Simulation and Analysis of Mesh Interconnection Networks

https://github.com/aadulla/15418_Interconnection_Network_Simulator

Ashwin Adulla, aadulla

Summary: I designed a multi-threaded interconnection network simulator capable of simulating any combination of network topologies, flow control algorithms, and routing algorithms under different message traffic patterns. The simulator provides accurate metrics regarding the average latency, throughput, and transmission speed of a user-configured network as well as time-series data that can be used to analyze the network's performance during the simulation. For this project, I simulated a 10x10 Mesh Interconnection Network with various flow control and routing algorithm configurations, and from my simulations, I was able to draw qualitative insights regarding how different mesh network configurations perform under different message traffic patterns.

Background: In various parallel processing applications where multiple processing and memory elements are required to communicate with each other, interconnection networks serve an important role in ensuring data is transmitted efficiently with minimal latency and maximal throughput. As these parallel systems scale up, such as by increasing the core count within a single processor or by adding more nodes to a supercomputer, properly designed interconnection networks are becoming more and more important to the overall performance of the system.

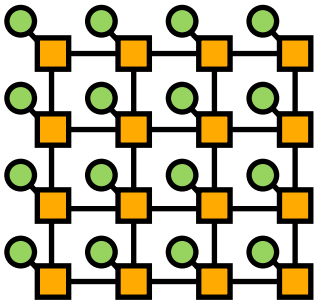

Figure 1: Mesh Interconnection Network

An interconnection network can be described by 3 attributes: a) topology, b) flow control, and c) routing algorithm.

A. The topology defines the physical hardware of the network and how all the nodes (i.e. processors, sensors, etc.) are connected to routers via channels. Figure 1 is an example of a "mesh" interconnection network where the green circles represent the nodes and the orange squares represent the routers. It is important to note that since each node is connected to a single router, a "mesh" interconnection network is also characterized as a "direct" network.

B. The flow control algorithm determines how a message is broken into smaller entities of data, and then how those entities are transmitted along a path from source to destination. In typicl interconnection networks, a message is first broken up into packets, and then those packets are broken up into flits. As shown in Figure 2, there are 3 types of flits: a) head flit, b) data flit, c) tail flit. The head flit contains the routing information (i.e. source and destination node ids) of the packet; the data flits compose the payload body of the packet and contain the actual data being transmitted; the tail flit contains the error-checking/correction information of the packet (i.e. parity, checksum, etc.)

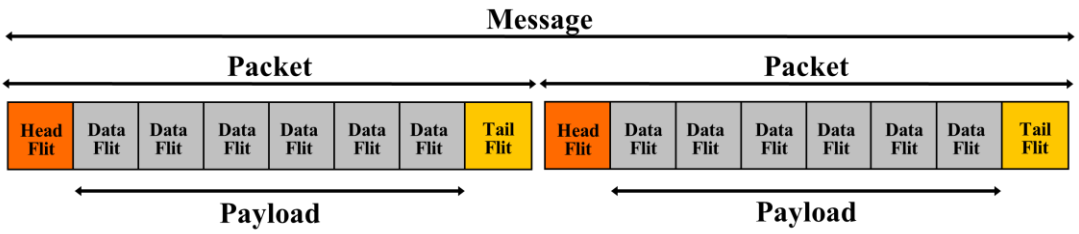

Figure 2: Physical Message Structure

The flow control algorithm is characterized by 1) its transmission granularity which specifies the entity of data that is atomically transmitted from one router to another and by 2) its buffering scheme which specifies how long data must reside inside the current router's buffer before being transmitted to the next router's buffer.

1. The transmission granularity is typically either a a) flit or b) packet. a) In the case of transmitting a flit from Router 1 to Router 2, the flit can simply be sent on the physical link without having to lock it as the flit size is determined by the link's bus width. As a result, atomically transmitting a flit requires a single send and receive action pair. b) In the case of transmitting a packet from Router 1 to Router 2, the physical link connecting the two routers must first be locked by the packet's head flit. Once the link is locked, the packet is transmitted flit-by-flit from Router 1 to Router 2. After the packet has been fully transmitted, the packet's tail flit unlocks the link, allowing the next packet to lock it and begin its transmission. As a result, atomically transmitting a packet requires multiple send and receive action pairs.

2. The buffering scheme is typically either a) store-and-forward buffering or b) cut-through buffering (also called "pipelined" or "wormhole" buffering). a) In the case of store-and-forward buffering, an entire packet must first be fully "stored" in the router's buffer before the router can begin "forwarding" that packet to the next router. As a result, the router's buffer must at least be sized to the packet size. b) In the case of cut-through buffering, flits do not need to wait for their corresponding packet to be fully stored before they can begin moving down the path. Instead, they can "cut-through" the router's buffer and be transmitted to the next router. As a result, the router's buffer must at least be sized to the flit size.

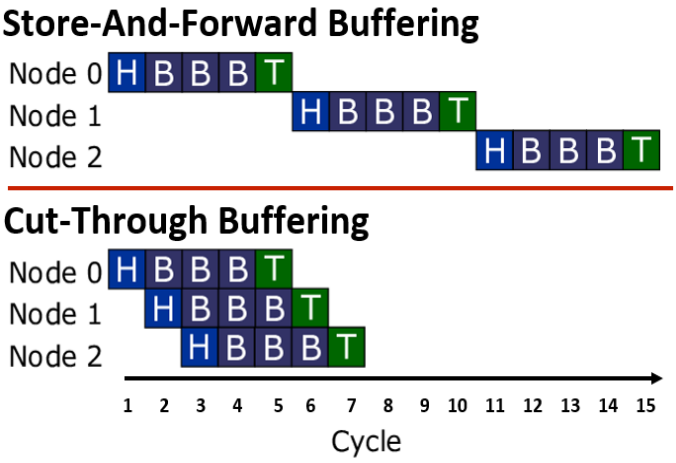

Figure 3: Store-And-Forward and Cut-Through Buffering

C. The routing algorithm determines how the path from source to destination is generated either before the transmission in the case of circuit-switching or during the transmission in the case

of packet-switching. Routing algorithms can either be characterized as 1) non-adaptive or 2) dynamic/adaptive.

1. Static/non-adaptive routing algorithms do not consider the current interconnection network environment conditions (i.e. traffic, power, etc.) when determining where next to route a packet to. Rather, they simply determining the next router to send a packet to as some function of the packet's current position and its final destination. Figure 4 shows the pseudocode for Dimension-Ordered XY Routing, a static routing algorithm for mesh networks where a packet is first routed along the X dimension and then along the Y dimension.

Algorithm 1 Dimension-Ordered XY Routing

```
function DOR_XY(curr_pos, final_pos)
    next_pos.x ← curr_pos.x
    next_pos.y ← curr_pos.y
    if curr_pos.x != final_pos.x then:
        if curr_pos.x < final_pos.x then:
            next_pos.x ← curr_pos.x+1
        else
            next_pos.x ← curr_pos.x-1
    else if curr_pos.y != final_pos.y then:
        if curr_pos.y < final_pos.y then:
            next_pos.y ← curr_pos.y+1
        else
            next_pos.y ← curr_pos.y-1
    return next_pos
```

Figure 4: Pseudocode for Dimension-Ordered XY Routing

2. Dynamic/adaptive routing algorithms do employ the network's conditions as additional parameters when determining the next router to send a packet to. Figure 5 shows the pseudocode for Mesh Adaptive routing, a dynamic routing algorithm for mesh networks where a packet either moves along the X dimension or the Y dimension depending on which direction has the least overall traffic.

