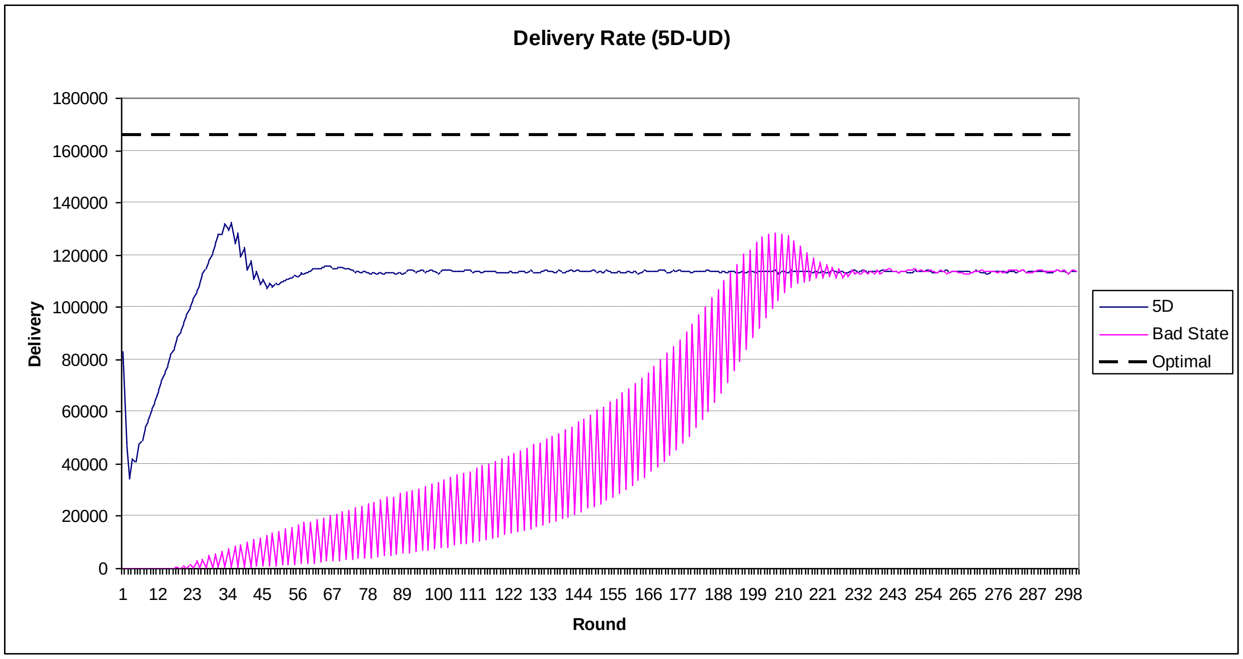


The 3D data is even smoother. Similar to the 2D case, the delivery rate drops slightly, then increases consistently until hitting a peak around round 37, then drops again and reaches the stable point very quickly around round 60. The rate stays there for the rest of the simulation.

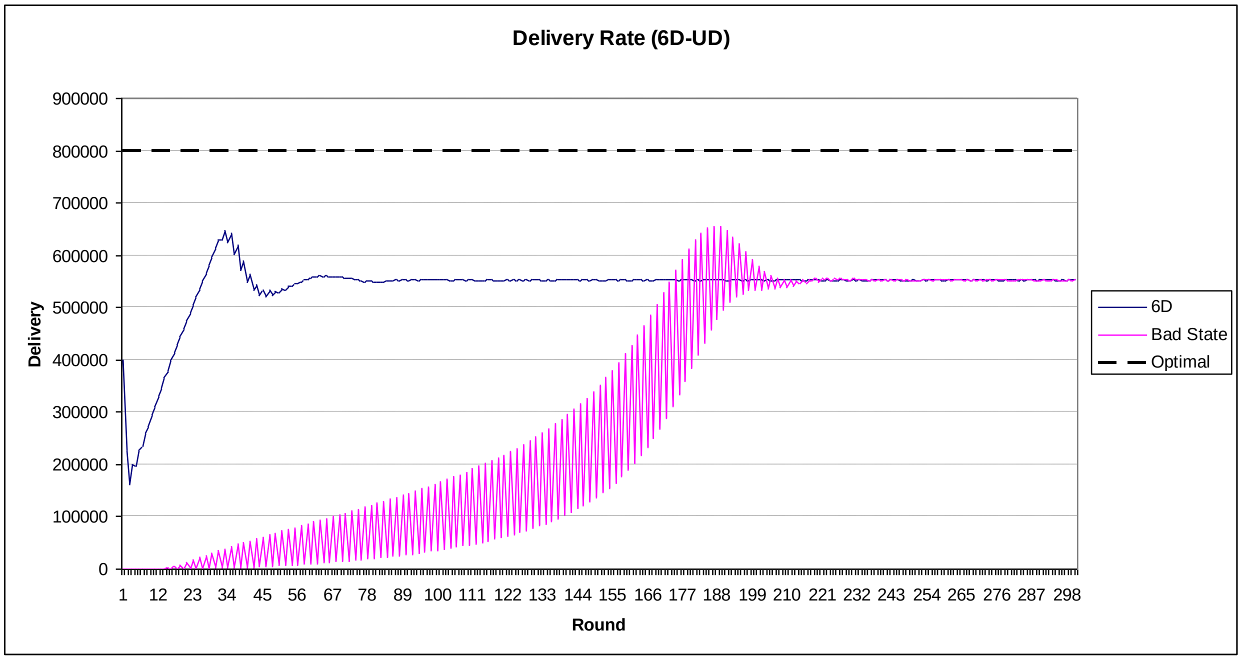
Starting from bad state, we reach steady state at round 220.



4D stabilizes around round 80. The bad state reaches steady state at 500 rounds. Notice that the zigzagging is smaller in 4D, due to packets wrapping around and can be delivered in odd and even time steps.



5D, normal case stable at round 90, bad state case reaches steady state at round 250.



6D, normal case stable at round 90, bad state case reaches steady state at round 230.

Here's a table showing the stable and bad state catching up points for the various simulations:

	Normal case stable at round	***Bad state stable at round***
1d	50	250
2d	50	220
3d	60	220
4d	80	500
5d	90	250
6d	90	230

From these plots and the data, we can make several observations.

- Packet delivery rate stabilizes very quickly for the normal case.

From all the plots, we see that the packet delivery rate stabilizes rather quickly. After 120 rounds (8x average distance), all systems are stable, and remain pretty constant after stabilization. This is very encouraging, indicating that the system reaches a steady delivery state for all dimensions, and once there it stays there. This indicates the routing is performing well, there are no hidden cost accumulating in the system that negatively affect routing in later rounds. A steady state exists and is maintained.

- Packet delivery rate when starting from the bad starting state stabilizes to the normal state, and stays at the normal rate.

For the system that starts with the bad state, we can see that the initial delivery rate is significantly lower than that of the normal state. However it slowly catches up, and matches the normal state at steady state.

This is very important. It indicates that if the system does get into a bad state (however unlikely, but none the less statistically possible), it will recover, and return to the steady state.

- Delivery curve for normal starting state.

The shape of the curve reaching steady state also provides some insights into how the routing is performing. In all dimensions, the delivery rate drops significantly in the first 3 rounds, then increases steadily to a value higher than steady state, then gradually approaches steady state, and stays there.

In the 1st round, only zero-distant packets are delivered. In the 2nd round, only packets that are 1 away, and get their 1st choice, are delivered. Number of packets that are 1 away is the same as number of packets that are 0 away, but those getting 1st choices will be smaller. Hence the drop in delivery rate compare to the 1st round. In the 3rd round, only packets that are 2 away, and got their 1st choice both times, are delivered. Packets that were 1 away, but didn't get their 1st choice, cannot be delivered. Therefore, the delivery rate drops even further.

In the 4th round, delivery rate picks up because the number of "eligible" packets was increased. Those that were 3 away and got their 1st choices 3 times, and those that started 1 away but did not get their 1st choice in round one, can all be delivered. As the simulation progresses, there are more eligible packets at each round, pushing the delivery rate higher. At each round, the average packet distance of packets that were delivered is smaller than the average distance of packets that were generated, as only initially close packets can be delivered in the first few rounds Eventually the system runs out of these packets, and have to work on those replacement longer distance packets, and the rate drops. Eventually the delivery rate stabilizes at the steady state, indicating the packet delivery distribution and packet generation distribution match.

- Delivery curve for starting from the bad state.

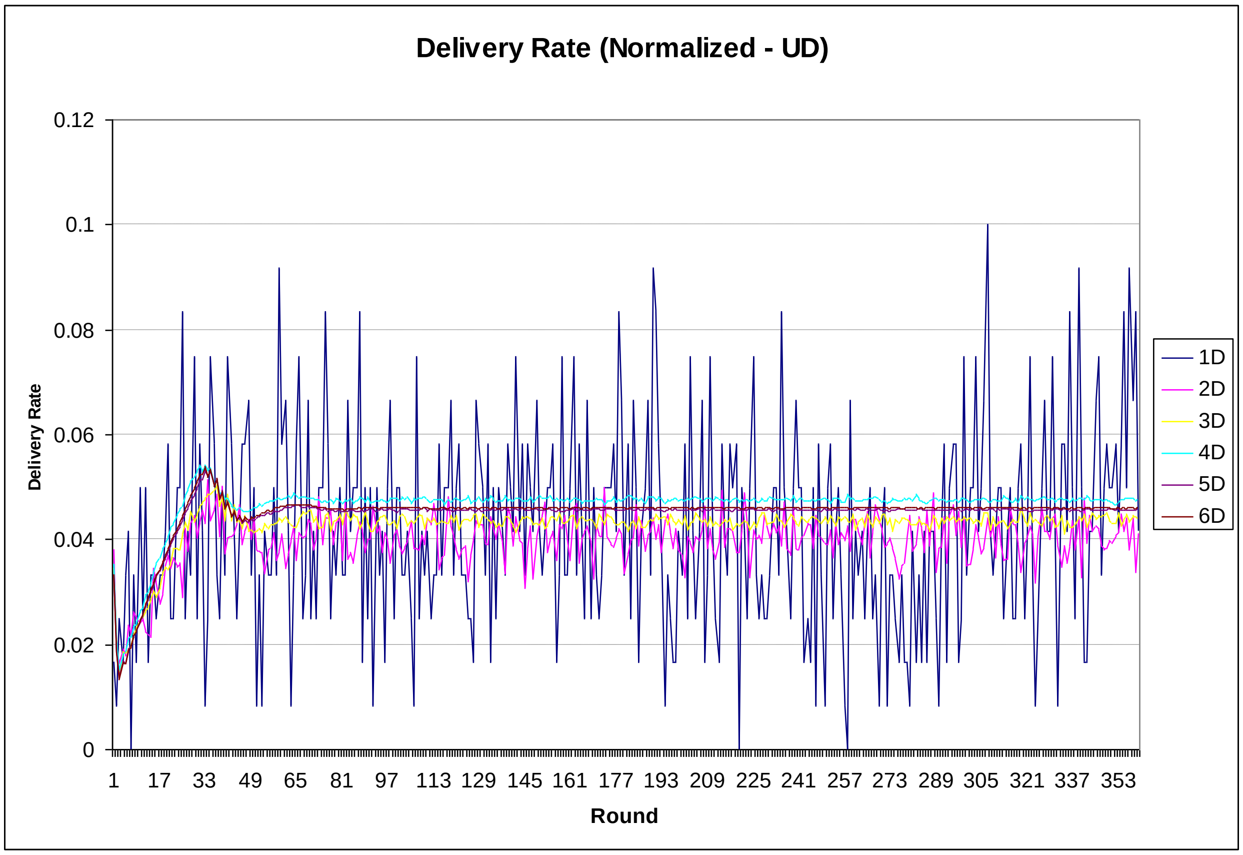
In starting from the bad state, in the first few rounds, every packet wants to go in the same 'direction', therefore as a result only few packets are making progress. Therefore, as we can see from the plot, the initial delivery rate is very poor. The delivery rate does catch up after a number of rounds, and stabilizes. The zigzagging in delivery rate during catching up is because the original packets can only be delivered in odd numbered steps. Replacement packets can be delivered in all steps.

5.3.3 Normalized Delivery Rate

We now compare the delivery rate across dimensions, to see how dimensions changes delivery rate.

The delivery rate by themselves can't be compared directly, as the number of packets in the system is different between torus of different dimensions. We normalize the data by computing the delivery rate in terms of percentage of the total packets, and compare the percentage across dimensions.

Here's a chart showing the normalized delivery rate for each round:

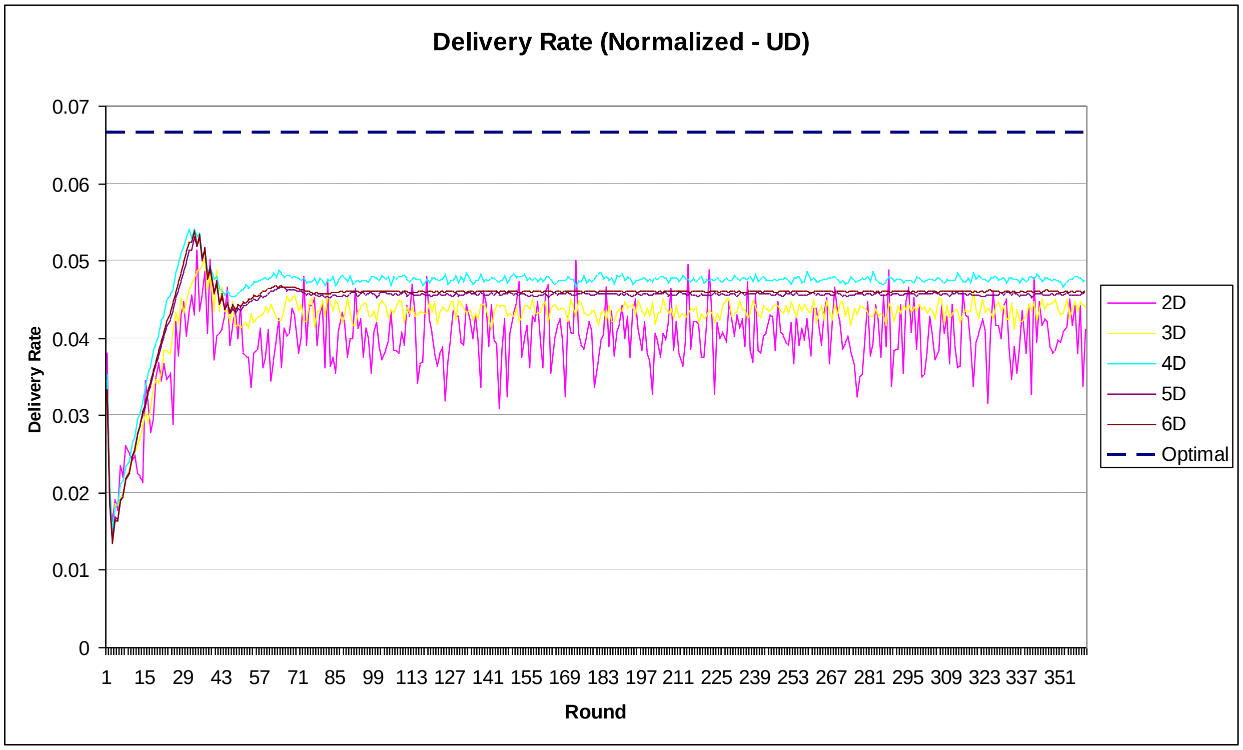


The delivery rate is very similar, around 4%, with some differences between them. The calculated average for the delivery rate is:

1D	***2D***	***3D***	***4D***	***5D***	***6D***
4.2	3.976	4.262	4.661	4.467	4.503

With the exception of 1D, the delivery rate increases steadily from 2D to 6D, with the exception of a big increase for 4D (resulting in a drop for 5D). The 4D delivery rate is higher and throwing off our number because the average packet distance is 14, not 15, as we have explained in previous sections. Therefore packets were being delivered faster than it would have if the average distance is 15. If we remove the 4D data, the rate increase will be consistent.

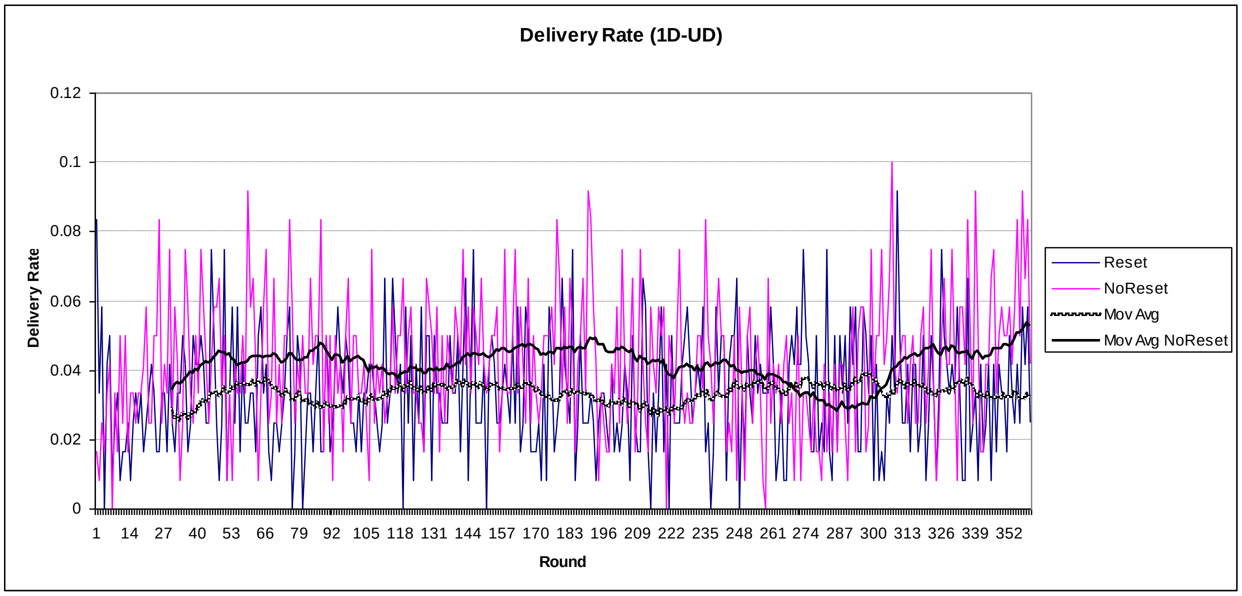
Here's the chart again, but with the 1D data removed:



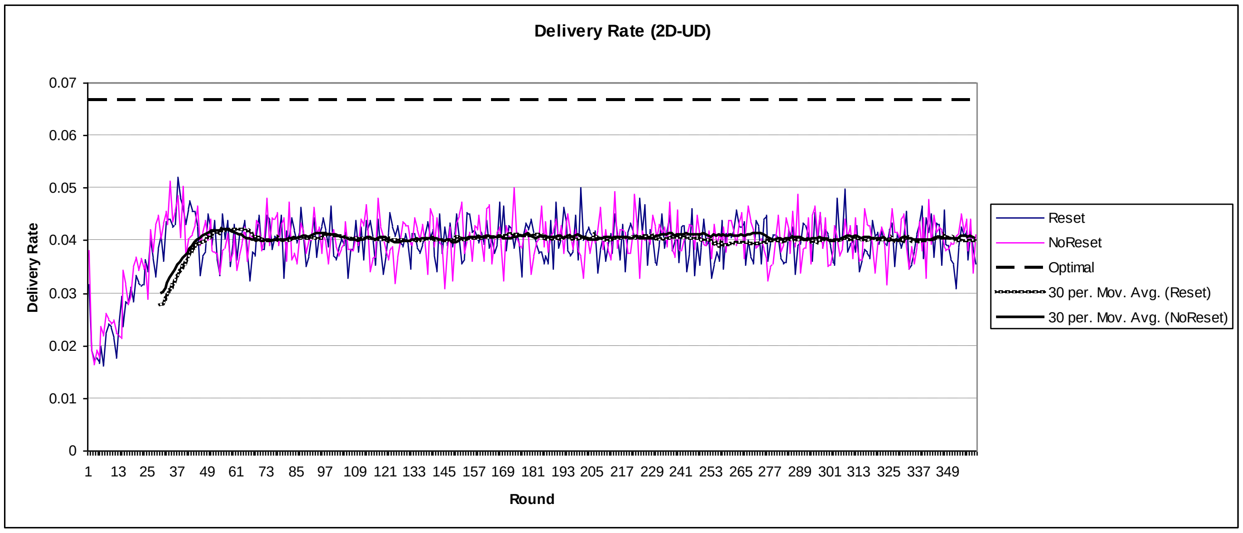
Except 4D, we see increase from 2D to 6D.

5.3.4 Delivery Rate (Normal & Reset Direction)

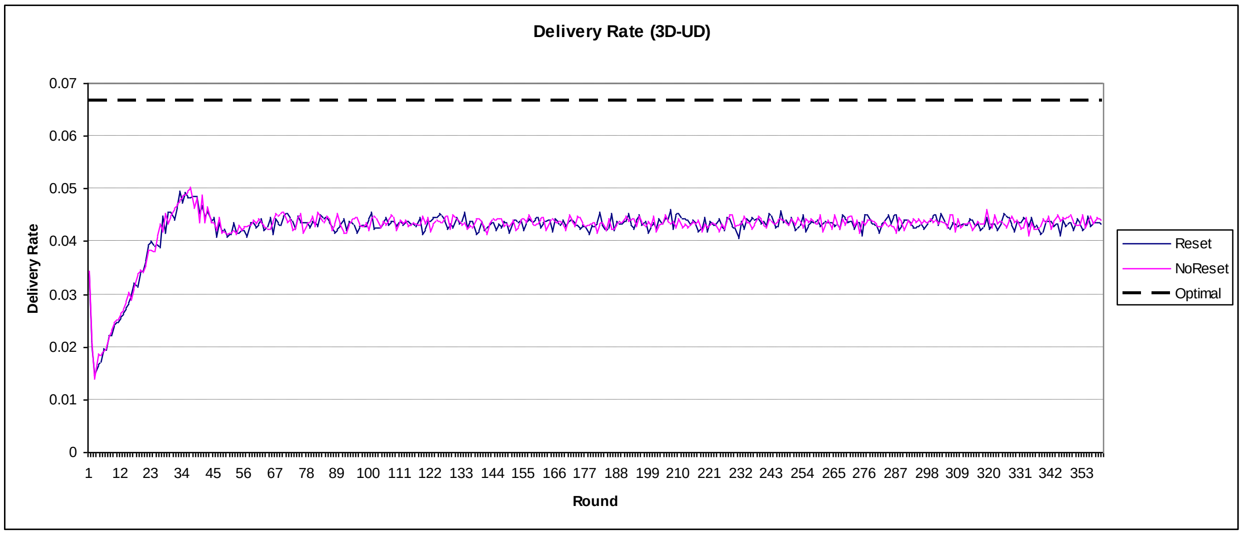
Let's now take a look at the packet delivery rate with the reset direction option. Recall this option is used to remove the directional dependency of the packet during routing, to see how directional dependency affects the system. Here are the results:

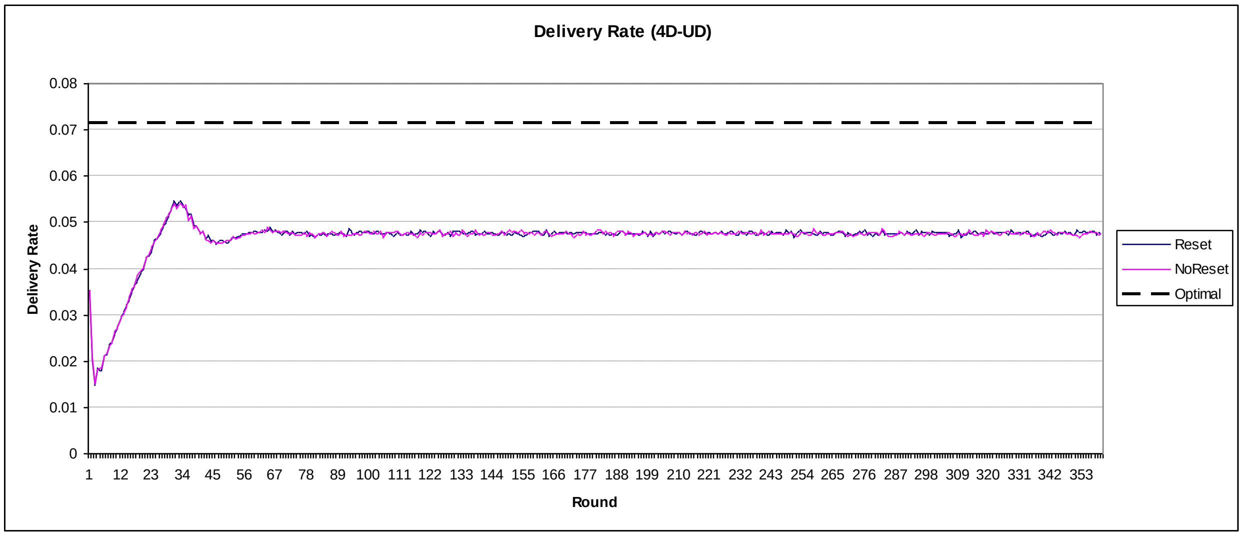


In 1D, the reset option yields a lower delivery rate.

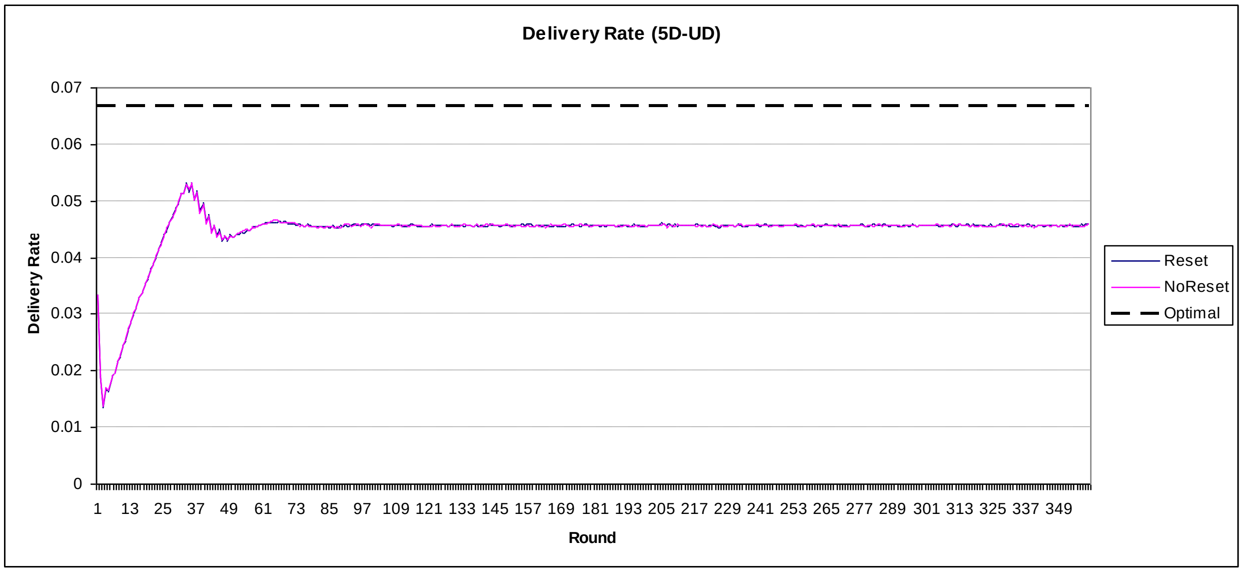


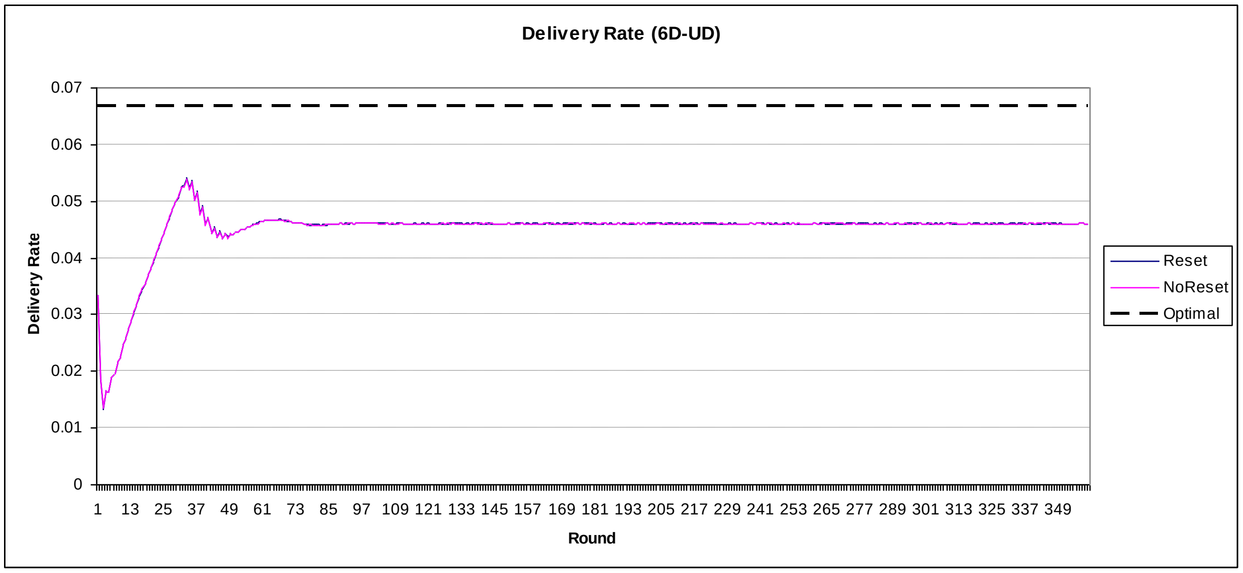
In 2D, it’s very close. The reset’s moving average just a bit below the not reset average.





In 3D and 4D, they are basically the same.





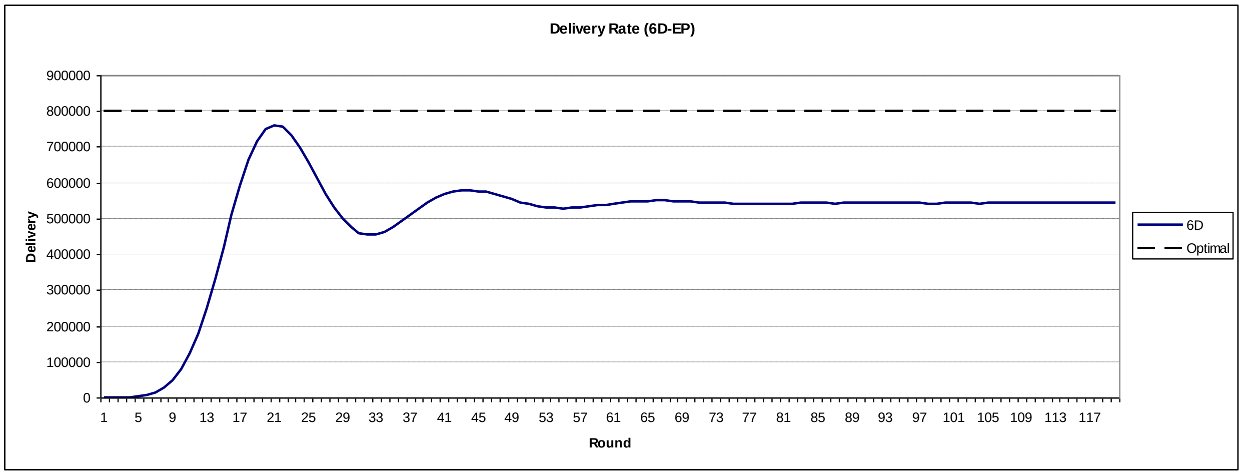
In 5D and 6D, the delivery rate for the normal case is smoother and more consistent than the reset case, but the rate is the same.

