ON THE EFFECTIVENESS OF SOURCE THROTTLING FOR NETWORKS-ON-CHIP IN CHIP MULTIPROCESSOR DESIGNS

A Thesis

by

RUIXIAO NI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jiang Hu
Co-Chair of Committee,	Paul V.Gratz
Committee Members,	E.J. Kim
Head of Department,	Chanan Singh

May 2014

Major Subject: Computer Engineering

ABSTRACT

In modern chip-multiprocessor (CMP) designs, with the increasing number of cores, traffic between different cores keeps increasing. Consequently, on-chip interconnection networks experience increasingly large communication bandwidth demand. This thesis focuses on Quality-of-Service (QoS) of Networks-on-Chip (NoC). NoC is considered as a scalable approach of interconnection network compared to conventional bus-based architecture. Like Ethernet, NoC faces common QoS issues such as bandwidth utilization and fairness. This thesis is a study on the effectiveness of source throttling for NoC, including fairness and overall performance such as program run time and packet latency. Source throttling is a well-known technique for traffic regulation. It is shown to be effective for bufferless NoC in previous studies. Due to different traffic behaviors and characteristics, however, it is not obvious if source throttling is effective for general buffered NoC. The first part of this research is a set of network simulations on various synthetic traffic cases. The results indicate that source throttling can reduce application runtime when (1) the network is congested, (2) there are dependencies among communication requests, and (3) the width of the dependence graph must be sufficiently large. The second part is full system simulations on public benchmark suites. Source throttling does not bring benefit for these relative realistic cases. Further experiment reveals that the aforementioned conditions are not satisfied. This explains why source throttling is of little use for general buffered NoC in CMP designs.

DEDICATION

To my parents and grandparents

ACKNOWLEDGEMENTS

I would like to thank all those who encouraged me and helped me during my study and research at Texas A&M University.

Especially, I would like to thank my family for their ceaseless support, encouragement and endless love, without which I would never be able to complete my master study so smoothly.

Also, I would like to extend my heartfelt gratitude to my advisor, Dr. Jiang Hu, who gave me constant guidance as well as warm encouragement throughout this research. I also would like to thank my co-advisor, Dr. Paul Gratz, and Dr. Srinivas Shakkottai for giving their helpful suggestions on this research. They were always patient, kind and helpful whenever I had questions on my academic life. I could not have completed this thesis without their guidance and generous support. And I would like to thank Dr. E. J. Kim for being my committee member.

Last but not least, thanks to all my friends and colleagues who accompanied me these years, and also the department faculty and staff for giving me a warm and kind environment during my life at Texas A&M University.

NOMENCLATURE

CMP	Chip-multiprocessor
LPH	Latency per Hop
MSHR	Missing Information/Status Holding Register
NI	Network Interface
NoC	Networks-on-Chip
QoS	Quality-of-Service
ROI	Region of Interest
RTT	Round Trip Time
ST	Source Throttling

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

In modern chip multi-processor (CMP) designs, the demand for higher performance never ceases to increase, and number of cores on a single chip grows accordingly. As more and more applications become communication intensive, the interconnection network, which handles all kinds of traffic between different cores, suffers from poor bandwidth utilization and unfair bandwidth allocation. Conventional bus-based architecture cannot fulfill the demand for bandwidth when on-chip traffic grows along with the number of cores.

Networks-on-chip is introduced as a scalable substitute to bus-based designs. In NoC based CMP designs, every core has its private uncore part, such as shared cache and network interface (NI). Through the NI, each core is connected with a router. Routers together with communication links constitute a network. NoC is more scalable because its resources are distributed and supports massively parallel data transportation as opposed to bus, which is a centralized resource and is used by one or a few packets at a time. However, resource of NoC is still limited especially for coping with peak traffic demands. Some applications are so communication intensive that the limited NoC resource becomes a bottleneck of entire chip performance.

Quality-of-Service is to deal with the resource allocation issue for networks. It is a concept introduced from the field of computer network. It can be achieved in different approaches. One is through routing policy, such as adaptive routing. When the utilization of a network is below its saturation point, it is effective for reducing congestion.

However, it requires additional effort to collect congestion information and conduct routing computation. Another approach is through router design, especially the flow control design. It can help to improve performance but it lacks the global view of a network. Last but not the least, source throttling is an approach that is studied in this thesis research.

Source throttling is a well-known technique. Previous studies show that it is an effective to regulate traffic, either in road systems or in Ethernet. Its main idea is to hold some packet injection to the network so that network congestion is reduced and the overall performance is improved. The signal light at highway entrances in California is an example of source throttling. Previous works on NoC source throttling are mostly restricted to bufferless designs and benefits are observed in such cases. However, buffered NoC is more typical and the mainstream technique. Traffic behaviors and characteristics in buffered NoC are remarkably different from those in bufferless NoC.

The effectiveness of source throttling for buffered NoC is hardly studied before and is the main focus of this thesis research. Our study includes network simulations on synthetic cases as well as full system simulations on public benchmark suites. We find it is very difficult, if not impossible, to observe any benefit of using source throttling for NoC in realistic CMP designs. By conducting various simulation experiments, we also analyze the reason of this ineffectiveness and identify several necessary conditions that may allow it to be useful.

2. RELATED WORK

Quality-of-Service is an important issue for NoC designs. As more applications become communication intensive, congestion shows up and traffic regulation is needed. There are many different QoS techniques. Lu, et al., [9] propose a Time-Division-Multiplexing Virtual-Circuit configuration for NoC. In this work, time is divided into a number of slots evenly. Traffic of different directions reserves a fixed number of slots in one period. It has hard bandwidth guarantee, but due to this guarantee and non-adaptive configuration, utilization of bandwidth is not high in some traffic patterns. Globally Synchronized Frames (GSF) is a frame-based QoS approach proposed by Lee, et al. [10]. It uses deadline-based arbitration to guarantee the bandwidth, and help to improve fairness. However, this technique may degrade network throughput when there is a congestion hotspot. Preemptive Virtual Clock (PVC) is another frame-based QoS Scheme [11]. It allows high-priority packets to preempt low-priority packets. It is a cost-effective mechanism in terms of area and energy. On the other hand, as it allows packet dropping and therefore needs a retransmission scheme with an additional ACK/NACK protocol.

Similar as the aforementioned approaches, source throttling also proves to be an effective technique to reduce congestion in interconnection [7], [8]. It is first applied to off-chip networks. Baydal, et al., propose an injection restriction mechanism to reduce congestion for wormhole network [1]. Gran, et al., present an implementation of infiniband hardware for congestion control [2]. This interconnection is between many large-scale computers and it is different from on-chip network: each node has more computing power and storage space; the latency is significantly large compared with on-chip network. Later, a few prior works apply source throttling to on-chip network. Nychis, et al., [3], [4] propose an application-aware mechanism to achieve higher network utilization for bufferless NoC. In these two works, starvation rate is used as a metric to indicate the congestion of a network. The starvation rate is applied into a function together with another parameter - Instructions-per-Flit. When the value of this function is higher than a preset threshold, source throttling is started. It is an effective scheme but it does not consider network conditions. Its focus is on the awareness of applications. Chang, et al., make an improvement of the work and introduce a mechanism that is both application-aware and network-load-aware [5]. This mechanism detects L1 misses per thousand instructions (MPKI) to indicate application load. In addition, it uses different throttling rate steps to make it network-load-aware. When the throttling rate is low, the rate step is large. When most of the traffic is blocked, the rate step is small. As such, this scheme avoids over- and under-throttling. To the best of our knowledge, there is no published work on latency-based source throttling on NoC.

3. BACKGROUND

In this section, we introduce some basic concepts that will be employed in this thesis research, including basics in CMP architecture, cache coherence protocol, metrics, and techniques in source throttling.