Algorithm 2 Adaptive XY Routing

```
function ADAPT_XY(curr_pos, final_pos)
    x_dir_next_pos.y ← curr_pos.y
    y_dir_next_pos.x ← curr_pos.x
    if curr_pos.x != final_pos.x then:
        if curr_pos.x < final_pos.x then:
            x_dir_next_pos.x ← curr_pos.x+1
        else
            x_dir_next_pos.x ← curr_pos.x-1
    if curr_pos.y != final_pos.y then:
        if curr_pos.y < final_pos.y then:
            y_dir_next_pos.y ← curr_pos.y+1
        else
            y_dir_next_pos.y ← curr_pos.y-1
    x_traffic = traffic_fn(x_dir_next_pos)      ▷ traffic at x_dir_next_pos
    y_traffic = traffic_fn(y_dir_next_pos)      ▷ traffic at y_dir_next_pos
    if x_traffic ≤ y_traffic then
        next_pos ← x_dir_next_pos
    else
        next_pos ← y_dir_next_pos
    return next_pos
```

Figure 5: Pseudocode for Mesh Adaptive Routing

Implementation:

1. User Configuration and Simulation Setup

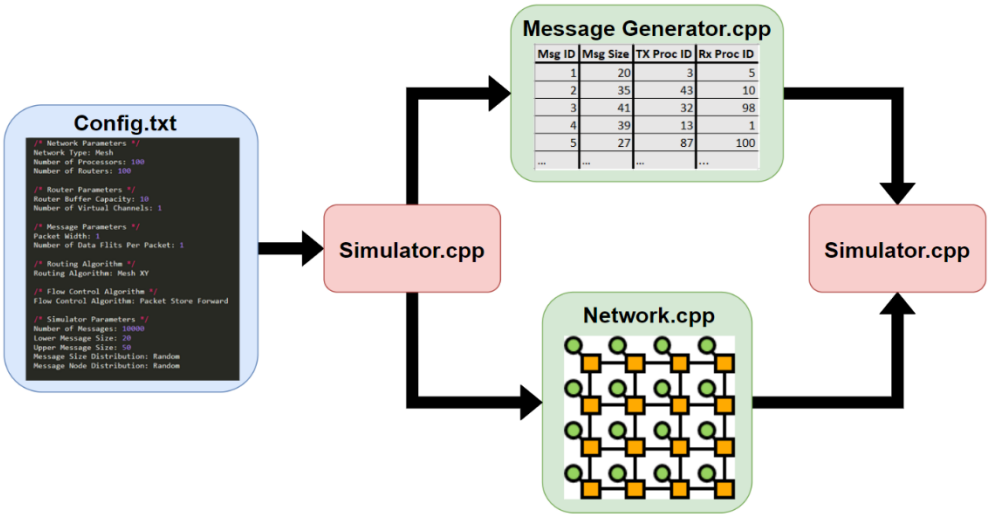

Figure 6: Configuration and Setup Workflow

The user starts by creating a 'config.txt' file which specifies the parameters (i.e. network size, routing algorithm, flow control algorithm, etc.) of the network he would like to simulate. The simulator then parses 'config.txt' and generates a) a list of messages whose size, source, and destination are assigned according to the traffic pattern specified and b) the physical mesh network. The message list and physical mesh network are then integrated together where the messages assigned to each processor are queued up inside that processors' buffers.

Since typical simulations require 1000's of messages to be transmitted on the network, the message generator utilizes multi-threading with the OpenMP API to create the message list. Depending on how the user has specified to distribute the messages across the processors (i.e. random, uniform), there needs to be synchronization between the threads to ensure that the nodes are assigned the correct amount of messages to transmit and receive. As a result, there is a global data structure that each thread atomically updates once it generates a message and adds it to the message list. When a thread generates a message, it first scans this global data structure to determine a) which nodes to assign as the source and destination of the message as well as b) the size of the message.

Depending on the number of virtual channels specified by the user, each router will dedicate a certain number of buffers to each of its physical input channels. In Figure 7, this is seen where each physical input channel, labelled "Input 1" to "Input 5" are connected to a group of input buffers labelled "VC 1" to "VC n". Upon receiving a flit on an input channel, the router runs through some internal logic to determine which input buffer to send the flit

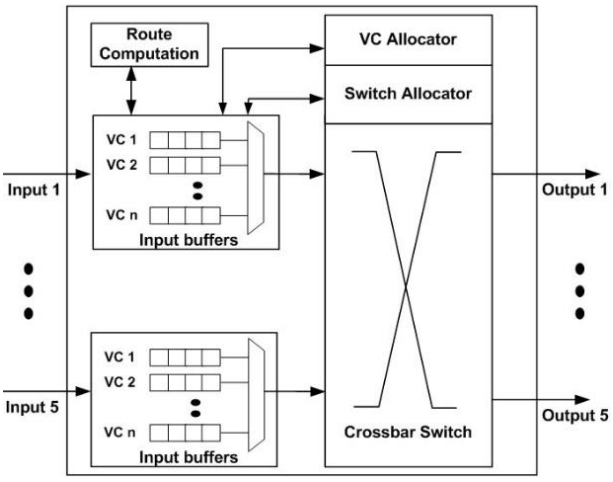

Figure 7: Inside the Router

according to some metadata that the flit stores regarding the packet and message it is a member of. Then, when transmitting a flit on an output channel, the "Switch Allocator" module adjusts the crossbar switch to multiplex one buffer at a time to an output channel. The flit at the front of the multiplexed buffer is then removed from the buffer and transmitted on the output channel.

2. Running the Simulation

Once the setup has been completed with the message list and network properly configured, the simulator initializes a global clock and begins the simulation. At each global clock tick, the simulator follows the procedure shown in Figure 8.

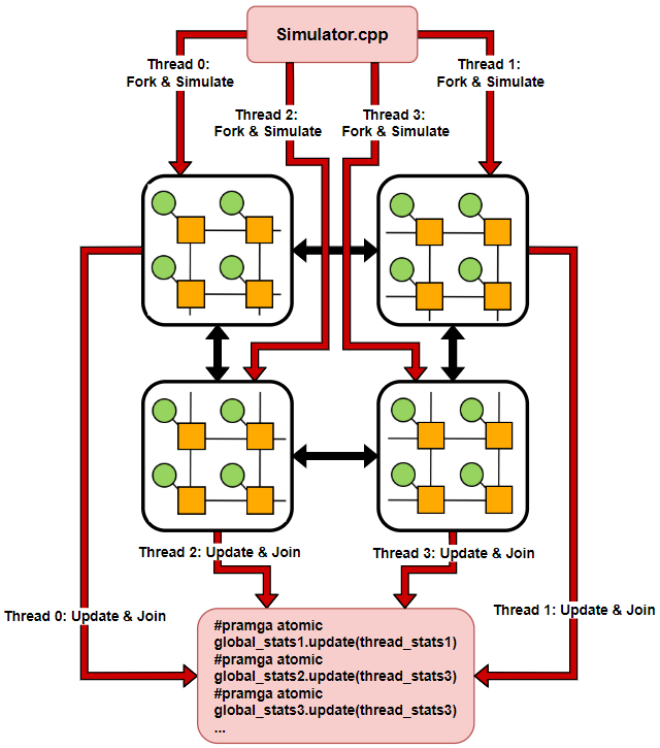

Figure 8: Simulation Workflow

First, the simulator divides up the mesh network among several threads in a block-wise fashion to reduce the communication required between threads (i.e. all the communication only happens at the perimeter of each block). Each thread then simulates its portion of the network in 2 steps: TX RX. Threads are synchronized between the 2 steps via an OMP barrier.