As the data shows, the delivery rate for the reset/independent system is lower in 1 and 2D, and identical in higher dimensions. This matches very well to our finding from the delivery time simulation with reset direction. Delivery rate and delivery time can be calculated from one another in steady state, but here we were mostly interested in the throughput while stabilizing to steady state.

5.3.5 Delivery Rate (Equal Probabilistic Destination)

We now look at the delivery rate when packets are generated using the equal probabilistic destination method.

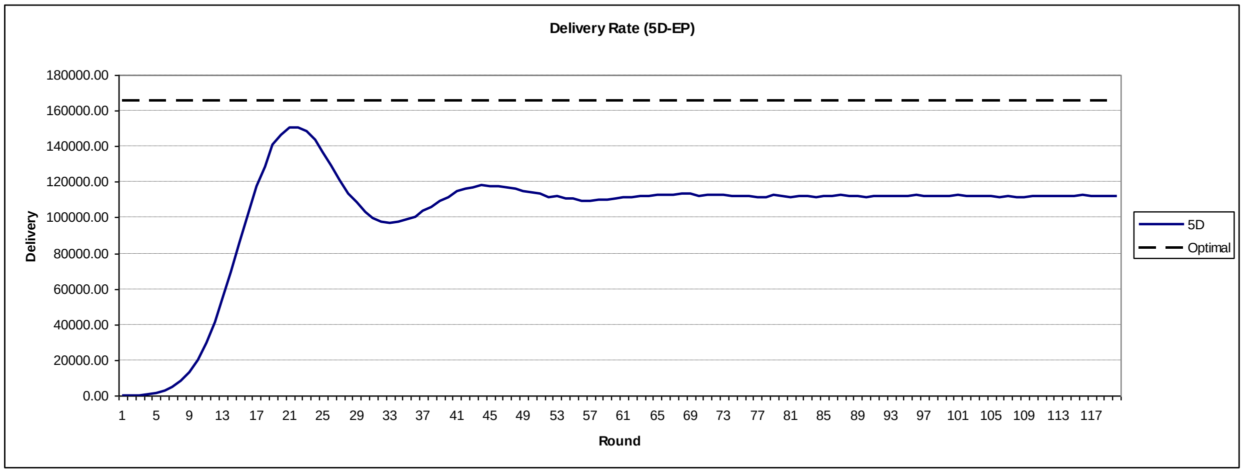
We show the result of the 6D plot because it's the most interesting one.

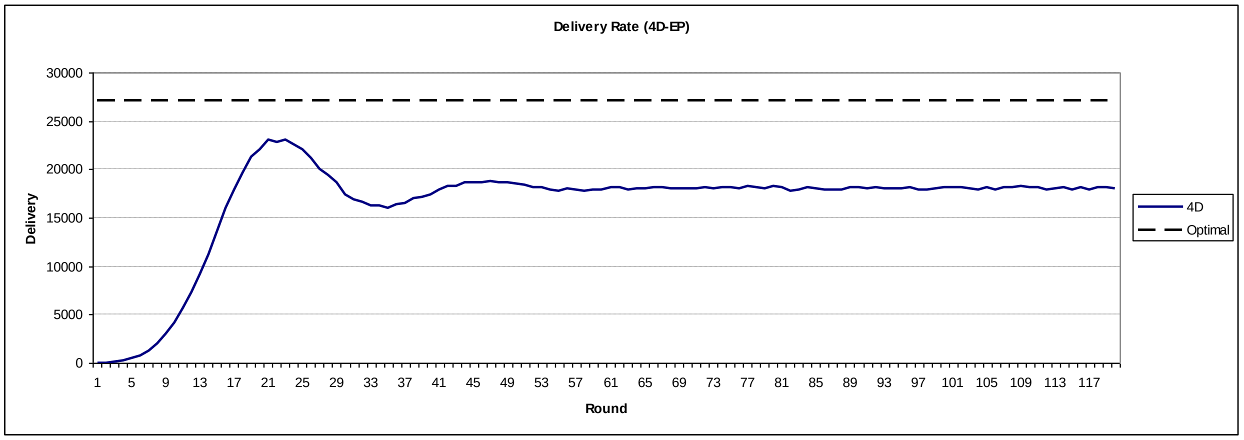


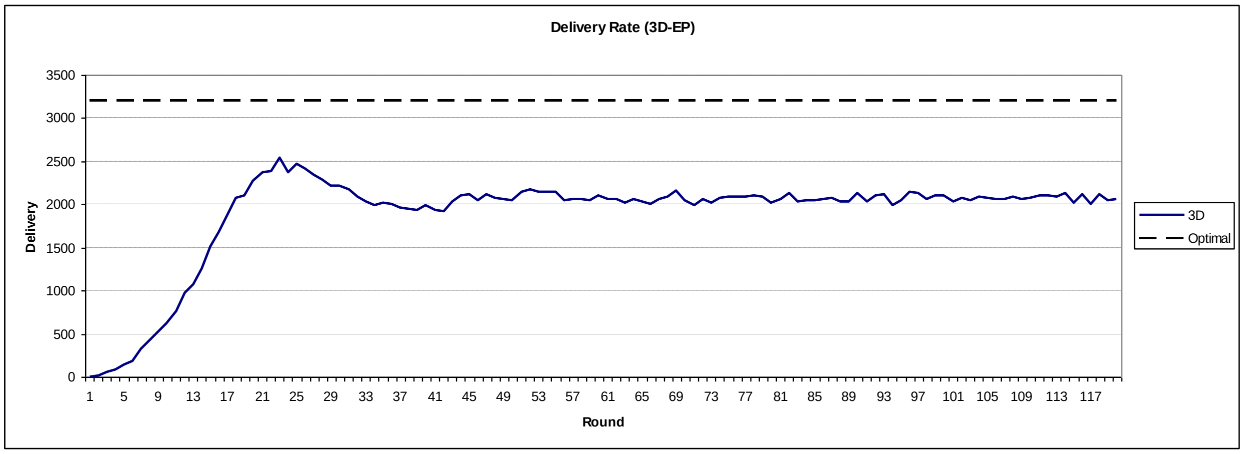
In higher dimensions such as 6D, using equal probabilistic packet generation, packets will have distance distribution on a narrow band centered on the average, 15. In other words, most packets have initial distance close to 15.

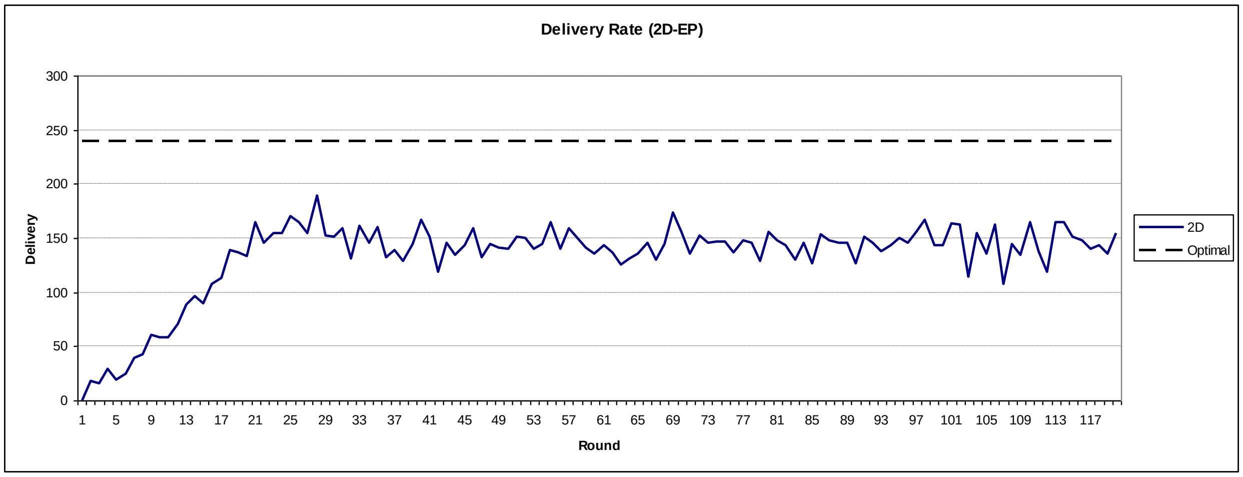
Initially there were no packets delivered, because there are very few packet with distance significantly smaller than 15. Eventually the simulation progresses enough rounds that some of the short distance packets are delivered. From that point on, at each round, we have more, longer distance packets that can be delivered. The rate reaches a peak when most of the packets at distance 15 or less are delivered, but replacement packets not yet close enough to be delivered, so after the initial packets are delivered, the rate drops until some of the 2nd generation packets delivery rate increases. It eventually stabilizes after the packet delivery distribution matches packet generation distribution.

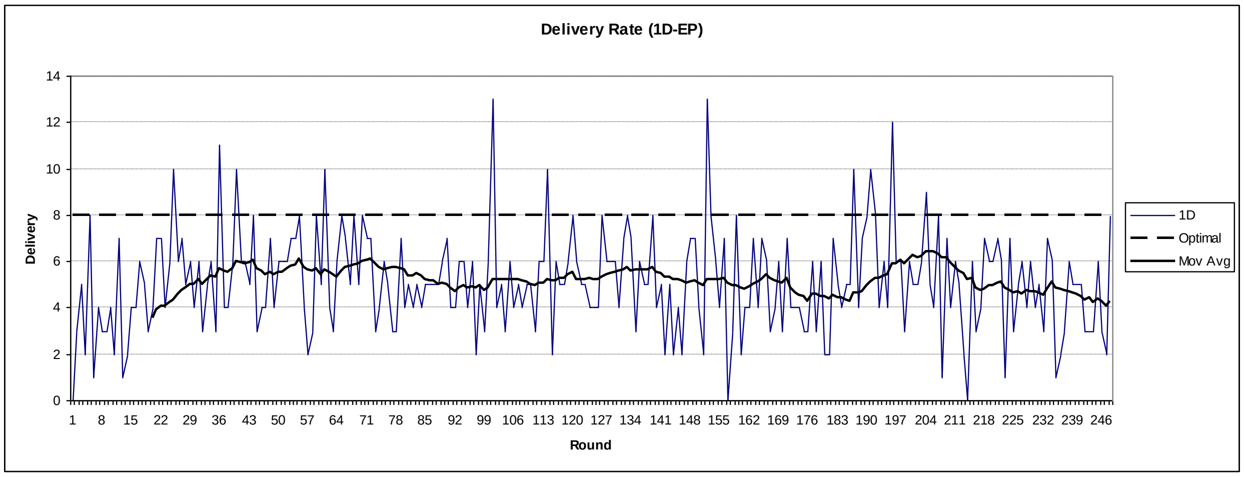
Here are the charts for the rest of the dimensions:











The trend is less obvious in lower dimensions. In 1D the data is very noisy, due to the small number of packets being averaged, so a moving average is provided.

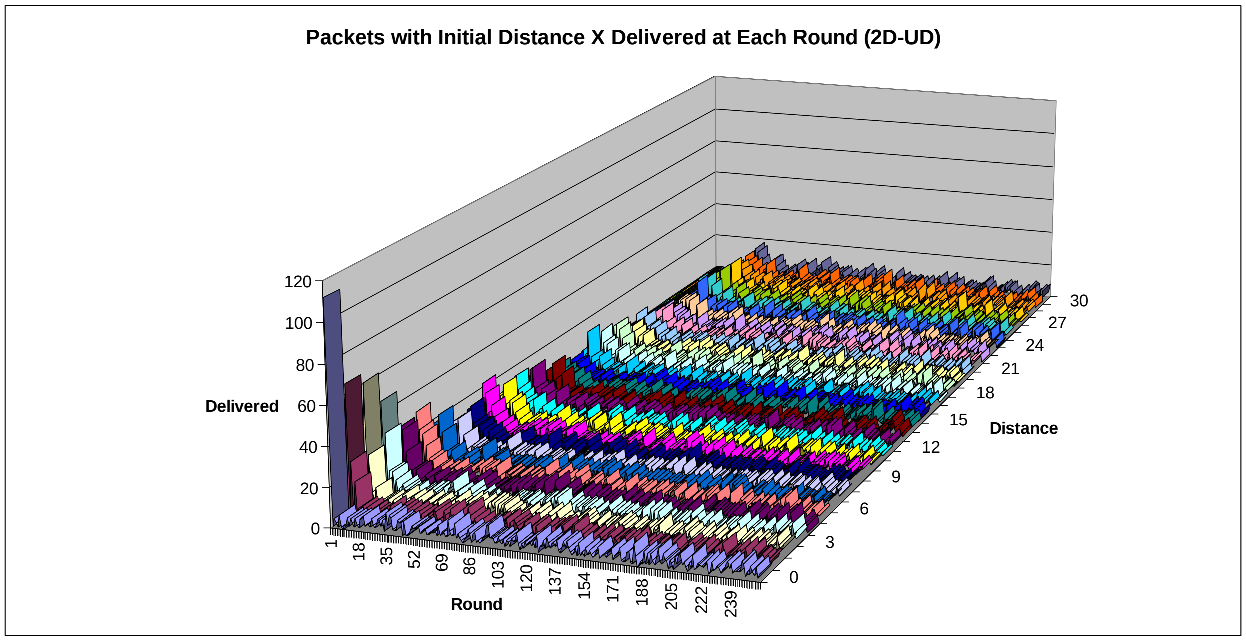
5.4 *Delivery Type*

We now take a look at the 3rd, and final packet delivery metric, delivery type. That is, for packets that were delivered, how many started 1 away, 2 away, etc?

We are generating packets with a specific distance distribution, at steady state, the delivery distribution should match the input distribution (otherwise there won't be a steady state). Let's see if this is the case.

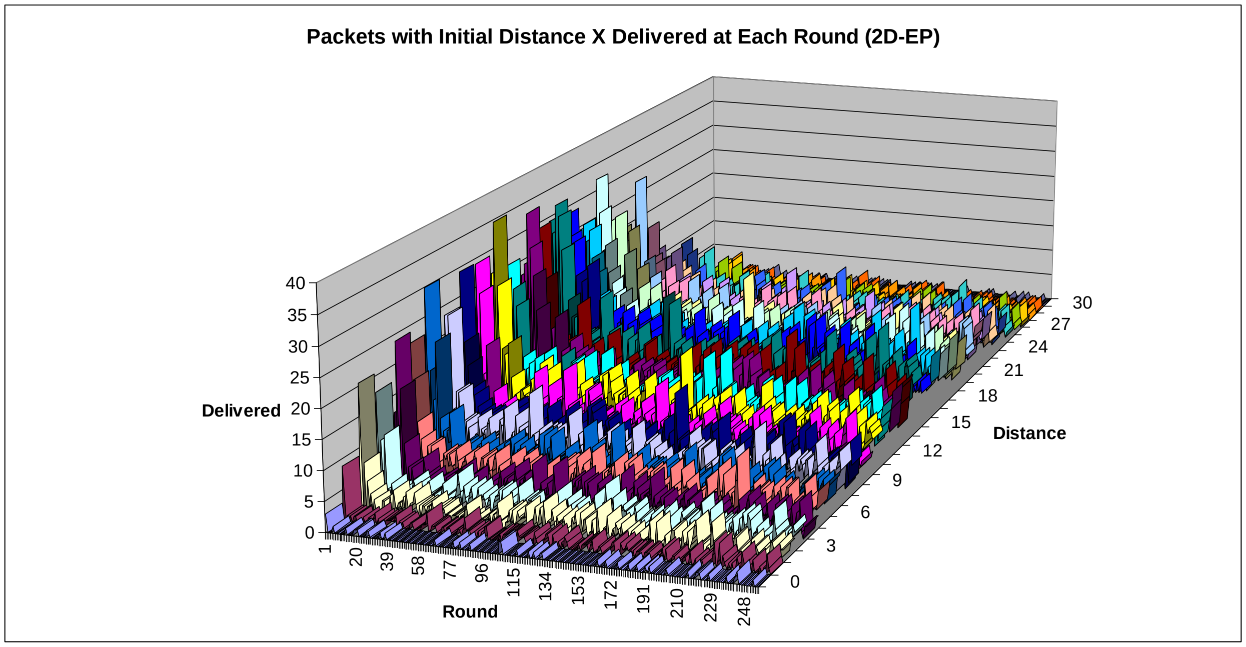
5.4.1 Delivery Type Analysis

Here's the chart of packet delivery per round by its initial distance, on a 2D system with uniform distance distribution.



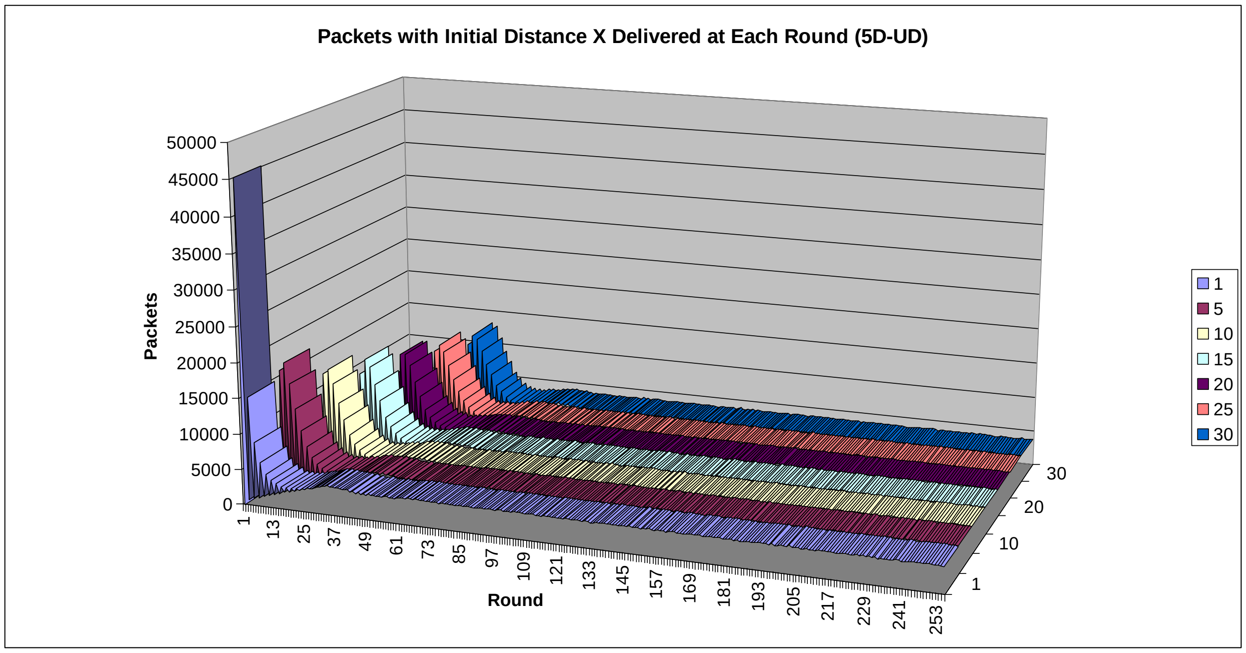
The x-axis is the number of rounds. The y-axis is the initial packet distance, and the z-axis is the actual deliveries per round. From the chart, we see that after some rounds, the delivery is uniform across all distances, matching the packet generation model.

For comparison, here's a plot of the same data, but using equal probabilistic method:



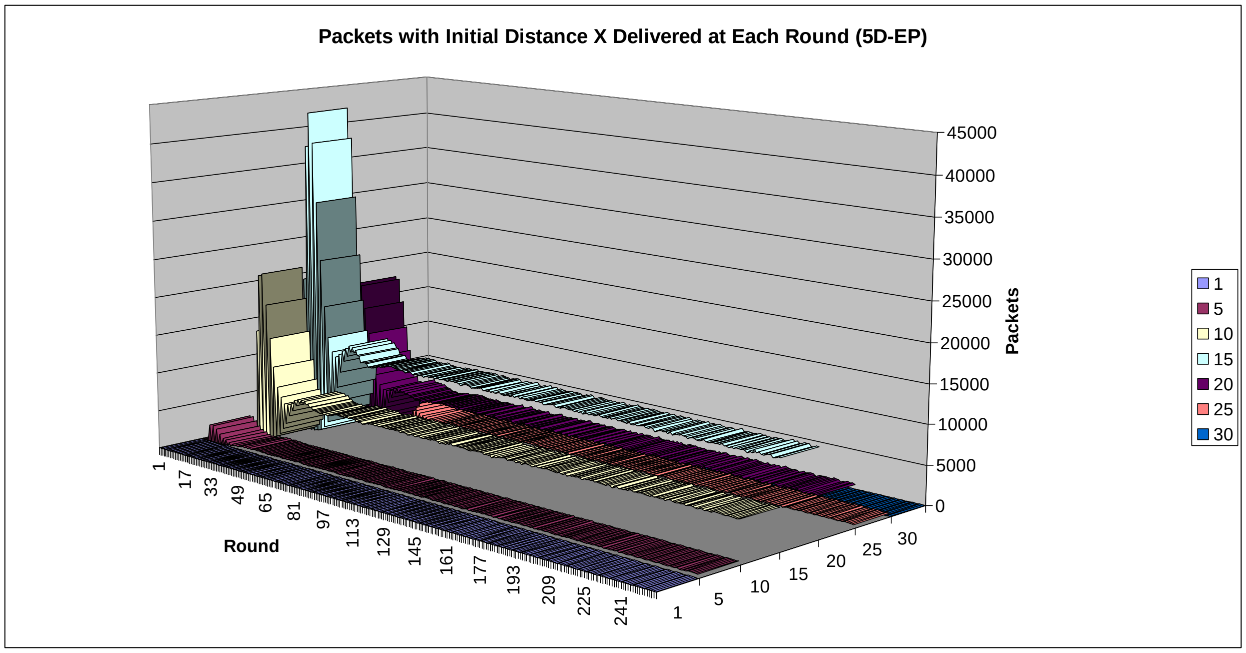
As we can see, at steady state the delivery matches the EP generation. Because there are more packets generated with distance close to average, at steady state, more packets with distance close to average are being delivered.

Let us now take a look at the same data, but in 5D:



The packet delivery by distance matches very well to the packet generation model. 5D torus have significantly more packets, therefore the data is very smooth.

Here's the result for 5D with equal probabilistic model:

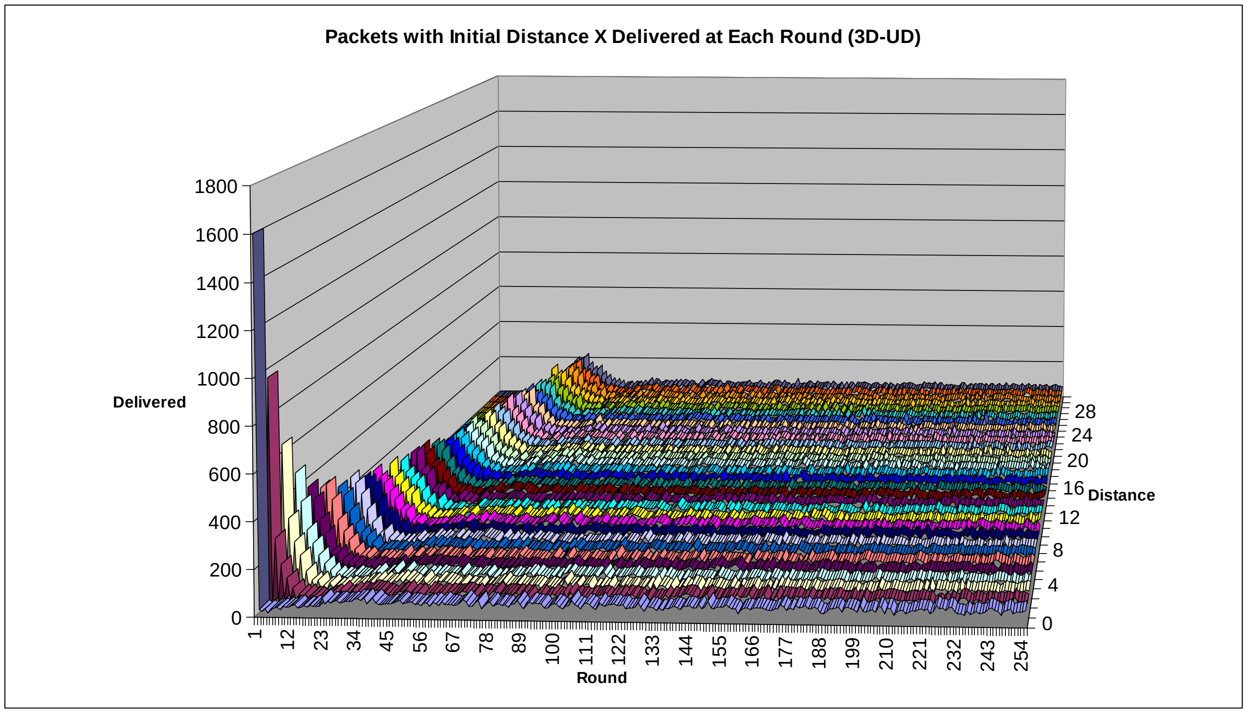


Although a bit hard to see, there are more delivery for packets with initial distance 15 than those of 1 or 30, which is very close to 0, matching our packet generation model.

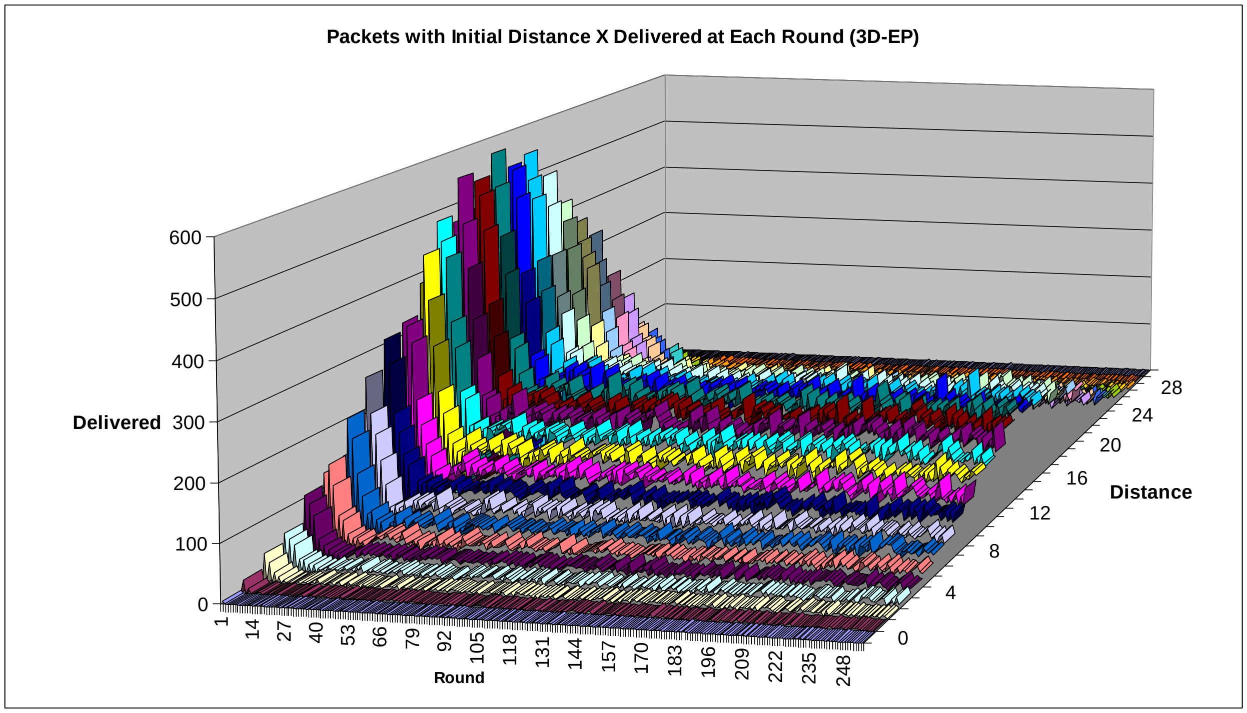
5.4.2 Delivery Type Charts

Here are the charts for the remaining dimensions:

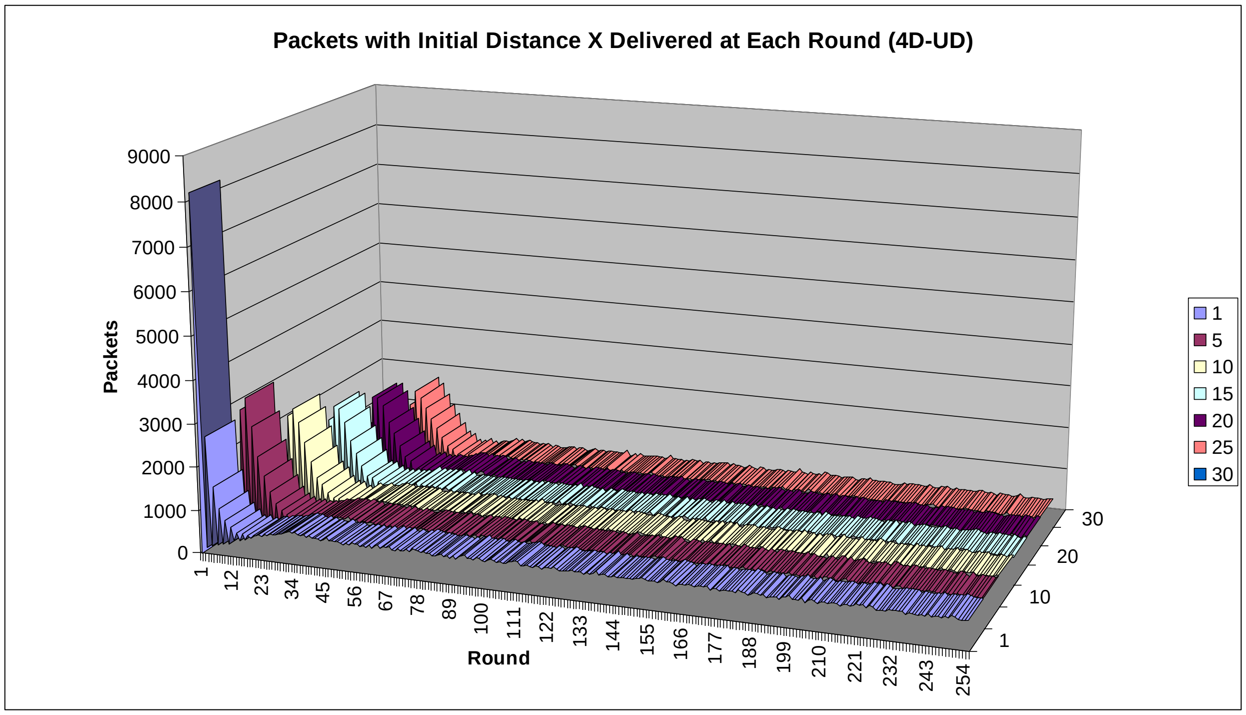
3D (UD)