3.1 CMP Architecture

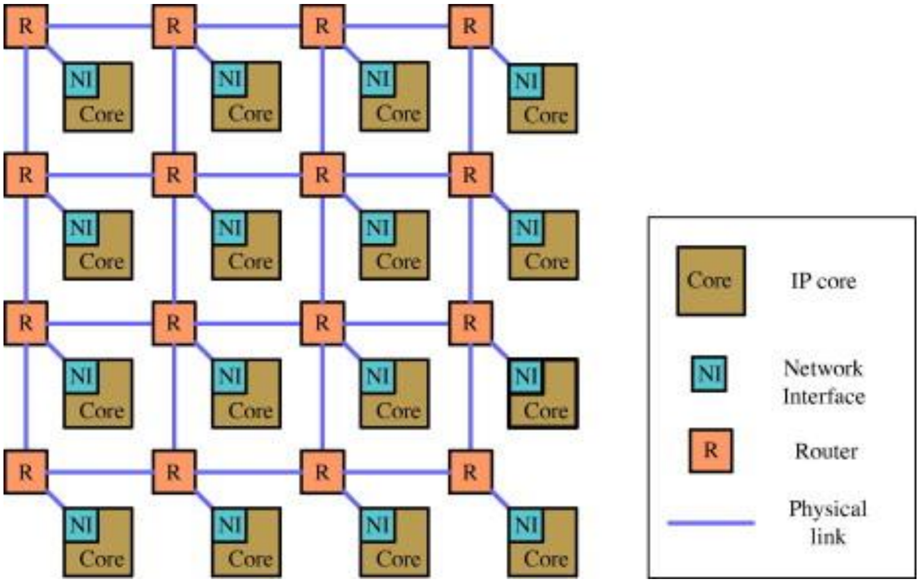


Figure 1. CMP and 2D mesh-based NoC architecture.

In CMP, each core has a corresponding uncore part, including shared caches and a network interface (NI). Through the NI, a core is connected to the network. Both injection and ejection traffic go through network interface to enter or leave the network. Each network interface is connected with one router. Between routers, there are physical links. There are different topologies of the interconnections between routers. The topology in Figure 1 is a 2D mesh [6].

For the uncore part, the detail is shown as Figure 2.

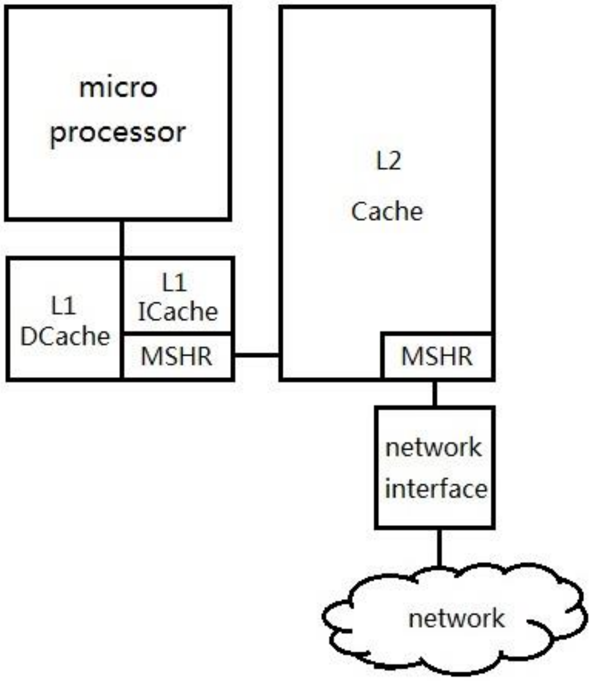


Figure 2. Components in CMP architecture

Each core will have its private L1 instruction cache and data cache. Next to it is L2 data cache. It can be configured either shared cache or private cache, depending on the cache coherence protocols. If L2 cache is shared, it is still physically distributed. L2 cache is connected with a network interface. For each level cache, there is a component called missing information/status holding registers (MSHRs). MSHRs [12] will help keep track of cache misses and allow CPU to go on execute without waiting for the response of cache misses. The size of MSHRs restricts the maximum number of outstanding requests that can be issued into the network at the same time.

3.2 Cache Coherence Protocols

Because of the execution requirement in modern CMP design, write-back cache is a dominant cache in today's cache hierarchy. To keep the consistency between caches of all different levels and between different private and physically distributed shared cache, cache coherence protocol is needed.

There are mainly two types of cache coherence protocols: one is directory-based and the other is snooping. Directory-based protocol means that every cache has a directory. Whenever a request attempts to get permission to access a cache, it needs to query the directory. As the directory knows whether the entry is currently updated or invalidated by some other processors, it decides whether or not to give the permission. Also, the directory may know who has the most recently changed data, which helps to direct the request to that location. Another type of cache coherence protocol is snooping. For this protocol, whenever there is a cache miss, the request is cloned and broadcasted to all the other caches. There is no directory-like component in snooping protocols. Local cache has no idea who has the copy. The only way to require the data is to broadcast. Evidently, snooping protocol causes more traffic and more easily results in congestion than directory-based protocol. In the simulation environment of this thesis work, MESI CMP directory is a directory-based protocol and MOESI hammer is a snooping protocol.

For both types of protocols, cache status plays an important role in the protocol. Transitions of cache status help to understand cache behaviors.

3.3 NoC Performance Metrics

NoC performance can be evaluated by several different metrics. Below are some common metrics which are employed in this thesis:

Round trip time (RTT): In NoC, for every transaction, there is a pair of request and response. Round trip time is the time from the start of a request being issued to the time the corresponding response is received. It includes three parts: request latency in the network, response time at the target and response latency in the network. Usually the response time at the target is a fixed value, like cache access time. But sometimes due to the limit to the number of target entries, the response time can be a multiple of the access time. RTT is a good metric to indicate the congestion of the network. When a network is congested, some packets must wait longer in buffer queues and hence lead to RTT increase.

Ideal round trip time: Ideal round trip time is the RTT if there is no congestion in the network. Ideal RTT is used as a reference in source throttling algorithm.

Latency per hop (LPH): In the 2D mesh topology, for every source-target node pair, the minimum number of edges in-between is the number of hops. Every packet, either request or response, has network latency. This latency divided by the number of hops is the latency per hop. This metric can also indicate the network congestion, but since its value is typically small compared with RTT, it is not an ideal parameter to use in the control policy. However, due to the various numbers of hops of different requests, LPH is a good measure of fairness. It makes long distance packets and short distance packets comparable.

Queue occupancy: This metric measures the occupancy of queues in buffered NoC. It indicates the resource utilization.

Link Utilization: This is another metric to estimate the NoC resource utilization. It measures the usage of the links between routers. Under dimension order routing, links at the cross section tend to have more traffic than the others. Thus, these links are to be used to estimate the utilization.

Injection rate: This metric is used to indicate the source feeding rate. If it is high, it is likely that it has a high throughput. The standard deviation of this metric can also indicate the fairness.

3.4 TCP-Vegas

TCP-Vegas is a congestion avoidance algorithm based on latency. It uses RTT to estimate the network congestion and executes control policy accordingly. In TCP-Vegas, there is a window at the source queue. Records of packets in the source queue are placed in the window. As source throttling is only applied to the source, only request packets are throttled, not response packets. When a request record is placed into the window, the request packet can be injected into the network. A record is removed when the corresponding response packet is received. The original TCP-Vegas keeps track of the sequence of request records. It allows window shifting only when the top request is replied. But in NoC, the number of hops of different requests varies. There are long distance requests and short distance requests. Obviously, long distance requests have relatively long RTT. If sequence of the records is still traced as the original TCP-Vegas algorithm, short distance requests could be stalled unintentionally and degrades the performance. To break the sequence trace of record, request record can be added into the window as long as there is an available slot. Then the window basically means the maximum outstanding requests allowed. Changing the window size can help control the congestion in the network.