TX: For each node inside a thread's region, that thread scans over all the buffers inside that node, looking at the first flit inside the buffer. It runs the routing algorithm to determine which node to send the flit to next, finds the corresponding output channel connected to that node, and then, if the flit is able to be transmitted according to the flow control algorithm, it queues the flit up on the output channel. Once a thread has finished looping through all its nodes, it is synchronized with the other threads at an OMP barrier.

Algorithm 3 Node-to-Node Transmission

```
1:  function TX(curr_node)
2:      for i = 1 to num_input_channels do
3:          input_channel ← curr_node.input_channels[i]
4:          for j = 1 to num_virtual_channels do
5:              buffer ← input_channel.buffers[j]
6:              flit ← first item in buffer
7:              next_node ← routing_func(curr_node, flit)
8:              output_channel ← channel connecting curr_node to next_node
9:              can_transmit ← is_open(output_channel)
10:             should_transmit ← flow_ctrl_func(buffer, flit)
11:             if (can_transmit and should_transmit) then
12:                 queue up flit on output_channel
13:     return
```

Figure 9: Pseudocode for TX

RX: For each node inside a thread's region, the thread scans over all the input channels connected to that node to determine if there is flit queued up on that channel that is pending transmission. If so, all the buffers inside the node are scanned to determine if there is one that is able to receive this flit. If such a buffer exists, then the flit is removed from the input channel and stored in the buffer. If not, then the thread fails the transmission and does not remove the flit from the input channel. To indicate to the source node the status of the transmission (i.e. if it was successful or if it failed), the thread locks and modifies some state in the source node. This locking is especially necessary in the case where the source node is not inside the thread's assigned region (i.e. at the borders of the thread regions).

Algorithm 4 Node-to-Node Reception

```
1:  function RX(curr_node)
2:      for i = 1 to num_input_channels do
3:          input_channel ← curr_node.input_channels[i]
4:          if has_flit(input_channel) then
5:              find appropriate buffer in input_channel.buffers
6:              if buffer is found then
7:                  remove flit from input_channel and append to buffer
8:              else
9:                  fail transmission on input_channel
10:     return
```

Figure 10: Pseudocode for RX

Second, following the simulation period, each thread scans through and locally accumulates certain statistics from each of its nodes, and then atomically updates the corresponding global statistics with its local statistics. Once it has finished updating, it joins back with the main thread.

This procedure is repeated until the simulation has finished (i.e. until all messages have been received by their corresponding processors). By having a global clock synchronize each simulation cycle, the simulator is able to collect cycle-accurate metrics of the network's performance.

```
void Network::simulate () {
    #pragma omp parallel
    {
        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_routers; i++) {
            Router* router = this->router_lst[i];
            router->clear_internal_info_summary();
        }
        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_processors; i++) {
            Processor* processor = this->processor_lst[i];
            processor->tx();
        }
        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_routers; i++) {
            Router* router = this->router_lst[i];
            router->tx();
        }
        ////////////////////////////////////////////////
        #pragma omp barrier
        ////////////////////////////////////////////////
        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_routers; i++) {
            Router* router = this->router_lst[i];
            router->rx();
        }
        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_processors; i++) {
            Processor* processor = this->processor_lst[i];
            processor->rx();
        }

        #pragma omp for schedule(static) nowait
        for (uint32_t i=0; i < this->num_routers; i++) {
            Router* router = this->router_lst[i];
            router->update_internal_info_summary();
        }
    }
}
```

Figure 11: Simulation Code

3. Aggregating and Visualizing Results

After the simulation has completed, all the statistics which had been logged at each global clock cycle are aggregated together and analyzed to determine the overall network's performance. An example of the summary report provided to the user is shown in Figure 12. In addition, the statistics from each global clock cycle are formatted and dumped into text files which are then parsed by a python interface to display time-series plots of the network's performance across the whole simulation.

```
Average Message Latency in Clock Cycles: 453.227997
Average Message Distance in Channels: 8.682555
Average Message Size: 34.467999
Average Throughput in Messages/Clock Cycles: 0.188324
Average Speed in Distance/Latency: 0.019157
Total Simulation Time in Clock Cycles: 5310
```

Figure 12: Simulation Summary Statistics

Results and Analysis: I ran hundreds of simulations with various mesh network configurations and message traffic patterns to understand the correlation between network design parameters and network performance. This section details some of the key findings I was able to extract from my simulations.

Test #1: Effect of Varying Message Traffic Patterns on Network Performance

This test was intended to explore the correlation between routing and flow control with regards to overall network latency, throughput, and buffer efficiency while keep the total message data (message size * number of messages) constant across traffic patterns.

I simulated all combinations of the routing algorithm (M_XY: Mesh XY or M_Adapt: Mesh Adaptive), flow control granularity (Flit or Packet), and flow control buffering (SF: Store-and-Forward or CT: Cut-Through) for the following message traffic patterns:

- Traffic Pattern 1: Message Size = 10, Number of Messages = 5000
- Traffic Pattern 2: Message Size = 50, Number of Messages = 1000
- Traffic Pattern 3: Message Size = 100, Number of Messages = 500
- Traffic Pattern 4: Message Size = 500, Number of Messages = 100

Figure 11 details the base network configuration used in this test:

Network Size	Packet Size	Num DFs/Packet	Buffer Size	Num VCs	Msg Size Distr	Msg Node Distr
10x10	5	10	13	5	Uniform	Uniform

Figure 13: Test #1 Base Network Configuration

Figure 12 details the performance of each network configuration with each traffic pattern:

Routing: Mesh XY				
	Traffic Pattern 1			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	255	403.08	186.05	295.93
Throughput (msgs/cycle)	0.645	1.124	0.768	1.176
	Traffic Pattern 2			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	619.06	795.62	556.95	671.49
Throughput (msgs/cycle)	0.107	0.169	0.121	0.185
	Traffic Pattern 3			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	1126.42	1152.16	1052.72	1021.71
Throughput (msgs/cycle)	0.0516	0.0829	0.0509	0.0896
	Traffic Pattern 4			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	4062.31	3908.48	3994.65	3803.83
Throughput (msgs/cycle)	0.0103	0.00234	0.0104	0.0123

Routing: Mesh Adaptive				
	Traffic Pattern 1			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	257.71	465.96	188.15	382.79
Throughput (msgs/cycle)	0.667	0.925	0.683	0.821
	Traffic Pattern 2			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	630.11	912.92	569.98	931.05
Throughput (msgs/cycle)	0.105	0.174	0.108	0.148
	Traffic Pattern 3			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	1131.58	1339.77	1052.47	1291.97
Throughput (msgs/cycle)	0.0505	0.0847	0.0509	0.0817
	Traffic Pattern 4			
	SF + Packet	**SF + Flit**	**CT + Packet**	**CT + Flit**
Latency (cycles)	4062.31	3977.83	3994.65	4154.74
Throughput (msgs/cycle)	0.0104	0.0149	0.0104	0.0136