3D (EP)

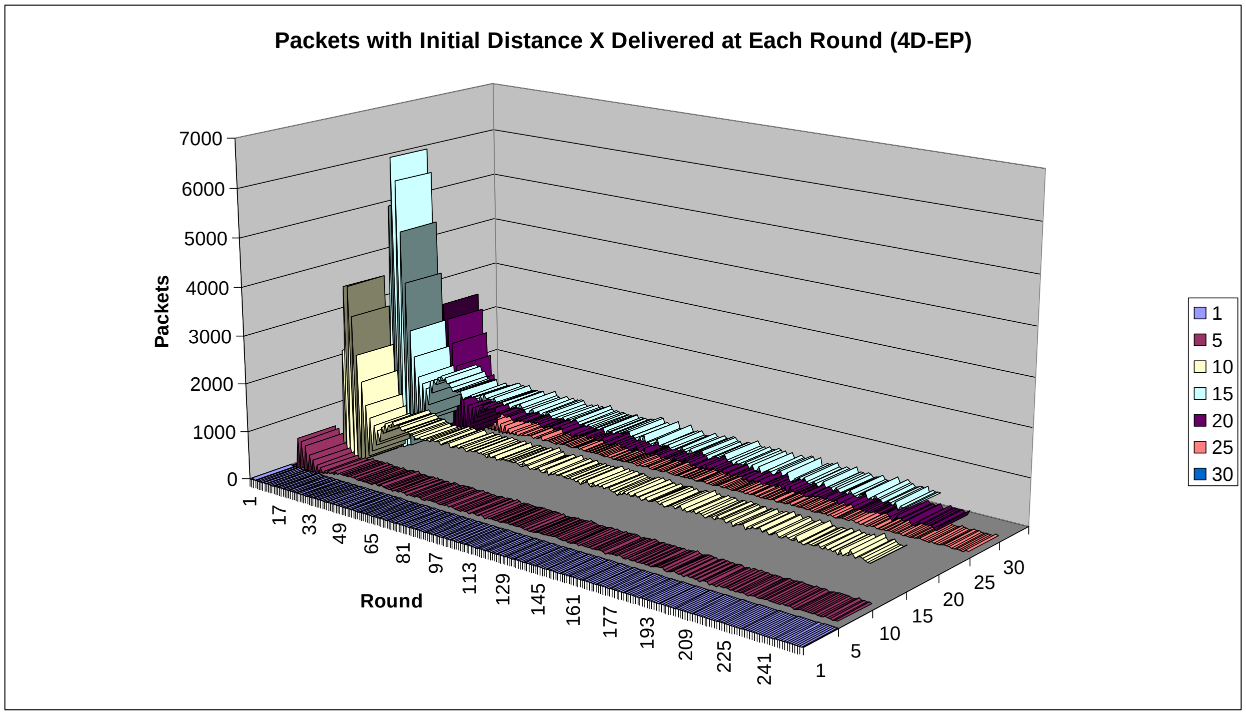


4D (UD)



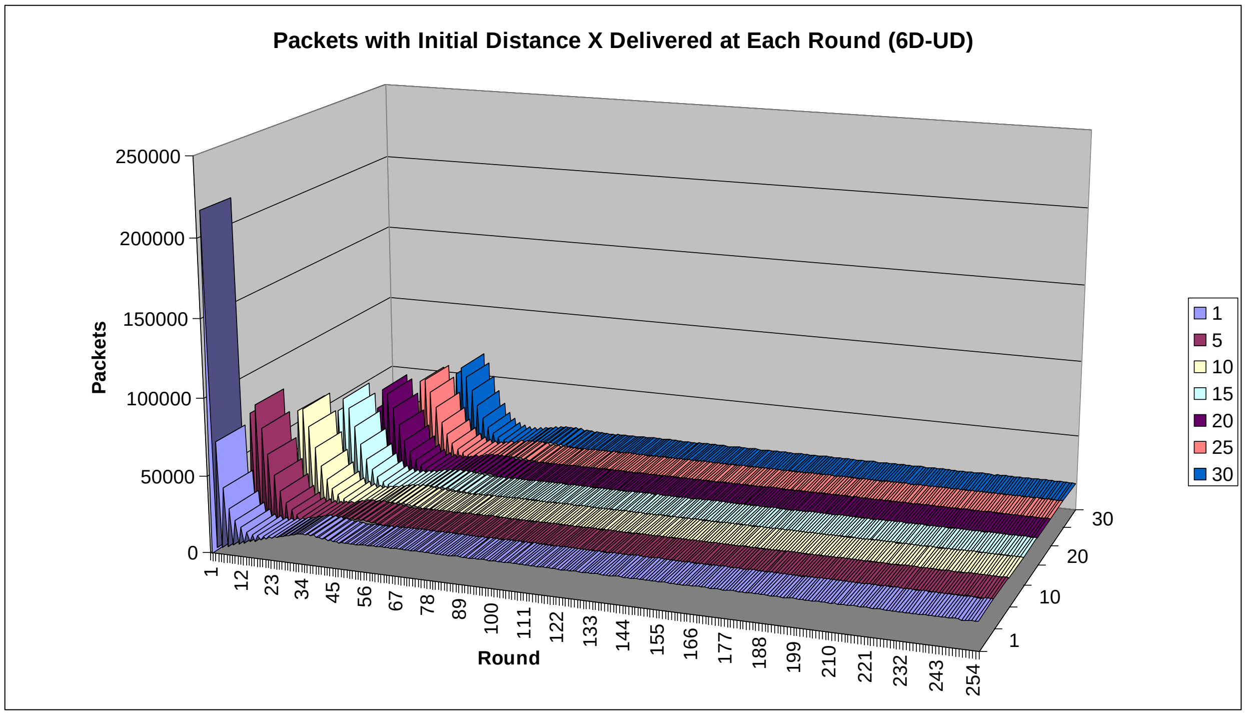
Packets with Initial Distance X Delivered at Each Round (4D-UD)
Packets
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Round
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
1
10
20
30
1
5
10
15
20
25
30

4D (EP)

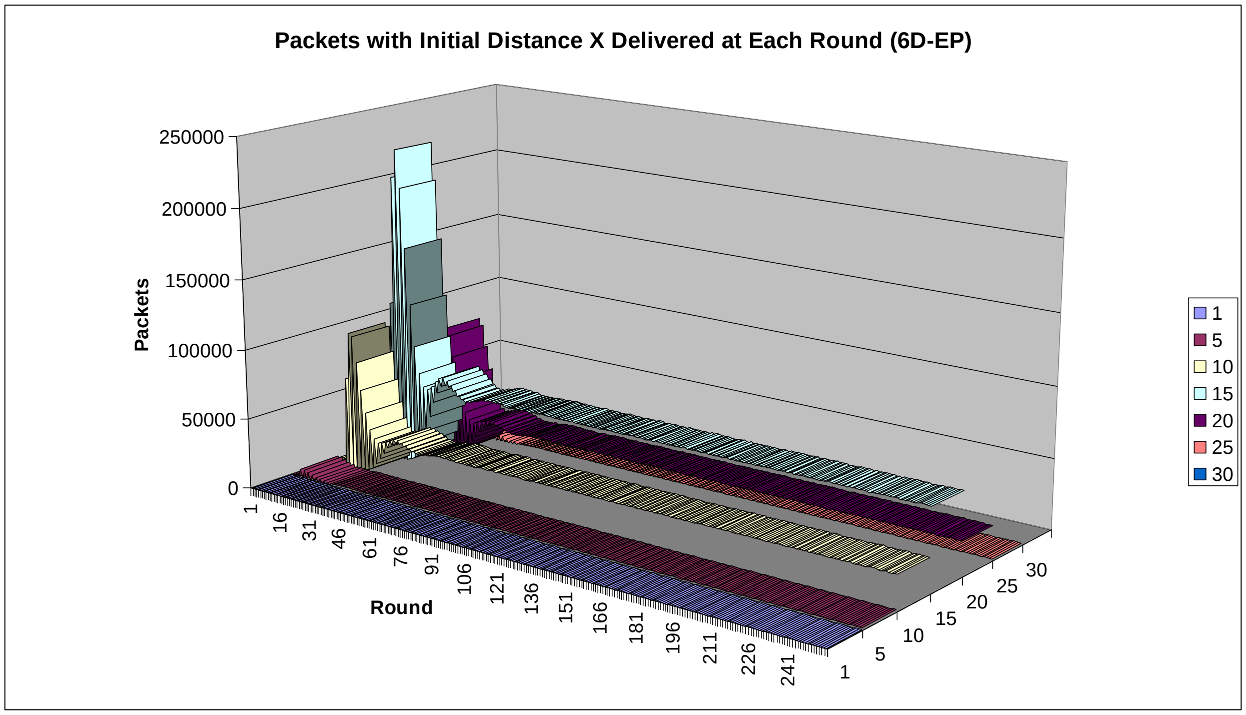


Packets with Initial Distance X Delivered at Each Round (4D-EP)
Packets
7000
6000
5000
4000
3000
2000
1000
0
Round
1
17
33
49
65
81
97
113
129
145
161
177
193
209
225
241
1
5
10
15
20
25
30
1
5
10
15
20
25
30

6D (UD)



6D (EP)



As expected, in the steady state, the delivered packet distribution matches the generated packet distribution.

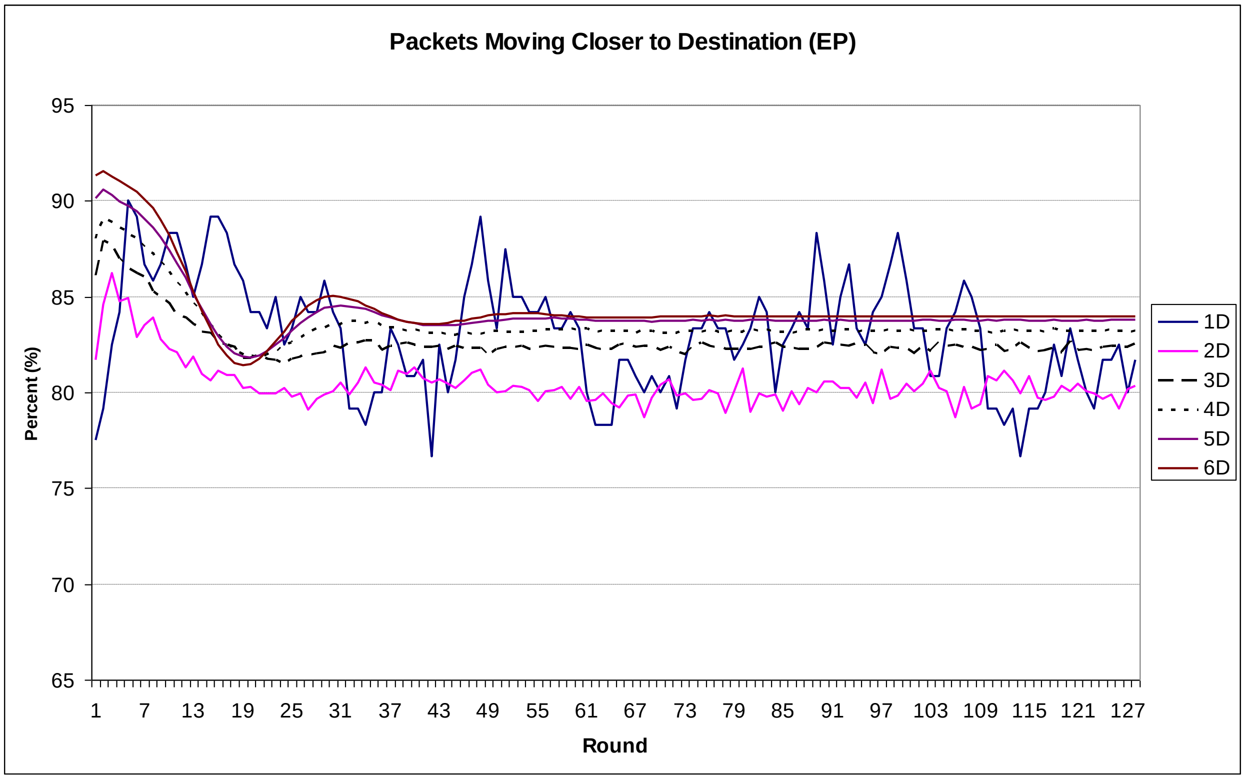
5.5 Routing Choices

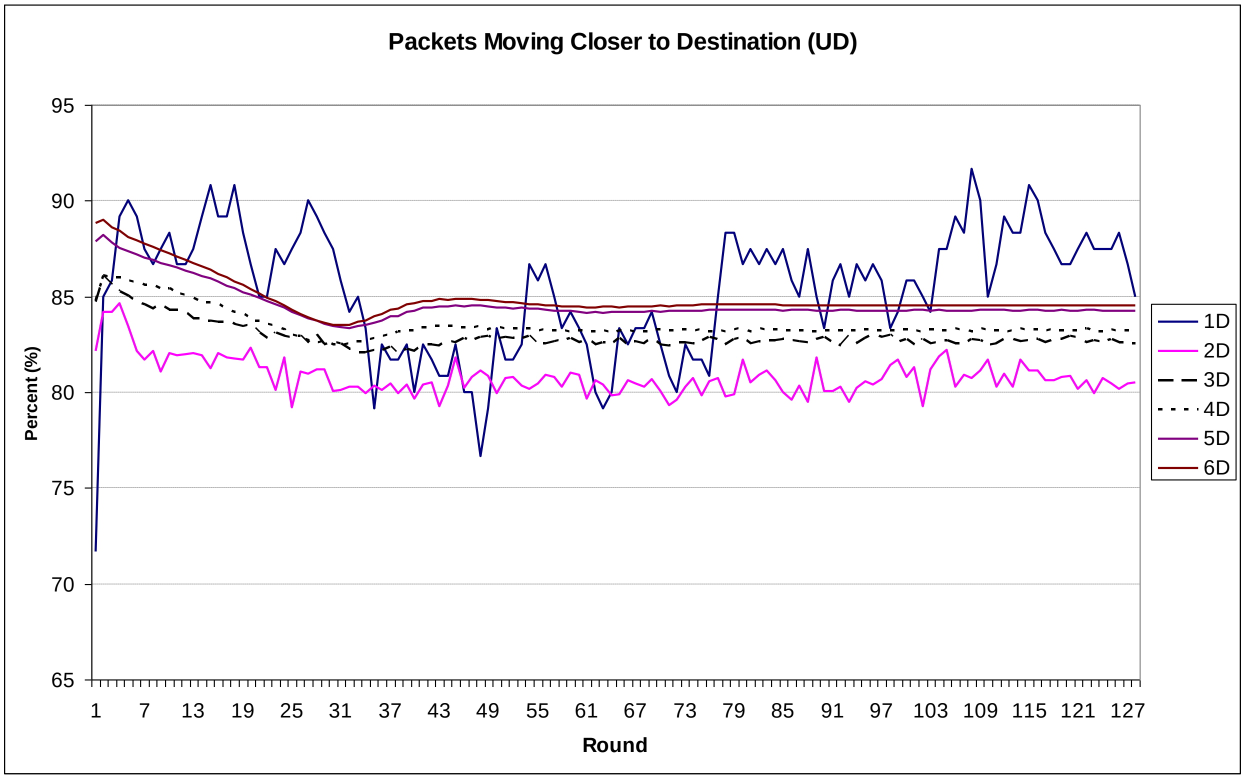
Now we know that the routing is performing well, packets are being delivered in linear time with respect to initial distance, and the system reaches a steady state after a set number of rounds. Let us now take a look in detail the routing itself. That is, what % of packets are moving closer to destination? What routing choices are packets getting?

We ran simulations similar to the previous section, but with number of rounds set to 128, roughly the point where all systems reached steady state.

5.5.1 Packets Moving Closer

We first take a look at the percentage of packets that are moving closer to their destination. In other words, packets that are making progress.





As we can see, about 80-85% of the packets are moving closer to destination. This is very good, indicating that the routing is working well for majority of the packets.

Furthermore, we notice that in higher dimensions, slightly higher percentage of packets are traveling in the right direction. This makes sense, as higher dimensions allows for more flexible packet routes. In 1D for example, you are either traveling in the right direction, or the wrong direction. In 6D, you'll be only traveling in the wrong direction if all your first 6 choices are taken (assuming the packets have some distance to go in each dimension).

The change between 2D and 3D is quite significant, indicating the extra dimension contributes significantly to the packet flexibility. The change between 5d and 6d is a lot smaller, indicating that the extra dimension helps, but we are reaching a diminishing point of returns, especially since, with a fixed average distance to destination, higher dimensional systems will likely have packets that already match their destination in some dimensions.

5.5.2 Routing Choices

Now that we know most packets are getting closer to their destination, we want to know what choices these packets take. That is, what % of packets are getting their 1st choice, 2nd, choice, etc. This would be a very good indicator of how our routing algorithm is performing.

5.5.2.1 Routing Choices Calculation

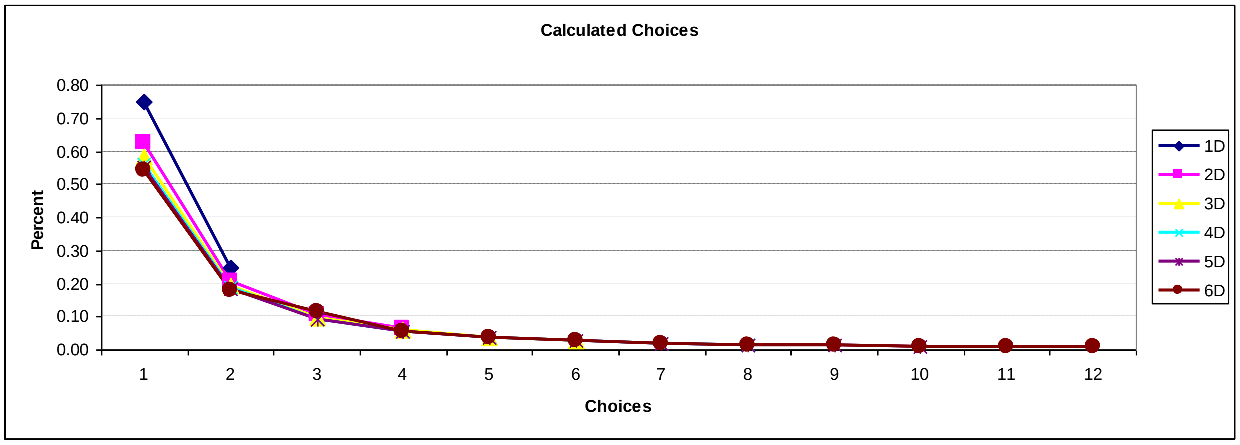
Before we present the routing choices data, it'll be interesting to analyzing the routing choices from a theoretical standpoint. That is, what is the expected probability of getting the 1st choice, the 2nd choice, etc, in a *d* dimensional torus? Analyzing the full probability of the choices in our system is difficult, since packets have dependencies among them. Therefore, we consider a slightly different model where the packets and choices are independent from one another. This allows us to calculate some results and compare it with our system. Although the model is different from our system, it should be somewhat close, and the result and comparison will provide some insights into our system.

In our system, packets are not independent from one another. They are on the very first time stamp, but are not afterwards due to the way packets are being routed. In our calculation, we first make the assumption that packets are independent from one another. Second, packet choices are not independent. A packet's 2nd choice is perpendicular to the 1st choice in our system. In our model, we make the additional assumption that a packet's choices are independent from each other. That is, the 1st choice is independent of the 2nd choice, etc. We assume the choices are a random permutation.

Given these two assumptions, we can calculate the probability of a packet getting its nth choice directions. Here are the results of our calculation. We will explain the calculations later.

Calculated Result:

Choices	***1D***	***2D***	***3D***	***4D***	***5D***	***6D***
1	3/4	5/8	7/12	9/16	11/20	13/24
2	1/4	5/24	7/36	3/16	11/60	13/72
3		5/48	7/72	3/32	11/120	13/114
4		1/16	7/120	9/160	11/200	13/240
5			7/180	3/80	11/300	13/360
6			1/36	3/112	11/420	13/504
7				9/448	11/560	13/672
8				1/64	11/720	13/864
9					11/900	13/1080
10					1/100	13/1320
11						13/1584
12						1/144



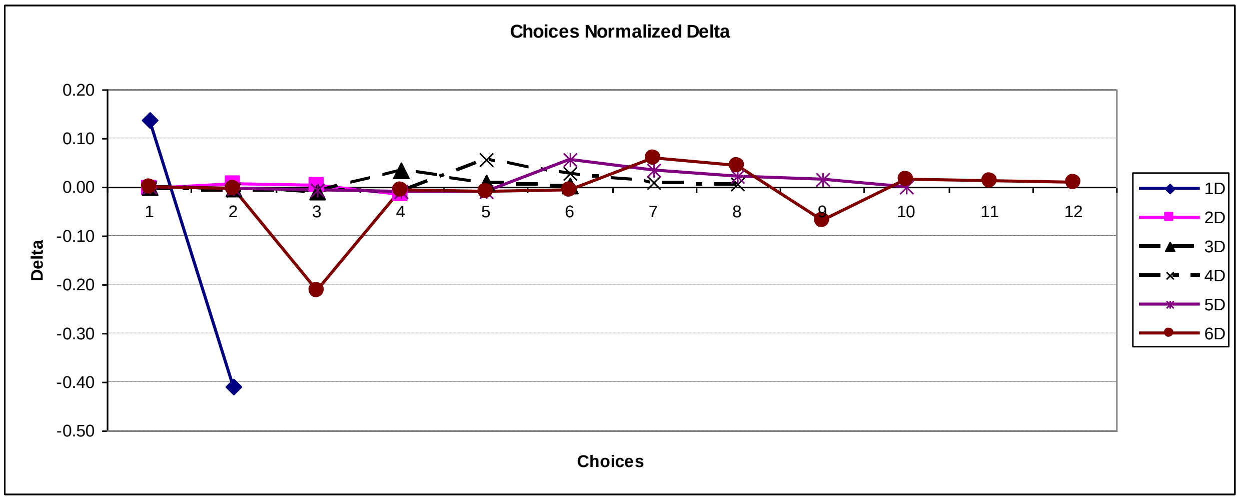
Actual Simulation Result

Choices	*1D*	*2D*	*3D*	*4D*	*5D*	*6D*
1	0.8524	0.6239	0.5826	0.5615	0.5497	0.5415
2	0.1476	0.2099	0.1937	0.1865	0.1827	0.1799
3		0.1045	0.0963	0.0931	0.0911	0.0897
4		0.0616	0.0603	0.0557	0.0545	0.0538
5			0.0392	0.0396	0.0363	0.0358
6			0.0279	0.0275	0.0277	0.0256
7				0.0203	0.0203	0.0205
8				0.0157	0.0156	0.0157
9					0.0124	0.0112
10					0.0100	0.0100
11						0.0083
12						0.0070

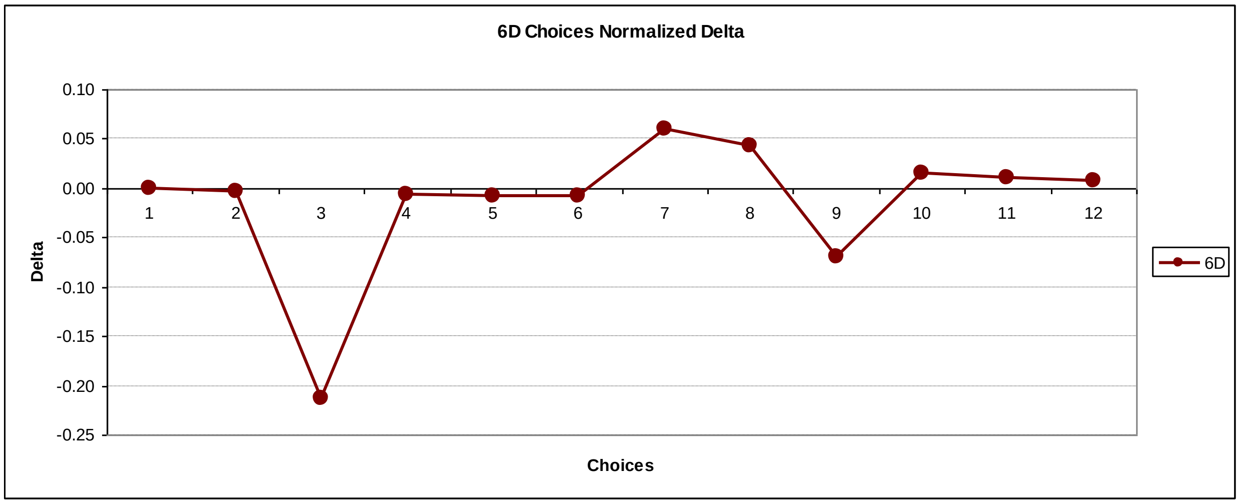
Result Difference Normalized ((Actual – Calculated) / Calculated)

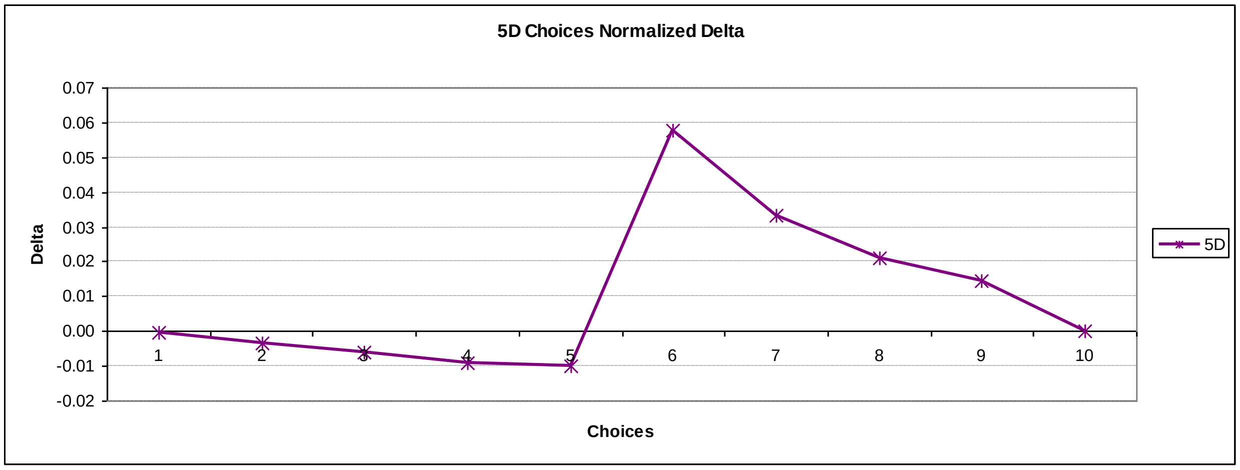
Choices	*1D*	*2D*	*3D*	*4D*	*5D*	*6D*
1	0.136533	-0.001760	-0.001257	-0.001778	-0.000545	-0.000308
2	-0.409600	0.007520	-0.003829	-0.005333	-0.003455	-0.003631
3		0.003200	-0.009486	-0.006933	-0.006182	-0.213400
4		-0.014400	0.033714	-0.009778	-0.009091	-0.006769
5			0.008000	0.056000	-0.010000	-0.008615
6			0.004400	0.026667	0.057636	-0.007508
7				0.010489	0.033455	0.059692
8				0.004800	0.021091	0.043446
9					0.014545	-0.069538
10					0.000000	0.015385
11						0.011323
12						0.008000

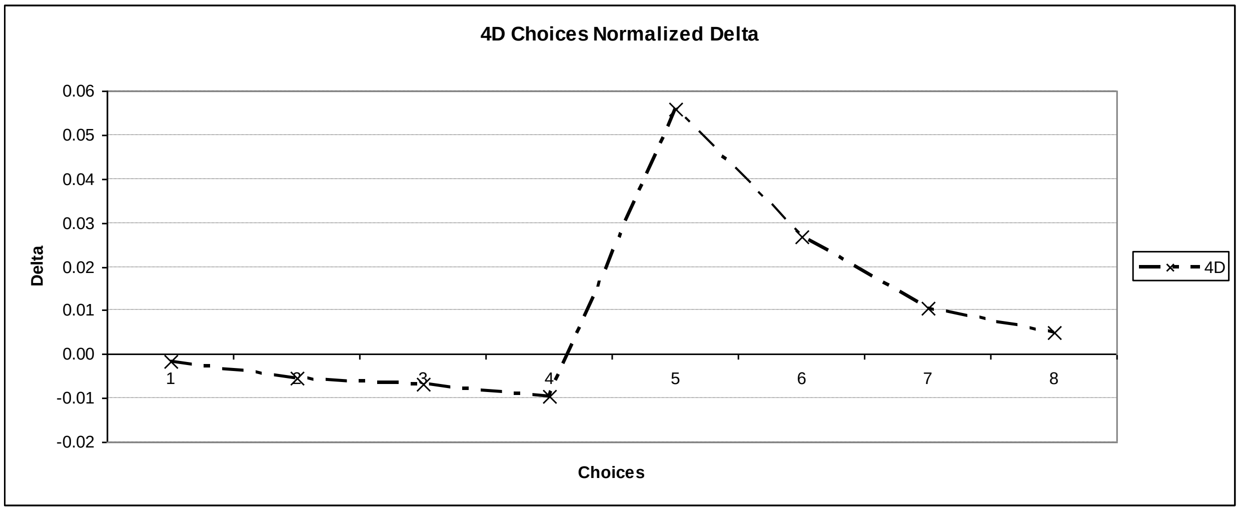
From the data, we see that except for 1D, the actual simulation choices are slightly lower than the calculated result for the first *d* choices. 1D is better by dependencies in section 5.2.3.