For the window size changing policy, the basic idea is to keep the actual throughput close to the estimate throughput, which is also the ideal throughput.

The ideal throughput is calculated as follows:

$$\text{ideal throughput e} = \frac{window\ size}{ideal\ average\ RTT}$$

The actual throughput is

$$\text{actual throughput a} = \frac{number\ of\ acknowledgements}{actual\ average\ RTT}$$

The ideal throughput is always greater than the actual throughput. The difference (e – a) is measured. There are two preset threshold values Bl and Bh (Bl < Bh). There could be three cases:

(1) Bl < (e – a) < Bh, it means that actual throughput is in the range as expected. The window size is then kept unchanged.

(2) Bl > (e – a), it means that the actual throughput is very close to the ideal case. Requests hardly meet any congestion. More requests can be issued into the network. Hence, the window size is increased.

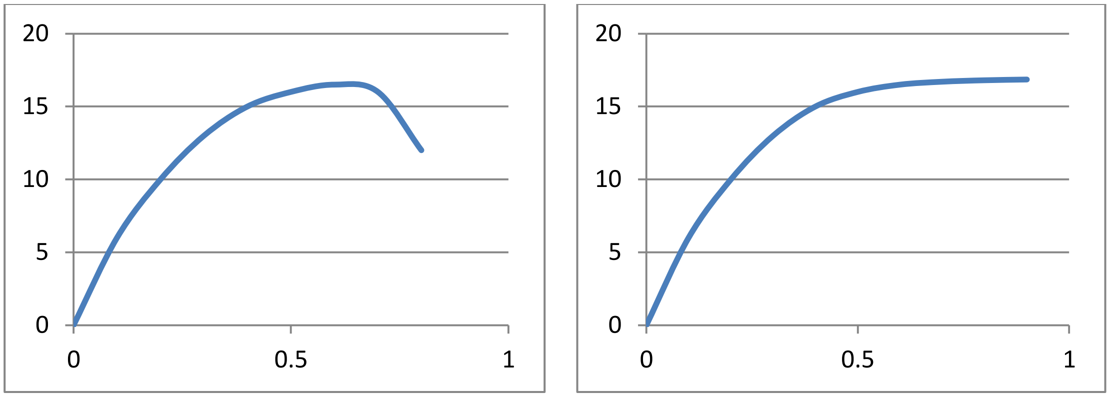
(3) Bh < (e – a), it means that the network is so congested that the actual throughput is too low. In this case, the window size needs to be decreased to block requests from injecting into the network.

The control policy is called periodically to change the window size. The period is named control interval. Actual RTT and ideal RTT is measured in each control interval. The size of control interval determines the granularity of the algorithm. It may lead to different simulation results. This issue will be discussed in more details later.

3.5 Properties of Buffered Network

In this thesis work, buffered network is chosen as the interconnection network. It has some properties that are related to source throttling.

In most previous works, source throttling is applied on the bufferless networks. It seems to gain a good benefit using source throttling. But the properties of bufferless network and buffered network are different. The difference is shown in Figure 3.



(a) Bufferless network [4]

(b) Buffered network [13]

Figure 3. Throughput vs. injection rate

It is very clear that two types of networks act differently when injection rate is very high. The bufferless network suffers from a throughput drop, while the throughput of buffered network saturates and stays at the maximum. The effect of source throttling is to reduce injection rate when it is too high, to improve the performance or throughput. For bufferless network, when reducing the injection rate from a very high value, the throughput can get closer to the maximum value and thus gain benefit. But for buffered network, when throttling reduces the injection rate, there is not much change in the throughput. The potential benefit in buffered networks is not expected to be as high as that in bufferless networks.

Another property that buffered network has is the tree saturation [7]. As it is on-chip network for NoC, its resource, such as queue depth, is very limited. Once there is congestion, it means some packets are stalled in the network. It occupies one certain queue in the network, and blocks some upstream traffic from injecting into this queue. Then the upstream traffic will be stalled. The back-pressure is propagated to the source queue. The buffers are filled up quickly and part of the network is stalled, leading to performance degradation. Source throttling helps to reduce the effect of tree saturation as it blocks traffic at the source node.

4. PROBLEM STATEMENT AND METHODOLOGY

This section mainly discusses the problem formulation and methodology. It introduces the mathematical model of the problem and an analysis is provided.

4.1 Problem Formulation

The problem formulation is:

Maximize

$$V(x) = \sum_{r\in S} U_r(x_r) - \sum_{l\in L} B_l\,(\sum_{s:l\in s} x_s)$$

where U_r is the utility function, according to a certain fairness metric, B_l is a “barrier” associated with link *l* and *x* is the injection rate [13].

The utility function can be built depending on the type of fairness desired. There are three basic types of fairness: proportionally fairness, max-min fairness and minimum potential delay fairness. In this thesis work, proportionally fairness is the adopted. In this case, the utility function can be written as:

$$U_r(x_r) = \log(x_r)$$

Then the problem is to find the optimal $\{\widehat{x_r}\}$.

4.2 Methodology

To solve the mathematical formulated in section 4.1, an algorithm similar to steepest decent is used. In each control interval, throttle is made according to the following equation.

$$\Delta x_r = k_r(x_r)(U_r'(x_r) - \sum_{l \in r} f_l \, (\sum_{l \in s} x_s))$$

where k_r is an empirical parameter, U_r' is the gradient of the utility function, x is the injection rate and f_l is the link price. In this project, the round trip time of the request is used as the link price. The injection rate is adjusted using the equation above at the beginning of each control interval.

To solve this problem, TCP-Vegas based source throttling is applied. The basic idea of TCP-Vegas is to keep the difference between the ideal throughput and the actual throughput in a certain range. This difference is used to control the window size at the network interface. The window keeps the number of outstanding requests injected into the network by its core. If the window is fully filled, all the requests trying to inject into the network will be stalled. Until at least one outstanding request gets replied can new request get injected. This method is revised and different from the basic TCP-Vegas. Because for on-chip network, resource is limited and per-destination queues cannot be implemented, the window controls only one queue that stores requests to all destinations. This makes the method keep track of all outstanding requests.

Metrics to be measured are round trip time, link utilization, queue occupancy, latency per hop and injection rate. They can present the network utilization and fairness for the baseline case and throttled case. Comparing these two groups of data, it can be concluded whether benefit can be gained by source throttling.

The simulations are based on both synthetic traffic and practical application traffic. Synthetic traffic is used to identify the conditions for source throttling to be effective. The practical application is to test the method in the full system simulation. It is to show whether or not source throttling is useful in realistic cases.

5. SYNTHETIC TRAFFIC SIMULATION

This section describes synthetic traffic simulation. The simulation is based on ocin-tsim simulator [17]. Since the cases are prepared by ourselves, high controllability and observability are allowed in these cases. It starts from introducing the simulation environment. Then using one specific self-built case to find how the benefit of source throttling is generated. Finally, we try more synthetic cases to see the potential average gain.

5.1 Ocin-Tsim Simulator and Simulation Environment

In the synthetic traffic simulation, ocin-tsim simulator is used. It is claimed as a DVFS (Dynamic Voltage and Frequency Scaling) aware simulator for NoC. However, since it models the cache system and network interconnection, it can be used as a simulator for source throttling. It can also run some benchmarks by reading trace files. But the traces are fixed files and every packet is generated at a fixed time according to the trace files. Its simulation still has significant difference from practical CPU feeding mechanism on packet generation. Therefore, we use it only for synthetic traffic simulation.

The configuration of ocin-tsim simulation environment is shown as Table 1.