Figure 14: Test #1 Summary Statistics

1. Granularity is more important than buffering for throughput

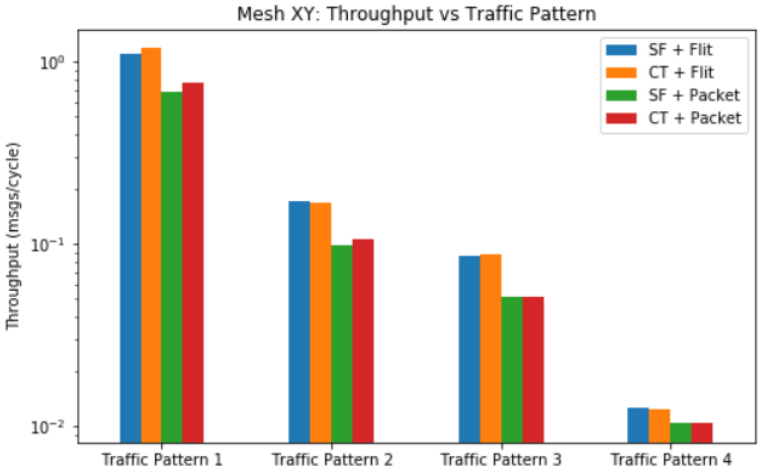

Figure 15: Test #1 Mesh XY Throughput Bar Plot

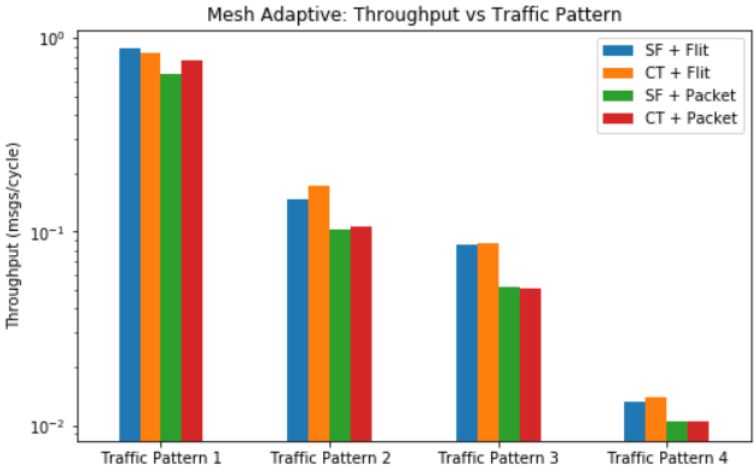

Figure 16: Mesh Adaptive Throughput Bar Plot

As shown in Figures 15 and 16, regardless of the other configured parameters and message traffic patterns, a network's performance is predominantly dictated by its granularity (flit or packet) rather than its buffering (store-and-forward or cut-through). This is so because when using packet-based granularity, the channel must be locked for the entire duration of the packet transmission from node to node. As a result, if for some reason a flit in the currently transmitting packet is blocked from being queued on the channel for transmission, then that channel remains inactive as no other flit from a different packet can use it. Such a situation is especially prominent in highly trafficked networks where there are many packets contending to use the same channel and the packet currently holding the channel is blocked from transmitting because the buffers at the channel's endpoint are full.

In addition, when looking at the network performance over time, we can see that flit-based granularity allows for more flits to be transmitted and received earlier than packet-based granularity. Figure 17 and 18 are time series plots of the number of flits transmitted and received by all processors at each clock cycle, respectively. These plots highlight that flit-based granularity allows nodes to not be stalled trying to transmit packets, and thus allows for more messages to be transmitted in a shorter period of time.