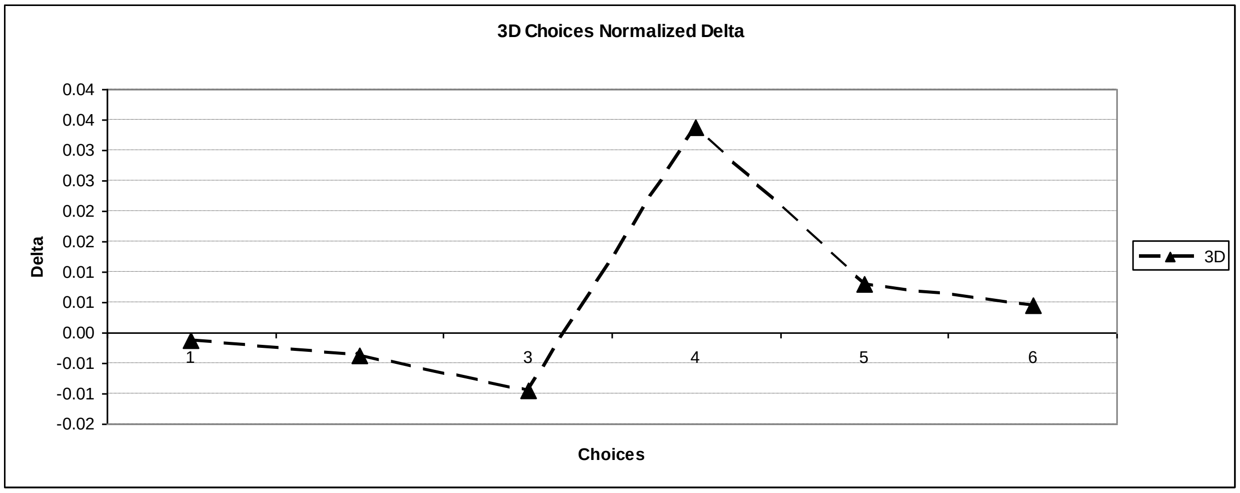


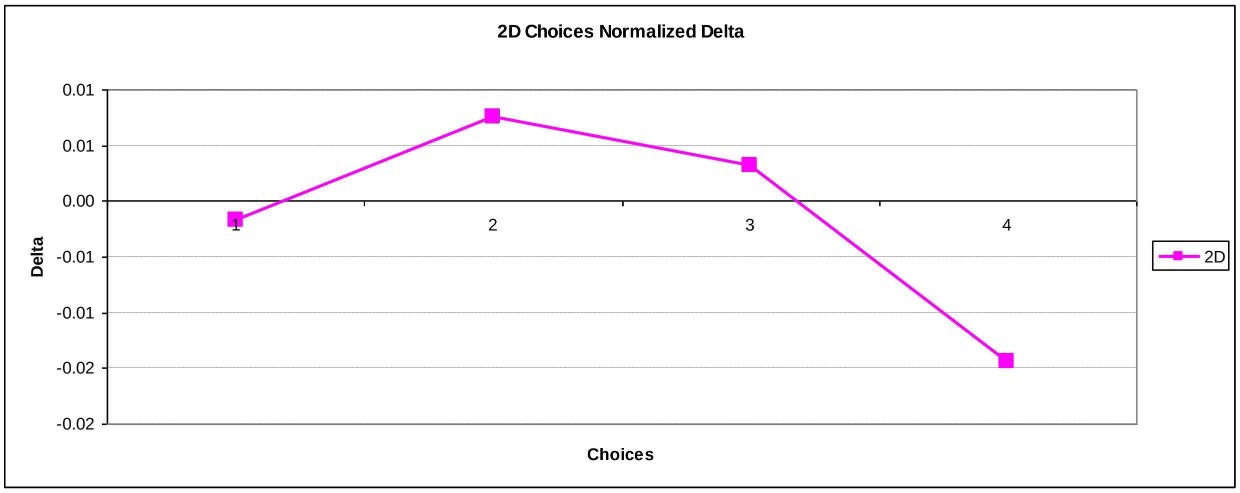
Here are the delta plot for each dimension:


6D Choices Normalized Delta
Delta
Choices
6D
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
1 2 3 4 5 6 7 8 9 10 11 12


5D Choices Normalized Delta
Delta
Choices
5D
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
-0.01
-0.02
1 2 3 4 5 6 7 8 9 10


4D Choices Normalized Delta
Delta
Choices
4D
0.06
0.05
0.04
0.03
0.02
0.01
0.00
-0.01
-0.02
1 2 3 4 5 6 7 8





We now take a look at how we obtained the results for the first d choices of each dimension.

5.5.2.1.1 1D Calculation

The probability of a packet getting its first choice in 1D is:

1/2 * (2/2 + 1/2) = 75%

The first multiplier, 1/2, is the probability of being the *i*-th packet. 1D torus have 2 packets at each node, therefore, probability of being first is the same as being 2nd, 1/2. The equation inside the parenthesis represents the probability of the packet getting its first choice, given that it's the *i*-th packet. If it's the first packet, it must have gotten its first choice, hence 1. If its' the 2nd packet, then with 50% probability its first choice would be available, hence 1/2.

5.5.2.1.2 2D Calculation

Applying similar logic, here is the first choice equation for 2d:
1/4 * (4/4 + 3/4 + 2/4 + 1/4) = 62.5%

Here's the equation for 2nd choice:
1/4 * (0 + 1/4 + 2/4 * 2/3 + 3/4 * 1/3) = .208333%

5.5.2.1.3 3D Calculation

First Choice:
1/6 * (6/6 + 5/6 + 4/6 + 3/6 + 2/6 + 1/6) = 58.333%

Second Choice:
1/6 * (0 + 1/6 + 2/6 * 4/5 + 3/6 * 3/5 + 4/6 * 2/5 + 5/6 * 1/5) = 19.444%

3rd choice:
1/6 * (0 + 0 + 2/6 * 1/5 * 4/4 + 3/6 * 2/5 * 3/4 + 4/6 * 3/5 * 2/4 + 5/6 * 4/5 * 1/4) = 9.7222%

5.5.2.1.4 4D Calculation

First Choice:
1/8 * (8/8 + 7/8 + 6/8 + 5/8 + 4/8 + 3/8 + 2/8 + 1/8) = 56.25%

Second Choice:
1/8 * (0 + 1/8 + 2/8 * 6/7 + 3/8 * 5/7 + 4/8 * 4/7 + 5/8 * 3/7 + 6/8 * 2/7 + 7/8 * 1/7) = 18.75%

Third Choice
1/8 * (0 + 0 + 2/8 * 1/7 + 3/8 * 2/7 * 5/6 + 4/8 * 3/7 * 4/6 + 5/8 * 4/7 * 3/6 + 6/8 * 5/7 * 2/6 + 7/8 * 6/7 * 1/6) = 9.375%

Fourth Choice
1/8 * (0 + 0 + 0 + 3/8 * 2/7 * 1/6 + 4/8 * 3/7 * 2/6 * 4/5 + 5/8 * 4/7 * 3/6 * 3/5+ 6/8 * 5/7 * 4/6 * 2/5+ 7/8 * 6/7 * 5/6 * 1/5) = 5.625%

5.5.2.1.5 5D Calculation

First Choice:
1/10 * ((10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1)/10) = 55%

Second Choice:
1/10 * (0 + 1/10 + 2/10 * 8/9 + 3/10 * 7/9 + 4/10 * 6/9 + 5/10 * 5/9 + 6/10 * 4/9 + 7/10 * 3/9 + 8/10 * 2/9 + 9/10 * 1/9) = 18.333%

Third Choice:
1/10 * (0 + 0 + 2/10 * 1/9 + 3/10 * 2/9 * 7/8 + 4/10 * 3/9 * 6/8 + 5/10 * 4/9 * 5/8 + 6/10 * 5/9 * 4/8 + 7/10 * 6/9 * 3/8 + 8/10 * 7/9 * 2/8 + 9/10 * 8/9 * 1/8) = 9.1666%

Fourth Choice:
1/10 * (0 + 0 + 0 + 3/10 * 2/9 * 1/8 + 4/10 * 3/9 * 2/8 * 6/7 + 5/10 * 4/9 * 3/8 * 5/7 + 6/10 * 5/9 * 4/8 * 4/7 + 7/10 * 6/9 * 5/8 * 3/7 + 8/10 * 7/9 * 6/8 * 2/7 + 9/10 * 8/9 * 7/8 * 1/7) = 5.5%

Fifth Choice:
1/10 * (0 + 0 + 0 + 0 + 4/10 * 3/9 * 2/8 * 1/7 + 5/10 * 4/9 * 3/8 * 2/7 * 5/6 + 6/10 * 5/9 * 4/8 * 3/7 * 4/6 + 7/10 * 6/9 * 5/8 * 4/7 * 3/6 + 8/10 * 7/9 * 6/8 5/7 * 2/6 + 9/10 * 8/9 * 7/8 * 6/7 * 1/6) = 3.6666%

5.5.2.1.6 6D Calculation

First Choice:
1/12 * ((12 + 11 + 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1)/12) = 54.1666%

Second Choice:
1/12 * (0 + 1/12 + 2/12 * 10/11 + 3/12 * 9/11 + 4/12 * 8/11 + 5/12 * 7/11 + 6/12 * 6/11 + 7/12 * 5/11 + 8/12 * 4/11 + 9/12 * 3/11 + 10/12 * 2/11 + 11/12 * 1/11) = 18.0555%

Third Choice:
1/12 * (0 + 0 + 2/12 * 1/11 + 3/12 * 2/11 * 9/10 + 4/12 * 3/11 * 8/10 + 5/12 * 4/11 * 7/10 + 6/12 * 5/11 * 6/10 + 7/12 * 6/11 * 5/10 + 8/12 * 7/11 * 4/10 + 9/12 * 8/11 * 3/10 + 10/12 * 9/11 * 2/10 + 11/12 * 10/11 * 1/10) = 9.02777%

Fourth Choice:
1/12 * (0 + 0 + 0 + 3/12 * 2/11 * 1/10 + 4/12 * 3/11 * 2/10 * 8/9 + 5/12 * 4/11 * 3/10 * 7/9 + 6/12 * 5/11 * 4/10 * 6/9 + 7/12 * 6/11 * 5/10 * 5/9 + 8/12 * 7/11 * 6/10 * 4/9 + 9/12 * 8/11 * 7/10 * 3/9 + 10/12 * 9/11 * 8/10 * 2/9 + 11/12 * 10/11 * 9/10 * 1/9) = 5.41666%

Fifth Choice:
1/12 * (0 + 0 + 0 + 0 + 4/12 * 3/11 * 2/10 * 1/9 + 5/12 * 4/11 * 3/10 * 2/9 * 7/8 + 6/12 * 5/11 * 4/10 * 3/9 * 6/8 + 7/12 * 6/11 * 5/10 * 4/9 * 5/8 + 8/12 * 7/11 * 6/10 * 5/9 * 4/8 + 9/12 * 8/11 * 7/10 * 6/9 * 3/8 + 10/12 * 9/11 * 8/10 * 7/9 * 2/8 + 11/12 * 10/11 * 9/10 * 8/9 * 1/8) = 3.6111%

Sixth Choice:
1/12 * (0 + 0 + 0 + 0 + 0 + 5/12 * 4/11 * 3/10 * 2/9 * 1/8 + 6/12 * 5/11 * 4/10 * 3/9 * 2/8 * 6/7 + 7/12 * 6/11 * 5/10 * 4/9 * 3/8 * 5/7 + 8/12 * 7/11 * 6/10 * 5/9 * 4/8 * 4/7 + 9/12 * 8/11 * 7/10 * 6/9 * 5/8 * 3/7 + 10/12 * 9/11 * 8/10 * 7/9 * 6/8 * 2/7 + 11/12 * 10/11 * 9/10 * 8/9 * 7/8 * 1/7) = 2.57936%

5.5.2.1.7 General Formula

The general formula for calculating the probability of the *i*-th packet being routed *j*-th choice in a d dimensional torus:

d = dimension
i = *i*-th packet
j = *j*-th choice

$f1(d, i, j) = [\prod_{(k=0 \text{ to } j-2)} ((i - 1 - k) / (2d - k))] * (1 - (i - j) / (2d - j + 1))$

For the 1st choice, the formula is simply $(2d + 1) / 4d$.

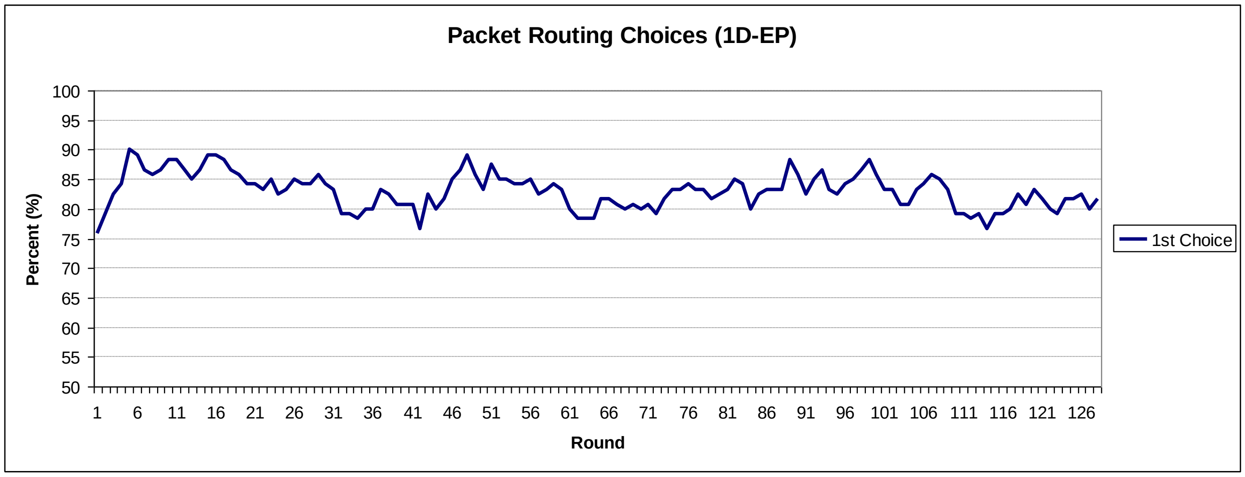
The general formula for calculating the j-th choice in d dimension is:

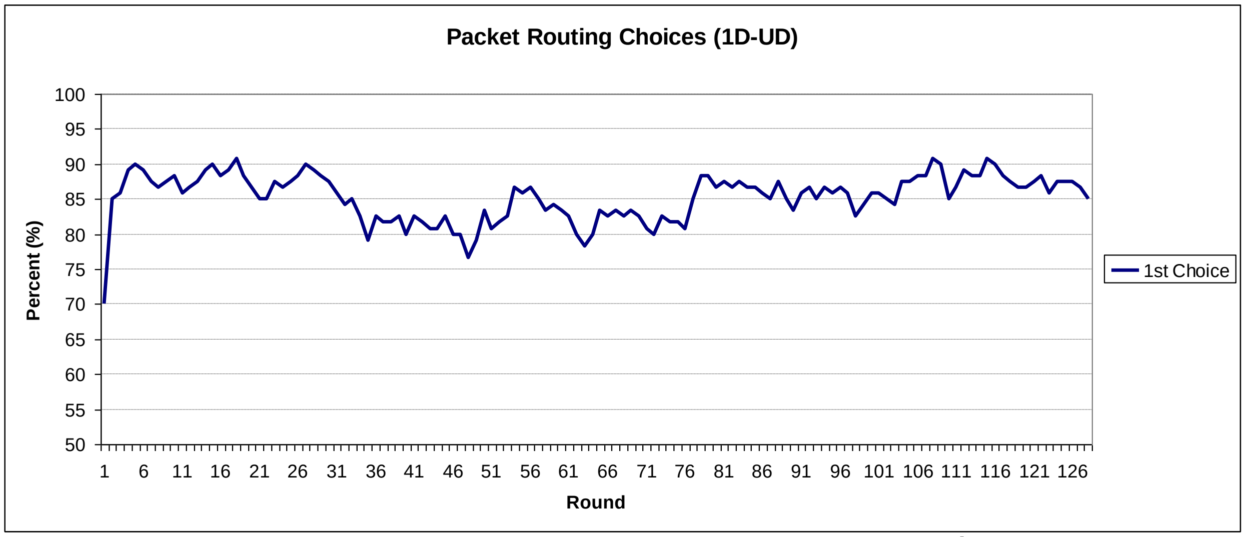
$f(d, j) = 1/(2d) * \sum_{(i=1 \text{ to } 2d)} f1(d, i, j)$

5.5.2.2 Routing Choices Per Round

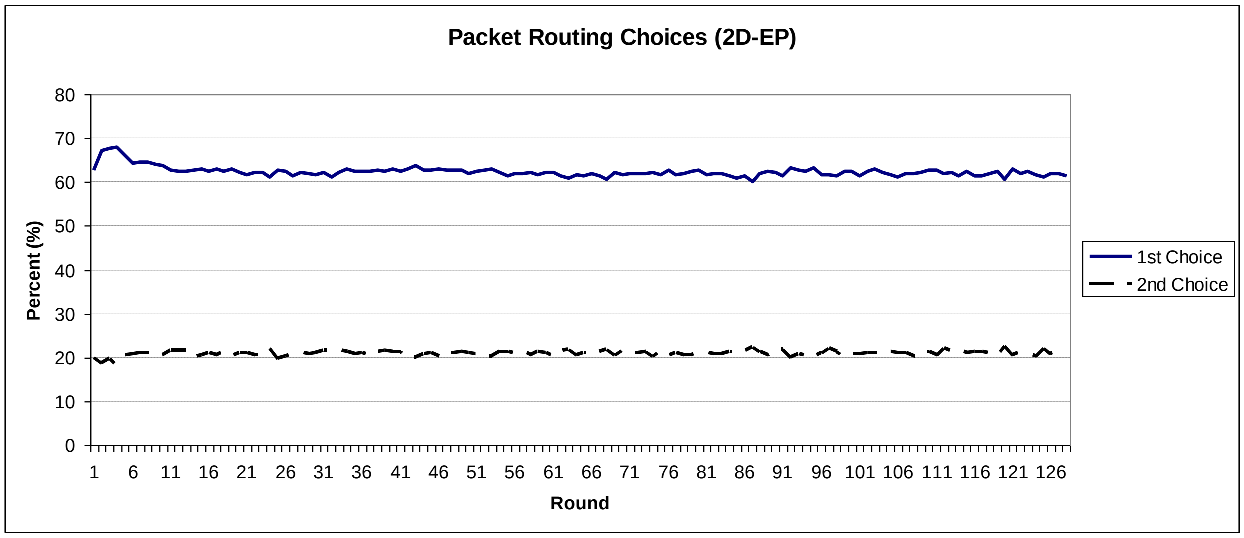
We present the data in 2 forms. First we show the packet choices per round for the different dimensions. Then we show a histogram of the average.

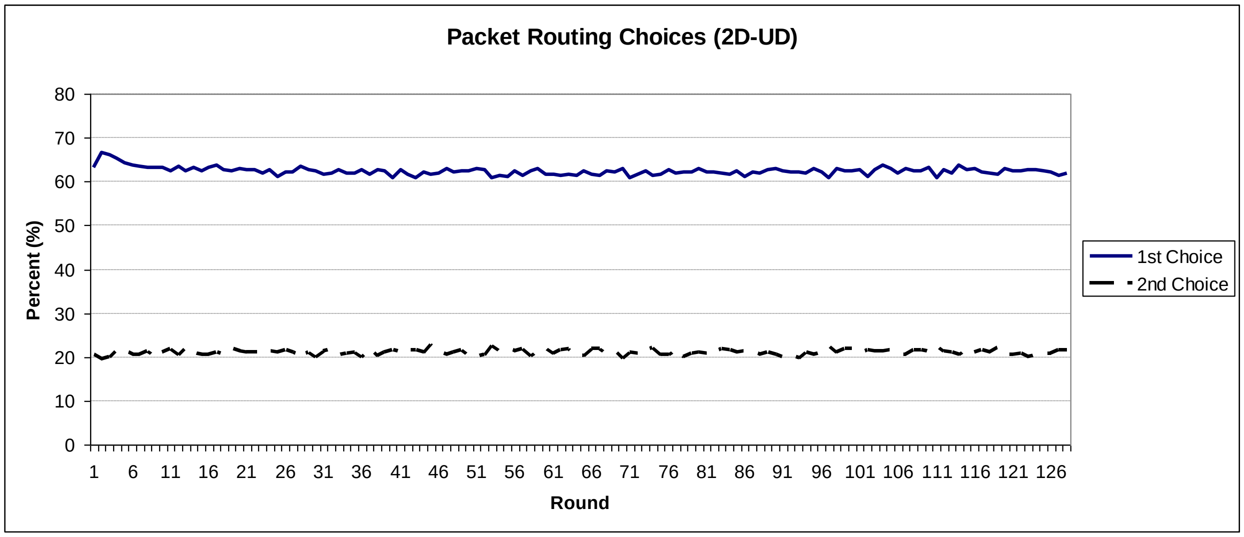
Here are the routing choices per round for 1-6D, with both EP/UD models:





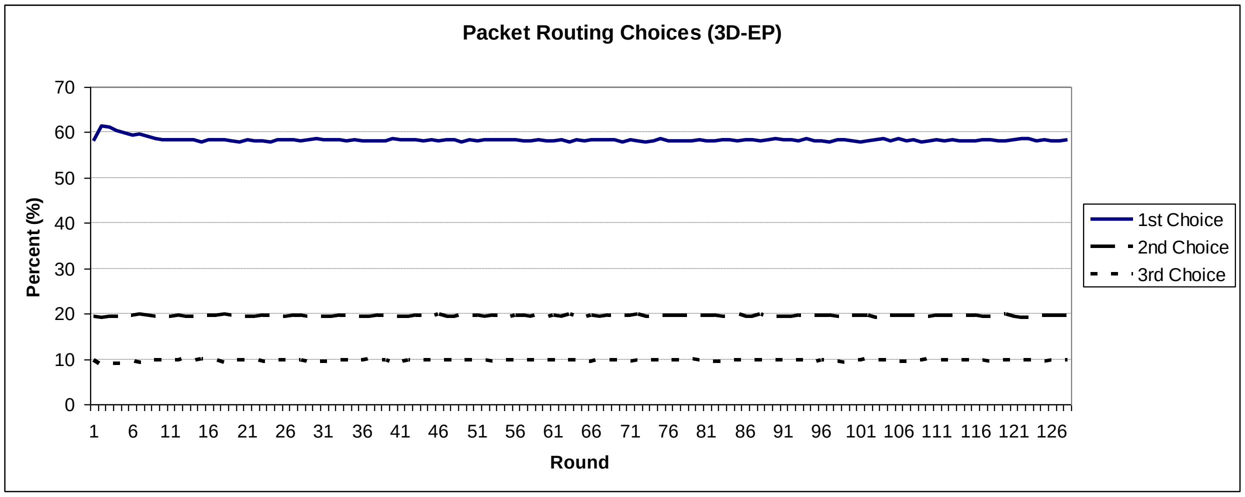
In 1D, this plot corresponds exactly to the plot for moving closer, as the 2nd choice will not bring you closer in 1D, except in the wrap-around case.

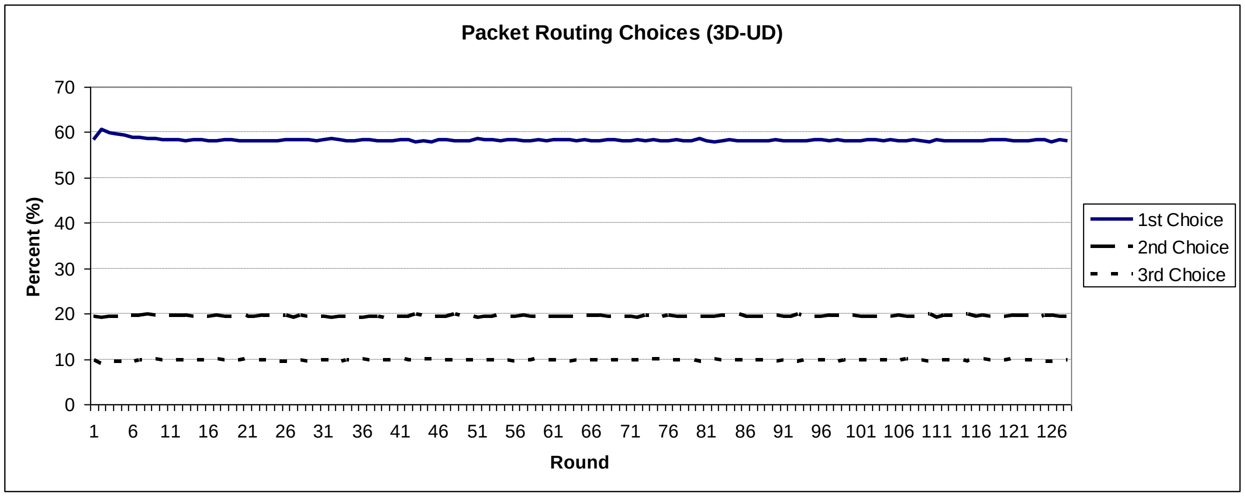


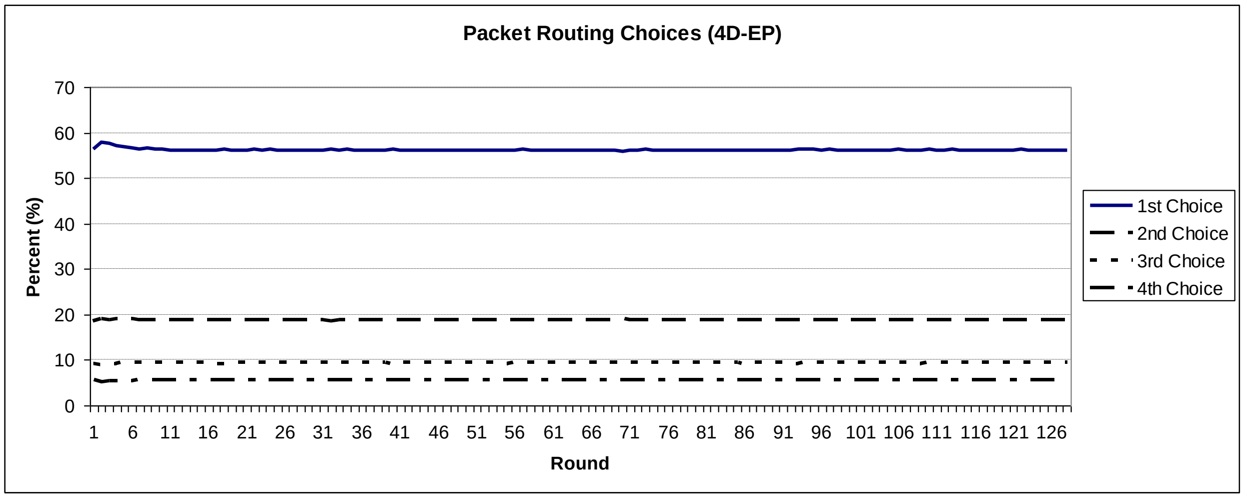


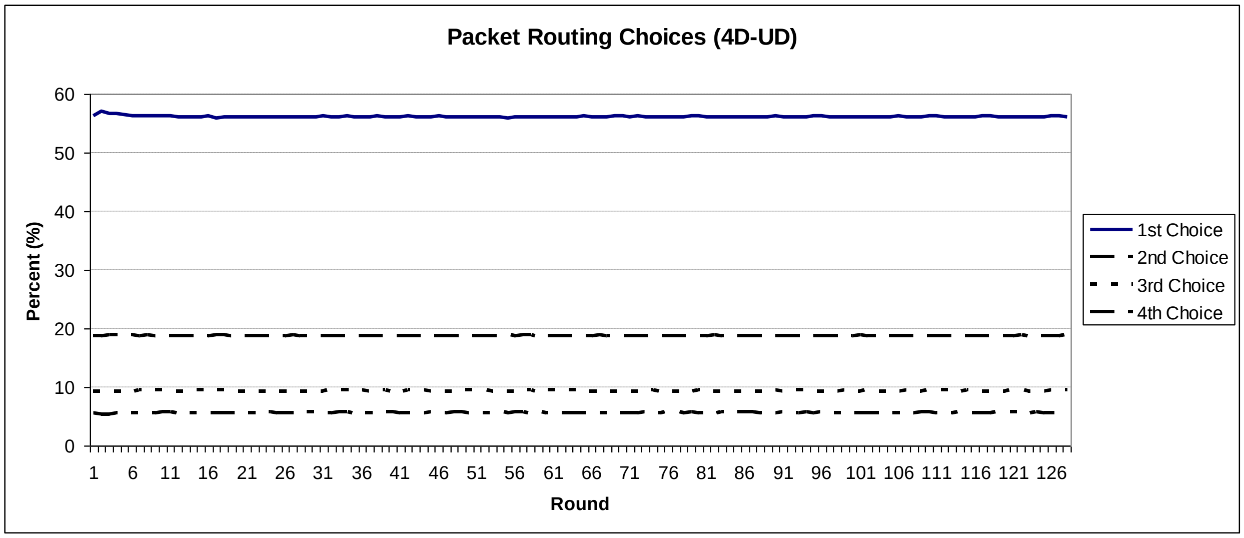
In 2D, the data becomes more interesting, we see that more than 60% of the packets are getting their 1st choice, and about 20% are getting 2nd choice. The 1st and 2nd choice added together should correspond to the packets moving closer plot for 2D.