Table 1. Simulation environment configuration

Network	64 nodes, 8 by 8 mesh, dimension order routing
Synthetic benchmarks	Based on uniformly random. 1- and 4-flit packets, stochastically generated
Baseline network	4 VCs per network port, 5 flits per VC; 2 injection VCs; 2 ejection VCs; round robin policy
Network latency	1 cycle per hop, 1 cycle wire delay, 1 cycle router pipeline latency

5.2 Synthetic Traffic Simulation and Analysis

There are a lot of built-in synthetic traffic patterns in ocin-tsim, such as uniformly random. The first synthetic simulation is on uniformly random traffic. By tuning the injection rate, we can generate congested cases. Because it is a uniformly random traffic, the window size is expected to be a fixed number. The result is as Table 2 follows.

Table 2. Source throttling on uniformly random traffic

window size	baseline	3	4	5	6	7	8
Network latency	83.6886	99.7884	88.1761	83.6539	83.5956	83.6622	83.727
Q time	3.51036	27.1341	12.8121	5.1118	3.90617	3.6044	3.55379
RTT	87.199	126.923	100.988	88.766	87.502	87.267	87.281
ideal RTT	45.0023	44.8212	45.0196	45.0026	45.0026	45.0026	45.0023
std dev Q time	0.680419	23.22	3.22624	0.599675	0.574554	0.640407	0.683617
std dev LPH	0.484686	1.75618	0.773421	0.694075	0.683012	0.678686	0.675783

This simulation is run with 15% injection rate, which means packets are injected into the network by 15% of the time. The number of packets in this simulation is

1,000,000 and the control interval is 100 cycles. In this table, Q time is the amount of time packet stalled at the source node queue. RTT is the sum of Q time and network latency. It can be seen that, with increasing window sizes, the result gets closer to the baseline case, but none is better than the baseline case. It means that no performance improvement is observed. This is conceivable as uniformly random is a case where every core has about the same priority. It is already fair even without the throttling.

To test source throttling on an intrinsically unfair case, the parking lot case is simulated. The traffic pattern is shown as Figure 4.

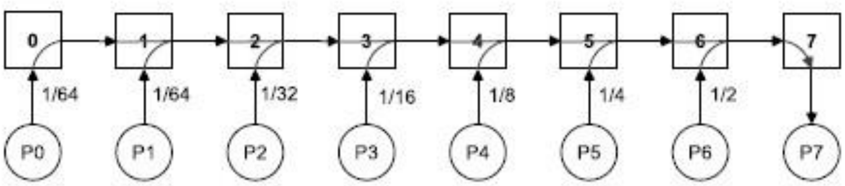


Figure 4. Parking lot case traffic pattern [16]

In the simulation, the network is 8 by 8 mesh. Figure above shows the traffic pattern in one line of the mesh. For every line, the left 7 nodes issue requests to the right-most node. The injection rate of every node varies as marked on the edge in the figure. The injection rate can be tuned simultaneously to create a congested case. The result is Table 3.

Table 3. Simulation results for parking lot case.

window size	baseline	3	4	5	6	7	8	9
Network latency	63.3769	59.9647	62.0633	62.9336	63.242	63.3398	63.3678	63.3754
Q time	2.25361	7.46924	3.80306	2.71884	2.3913	2.29102	2.26295	2.25514
RTT	65.631	67.434	65.866	65.652	65.633	65.631	65.631	65.630
ideal RTT	37.0025	37.0025	37.0025	37.0025	37.0025	37.0025	37.0025	37.0025
std dev Q time	0.861663	4.96089	2.07718	1.12088	0.892728	0.863543	0.862471	0.861487
std dev LPH	2.8938	2.90012	2.90242	2.89848	2.8955	2.89434	2.89392	2.89382
runtime	4440013	13380000	13470000	12245000	13060000	13170000	12835000	14260000
avg req time	65.63051	67.43394	65.86636	65.65244	65.6333	65.63082	65.63075	65.63054

The injection rate of P0 is 3%. The number of packets in the simulation is 1,000,000 and the control interval is 100 cycles. The simulation result is similar as that of the uniformly random case. We also tried some cases with different injection rate of each node, but similar result is observed. That is, when the window size keeps increasing, the result gets closer to the baseline case. Under the source throttling, the unfairness still exists. There is one bottleneck link in the network - the link between node 6 and node 7 in Figure 4. All traffic needs to pass through this link and source throttling cannot overcome this bottleneck. The TCP-Vegas algorithm actually hurts the long distance request, which makes the unfairness even worse.

For all of the simulation above, the priorities of all packets are the same. But in realistic cases, some packets have higher priority than others. The next simulation is the case with different packet priorities.

In an application, quite often some tasks must be finished before some others begin. Such precedence constraint can be modeled in a data dependency tree. For example, if request A has dependency on request B, request A cannot be issued until request B gets replied. Due to this constraint, request A has a lower priority than request B. A data dependency tree is built for the simulation and is shown in Figure 5.

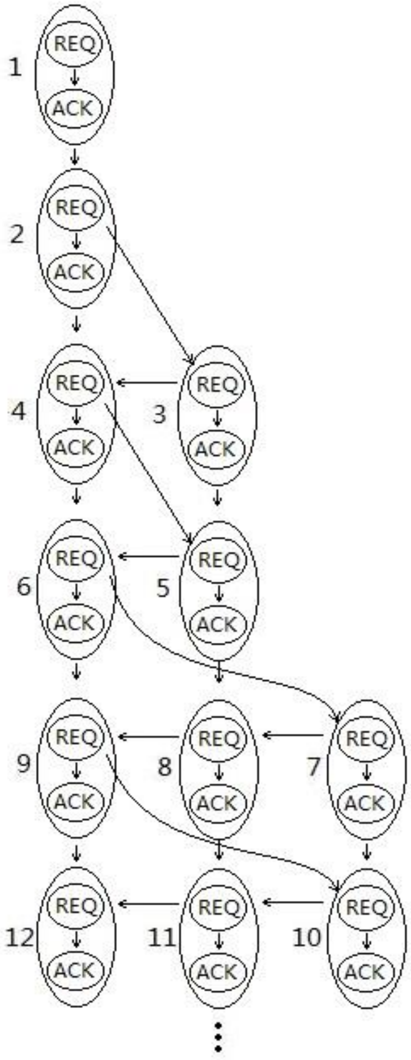


Figure 5. Data dependency tree in synthetic traffic simulation

In the figure, REQ means request and ACK means responses. Every request has a corresponding response. In the figure, if there is a data dependency between two packets, the two nodes are connected by an edge. The lower node cannot be issued until the upper node packet reaches its target. Every core has similar dependency tree for its requests. In this case, different packets have different priorities. The simulation result is shown in Table 4.

Table 4. Different packet priority case with dependency tree

	Baseline	ST(0.1, 0.15)
Network latency	154.909	157.543
Queue time	31.7794	47.4594
Runtime	26037	24519
LPH std dev	1.35262	1.1673
Q time std dev	13.9184	17.1804

The injection rate is 40%. The number of packets in the simulation is 70400 and the control interval is 100 cycles. The simulation result shows that both the network latency and source queuing time increased. Nevertheless, the total runtime is reduced by around 5.83%. It seems strange as lower network latency is expected after source throttling. Further analysis needs to be done for this case. More information is collected from the simulation and shown as Figure 6.