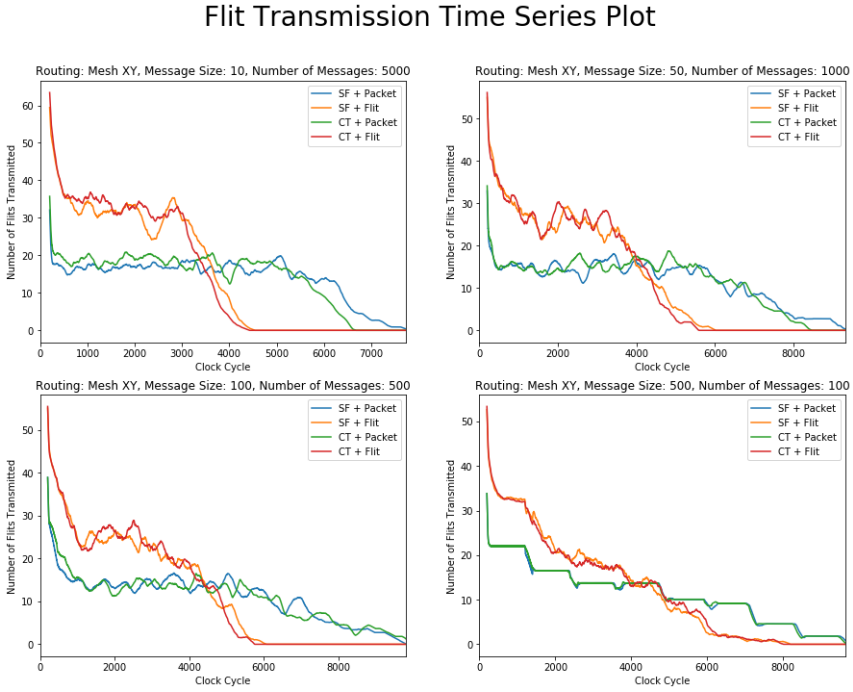

Figure 17: Test #1 Flit Transmission Time Series Plot

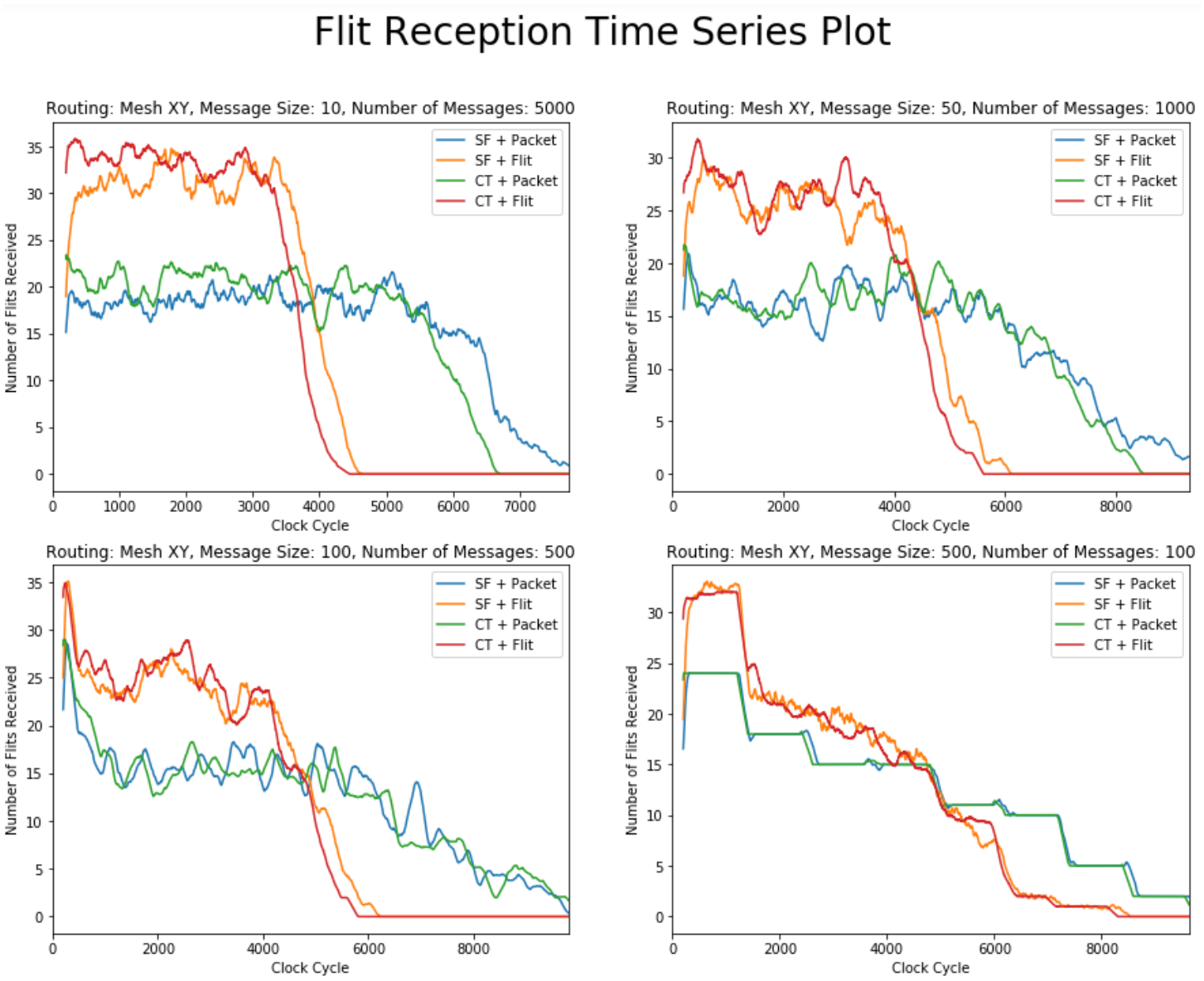

Figure 18: Test #1 Flit Reception Time Series Plot

2. Granularity is more important than buffering for buffer efficiency

Buffer efficiency can be characterized as the ratio of the amount of buffer space occupied with data to the total amount of buffer space in the network. Similar to the previous finding and with similar reasoning, flow control granularity plays a larger role than flow control buffering in the buffer efficiency of a network. Figures 19 demonstrates that in all traffic patterns, flit-based granularity yields higher buffer efficiencies because flits from various packets are able to fill up several buffers within a router with little contention. These plots also reinforce the plots in Figures 13 and 14 because a higher buffer efficiency implies more flits are moving around the network and thus are being transmitted and received by processors.

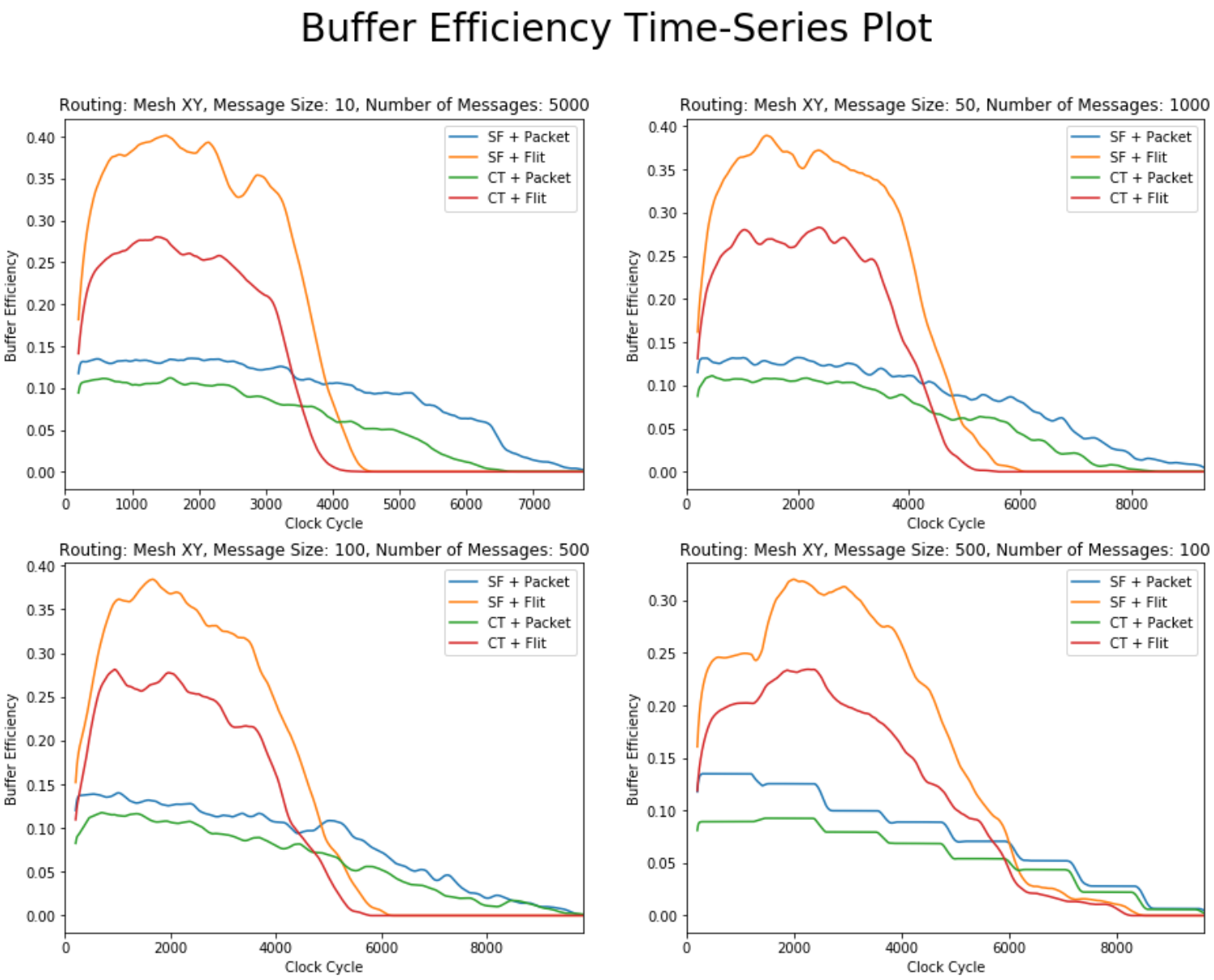

Figure 19: Test #1 Mesh XY Buffer Efficiency Time Series Plot

3. Packet granularity exhibits lower and more deterministic latency

In Figure 14, the latencies of the packet-based network configurations were less than the latencies of the flit-based network configurations when simulated with Traffic Patterns 1-3. This is so because since packet-based granularity requires locking a channel to atomically transmit a packet, there is no possibility for interruption in the packet transmission. As a result, the number of clock cycles required for the packet transmission once the channel is locked is simply bounded by the size of the packet. This channel-locking scheme thus both reduces the overall time for a packet to move from its source to destination and ensures that there is little variance in the latency measure across different messages. This trend is highlighted in Figure 20 which is a KDE (Kernel Density Estimate) of the distribution of message latencies. All packet-based configurations have a latency distribution which mirrors that of an almost point-wise Gaussian with small variance while the flit-based configurations have a latency distribution which mirrors that of a heavily skewed and multi-modal Gaussian.