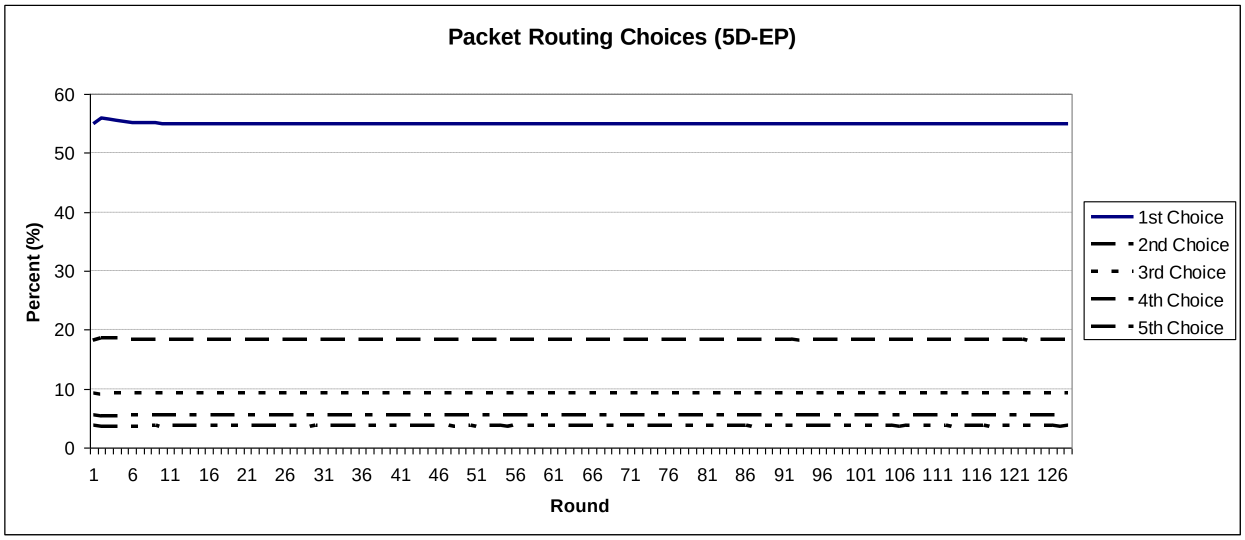
Here are the remaining plots:

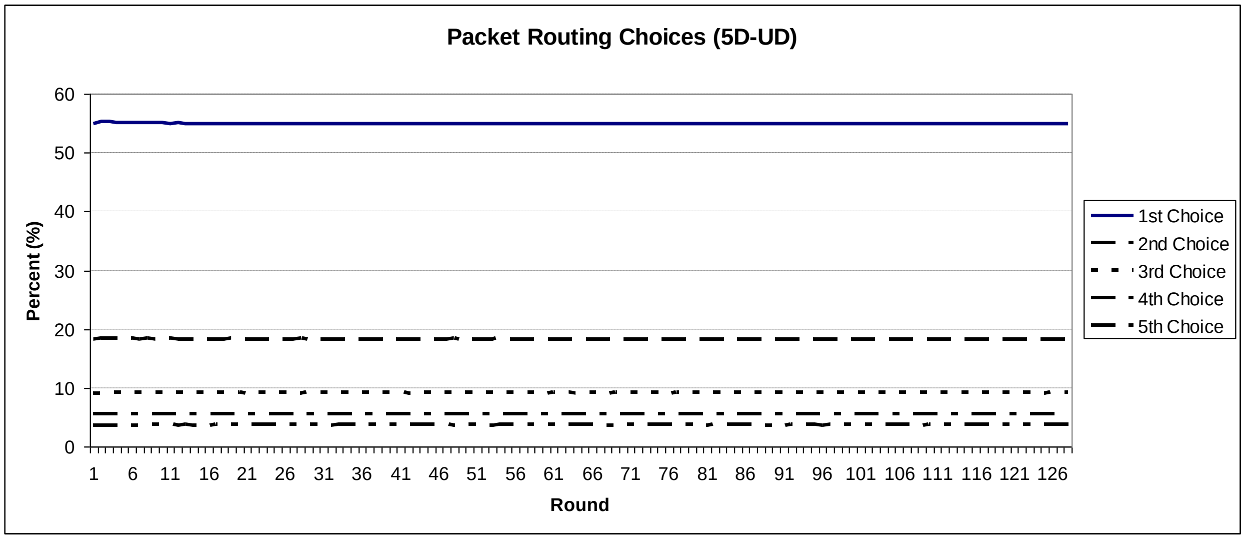


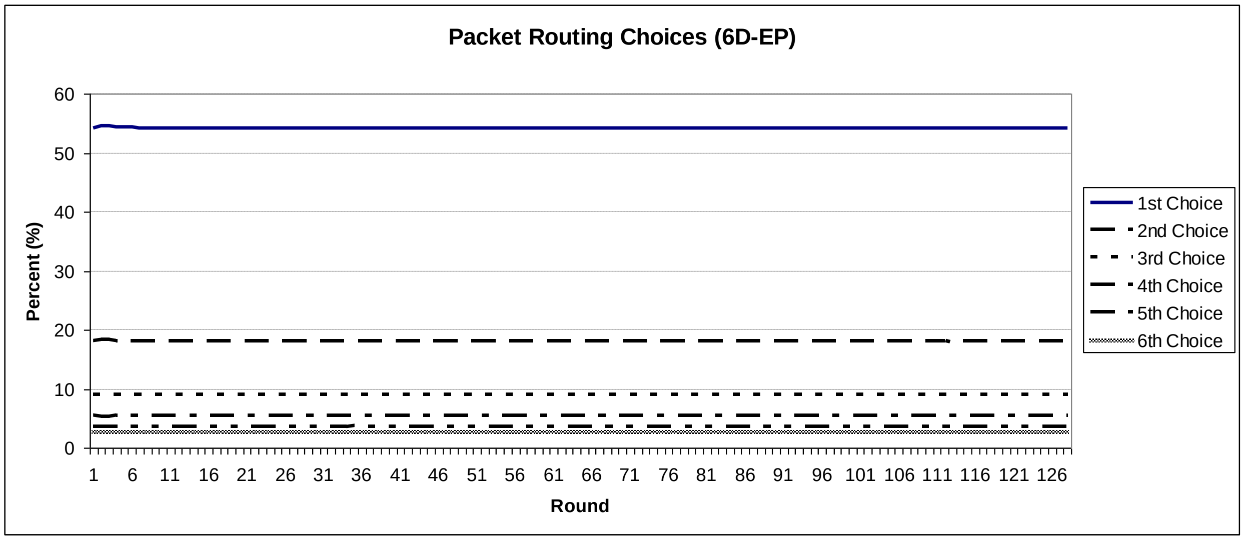


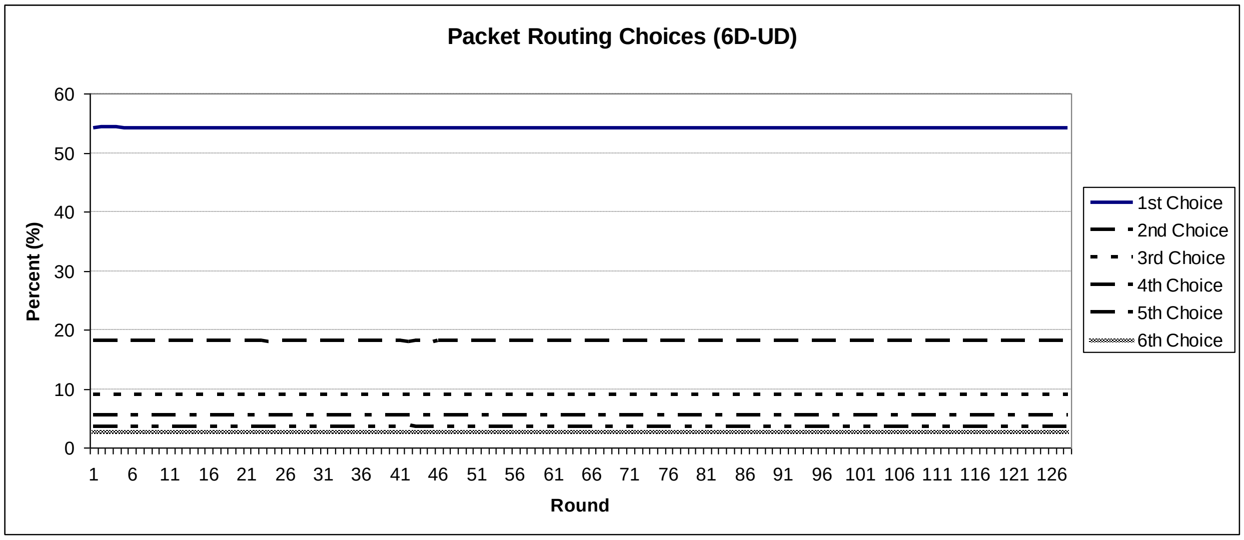



Packet Routing Choices (4D-UD)
60
50
40
30
20
10
0
Percent (%)
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
Round
1st Choice
2nd Choice
3rd Choice
4th Choice


Packet Routing Choices (5D-EP)
60
50
40
30
20
10
0
Percent (%)
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
Round
1st Choice
2nd Choice
3rd Choice
4th Choice
5th Choice


Packet Routing Choices (5D-UD)
60
50
40
30
20
10
0
Percent (%)
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
Round
1st Choice
2nd Choice
3rd Choice
4th Choice
5th Choice



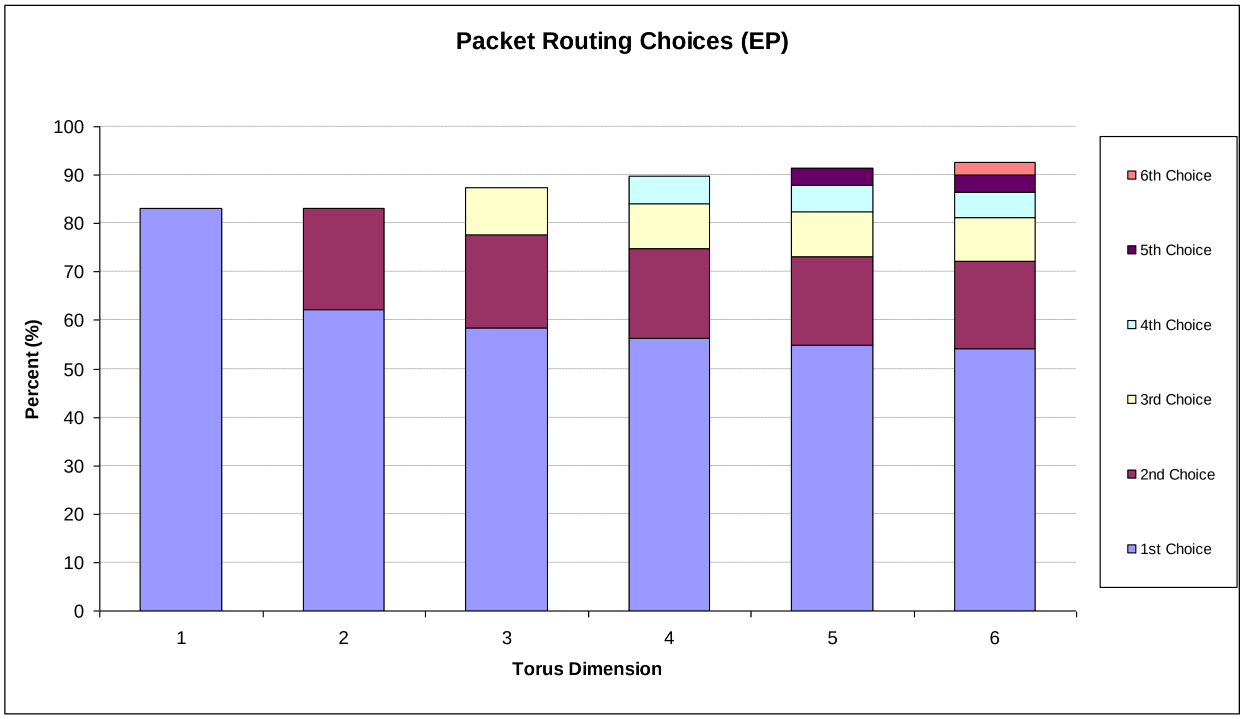


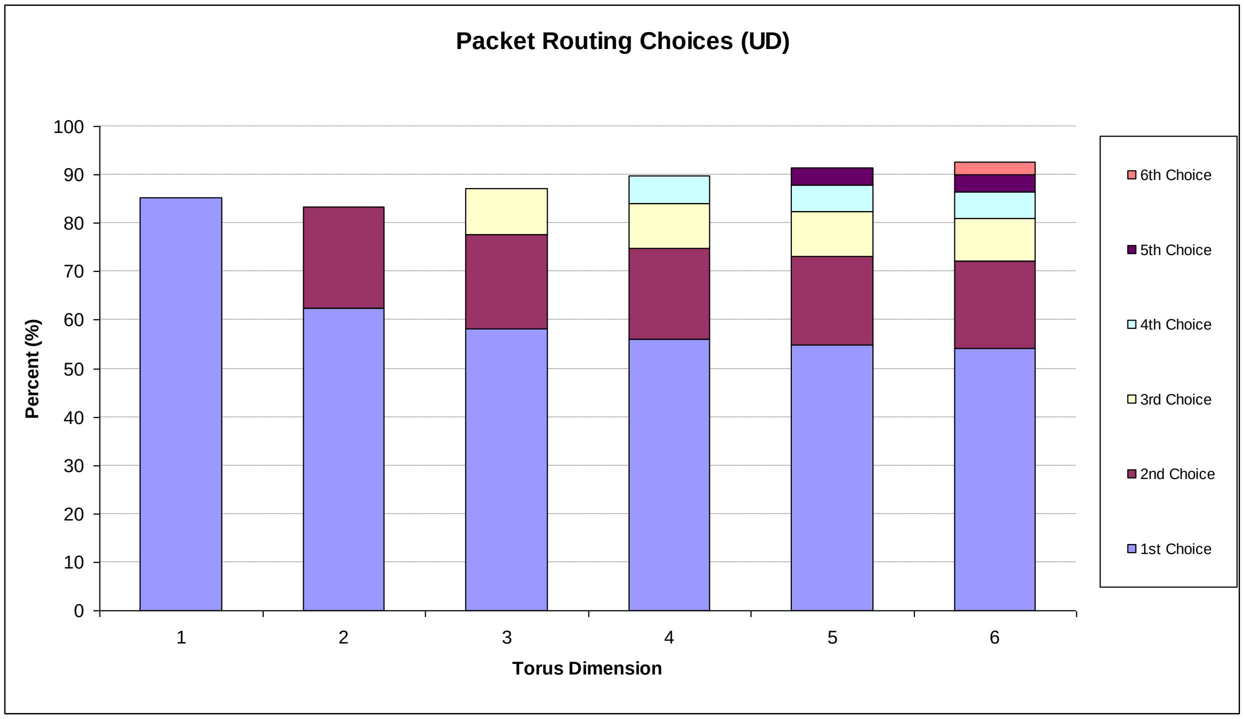
We can see from the plot that the 1st choice packets are more than 50% for all these case, generally close to the $(2d+1)/4d$ expected in an independent system. The 2nd choice is less than half, the 3rd choice half of 2nd, etc.

We can also see the routing choices are very consistent across rounds, with very little fluctuation. This is another indication that the system is very stable.

5.5.2.3 Routing Choices Comparison

Let us now present the data in a slightly different form. We plot a histogram of the choices for each dimension. This allows us to more easily see the relative proportion of the choices compare to one another.

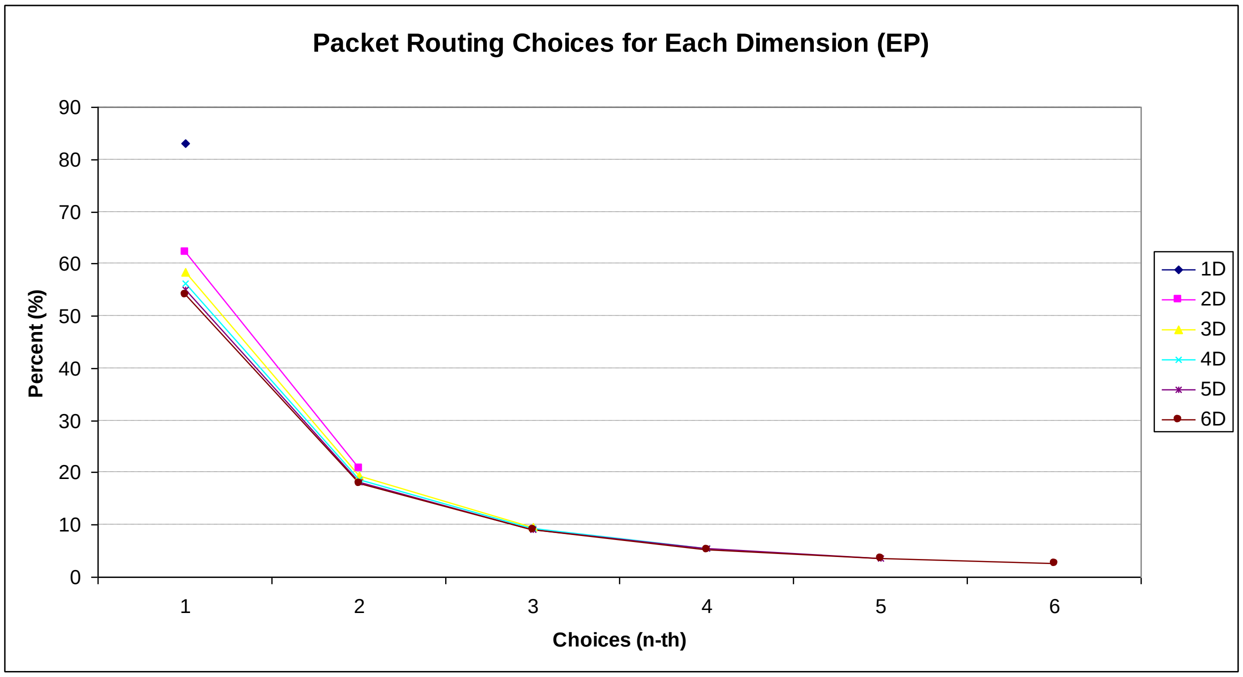


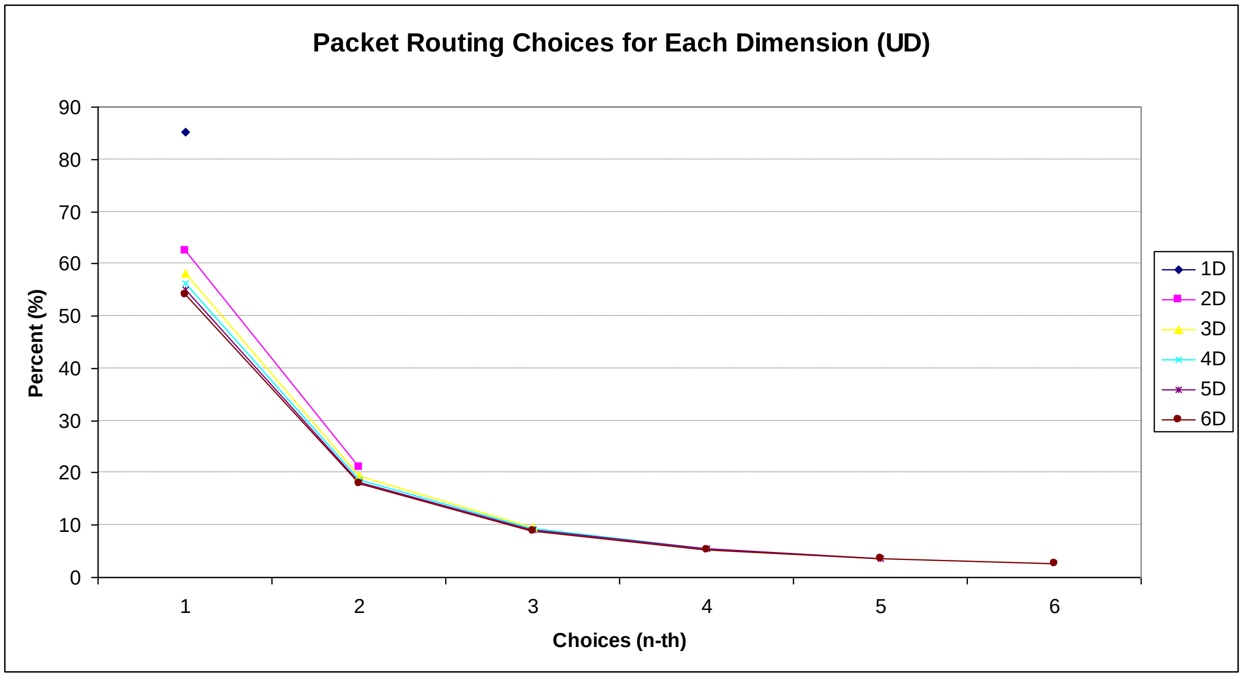


Each bar represents packets that are moving closer for that dimension. The bar is further divided among the various routing choices, those getting 1st choice, 2nd choice, etc. The section from the top of the bar to 100% would represent the packets that are going in the wrong direction. (Of course, for some packets, even the 2nd choice might be a wrong direction, for instance, packets 1 away from their destination).

We notice that for all dimensions, 1st choice packets are the majority of the packets. I.e. number of packets getting 2nd choice is significantly less than those getting 1st choice. This is very good, as first choice represents the preferred route, and knowing most packets are getting the preferred route is a good indication that the routing is going well.

Let us now take a look at the change in routing choices for each dimension:





As it is more apparent from this plot, the packet choices drop significantly for the 1st three choices, and levels off for the remaining choices.

This is very good. It confirms that packet routing is working well. More than half of the packets are getting their 1st choice route, and in every case, higher preferences occur significantly more often than lower preferences.

5.5.3 Packet Route Flexibility

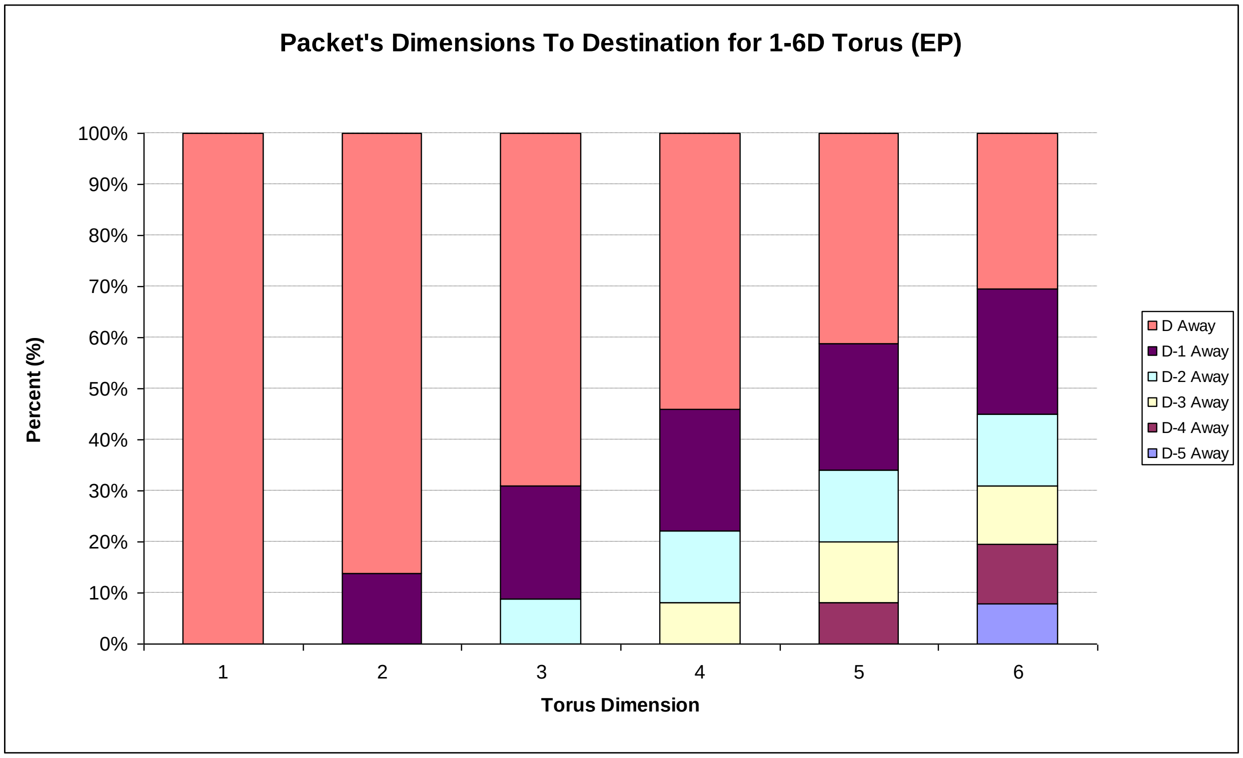
Another interesting metric to consider when evaluating routing performance is the packet's route flexibility. For the discussion, we consider packets with x+1 directions to go to be more flexible than packets with x directions to go, simply because the packet will have less chance of traveling in the wrong direction.

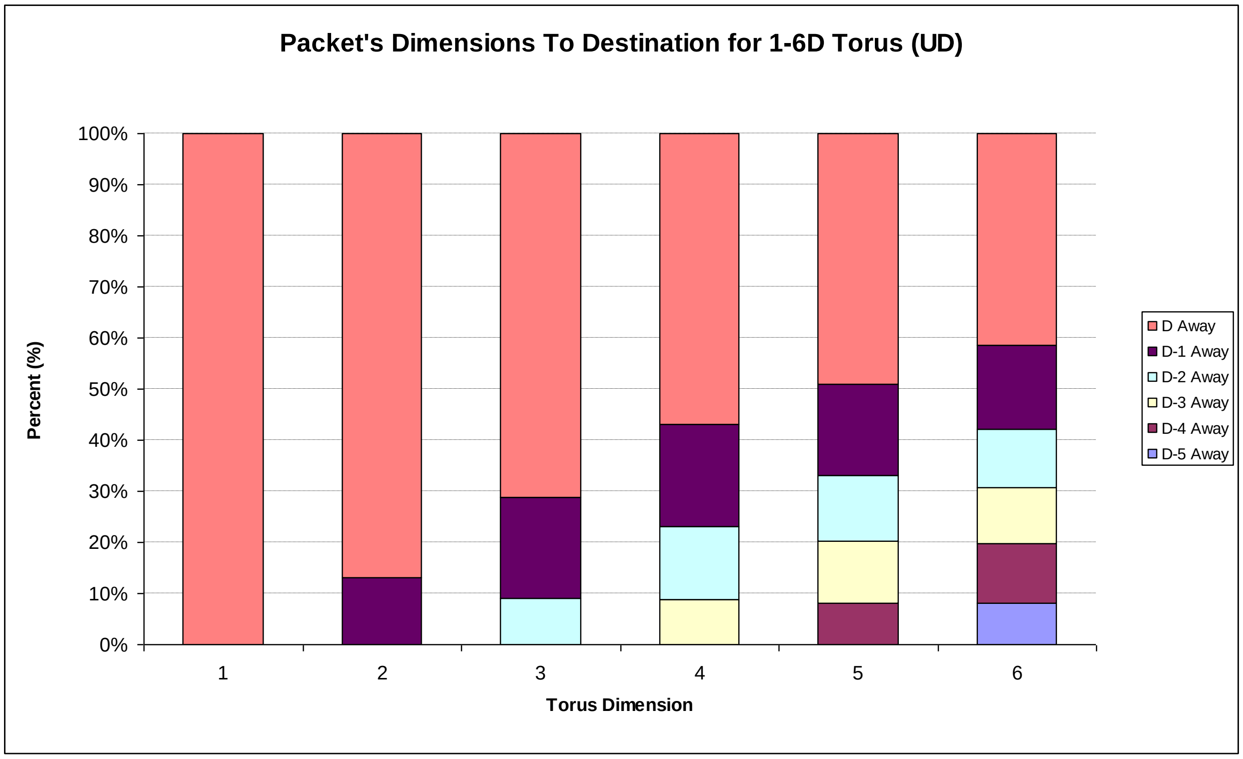
For our system, we want to the percentage breakdown of packets by their flexibility ranking. That is, what % of packets have 1 directions to go, 2 directions, 3, etc.

We present the data in 2 forms similar to the choices data.

5.5.3.1 Route Type Histogram

We first show the route type data in a histogram. Unlike the choices plot which does not reach 100%, the bar does reach 100%, as we are plotting everything.





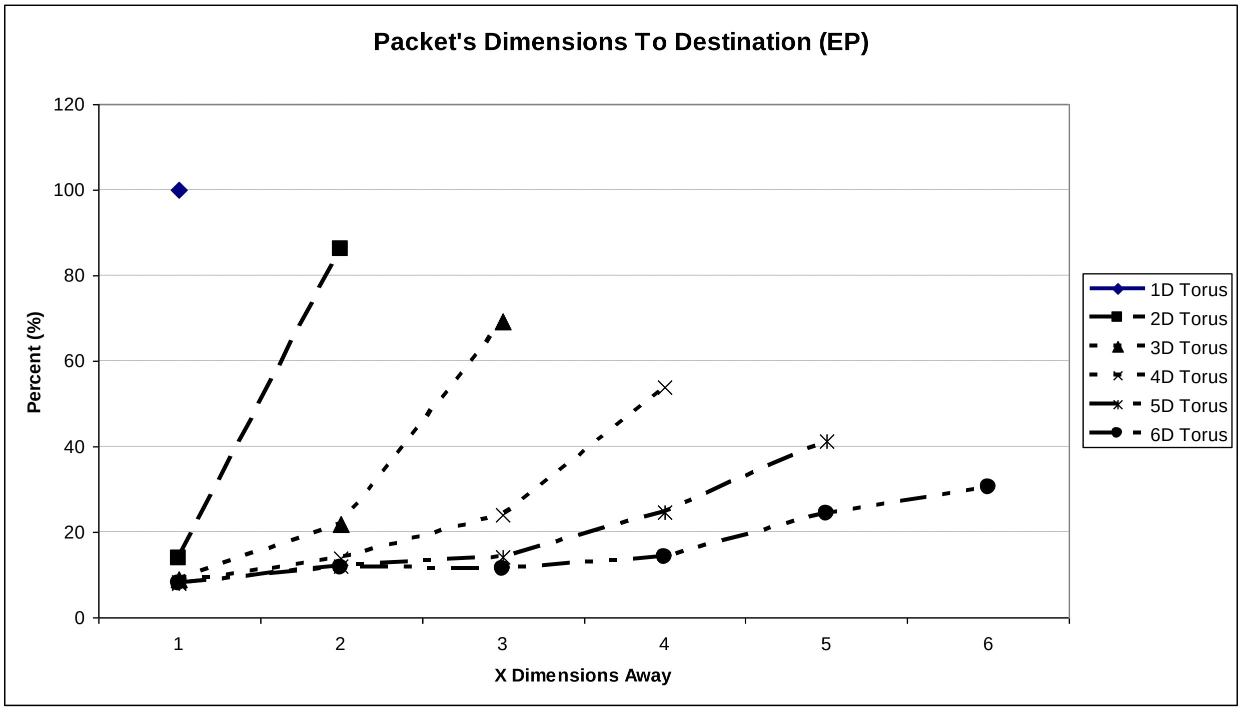
From the chart, we can see that a large percentage of packets, more than 50% in many cases, have the maximum number of dimensions it can go. This implies those packets will have a higher degree of flexibility. If one of their preferred directions is taken, they still have other directions to take to get closer to their destination. Only if all its preferred directions are taken will they go in the wrong direction. The probability of this happening is low.