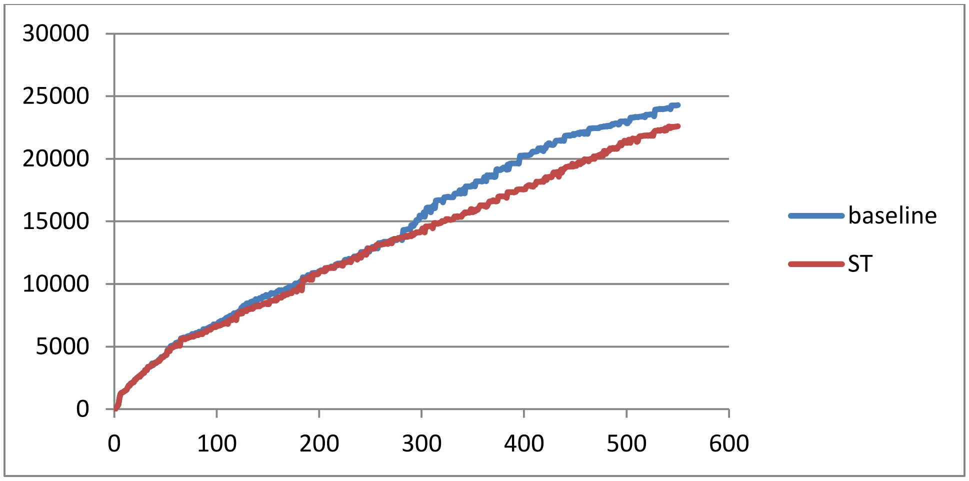


Figure 6. Injection time vs. packet ID

The horizontal axis is packet ID and vertical axis is the injection time of the corresponding packet. It can be seen that after 300 packets, the packets are injected into the network earlier if source throttling is conducted. We also check the injection rate over control interval, which is 500 cycles. The result is plotted in Figure 7.

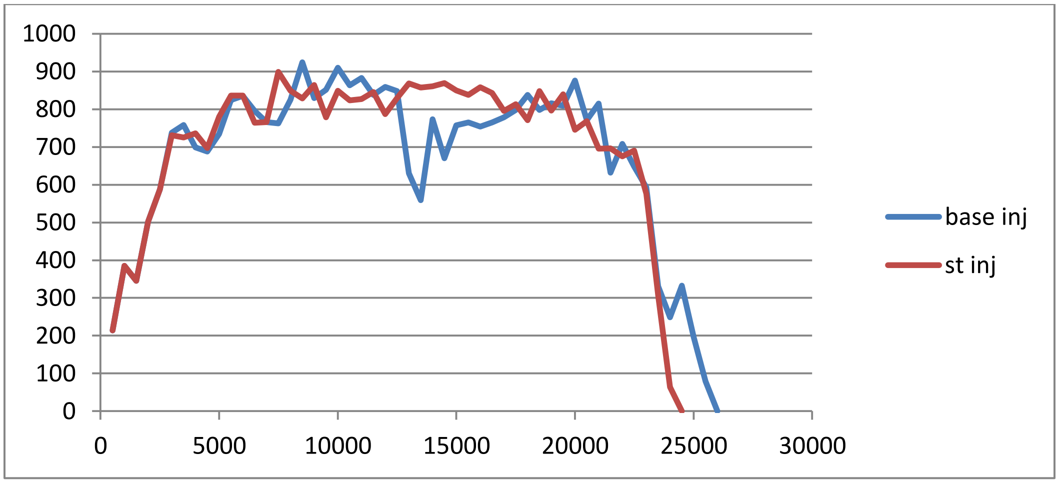


Figure 7. Packet injection number vs. cycles

It is clear that the injection rate has a drop at around 13000 cycles in the baseline. From then on, source throttling allows packets to be injected earlier than the baseline. The injection drop is caused by the data dependency. It means some packets are blocked from injection due to data dependency. Since the source throttling blocks the injection of some low priority packets, high priority packets can be injected earlier. There are less packets waiting compared with the baseline case.

For the network latency histogram in Figure 8, the RTT tends to be crowded in the middle range, between 150 cycles to 400 cycles, after source throttling. There are less very short RTT packets after source throttling, but there are also less very long RTT packets. The maximum RTT during overall runtime decreases from 1059 to 825 cycles. In this sense, source throttling does improve fairness.

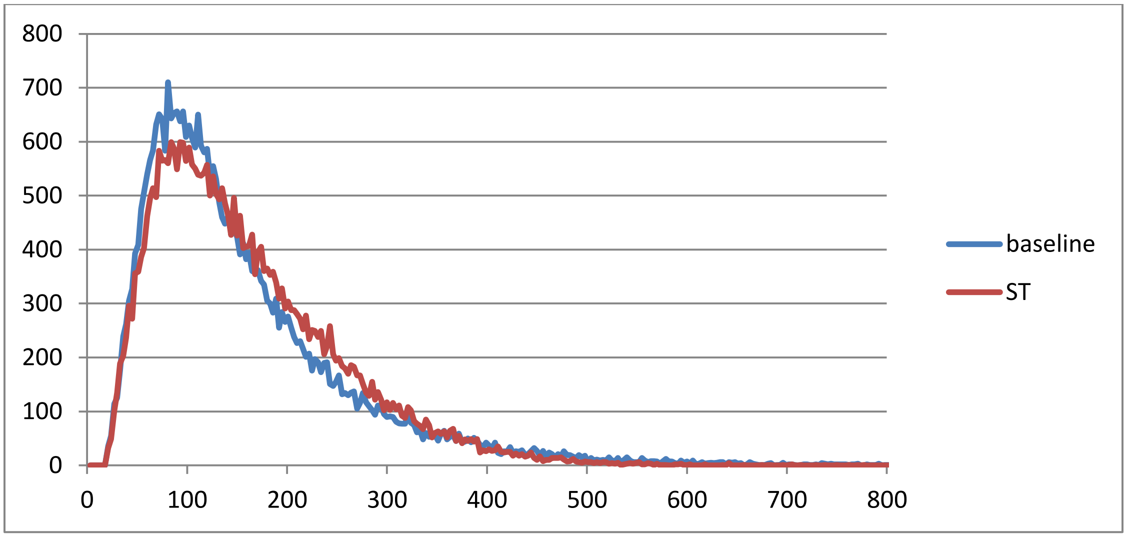


Figure 8. Network latency histogram

CONCLUSION: Data dependency may cause injection rate drop. Source throttling can prevent low priority packets from blocking high priority packets.

From this case, it is observed that data dependency is necessary for source throttling to show its effect. Then a series of simulations are run. The first simulation is to make perturbation of the destination of every packet. In the previous simulation, the destination is calculated by packet ID and core ID. This causes the traffic to have a moving hotspot. To make the case more general, for each case, 20 simulations are run. The average is used to measure the performance. For the 20 simulations, the simulator reads randomly from an existing file to generate packet destinations. We also vary the width of dependency trees to see the effect in Table 5.

Table 5. Perturbation of different tree width cases

tree width = 4			tree width = 8			tree width = 10		
Baseline runtime	ST runtime	improve	Baseline runtime	ST runtime	improve	Baseline runtime	ST runtime	improve
35190	35200	-0.028%	47177	47039	0.293%	54309	54090	0.403%
35366	35539	-0.489%	47884	47979	-0.198%	56151	55211	1.674%
35479	35312	0.471%	47408	46900	1.072%	55315	54552	1.379%
35313	35174	0.394%	47818	47505	0.655%	55452	54859	1.069%
35091	35260	-0.482%	47270	46970	0.635%	55155	55022	0.241%
35336	35228	0.306%	47548	47340	0.437%	56515	55435	1.911%
35266	35273	-0.020%	46995	47248	-0.538%	54078	55136	-1.956%
35398	35522	-0.350%	47315	47659	-0.727%	55398	54255	2.063%
35300	35157	0.405%	46797	47344	-1.169%	54852	53647	2.197%
35111	35183	-0.205%	46822	47307	-1.036%	54139	53864	0.508%
35490	35279	0.595%	48095	47380	1.487%	54967	55924	-1.741%
35195	35237	-0.119%	47972	47416	1.159%	54403	54097	0.562%
35446	35378	0.192%	47257	47205	0.110%	54964	54854	0.200%
35547	35386	0.453%	48028	47139	1.851%	54146	55428	-2.368%
35084	35115	-0.088%	47112	47100	0.025%	55443	54762	1.228%
35260	35069	0.542%	47599	48051	-0.950%	54551	54661	-0.202%
35114	35266	-0.433%	47366	47609	-0.513%	55452	54681	1.390%
35148	35111	0.105%	47412	47662	-0.527%	55434	54468	1.743%