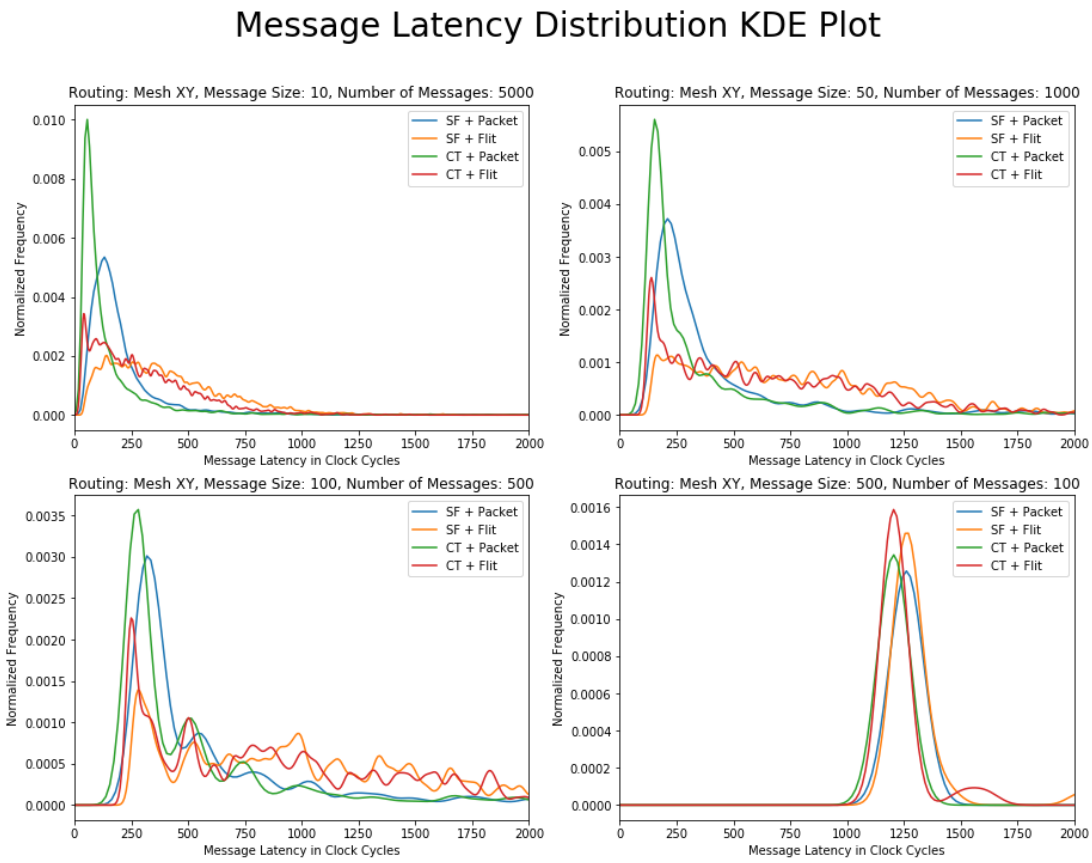

Figure 20: Test #1 Mesh XY Message Latency Distribution KDE Plot

Another interesting point is that as we move from Traffic Pattern 1 (upper left corner) to Traffic Pattern 4 (lower right corner), the latency distributions of the flit-based network configurations slowly converge to that of the packet-based network configuration. A possible explanation for this is that the number of unique messages and thus unique packets in the network decreases, then at a given node, there is less contention among the flits inside a node's buffers for the same channel simply because the probability that multiple flits are following the same path is less.

4. Adaptive routing algorithms are better exploited with flit-based granularity

Adaptive algorithms attempt to evenly distribute flits across all the buffers in the network to avoid any highly trafficked node in the network that could be hotspots for significant contention. Thus, the effectiveness of an adaptive algorithm can be measured by its buffer efficiency (i.e. how well it utilizes the buffer space provided in the network).

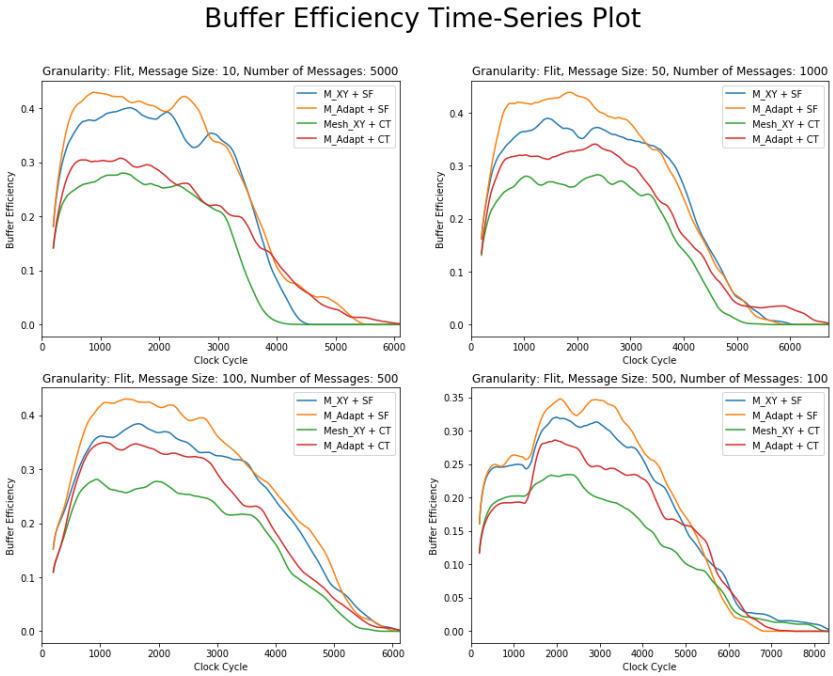

Figure 21: Test #1 Flit-Based Buffer Efficiency Time Series Plot

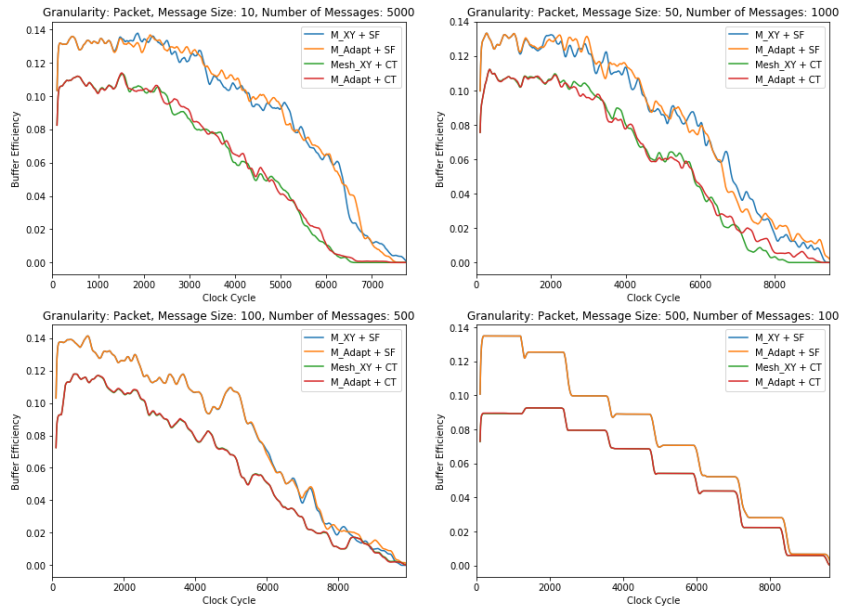

Figure 22: Test #1 Packet-Based Buffer Efficiency Time Series Plot

Figure 21 shows the buffer efficiency of networks configured with flit-based granularity and different combinations of routing algorithms (Mesh XY or Mesh Adaptive). Clearly, in all 4 traffic patterns, the Mesh Adaptive algorithm achieves higher buffer efficiency as expected.

Figure 22 follows the same layout and network configurations as Figure 21, however the one difference is that all network are instead configured with a packet-based granularity. In these networks, there is no noticeable difference in the buffer efficiency between Mesh Adaptive and Mesh XY routing. In addition, the buffer efficiencies shown in Figure 22 are significantly lower than those shown in Figure 21.