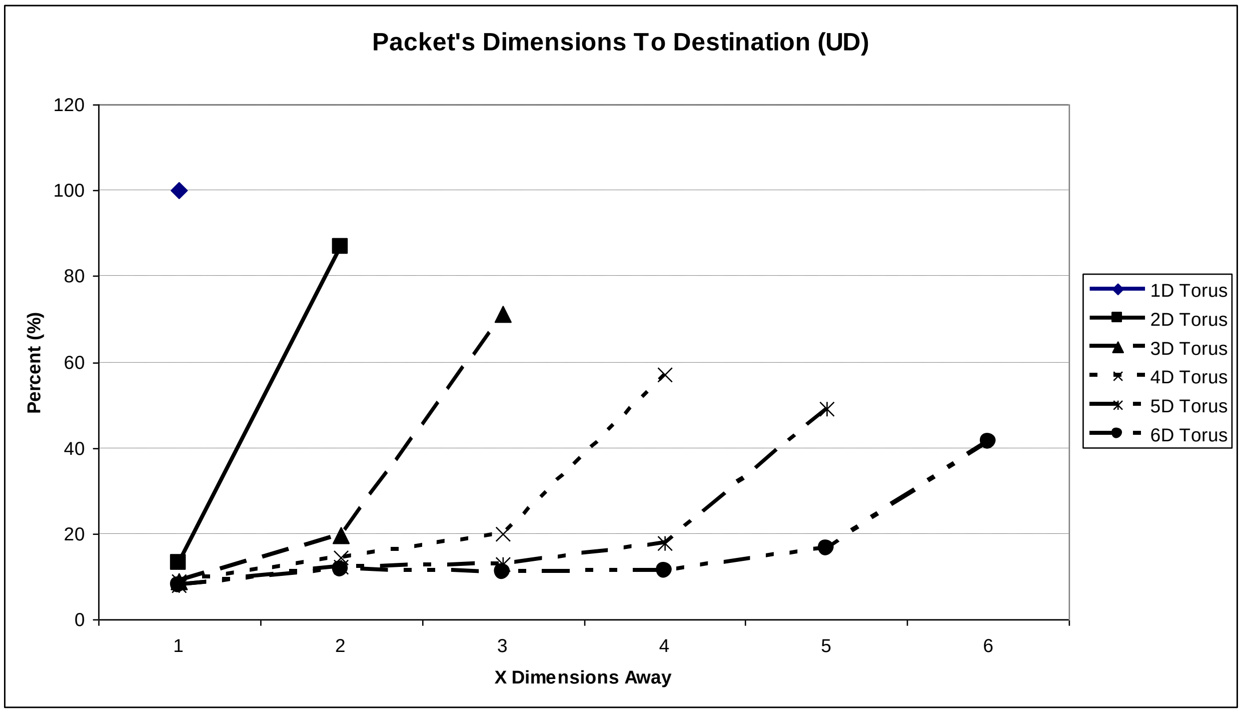
The 2nd observation we can make is the percentage of inflexible packets, that is, those having only one direction to go, is very low. Roughly around 10%. These are the packets that if they don't get to go where they want to go, will be traveling in the wrong direction. Since there are only 10% of them, the penalty they may incur is offset by the positive routing on the other packets. Of course, we can not get rid of these, as all packets one step before delivery are only 1 dimensional off. With a 4% delivery rate, we have a lower bound on 1D packets: at each time step, 4% of packets were 1D packets 1 step before delivery.

In addition, the breakdown indicates that the routing should be pretty close to the optimal. For example, in the 4D case, the % of packets having 4 direction to go is more than 50%, the % of packets having 3 direction to go is 25%, 2 direction to go is 12%, etc. This implies the packets have a very high degree of flexibility. 3 dimensional packets have less 'flexibility' than 4 dimensional packets, but they are only half as many of them. Although this doesn't quite generalize to 5-6 dimensions, it nonetheless is an indication that the packets in the system retains a high degree of flexibility and the routing is performing well. (In 5-6D systems, size of the torus is a limiting factor. Any packet within 5 of destination can't be a 6D packet).

5.5.3.2 Route Type Chart

We now present the data in a slightly different form, to allow us to better see the change in route type for a particular dimension.

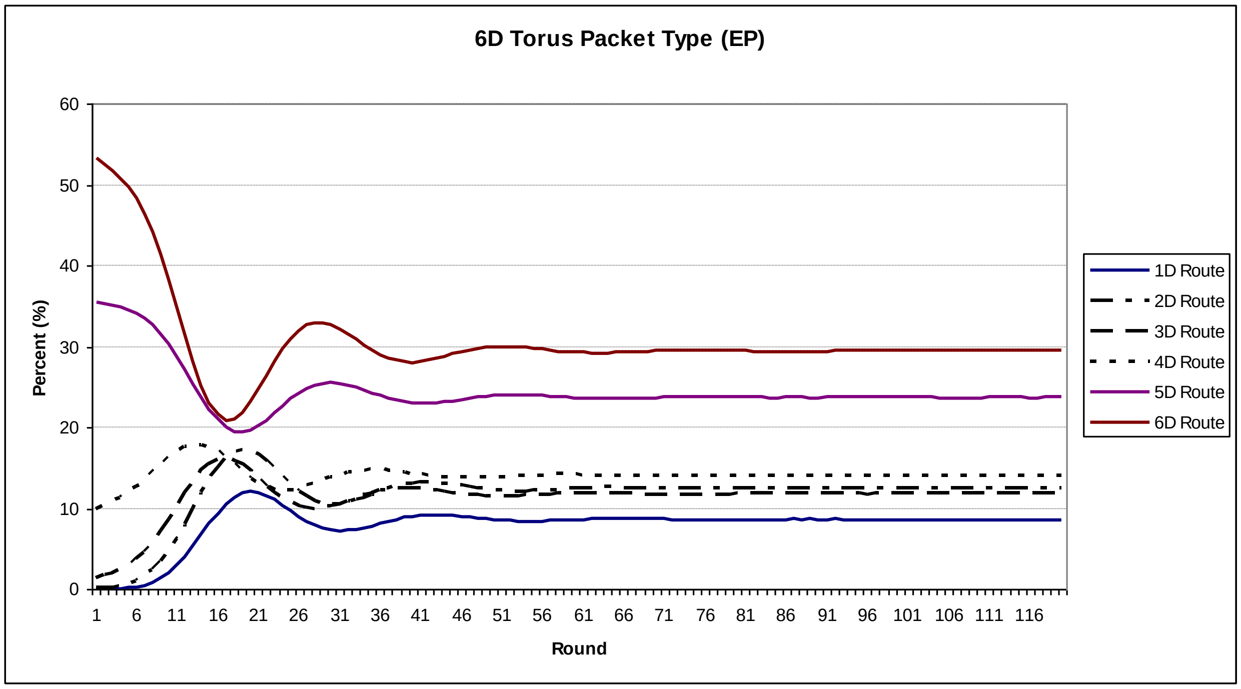


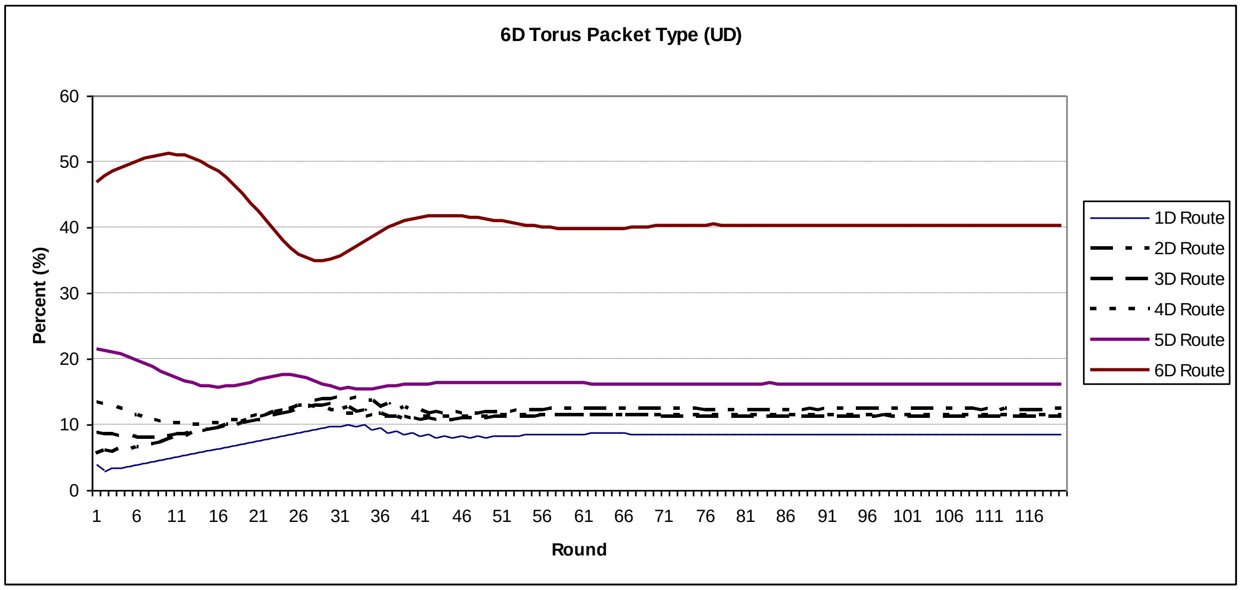


The data indicates that for each dimension, the most flexible packets are significantly higher than the rest of the packets, especially for lower dimensions. Changes between packets having (*d*-1) dimensions to 1 dimension are not as dramatic. The difference between the number of *d* dimension packets compare to (*d*-1) dimensions seems to drop as dimensions increase. Perhaps for larger dimensions, the distribution looks like it will be uniform.

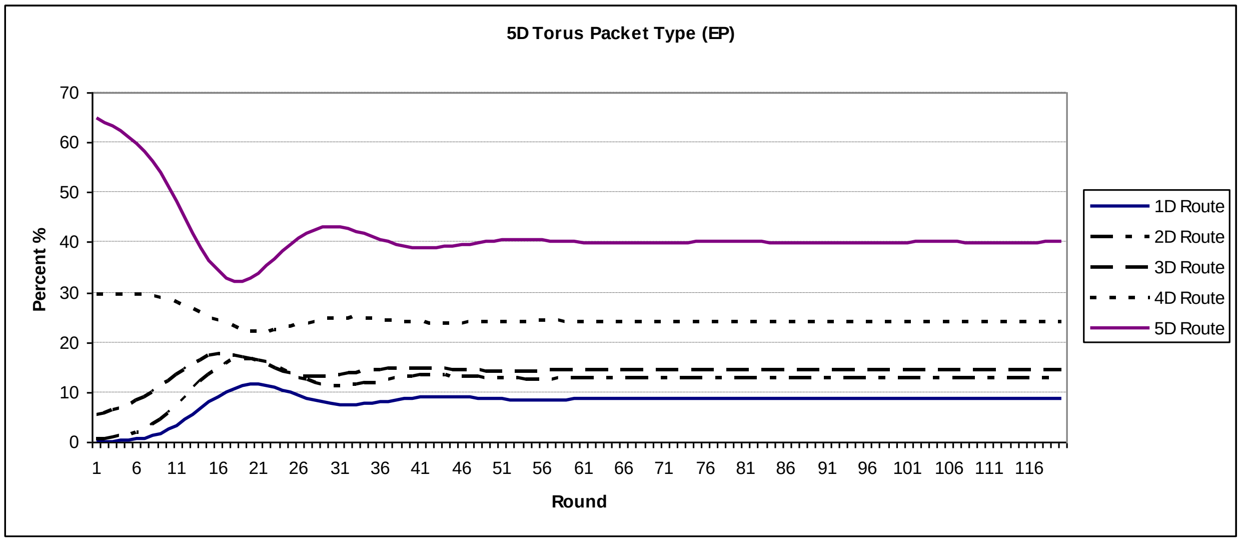
5.5.3.3 Route Type Per Round

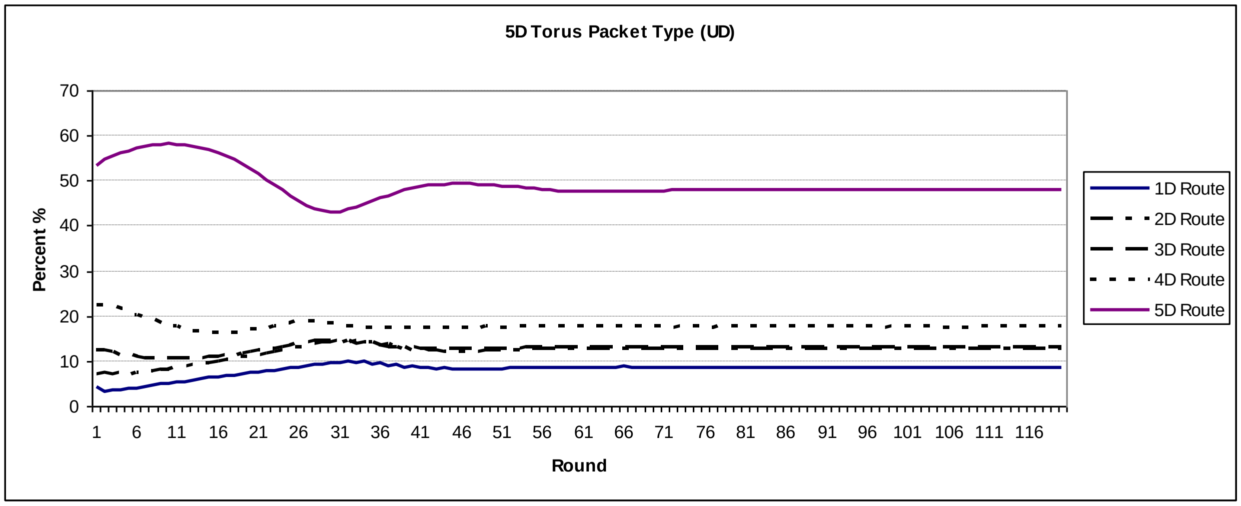
Let's now take a look at how the route type changes during the simulation.

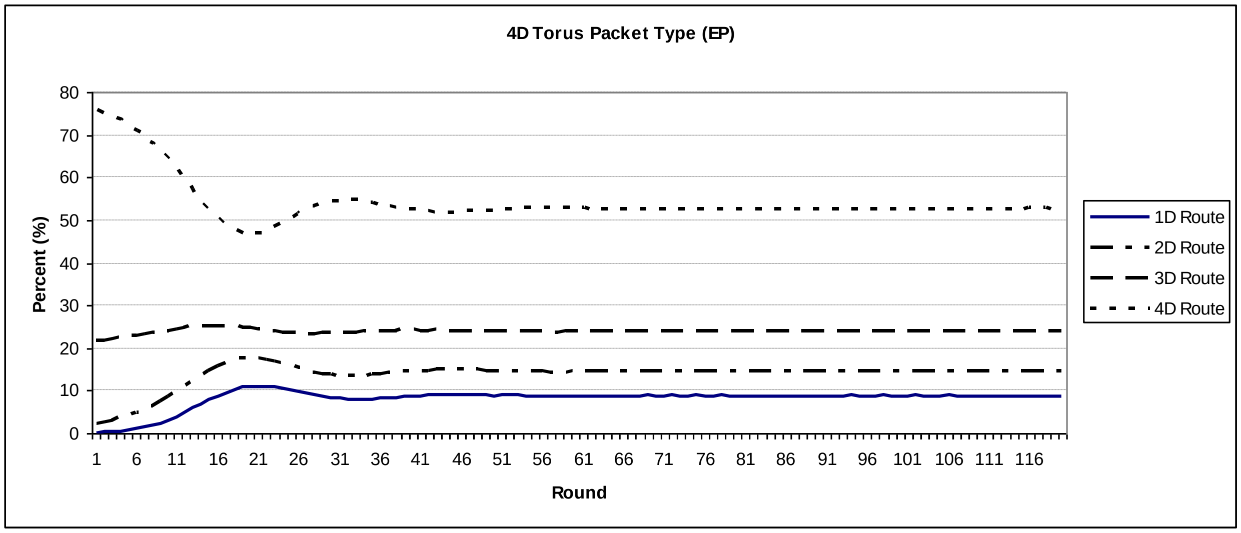


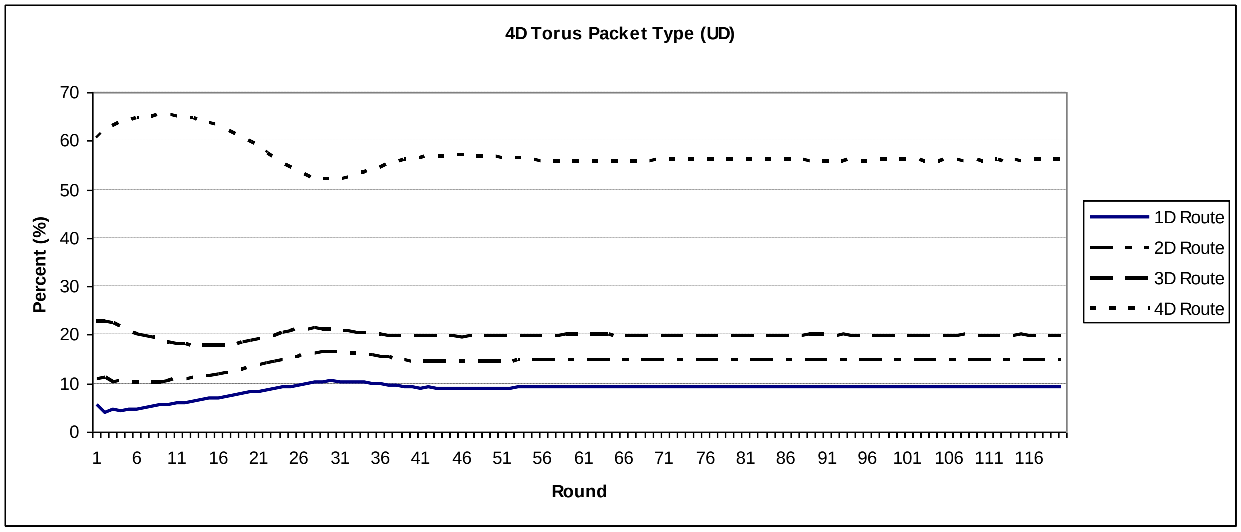


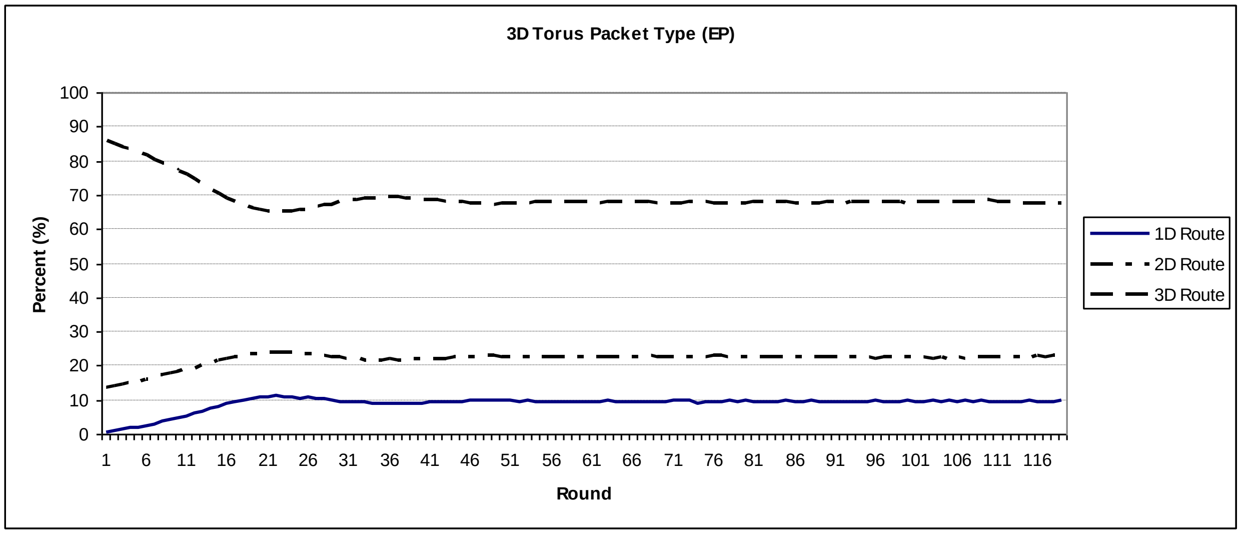
The curve for each packet type is similar to the delivery curve examined in previous sections. It indicates the various types do change initially, and reach a stable ratio just like delivery rate. Here are the remaining dimensions:

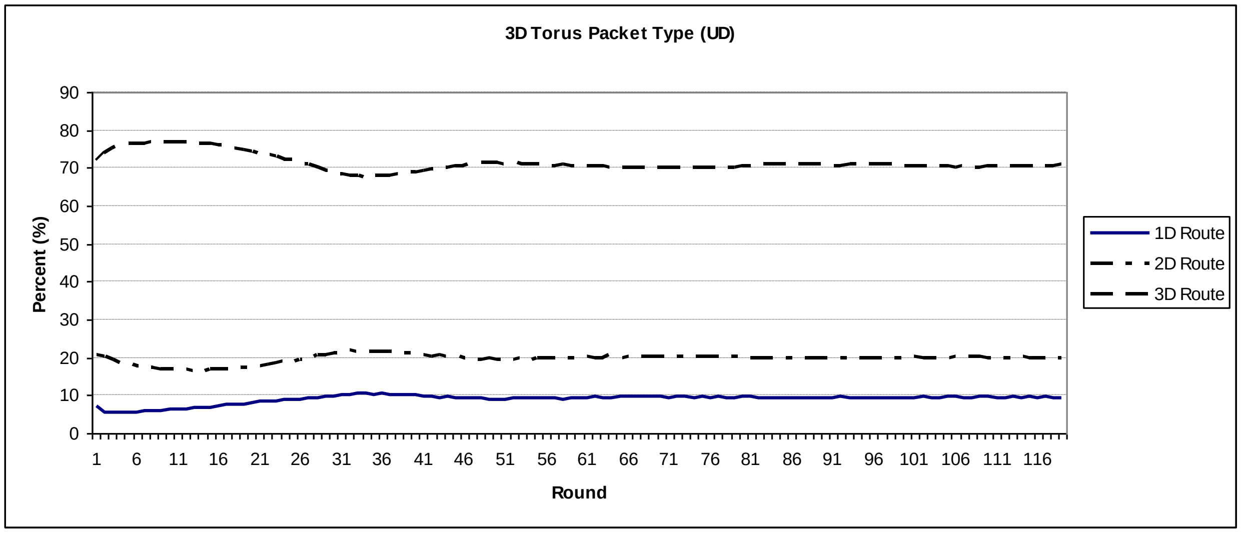


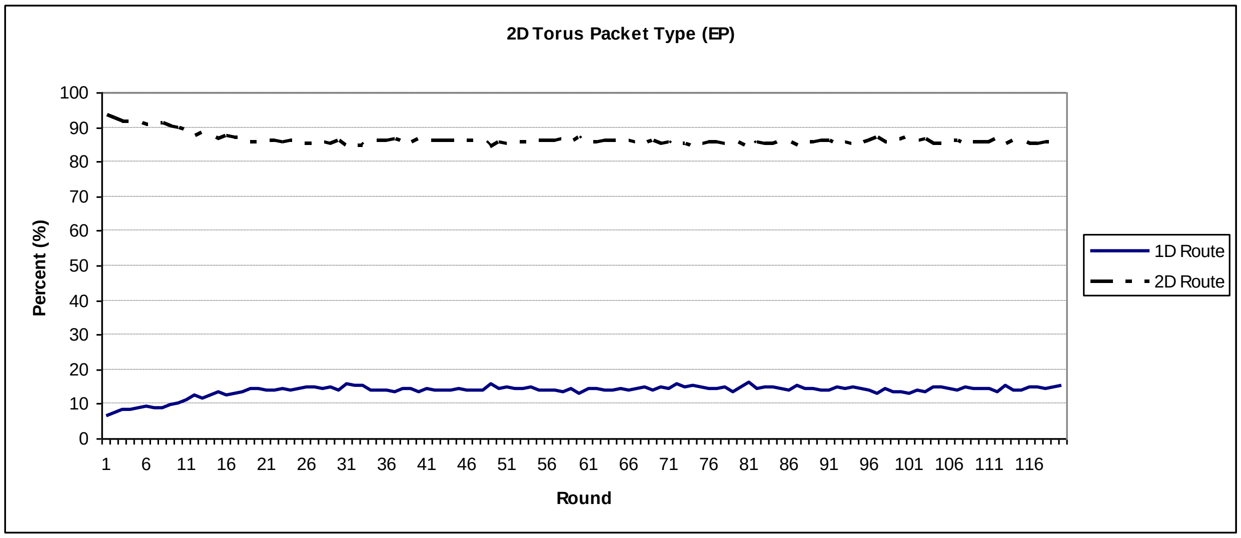


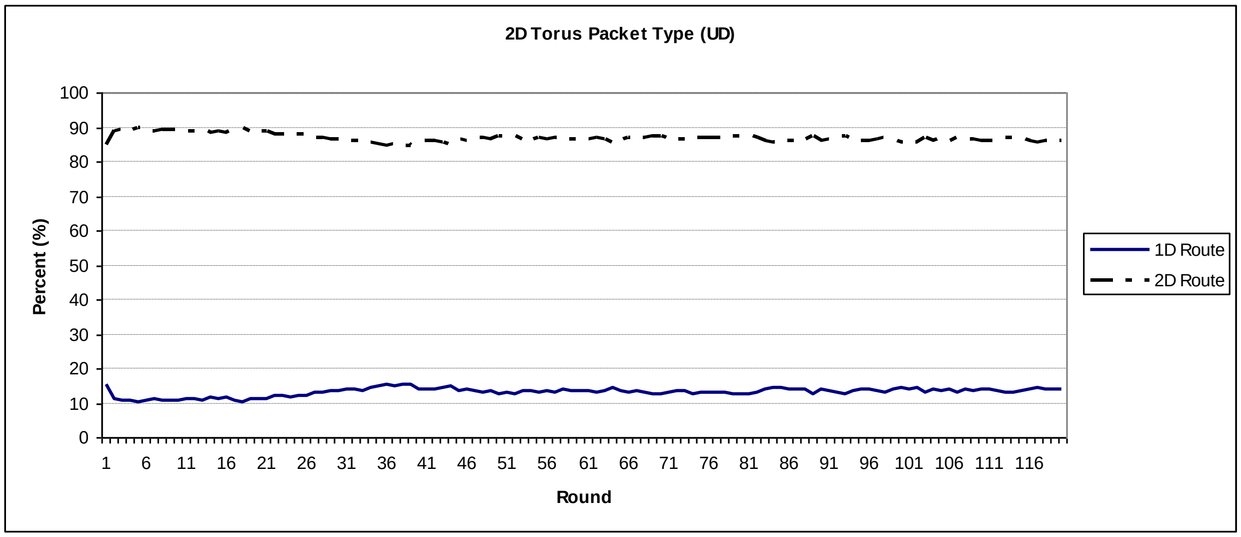



4D Torus Packet Type (UD)
Percent (%)
0
10
20
30
40
50
60
70
Round
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
1D Route
2D Route
3D Route
4D Route


3D Torus Packet Type (EP)
Percent (%)
0
10
20
30
40
50
60
70
80
90
100
Round
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
1D Route
2D Route
3D Route


3D Torus Packet Type (UD)
Percent (%)
0
10
20
30
40
50
60
70
80
90
Round
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
1D Route
2D Route
3D Route





In lower dimensions, the route type does not change as much compared to the stable state. In higher dimensions, there's more oscillation during the rounds.

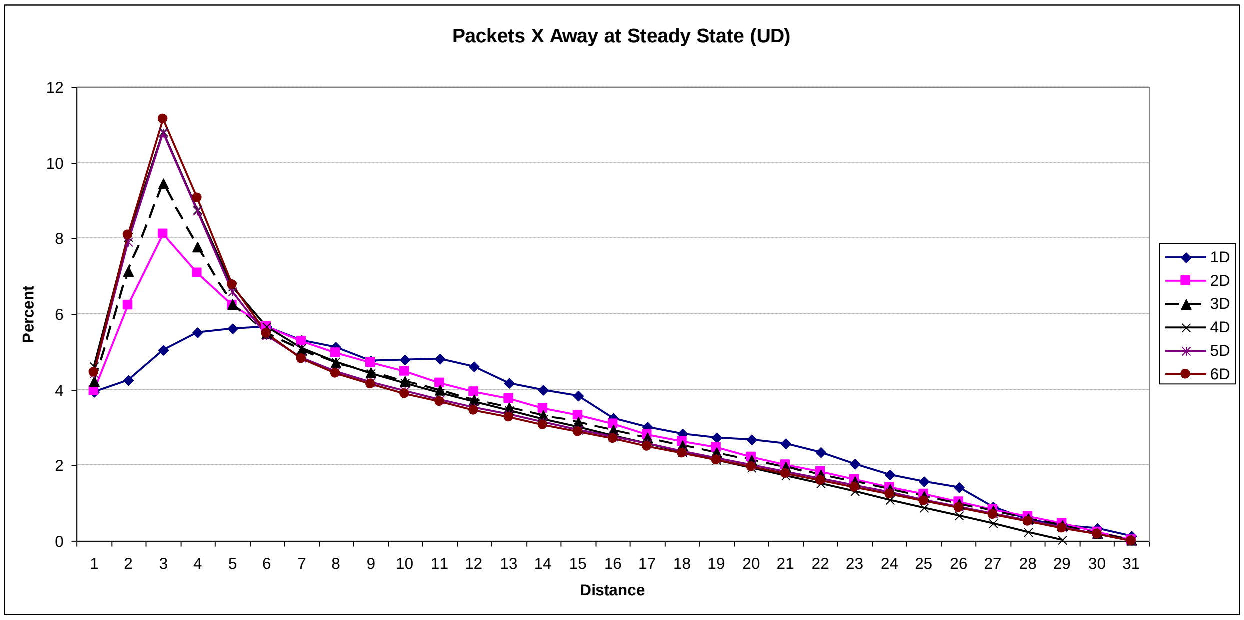
5.6 Steady State Configuration

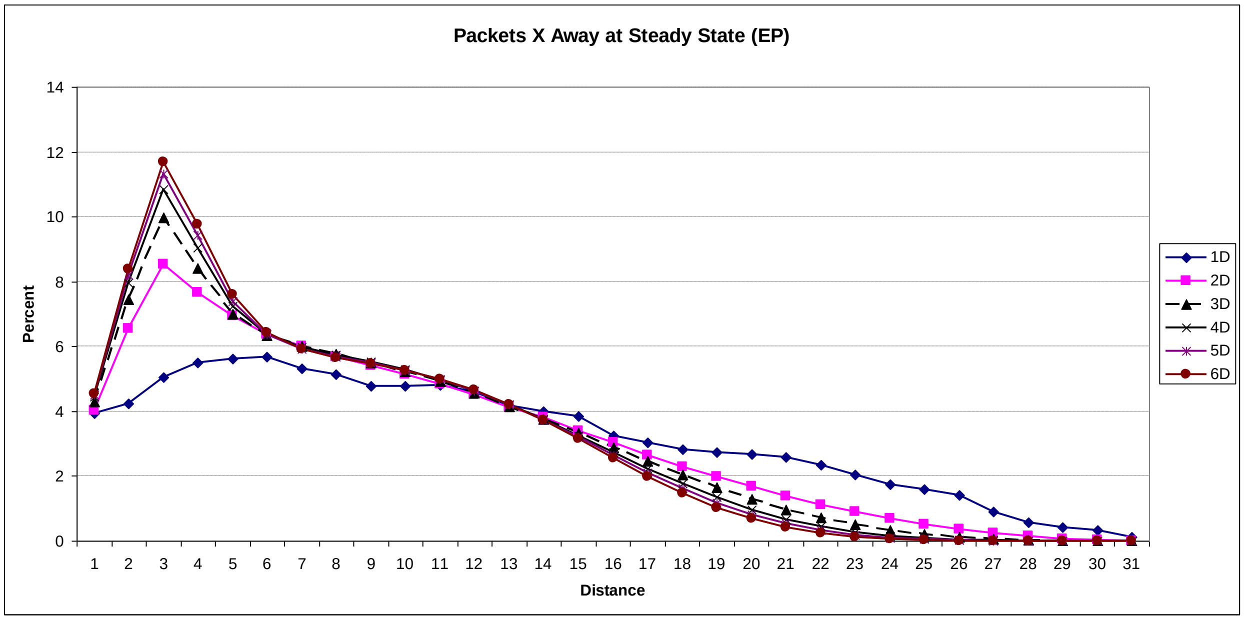
In previous sections, we examined the delivery time, delivery rate, and routing choices. From this data, we gained some understanding of the system and the routing performance.