Table 5. continued

35190	35502	-0.887%	47242	47294	-0.110%	54363	55381	-1.873%
35304	35343	-0.110%	47549	46852	1.466%	54513	54306	0.380%
705628	705534	**0.013%**	948666	946999	**0.176%**	1099600	1094633	**0.452%**

The injection rates of all three cases are 40%. And two preset threshold of source throttling is 0.05 and 0.1 for all above cases. The numbers of packets for these cases are 12800, 46080 and 70400. The control interval is 100 cycles. In the table above, it shows the runtime of baseline case and throttled case. The last line shows the average improvement. When the tree is narrow in width, like 4, there is hardly any benefit for source throttling. When the tree width increases, the benefit of source throttling becomes discernable.

CONCLUSION: Wider dependency tree width is necessary for source throttling to show effect.

The next simulation is to see the impact from different dependency tree depths.

Table 6. Simulation result of deeper tree (tree width = 8)

tree width = 8 pktNum = 46080			tree width = 8 pktNum = 92160		
Baseline runtime	ST(0.05,0.1) runtime	improvement	Baseline runtime	ST(0.05,0.1) runtime	improvement
47177	47039	0.293%	61679	61675	0.006%
47884	47979	-0.198%	62780	61756	1.631%
47408	46900	1.072%	61252	61233	0.031%
47818	47505	0.655%	62461	61891	0.913%
47270	46970	0.635%	59934	61161	-2.047%
47548	47340	0.437%	61010	61400	-0.639%
46995	47248	-0.538%	61687	60929	1.229%
47315	47659	-0.727%	61296	61539	-0.396%
46797	47344	-1.169%	61648	61167	0.780%
46822	47307	-1.036%	61625	60608	1.650%
48095	47380	1.487%	61694	61645	0.079%

Table 6. continued

47972	47416	1.159%	61101	61519	-0.684%
47257	47205	0.110%	61171	61916	-1.218%
48028	47139	1.851%	62913	61679	1.961%
47112	47100	0.025%	61966	62301	-0.541%
47599	48051	-0.950%	61896	62606	-1.147%
47366	47609	-0.513%	61410	61512	-0.166%
47412	47662	-0.527%	61591	61198	0.638%
47242	47294	-0.110%	62126	62202	-0.122%
47549	46852	1.466%	62119	61809	0.499%
948666	946999	**0.176%**	1233359	1231746	**0.131%**

Table 7. Simulation result of deeper tree (tree width = 4)

tree width = 4 pktNum = 12800			tree width = 4 pktNum = 25600		
Baseline runtime	ST(0.05,0.1) runtime	improvement	Baseline runtime	ST(0.05,0.1) runtime	improvement
35190	35200	-0.028%	40336	40303	0.082%
35366	35539	-0.489%	41126	41335	-0.508%
35479	35312	0.471%	40725	40579	0.359%
35313	35174	0.394%	41016	40898	0.288%
35091	35260	-0.482%	41051	41071	-0.049%
35336	35228	0.306%	40769	40998	-0.562%
35266	35273	-0.020%	40660	40816	-0.384%
35398	35522	-0.350%	40390	40852	-1.144%
35300	35157	0.405%	40838	40981	-0.350%
35111	35183	-0.205%	41134	41033	0.246%
35490	35279	0.595%	40964	40948	0.039%
35195	35237	-0.119%	40934	40904	0.073%
35446	35378	0.192%	40949	40858	0.222%
35547	35386	0.453%	41101	40700	0.976%
35084	35115	-0.088%	40760	40895	-0.331%
35260	35069	0.542%	40793	40825	-0.078%
35114	35266	-0.433%	40511	40763	-0.622%
35148	35111	0.105%	40601	40719	-0.291%
35190	35502	-0.887%	41089	41194	-0.256%
35304	35343	-0.110%	41106	40908	0.482%
705628	705534	**0.013%**	816853	817580	**-0.089%**

In Table 6 and Table 7, the injection rate for all cases above is 40%. The control interval is 100 cycles. From the two tables above, it is clear that varying tree depth makes almost no difference on the effect of source throttling. It is expected that the benefit permitted by data dependency does not change when the tree depth is doubled, as the overall runtime increases proportionally with or without source throttling.

CONCLUSION: Varying dependency tree depth does not affect the effect of source throttling.

Compared with other previous work (e.g., [3], [4], [5]), TCP-Vegas is somewhat more aggressive in the control policy, as it completely blocks requests from issuing when the source is throttled. By contrast, in a precious work, if time is divided into a set of time slots, the policy only blocks the NI for a certain percentage of the time slots. To reduce the aggressiveness of TCP-Vegas, we can use larger control intervals to make the window size change less frequently.

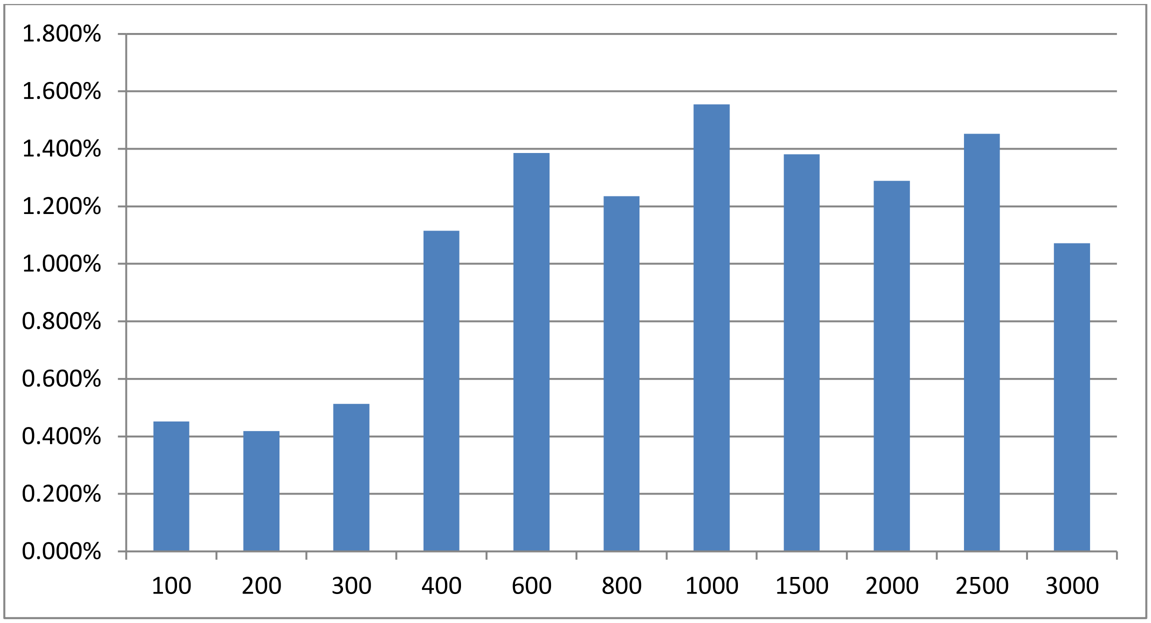


Figure 9. Runtime reduction percentage with different control interval

In Figure 9 above, horizontal axis is the control interval in cycles and vertical axis is the reduction in runtime compared with baseline case. This case is run with tree width equal to 10, which allows the largest benefit. The injection rate keeps being 40%. The original control interval is 100 cycles as it is about the same length of RTTs. It can be seen that the benefit from source throttling is greater when control interval is around 1000 cycles.