These results highlight that adaptive algorithms are best exploited in networks with flit-based granularity. This is so because packet-based granularity reduces the resolution by which the adaptive algorithm can evaluate the traffic at a node. Once a packet locks a channel and begins transmission to its next node, the adaptive algorithm can only route a flit of a different packet to that same node once the current packet finishes. And by the time the packet has finished, the traffic at that candidate node has increased on the order of the packet size. As a result, the adaptive

algorithm can only coarsely react to traffic at the scale of the packet size in packet-based granularity, while in flit-based granularity, it can finely react to traffic at the scale of the flit size.

Test #2: Effect of Varying Buffer Resource Sizing on Network Performance

This test was intended to explore the correlation between the buffer capacity and number of virtual channels with regards to overall network throughput and buffer efficiency while keep the total buffer space (buffer capacity * number of virtual channels) constant across all patterns.

I simulated all combinations of the routing algorithm (M_XY: Mesh XY or M_Adapt: Mesh Adaptive) and flow control granularity (Flit or Packet) for the following buffer/virtual channel patterns:

- Buffer/VC Pattern 1: Buffer Capacity = 3, Number of Virtual Channels = 20
- Buffer/VC Pattern 2: Buffer Capacity = 4, Number of Virtual Channels = 15
- Buffer/VC Pattern 3: Buffer Capacity = 6, Number of Virtual Channels = 10
- Buffer/VC Pattern 4: Buffer Capacity = 10, Number of Virtual Channels = 6
- Buffer/VC Pattern 5: Buffer Capacity = 15, Number of Virtual Channels = 4
- Buffer/VC Pattern 6: Buffer Capacity = 20, Number of Virtual Channels = 3

Figure 19 details the base network configuration used in this test:

Network Size	Packet Size	Num DFs/Packet	FC Buffering	Num of Msgs	Lower Msg Size	Upper Msg Size	Msg Size Distr	Msg Node Distr
10x10	5	10	Cut-Through	1000	20	50	Random	Uniform

Figure 23: Test #2 Base Network Configuration

Figure 20 details the performance of each network configuration with each buffer/vc pattern:

	Buffer/VC Pattern 1			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	363.42	349.11	359.09	347.6
Throughput (msgs/cycle)	0.135	0.3	0.136	0.299

	Buffer/VC Pattern 2			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	374.87	383.15	371.73	382.34
Throughput (msgs/cycle)	0.158	0.289	0.153	0.297

	Buffer/VC Pattern 3			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	368.22	435.06	365.33	485.62
Throughput (msgs/cycle)	0.164	0.271	0.171	0.334

	Buffer/VC Pattern 4			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	378.17	458.13	383.77	546.28
Throughput (msgs/cycle)	0.158	0.303	0.157	0.281

	Buffer/VC Pattern 5			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	406.209	523.43	412.373	DEADLOCK
Throughput (msgs/cycle)	0.182	0.251	0.181	DEADLOCK

	Buffer/VC Pattern 6			
	M_XY + Packet	M_XY + Flit	M_Adapt + Packet	M_Adapt + Flit
Latency (cycles)	451.4	573.52	453.23	DEADLOCK
Throughput (msgs/cycle)	0.19	0.235	0.188	DEADLOCK

Figure 24: Test #2 Summary Statistics

Note that in Buffer/VC Pattern 5 and 6, when the network was configured with Mesh Adaptive routing algorithm and flit-based granularity, the simulation deadlocked. This is a known issue with using adaptive algorithms in a mesh network and in real interconnection networks, this is typically handled either by dropping packets or by having the source node resend the message. For this project, I did not implement any deadlock recovery strategies as they would compromise the integrity of comparing results between networks which did and did not deadlock.

1. Flit-based granularity networks exploit the multiplexing of virtual channels. However, with larger buffer capacities and fewer virtual channels, flit-based granularity networks converge in performance to packet-based granularity networks

Flit-based granularity networks exploit the parallelizability built into the multiplexing of virtual channels by allowing flits from various messages and packets to quickly fill up the buffers attached to each physical channel. Figure 25 and 26 highlights this by showing that flit-based granularity thrives much more than packet-based granularity when buffer capacity is limited but virtual channels are plentiful. However, as we move from Buffer/VC Pattern 1 to 6, we see that the both flit- and packet-based granularity networks, regardless of the routing algorithm use, converge to a similar behavior. This is so because typically routers are designed such that only the flits from one packet can occupy a buffer at a time in order to reduce the hardware complexity required to store extra metadata about the buffer's contents. Thus, when virtual channels are sacrificed for buffer capacity, routers in flit-based granularity networks are now constrained to work with only few distinct packets, thereby serializing their transmissions similar to those of packet-based granularity networks.