In this section, we take a look at the configuration of the system in steady state to gain a better understanding of the steady state itself. We examine two important metrics, packets that are X away from destination, and packets that started X away.

5.6.1 Packets X Away

Let's first take a look at packets that are X away at steady state. That is, at steady state, what percent of packets are 1 away, 2 away, 3 away, etc. The data are the average of 40 rounds at steady state.





The chart indicates there are more packets that are closer to its destination than there are packets that are further away. However the plot is not completely linear, with peaks at 3. (In a system with perfect routing, the graph would be completely linear).

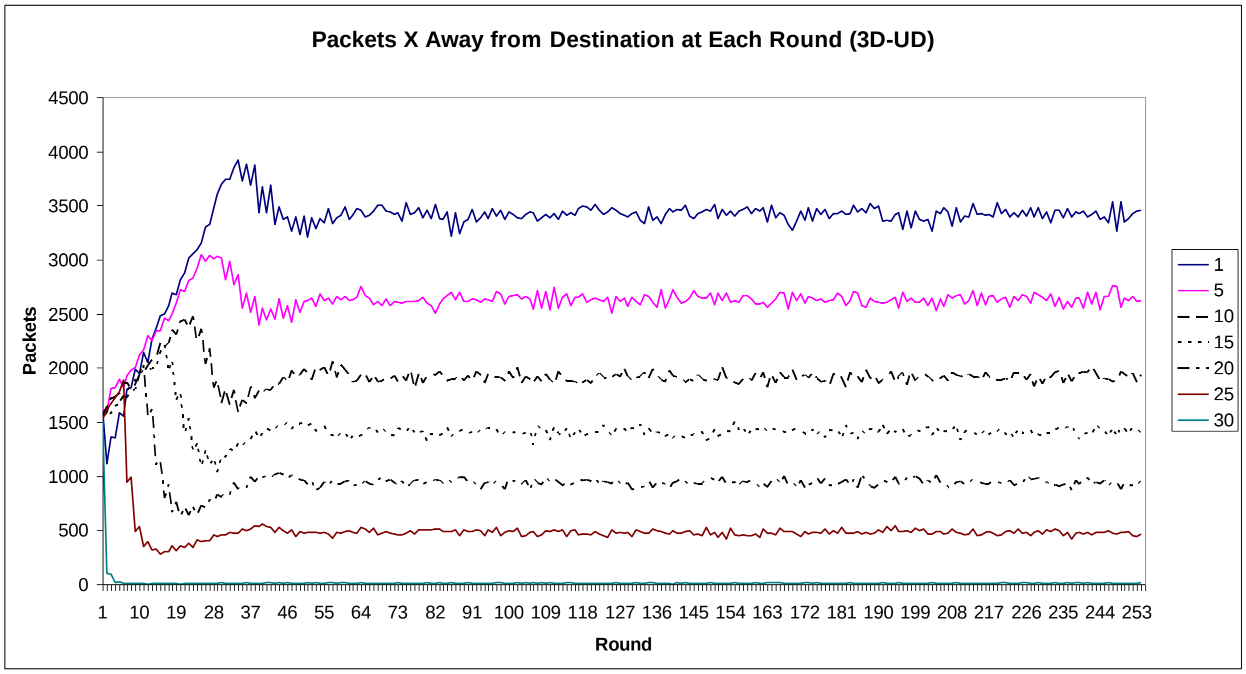
When the system starts out, the number of packets at each distance is the same. This would result in a flat line. The routing is not perfect, so at each time step, some packets move closer, and some packets move further from its destination. Let's say on average, 80% of the packets are moving closer at each time step. At the next time step, 80% of packets at distance *x* move to *x*-1, while 20% move to distance *x*+1. Over time, this would result in packets shifting from higher distance to lower distance, and we have the general shape of the curve sloping downward to the right of higher distance.

As packets get closer however, it is harder to make progress, due to decreased flexibility in its route (fewer dimensions to go). As a result, the percentage of packets moving closer decreases as well. This may explain, at least in part, the increase in slope from 7 to 4. At the minimum distance 1 however, we simply have less packets, because whatever percentage that is moving closer, is delivered, and removed from the system. So packets at 1 away are significantly lower than packets that are 2 away (We don't have distance 0 packets that can travel in the wrong direction to contribute to distance 1 packets. Fewer distance 1 packets also means fewer distance 2 packets since less packets will travel in the wrong direction.) This results in a similar drop in number of packets at 2 from 3. From our simulation, we see that distance 3 packets have the highest count, indicating this is the transition point which packet count for a particular distance starts to drop due to decreasing number of packets at distance 1 and 2. Distance 4 and above, the flexibility increases, more packets moves closer, and the packet count delta decreases until it reaches a stable rate for all higher distances.

5.6.1.1 Packets X Away per Round

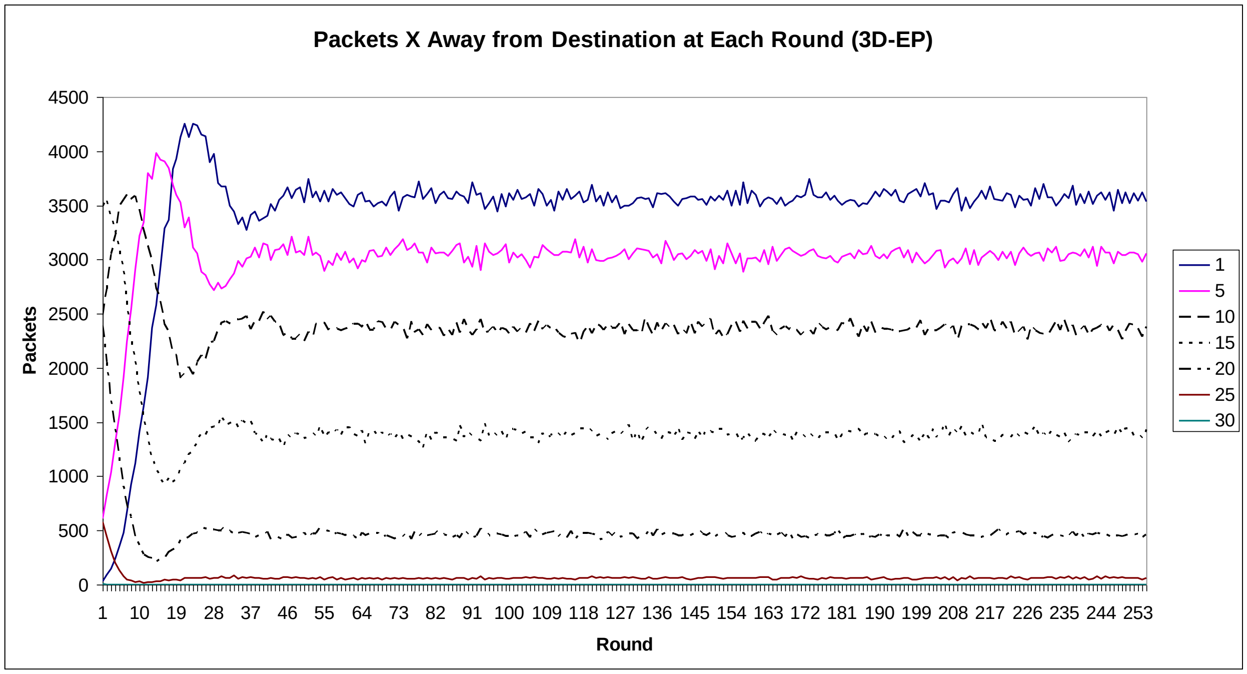
We now know what the system looks like at steady state, how does it get there?

3D-UD



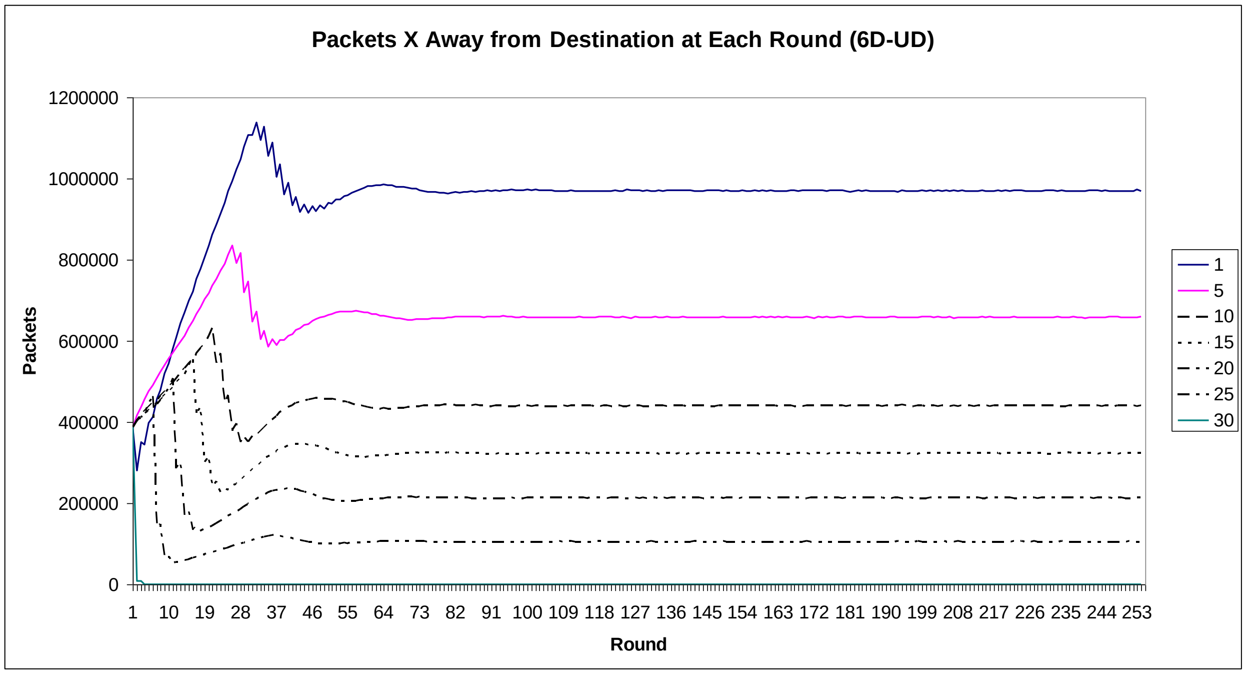
From high to low: [1], [5], … [25], [30].

3D-EP



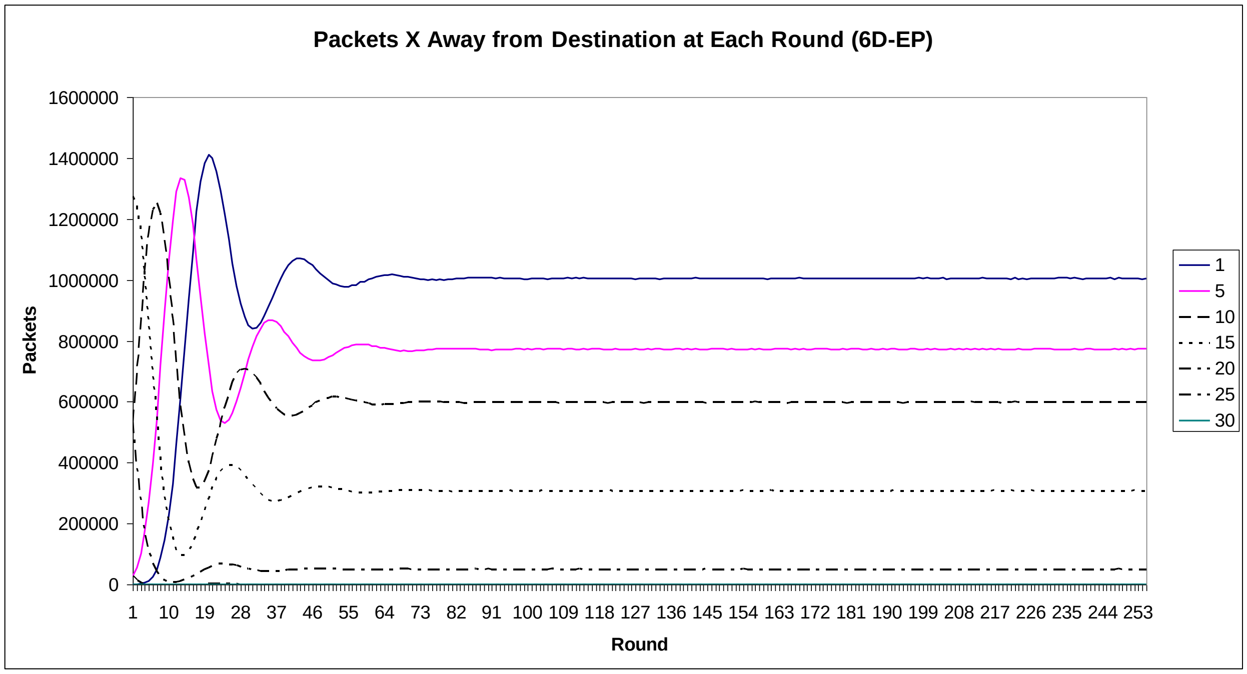
From high to low: [1], [5], … [25], [30].

6D-UD



From high to low: [1], [5], … [25], [30].

6D-EP



From high to low: [1], [5], … [25], [30].

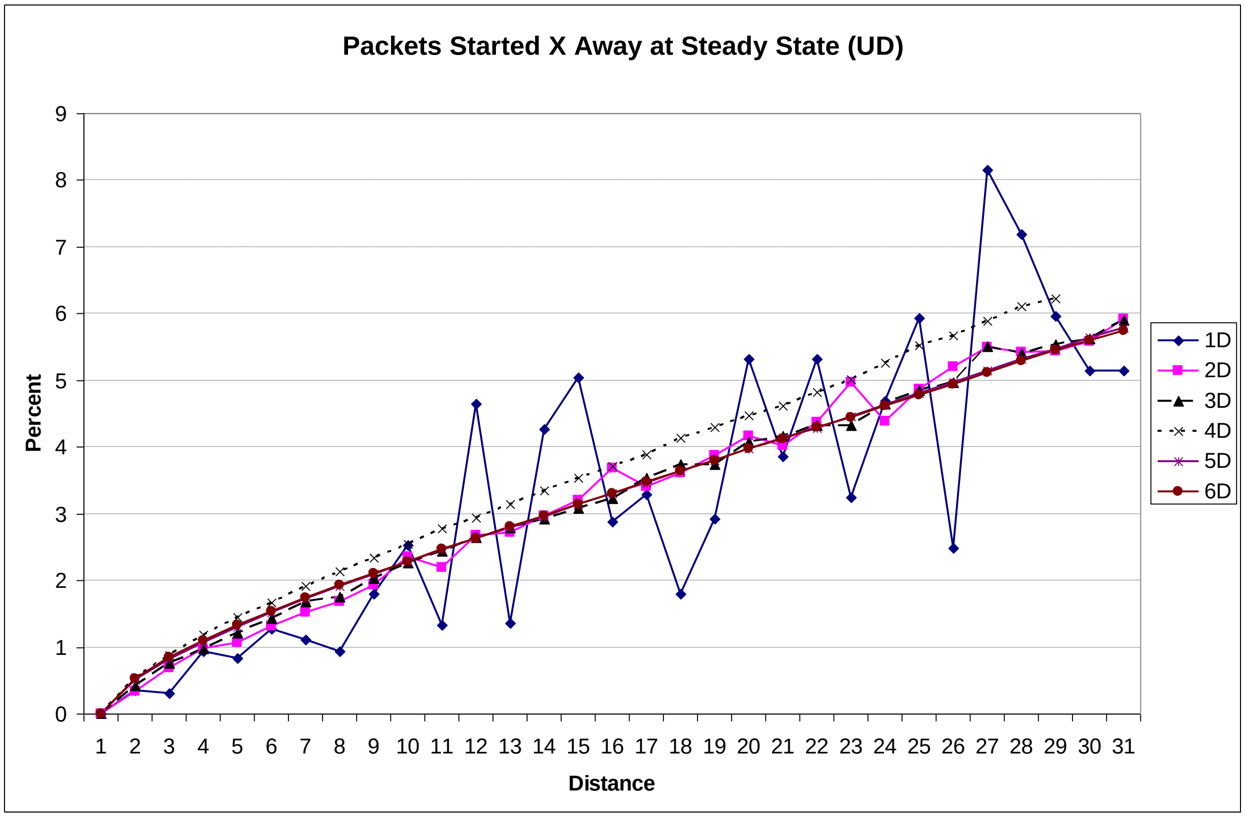
As we see, there are some oscillations initially, but the type stabilizes pretty fast just like the delivery rate, and does not change after that.

5.6.2 Packets Started X Away

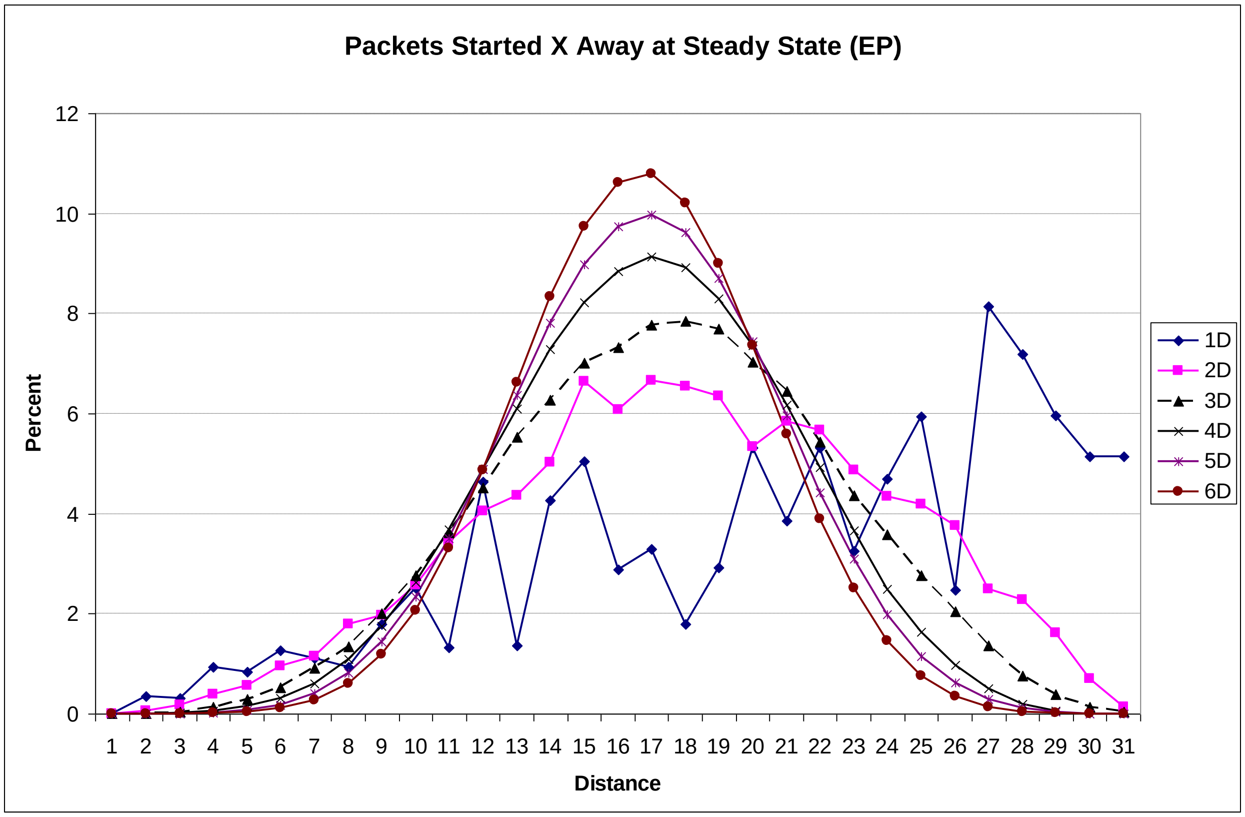
Let's take a look at the slightly different metrics of packets started X away at steady state. That is, how many packets were started 1 away, how many packets were started 2 away.

Naturally, we expect there will be fewer packets in the system that started 1 away than 15 away, as short packets would be delivered while longer packets need longer to be delivered. Therefore we expect the system would contain more packets which started further away than started closer.

Here are the charts. The data are the average of 40 rounds at steady state.



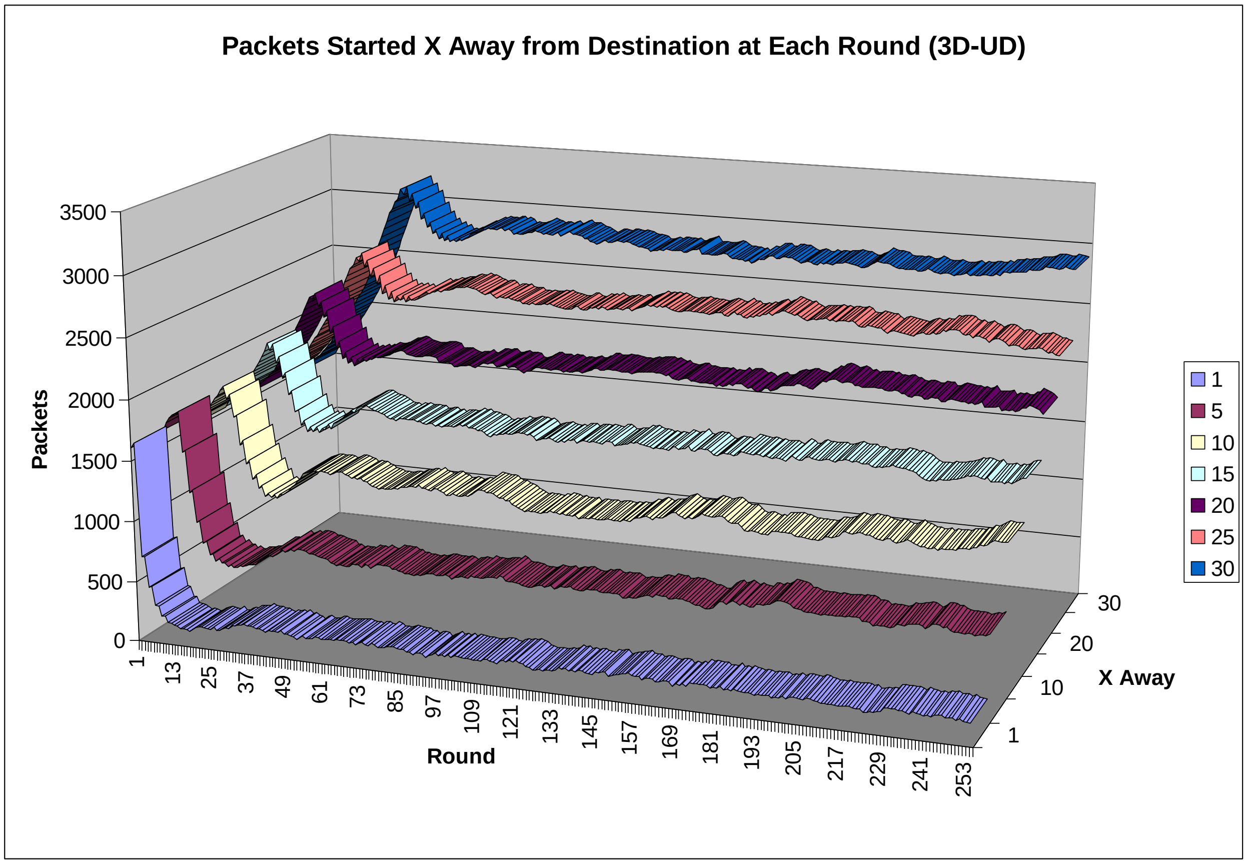
The chart indicates there are more packets that started further away in the system than there are packets that started closer to its destination. This makes perfect sense as packets that started closer got delivered faster, so over time, the system is left with more packets that started further than packets that started closer.



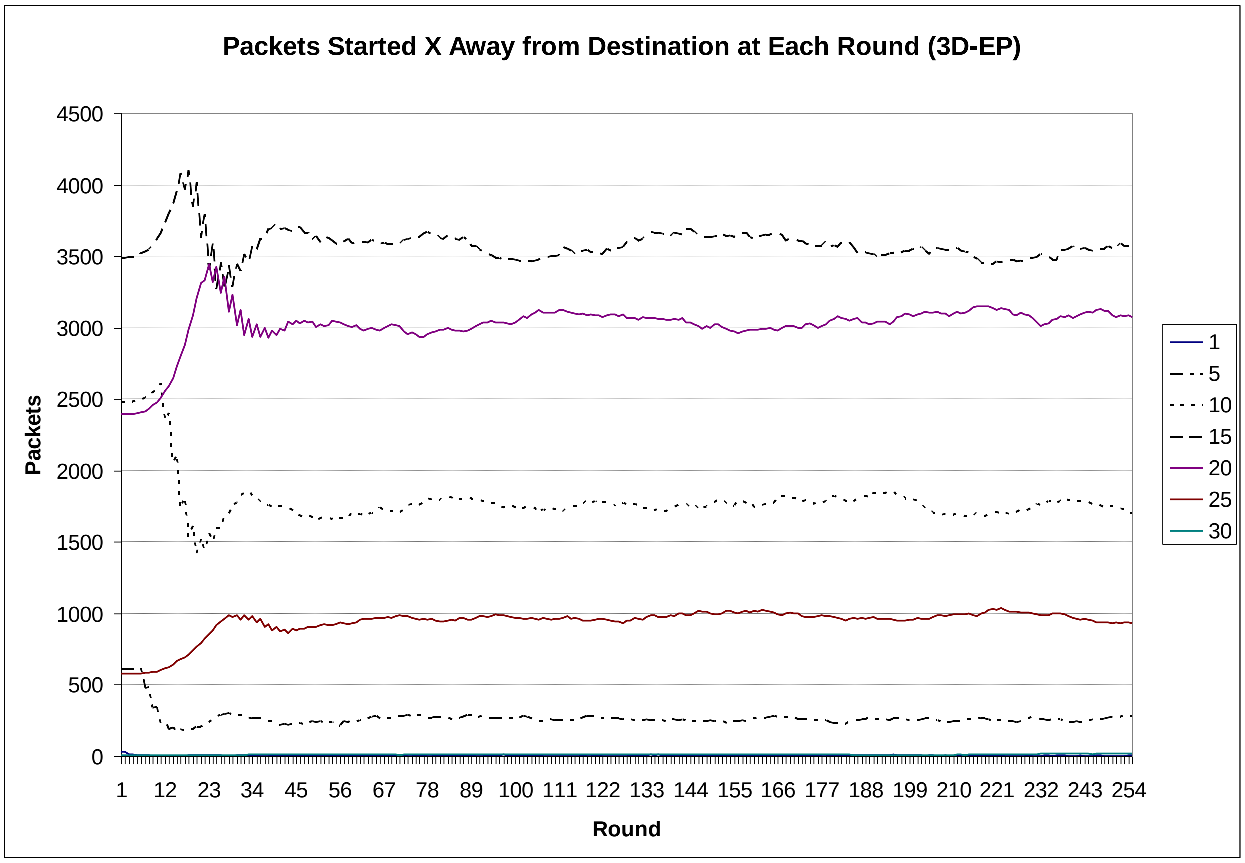
With EP packet generation, the chart looks quite different. It no longer shows a linear increase for packets with higher distance than lower distance like the UD, except in 1D (in 1D, UD and EP are the same). In higher dimensions, most of the packets have average distance, matching the packet generation model. Here, we see the tight distribution of packet generator, with just a small shift upward for longer packets staying the system longer.

5.6.2.1 Packets Started X Away per Round

Let us now take a look at the same data but by round, to observe how it changed over time. For the UD graph, we show a 3D chart, as it shows the data fairly well. For the EP chart, we show a 2D chart, as the data is hard to visualize in 3D.