CONCLUSION: The original TCP-Vegas is too aggressive in terms of the frequency of window size change. More benefit can be observed if the window size is changed less frequently.

Then it is wondered that if any benefit could be obtained with less data dependency. Another data dependency tree is built as Figure 10 below.

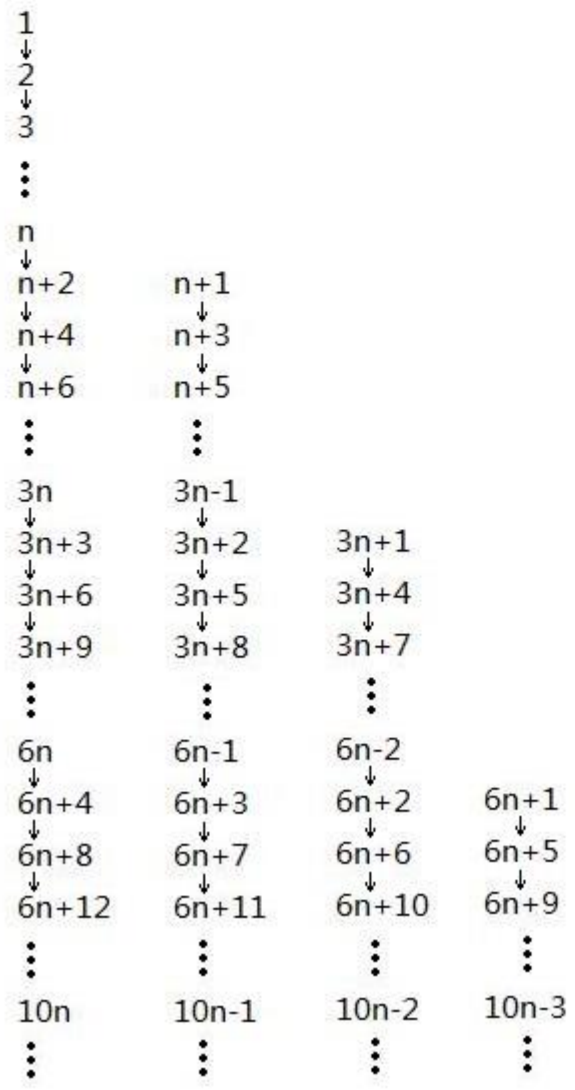
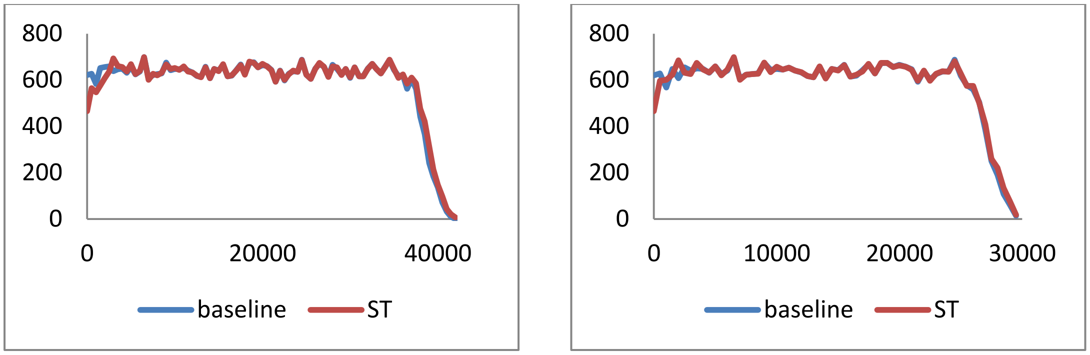


Figure 10. Data dependency tree with less dependency

It is a case with very much less data dependency. The simulation in Figure 11 results show that there is no injection rate drop in the middle and no obvious benefit can be observed.

The injection rate in this case is 40%. The control interval is 100 cycles.



(a) Tree width = 12

(b) Tree width = 10

Figure 11. Packet injection number vs. cycles (with less dependency)

CONCLUSION: If data dependency is low, source throttling provides almost no performance improvement.

The last simulation is on synthetic traffic where data dependency exists between different cores. According to statistics of realistic benchmarks, there is about one data dependency every hundreds of requests. In the simulation, cases are tried with inter-core-dependency every 500, 800, 1000, 1500 and 2000 requests.

Table 8. Simulation result of inter-core-dependency case

Inter-core dependency frequency	Improvement
1/500 reqs	1.68%
1/800 reqs	1.96%
1/1000 reqs	1.22%
1/1500 reqs	2.29%
1/2000 reqs	1.89%

In Table 8, the cases are built based on previous cases with less data dependency. There is no benefit for those cases, which makes the benefit of inter-core-dependency clear. The dependency tree width is 10 and the injection rate is 40%. The number of packets is 70400 and the control interval is 100 cycles. The performance improvement is around 2%. By tracing the injection rate of one case as Figure 12 and Table 9, it is found that reason for this improvement is almost the same as the case of table 4. However, the improvement is less than table 4, which is over 5%.

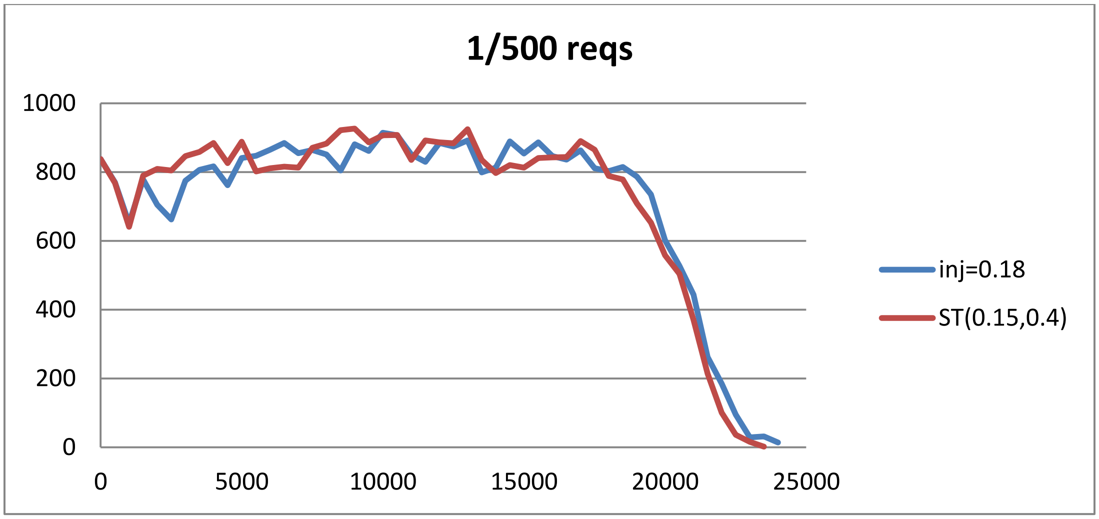


Figure 12. Packet injection number vs. cycles (with inter-core dependency)

To find out what happens around 1500 cycles, dependent packets are traced.

Table 9. Trace of dependent packets

Baseline:
bt-core-dep 4 ready @507
bt-core-dep 4 complete @1931

ST:
bt-core-dep 4 ready @731
bt-core-dep 4 complete @1040

It can be found that a certain dependent packet is completed earlier when throttled. Instead of injection rate drop, the injection rate increases a little bit at cycle 1000. The overall runtime is reduced.

CONCLUSION: Source throttling is still effective when there are inter-core-dependencies.