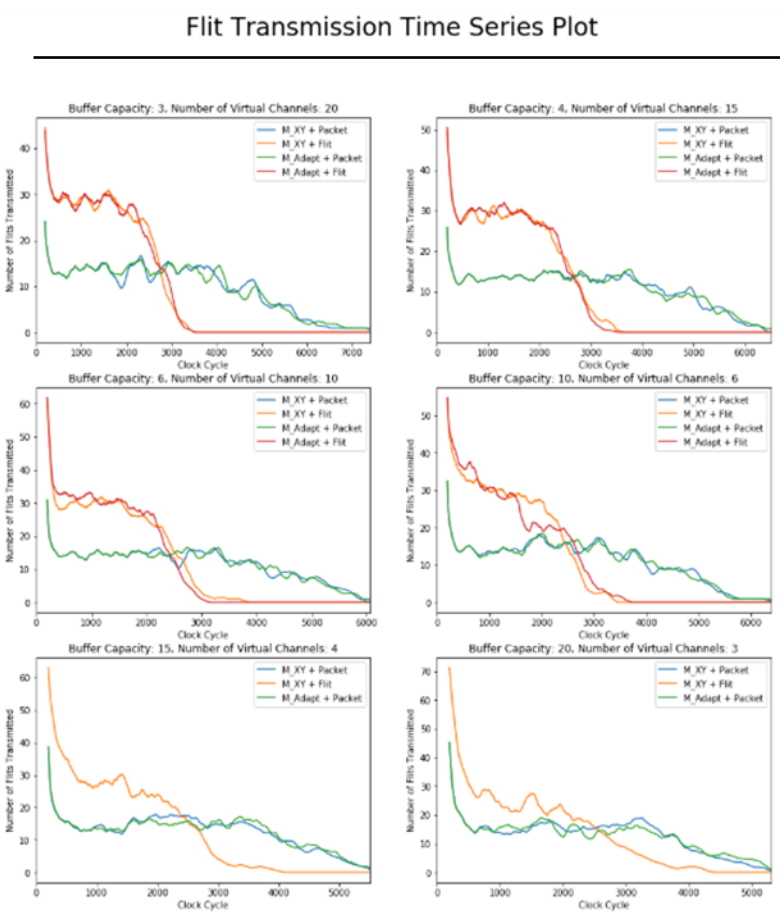

Figure 25: Test #2 Flit Transmission Time Series Plot

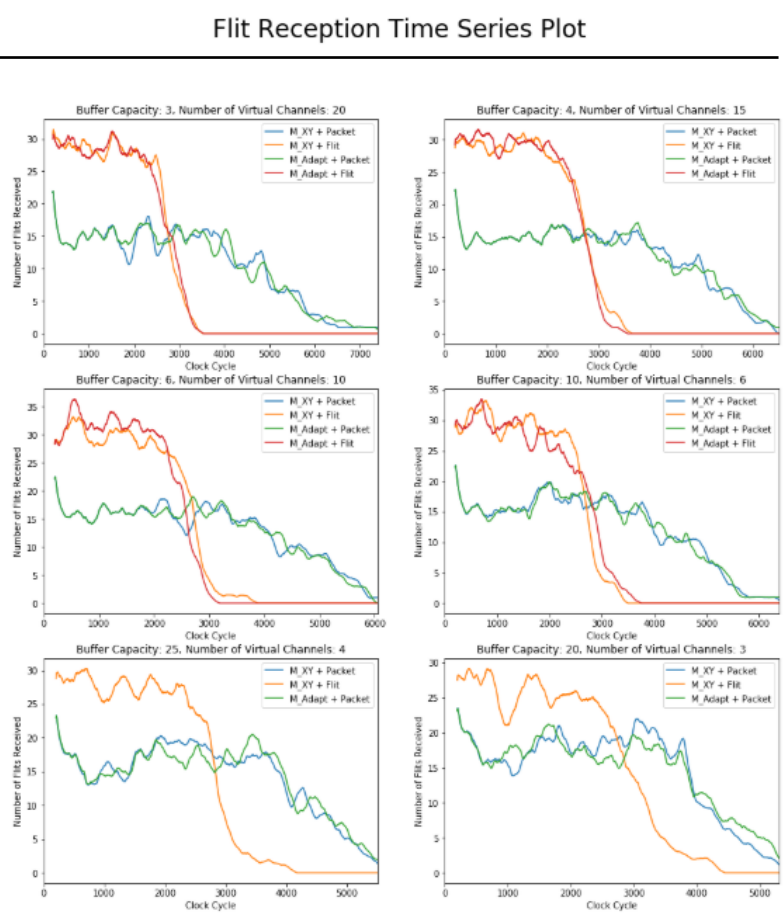

Figure 26: Test #2 Flit Reception Time Series Plot

2. Constraining the number of virtual channels reduces CT to SF buffering

When the number of virtual channels available to route a flit to are limited, flits are not as easily able to "cut-through" their router and be transmitted to the next. As a result, flits of a packet begin to accumulate within the router while waiting for the buffer at the next router to free up. This behavior resembles that of store-and-forward buffering where flits can only be transmitted once the entire packets has been received inside a node. Figure 27 validates this finding by showing an increasing trend in the buffer efficiency across all combinations of routing and flow control granularity network configurations as we move from Buffer/VC Pattern 1 to 6. As the number of available buffers decreases (i.e. the number of virtual channels decrease) and the overall buffer efficiency increases, this implies that the buffers are accumulating more and more flits before beginning to transmit them, thus resembling store-and-forward buffering.

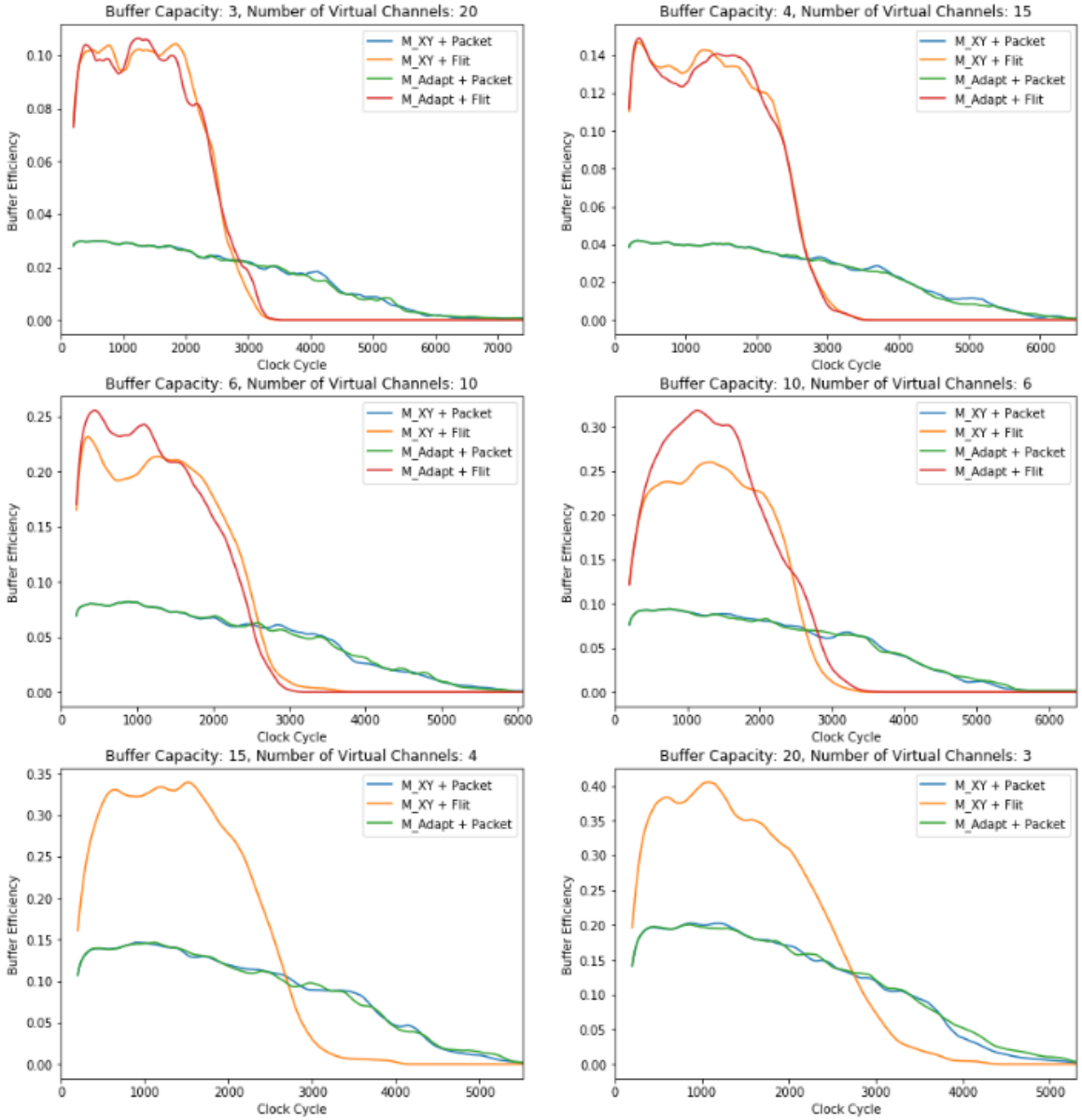

Figure 27: Test #2 Buffer Efficiency Time Series Plot

Conclusion: The choice of a) topology, b) routing algorithm, and c) flow control algorithm each individually and all holistically play a significant role in determining the overall interconnection network's performance. And depending on the message environment a network is employed in, certain combinations of these 3 attributes lend themselves to better network throughput, latency, and buffer efficiency than others. This project, through simulating and analyzing various network configurations, explored the vast design space available for interconnection networks and drew valuable insights correlating these design parameters to network performance.

References:

https://etd.lib.metu.edu.tr/upload/12609136/index.pdf

http://www.diva-portal.org/smash/get/diva2:11666/FULLTEXT01.pdf

https://www.hindawi.com/journals/jece/2012/509465/

http://pages.cs.wisc.edu/~tvrdik/7/html/Section7.html#Virtual

https://www.design-reuse.com/articles/10496/a-comparison-of-network-on-chip-and-busses.html

https://www.pdl.cmu.edu/PDL-FTP/associated/maze-routing_nocs15.pdf