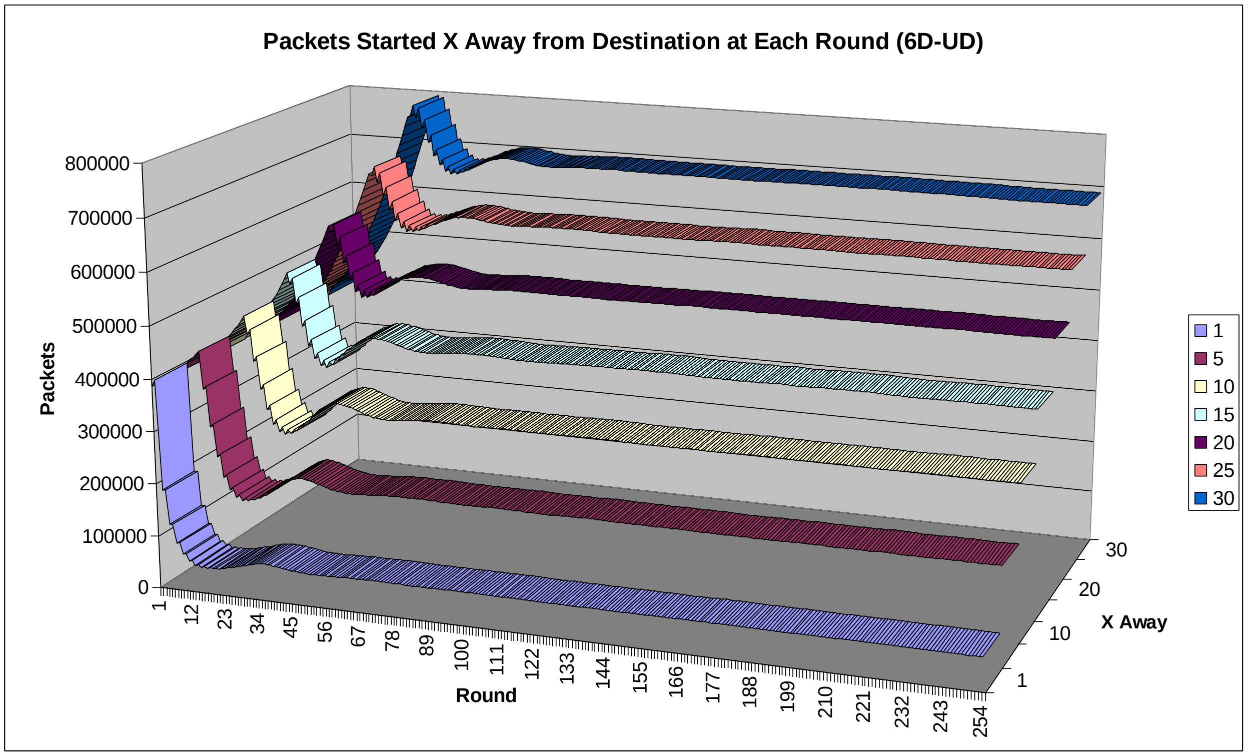


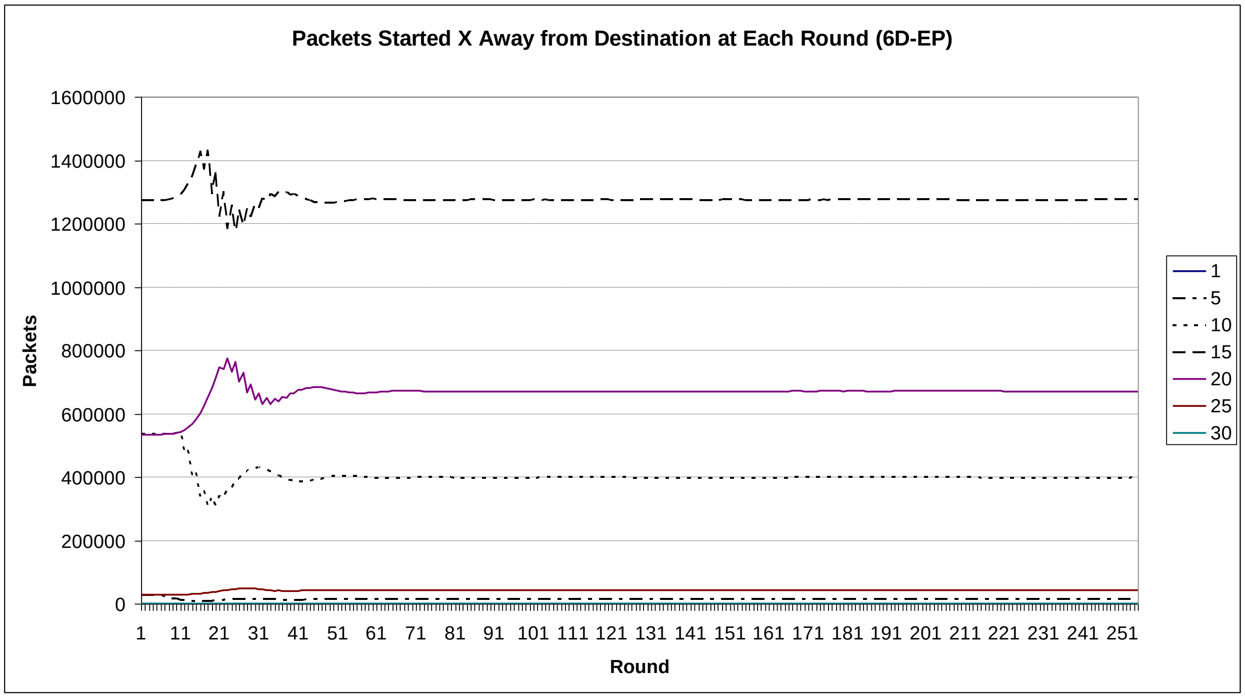
Packets Started X Away from Destination at Each Round (3D-UD)
Packets
Round
X Away
1
5
10
15
20
25
30



Packets Started X Away from Destination at Each Round (3D-EP)
Packets
Round
1
5
10
15
20
25
30

From high to low: [15], [20], [10], [25], [5], [30], [1].





From high to low: [15], [20], [10], [25], [5], [30], [1].

As we can see, similar to the packets X away data, there are some oscillations initially, but the system quickly gets to the stable state, and does not change after that.

5.7 2D Configuration

We did some calculations on the packet configuration of a node on the 2D torus system to help us understand the system better. We only analyzed 2D system to this level of detail.

5.7.1 First Choice Configuration

Each node of the 2D system has 4 packets. Each packet has a 1st choice direction, and a 2nd choice direction perpendicular to the first choice. Looking only at the first choice of the 4 packets, how many possible configurations are there? What are the probabilities of each configuration, if every packet is independent?

Table 1 lists all the possible configurations, and their probabilistic distribution assuming packets are independent (as they are on the first time step). In calculating the number of configurations, specific directions are not relevant and are grouped together. For example, all packets having ↑ as their 1st preference is the same as all packets preferring ←, etc. Directions are only relevant with respect to one another, the actual direction is not important. In other words, we categorize configuration as the same if they are equivalent by reflection or rotation.

Table 1:

	Direction	***Description***	***Prob***
1	↑↑↑↑	All packets have the same first choice direction.	1/64
2	↑↑↓↓	Two packets have the opposite first choice direction of the other two.	3/64
3	↑↑↑↓	One packet has the opposite first choice direction of the other three.	4/64
4	↑↑→→	Two packets have the same first choice direction, perpendicular to the other two.	6/64
5	↑↓←→	Every packet has a different first choice direction.	6/64
6	↑↑←→	Two packets have opposite first choice directions, perpendicular to the direction of the other two.	12/64
7	↑↑↑→	One packet has a first choice direction perpendicular to the other three.	8/64
8	↑↑←↓	Only one packet has same first choice direction as another packet, another has the opposite 1st choice, and the last is perpendicular	24/64

The 4 packets' first and 2nd choices belong to one of these configurations. In a completely independent system, the expected probability of a particular configuration is given above. It should be intuitive to see that the probability of all packets having the same 1st choice preference is a lot smaller than all packets having different 1st choice preferences.

We will show how each of these probabilities are calculated. In the following analysis, all directions are taken with respect to the 1st packet's direction.

1. ↑ ↑ ↑ ↑ (All packets have the same first choice direction)

The 2nd packet with probability 1/4 picks the same direction as the 1st packet. Similarly for the 3rd and 4th packet.

Therefore, 1 x 1/4 x 1/4 x 1/4 = 1/64.

2. ↑ ↑ ↓ ↓ (Two packets have the opposite first choice direction of the other two)

The 2nd packet with probability 1/4 has the same 1st choice as the 1st packet. The 3rd and 4th packets, with probability 1/4 each, have the opposite 1st choice. 1 x 1/4 x 1/4 x 1/4 = 1/64. It may seem at this point that the same logic can be applied to all of the above configurations, but this is clearly incorrect, as we only have 8 configurations, not 64. The sum of the probability of all the configurations must be 1.

The key is, not only is the relative direction important, but the ordering of the packets is also important. ↑ ↓ ↑ ↓ is the same configuration as ↑ ↑ ↓ ↓, yet it is not included in the above calculation. We must count the number of distinct configurations.

In counting the number of configurations, remember that specific directions and the orderings are not important. Two configurations are the same if rotation or reflection will yield one another. Only the relative directions with respect to one another and its orderings are important. For example, ↑ ↑ ↓ ↓ and ↓ ↓ ↑ ↑ are the same configuration because one can be obtained from the other by rotating rotation. However, ↑ ↓ ↑ ↓ is a different configuration, because the relative ordering is different, and you can't get this one by either rotation or reflection.

With that in mind, we count the number of configurations. They are:

↑ ↑ ↓ ↓ = ↑ ↓ ↑ ↓ = ↑ ↓ ↓ ↑.

Therefore, the correct probability is what we calculated above, multiply by the number of configurations: 1/64 * 3 = 3/64.

3. ↑ ↑ ↑ ↓ (One packet has the opposite first choice direction of the other three)

Same as #2, but the number of configurations is different.

↑ ↑ ↑ ↓ = ↑ ↓ ↑ ↑ = ↑ ↑ ↓ ↑ = ↓ ↑ ↑ ↑

Therefore, 4 * (1/4 * 1/4 * 1/4) = 4/64

4. ↑ ↑ → → (Two packets have the same first choice direction, perpendicular to the other two)

The 2nd packet has probability 1/4 of matching the first, but the 3rd packet has probability 1/2, as either of the perpendicular directions works, and the 4th packet has probability 1/4 of matching the 3rd. Therefore, 1/4 * 1/2 * 1/4 = 2/64.

The number of configurations is 3: ↑ ↑ → → = ↑ → ↑ → = ↑ → → ↑.

Therefore, probability is 3 * 2/64 = 6/64

5. ↑ ↓ ← → (Every packet has a different first choice direction)

The 2nd packet has probability 3/4 to have its first choice not match the first packet's first choice. Similarly, the 3rd packet has probability 2/4 of not matching the 1st and 2nd. The last packet has probability 1/4.

Therefore, 3/4 * 2/4 * 1/4 = 6/64.

6. ↑ ↑ ← → (Two packets have opposite first choice directions, perpendicular direction of the other two)

The 2nd packet has probability 1/4 to match the first packet. The 3rd packet has probability 1/2 to pick either of the perpendicular directions, and the 4th packet picks the opposite perpendicular with probability 1/4.

Therefore, 1/4 * 1/2 * 1/4 = 2/64, same as #4.

However, the number of configurations is not the same. There are 6 of them instead of 3, since left and right can be distinguished now.

↑ ↑ ← → = ↑ ← ↑ → = ↑ ← → ↑ = ← ↑ ↑ → = ← → ↑ ↑ = ↑ → ↑ ← = ← ↑ → ↑

Therefore, probability is 2/64 * 6 = 12/64

7. ↑ ↑ ↑ → (One packet has a first choice direction perpendicular to the other three)

The 2nd & 3rd packets have probability 1/4 to have its first choice match the first packet. The 4th packet has probability 1/2 for either of the perpendicular directions. Therefore, 1/4 * 1/4 * 1/2 = 2/64.

The number of configurations is 4:

↑ ↑ ↑ → = ↑ → ↑ ↑ = ↑ ↑ → ↑ = → ↑ ↑ ↑

Therefore, probability is 4 * 2/64 = 8/64.

8. ↑ ↑ ← ↓ (Only one packet has the same first choice direction as another packet, and one has the opposite direction)

The 2nd packet has probability 1/4 to match the 1st packet. The 3rd packet has probability 1/2 for either of the perpendicular directions. 4th packet has 1/4 probability to pick the remaining edge. Therefore, it is 1/4 x 1/2 x 1/4 = 2/64.

The number of equivalent configurations is fairly large, as this particular configuration offers the maximum amount of “flexibility”.

↑ ↑ ← ↓ = ↑ ↑ ↓ ← = ↑ ← ↑ ↓ = ↑ ← ↓ ↑ = ↑ ↓ ← ↑ = ↑ ↓ ↑ ← =
← ↑ ↓ ↑ = ← ↓ ↑ ↑ = ← ↑ ↑ ↓ = ↓ ← ↑ ↑ = ↓ ↑ ← ↑ = ↓ ↑ ↑ ←

Therefore, the probability is 2/64 * 12 = 24/64.

As a sanity check, the sum of all the probabilities is

(1 + 3 + 4 + 6 + 6 + 12 + 8 + 24) / 64 = 1.

5.7.2 Full Configuration

In a 2D system, every packet has a 1st and 2nd choice. By definition, the 2nd choice is perpendicular to the first choice. We will now analyze the probability of the full configuration, first choice plus second choice.

Given a particular 1st choice configuration, not all possible configurations are valid for the 2nd choice configuration, since the 1st choice and 2nd choice are not independent, but perpendicular to one another. For example, if all packets have the same direction as first choice, then their 2nd choice cannot be one in each different direction, since two of those directions are not perpendicular to the first choice. Note that for a particular 1st choice configuration, the same configuration (rotated and perhaps flipped) is always a valid second choice configuration.

Given these restriction, the following table lists the possible 2nd choice configuration for each 1st choice configuration, and the probability of each of them, if all packets are independent.

Table 2 (row = 1st choice, column = 2nd choice)

1st/2nd	↑↑↑↑	↑↑↓↓	↑↑↑↓	↑↑→→	↑↓←→	↑↑←→	↑↑↑→	↑↑←↓
↑↑↑↑	1/512	3/512	4/512					
↑↑↓↓	3/512	9/512	12/512					
↑↑↑↓	4/512	12/512	16/512					
↑↑→→				12/512	12/512	24/512		
↑↓←→				12/512	12/512	24/512		
↑↑←→				24/512	24/512	48/512		
↑↑↑→							16/512	48/512
↑↑←↓							48/512	144/512

Each row and each column sums up to the probability calculated in Table 1. In all, there are 22 unique 1st + 2nd choice configurations in the 2D system.

Let us now show how these probabilities are calculated. An interesting observation can be made from Table 2. The 8 configurations form 3 distinct groups with respect to first and second choices. If a packet has a specific 1st choice configuration, then its 2nd choice configuration only comes from the group it is in.

1. ↑ ↑ ↑ ↑, ↑ ↑ ↓ ↓, ↑ ↑ ↑ ↓ (#1, #2, #3)

If the 1st choice configuration is ↑ , then the 2nd choice configuration must be either ← or →. Rotate this 90 degrees and it becomes ↑ or ↓ . Therefore, the only possible 2nd choice configuration for the above 3 configurations are those having opposite choices of each other, namely #1, #2, and #3. (All other configurations have choices that are perpendicular to one another, therefore are not valid 2nd choice configuration for #1, #2, and #3.)

#2 and #3 have more than one equivalent configuration, as we have shown. Therefore, we expect the probability of 2nd choice having configuration #2 and #3 to occur more often than 2nd choice having configuration #1. Does the probability of these happening correspond to the number of equivalent configurations between #1, #2, and #3? If so, then calculating this is relatively simple.

However, as we will see later, not all possible equivalent configurations are valid for a specific instance of the 1st choice configuration. Therefore, a different approach is needed.

1. 1st choice configuration is ↑ ↑ ↑ ↑

- ↑ ↑ ↑ ↑ **x** ↑ ↑ ↑ ↑

  If the 1st choice configuration is ↑ ↑ ↑ ↑, then the 2nd choice configuration is either ←←←←, or →→→→. Both of them are configuration #1.

  The probability of either of them happening is 1/2 x 1/2 x 1/2 x 1/2 = 1/16, therefore the probability of the 2nd choice configuration is 2 x 1/16 = 1/8.

- ↑ ↑ ↑ ↑ **x** ↑ ↑ ↓ ↓

  2nd choice configurations are either

  ← ← → →, → → ← ←,
  ← → ← →, → ← → ←,
  ← → → ←, → ← ← →.

  Each of them happens with probability 1/16, as calculated above.

  Therefore, the probability of the 2nd choice configuration is 6 * 1/16 = 3/8.

- ↑ ↑ ↑ ↑ **x** ↑ ↑ ↑ ↓

2nd choice configurations are either

← ← ← →, → → → ←
← → ← ←, → ← → →
← ← → ←, → → ← →
→ ← ← ←, ← → → →

Each of them happens with probability 1/16, as calculated above.

Therefore, the probability is 8 * 1/16 = 4/8.

2. **1st choice configuration is ↑ ↑ ↓ ↓**

- ↑ ↑ ↓ ↓ **x** ↑ ↑ ↑ ↑

  The 2nd choice configuration is either ← ← ← ←, or → → → →. Both of them are configuration #1.

  The probability of either of them happening is 1/2 x 1/2 x 1/2 x 1/2 = 1/16, therefore the probability of the 2nd choice configuration is 2 x 1/16 = 1/8.

- ↑ ↑ ↓ ↓ **x** ↑ ↑ ↓ ↓

  The analysis is identical to when the 1st choice is ↑ ↑ ↑ ↑, since all equivalent configurations are valid. Therefore, the probability of the 2nd choice configuration is 6 * 1/16 = 3/8.

- ↑ ↑ ↓ ↓ **x** ↑ ↑ ↑ ↓

  Again, all equivalent configurations are valid, therefore the probability is 8 * 1/16 = 4/8.

3. **1st choice configuration is ↑ ↑ ↑ ↓**

- ↑ ↑ ↑ ↓ **x** ↑ ↑ ↑ ↑

  Same as analysis for #2 above. Therefore the probability of the 2nd choice configuration is 2 x 1/16 = 1/8.

- ↑ ↑ ↑ ↓ **x** ↑ ↑ ↓ ↓

  The analysis is identical to when the 1st choice is ↑ ↑ ↑ ↑, since all equivalent configurations are valid. Therefore the probability of the 2nd choice configuration is 6 * 1/16 = 3/8.

- ↑↑↑↓ **x** ↑↑↑↓

  Again, all equivalent configurations are valid, therefore, probability is 8 * 1/16 = 4/8.

It just happened that in group #1, all the equivalent configurations are possible for each of the configurations. Regardless of whether the 1st choice is ↑↑↑↑, ↑↑↓↓, or ↑↑↑↓, the 2nd choice analysis is the same. Therefore, the probability of the 1st+2nd choice is probability of the 1st choice * probability of the 2nd choice. The result is what we have in Table 2.

2. ↑↑→→, ↑↓←→, ↑↑←→ (#4, #5, #6)

For each case, we will list all the instance of the configuration of the 2nd choice. Since each instance of the configuration occurs with probability 1/16, the probability of the configuration is just # of instance * 1/16.

- **↑↑→→ x ↑↑→→ (4) =** ←←↑↑, ←←↓↓, →→↑↑, →→↓↓
- **↑↑→→ x ↑↓←→ (4) =** ←→↑↓, ←→↓↑, →←↑↓, →←↓↑

  ↑↑→→ x ↑↑←→ (8) = ←→↑↑, ←→↓↓, →←↑↑, →←↓↓
  ←←↑↓, ←←↓↑, →→↑↓, →→↓↑

- **↑↓←→ x ↑↑→→ (4) =** ↑→↑→, ↑←↑←, ↓→↓→, ↓←↓←
- **↑↓←→ x ↑↓←→ (4) =** ↑→↓←, ↑←↓→, ↓→↑←, ↓←↑→

  ↑↓←→ x ↑↑←→ (8) = ↑←↑→, ↑→↑←, ↓←↓→, ↓→↓←
  ↑←↓←, ↑→↓→, ↓←↑←, ↓→↑→

- **↑↑←→ x ↑↑→→ (4) =** ↑→→↑, ↓→→↑, ↑←←↑, ↓←←↓
- **↑↑←→ x ↑↓←→ (4) =** ↑→←↓, ↑←→↓, ↓→←↑, ↓←→↑

  ↑↑←→ x ↑↑←→ (8) = ↑←→↑, ↑→←↑, ↓←→↑, ↓←→↓
  ↑→→↓, ↑←←↓, ↓→→↑, ↓←←↑

The number of instances * 1/6 * prob[1st choice] gives the result in Table 2.

3. ↑↑↑→, ↑↑←↓ (#7, #8)

- **↑↑↑→ x ↑↑↑→ (4) =** ←←←↑, ←←←↓, →→→↑, →→→↓
- **↑↑↑→ x ↑↑←↓ (12) =**
  ←←→↑, ←←→↓, ←→→↑, ←→→↓, ←→←↑, ←→←↓
  →←←↑, →←←↓, →←→↑, →←→↓, →→←↑, →→←↓

- **↑ ↑ ← ↓ x ↑ ↑ ↑ → (4) =** ← ← ← ↑, ← ← ← ↓, → → → ↑, → → → ↓.

  This is a clear example where not all the possible equivalent configurations of ↑ ↑ ↑ → are valid, for example, ↑ → ↑ ↑ is not a valid 2nd choice for ↑ ↑ ← ↓.

- **↑ ↑ ← ↓ x ↑ ↑ ← ↓ (12) =**
  ← ← → ↑, ← ← → ↓, ← → → ↑, ← → → ↓, ← → ← ↑, ← → ← ↓
  → ← ← ↑, → ← ← ↓, → ← → ↑, → ← → ↓, → → ← ↑, → → ← ↓

  (Same as ↑ ↑ ↑ → x ↑ ↑ ← ↓).

The above calculations confirm the result in Table 2.

5.7.3 Comparing to our System

Given the probability of the 1st choice configuration, and the 1st + 2nd choice configuration in an independent system, we want to compare it with the result from our simulation to see how they differ. To enable this analysis, we updated our simulation to count the number of occurrences for each configuration during the simulation.

For each node, we look at the 1st choice configuration of all 4 packets, and assign it configuration 1 to 8 based on their first choices. Next, we look at the 2nd choice configuration, and again assign it configuration 1-8 based on their second choices. We then count the occurrences of 1st choice, and 1st choice + 2nd choice configuration (64 total). Even though the system tracks all 64 configurations, we don't expect to see data in all but 22 of them, as not all 1st choice and 2nd choice configuration are valid combinations.

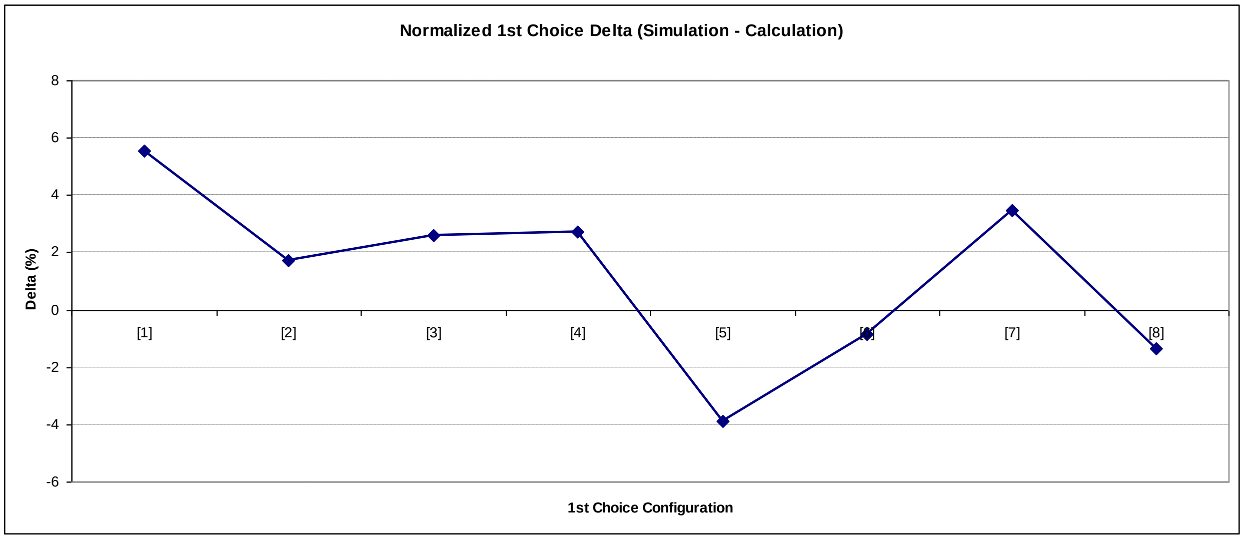
Here are the results of the simulation on a 2D torus of size 30. The uniform distance to destination model is used. Statistics are collected after 120 rounds, for a total of 1000 rounds.

We first take a look at the 1st choice configuration:

Config1	***Count***	***Calculation***	***Delta***	***Normalized Delta %***
[1]	13076	0.015625	686.9375	5.544709295
[2]	37806	0.046875	638.8125	1.718753941
[3]	50851	0.0625	1294.75	2.612687602
[4]	76351	0.09375	2016.625	2.712910413
[5]	71460	0.09375	-2874.375	-3.86681801
[6]	147424	0.1875	-1244.75	-0.837264052
[7]	102557	0.125	3444.5	3.475343675
[8]	293375	0.375	-3962.5	-1.332660697

Count is the occurrences of the specific configuration during the simulation. *Calculation* is the percentage the configuration occurs in our calculation. The delta and the normalized percentage ((simulation - calculation) / calculation) shows how our system differ from the calculated independent result.

Here's a plot of the data:

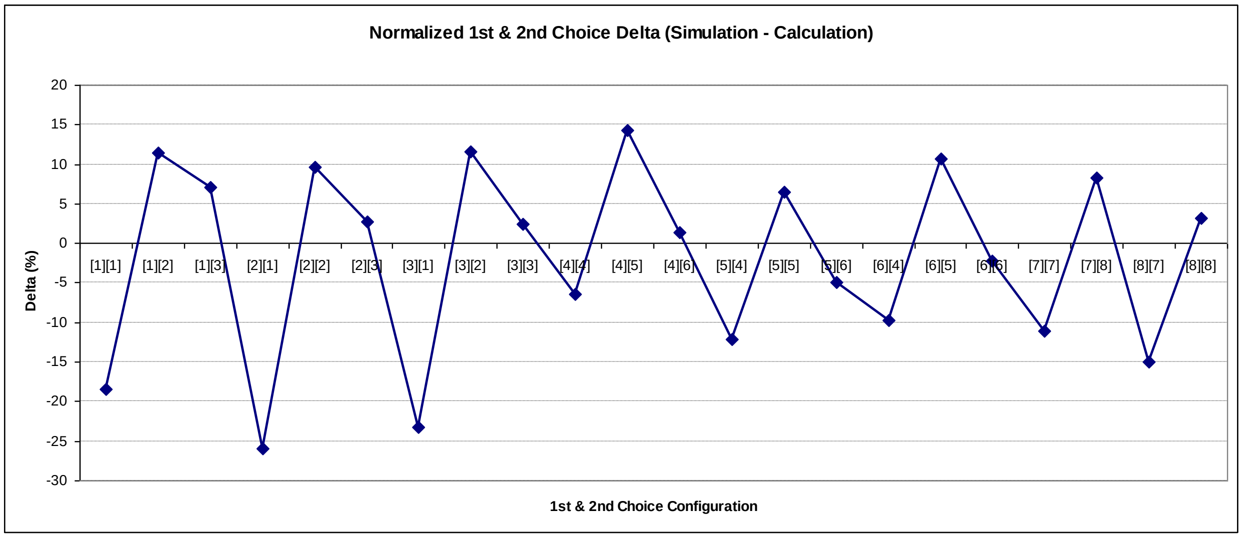


Our system differs from the calculated result by about +6 to -4 percent. The delta and percentage is normalized by the expected number of occurrences of each configuration, because some configurations occur much more often than others, an absolute delta is not very useful.

We now take a look at the 1st and 2nd choice configuration differences. Here are the data:

Config2	*Count*	*Calculated*	*Delta*	*Normalized Delta %*
[1][1]	1262	0.001953125	-286.6328125	-18.50876529
[1][2]	5180	0.005859375	534.1015625	11.4961954
[1][3]	6634	0.0078125	439.46875	7.094463362
[2][1]	3441	0.005859375	-1204.898438	-25.9346702
[2][2]	15286	0.017578125	1348.304688	9.673799414
[2][3]	19079	0.0234375	495.40625	2.665825871
[3][1]	4753	0.0078125	-1441.53125	-23.27103039
[3][2]	20726	0.0234375	2142.40625	11.52848194
[3][3]	25372	0.03125	593.875	2.396771346
[4][4]	17397	0.0234375	-1186.59375	-6.385168369
[4][5]	21250	0.0234375	2666.40625	14.34817337
[4][6]	37704	0.046875	536.8125	1.444318325
[5][4]	16325	0.0234375	-2258.59375	-12.1536974
[5][5]	19795	0.0234375	1211.40625	6.518686678
[5][6]	35340	0.046875	-1827.1875	-4.91613066
[6][4]	33571	0.046875	-3596.1875	-9.675705217
[6][5]	41115	0.046875	3947.8125	10.62176819
[6][6]	72738	0.09375	-1596.375	-2.147559591
[7][7]	22032	0.03125	-2746.125	-11.08286039
[7][8]	80525	0.09375	6190.625	8.328078362
[8][7]	63190	0.09375	-11144.375	-14.99222264
[8][8]	230185	0.28125	7181.875	3.220526618

Here's a plot of the data:



From the data, we see our system differs from the calculated result by about -25 to 15 percent. This is larger than the 1st choice only configuration differences. There are some patterns to be observed. For example, for configurations of type [1] for 1st choice, the shape of the chart is very similar to configurations of type [2] as 1st choice, and [3] as 1st choice. I.e. the 2nd choice configuration [1] is negative while 2nd choice configurations [2] and [3] are positive, and [2] is higher than [3]. Similar patterns can be observed for all the different 1st choice configurations.

This interesting pattern deserves further investigation, beyond this work. It is worth considering differences with the independent system, as the independent system is relatively easy to analyze, and we can prove expected linear delivery time for it.

6 CONCLUSION

We've presented our algorithm and analyzed the results from various simulation runs in detail. From the simulation results, we can see that packets are being delivered in linear O(n) time with respect to their initial distance. We see that the system reaches a steady state fairly quickly, even if it starts from a state where many collisions guaranteed, and stays at the steady state. We see that not only does initial distance affect delivery time, but the initial distance vector does too. Our simulation showed that delivery time is faster in higher dimensions, indicating the algorithm performs better with more dimensions.

Our simulation result also showed that in the system, the majority of the packets, between 80 to 85%, are moving closer to their destination at each time step. Furthermore, packets retain their route flexibility. There are consistently more packets which have i dimensions to move toward their destination than i-1 dimensions, for all i and for all dimensions. These all contributed to the system throughput, and delivery time.

Through our simulation and analysis, we have shown that the simple greedy hot potato routing algorithm performs well, delivering packets in linear time, and works better in higher dimensions. We hope our results will help future studies of the system where theoretical bounds, rather than empirical bounds, can be found.

REFERENCE

[Bar64] P. Baran. On distributed communications networks. *IEEE Transactions on Communications*, 12:1-9, 1964.

[Smi81] B. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Proc. (SPIE) Real Time Signal Processing IV*, pages 241-248, 1981.

[Hil85] W.D. Hillis. *The Connection Machine*. MIT press, 1985.

[BH85] A. Borodin and J.E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130-145, 1985.

[Max89] N.F. Maxemchuk. Comparison of deflection and store and forward techniques in the Manhattan street and shuffle exchange networks. In *IEEE INFOCOM*, pages 800-809, 1989.

[Sei92] C.L. Seitz. The Caltech Mosaic C: An experimental, fine-grain multicomputer. In *4th symp. on Parallel Algorithms and Architectures*, San Diego, June 1992. ACM.

[BHS94] A. Ben-Dor, S. Halevi, and A. Schuster. Potential function analysis of greedy hot-potato routing. In *Proc. 13th symp. on Principles of Distributed Computing*, pp. 225-234, Los Angeles, August 1994. ACM.

[BU96] A.Z. Broder and E. Upfal. Dynamic deflection routing on arrays. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pp. 348-355, 1996.

[Fei99] Uriel Feige. Nonmonotonic phenomena in packet routing. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pp. 583-591, 1999.

[BHW00a] Costas Busch, Maurice Herlihy, and Roger Wattenhofer. Randomized greedy hot-potato routing. *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 458-466, 2000.

[BHW00b] Costas Busch, Maurice Herlihy, and Roger Wattenhofer. Hard-Potato routing. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pp. 278-285, 2000.

[BCKV00] Constantinos Bartzis, Ioannis Caragiannis, Christos Kaklamanis, and Ioannis Vergados. Experimental Evaluation of Hot-Potato Routing Algorithms on 2-Dimensional Processor Arrays. *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pp. 877-881, 2000.

[BFU01] Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. A general approach to dynamic packet routing with bounded buffers. *Journal of the ACM, v.48 n.2*, pp. 324-349, 2001.

[BHW01] Costas Busch, Maurice Herlihy, and Roger Wattenhofer. Routing without flow control. *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 11-20, 2001.