6. FULL SYSTEM SIMULATION

This section introduces the full system simulation, including the full system simulator gem5 and its configuration, PARSEC benchmark and the simulation results and analysis.

6.1 Full System Simulator and Simulation Environment

Different from the simulator used in synthetic traffic simulation, the full system simulation needs a simulator that models not only the network and cache system, but also the CPUs. The simulator should act the same as the realistic applications on CMP. It can boot up an operating system on the CPU, and run an application on it. Lots of information on all kinds of metrics can be captured. Not surprisingly, the runtime of full system simulation is much longer than ocin-tsim.

The simulator is gem5 [14]. It is an open source simulator for chip multi-processor. It can be configured to different architectures, such as alpha, X86 and ARM, as well as different cache coherence protocols, like MESI CMP directory and MOESI hammer protocols. In addition, CPU frequency and network frequency can be set to different values. Since it is a full system simulator, there are a lot of parameters in the whole system that can be changed when setting up the configuration.

Some major configurations are listed in Table 10.

Table 10. Full system simulation environment configuration

Network	32 nodes, 4 by 4 concentrated mesh, dimension order routing
Core model	Out-of-order, 128-entry instruction window, 16 miss buffers, stall when buffers are full
Coherence protocol	MOESI hammer snooping protocol
Benchmarks	PARSEC benchmarks
Baseline network	4 VCs per network port, 5 flits per VC
Router	5 pipeline-stage router, round robin

In this simulation, concentrated mesh is used. The topology is shown as Figure 13 below.

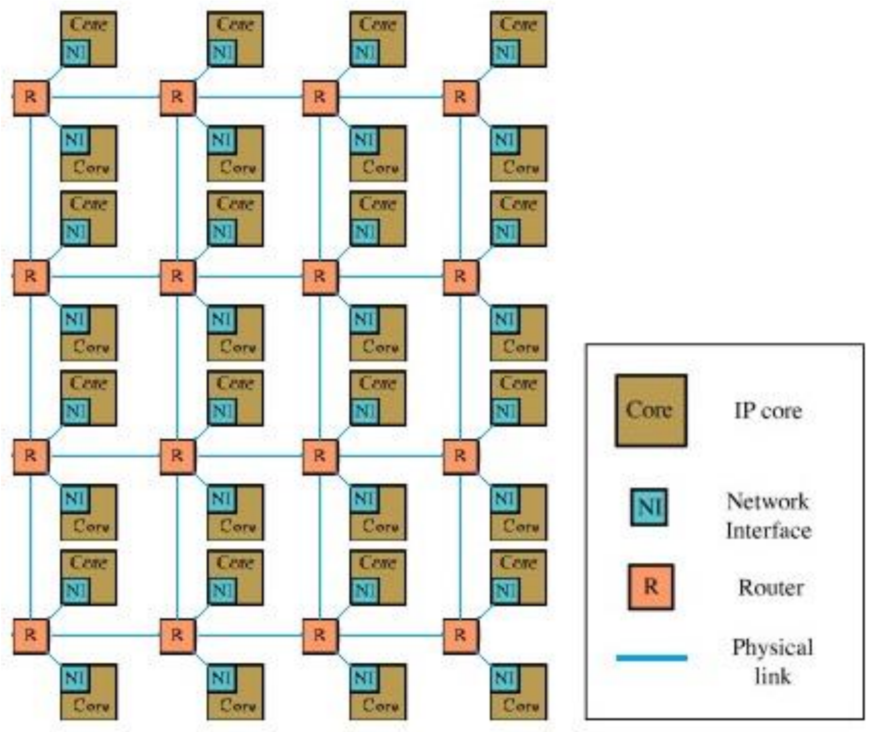


Figure 13. Concentrated mesh topology

It can be seen that two cores share one router. More traffic will be injected into the network. The network will be more congested. Upon that source throttling may be more useful.

6.2 PARSEC Benchmark

In the full system simulation, realistic benchmark can be run on the simulator. The PARSEC benchmark suite is used in this work.

PARSEC is a benchmark suite for CMPs that focuses on emerging applications. It includes a diverse set of benchmarks from different domains such as interactive animation or systems applications that mimic large-scale commercial benchmarks [15].

For the benchmarks that are used in our simulation are briefly introduced here. Blackscholes comes from financial analysis. It has little communication among cores. Canneal employs a simulated annealing algorithm to minimize the routing cost of a chip design. Both Freqmine and Streamcluster are data mining processes and have a lot of traffic in the network. Vips and x264 are abstracted from media processing.

Each benchmark trace mainly consists of four parts. The first is operating system boot up. The second part is a period of set up before running the application. Then there is the Region of Interest (ROI). This is a parallel phase in the benchmarks. Most researches are focused on ROI. Usually simulation will run in simple CPU mode to fast boot the system and switch to parallel mode at the beginning of ROI to simulate the hardware. The last part is tailing process. Source throttling is only applied in ROI, which shows distinctive characteristics among different cases. In the simulation, the start cycle of ROI and end cycle of ROI is set using a hook library in the disk loaded by the simulator.

6.3 Simulation Results and Analysis

From the simulation results, we first build the latency histogram, which helps to characterize different request types. The latency histogram of Blackscholes is shown in Figure 14.

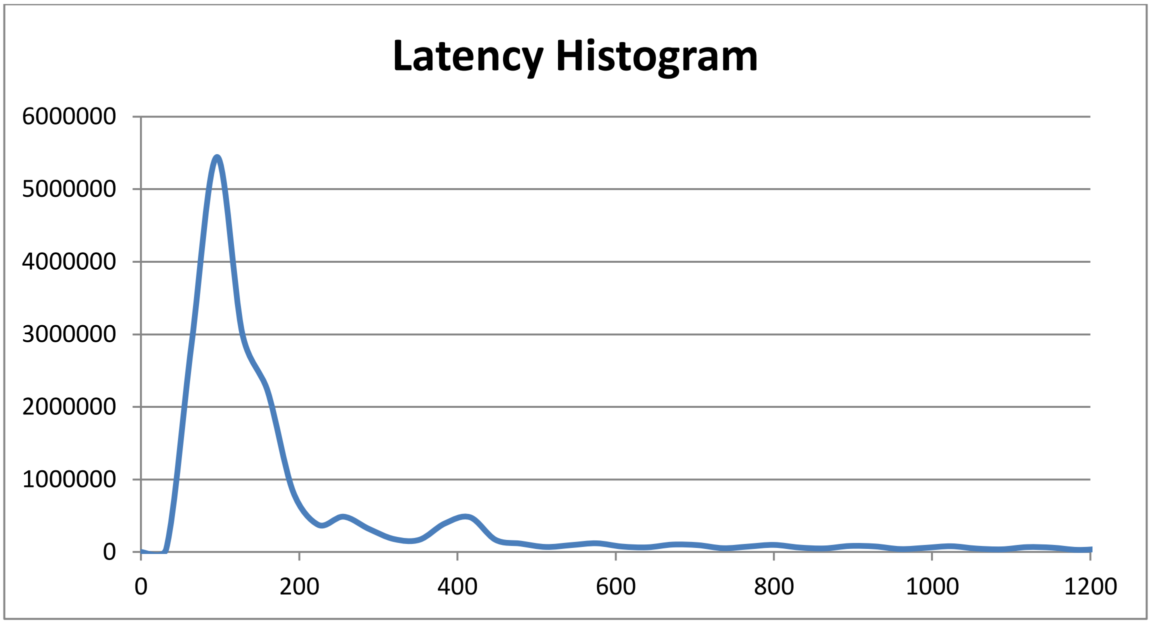


Figure 14. Latency histogram of Blackscholes

The histogram contains two sets of peaks, which correspond to two different types of communication requests. The main peak at 50 cycles to 200 cycles is the cache coherence traffic. Their latencies are mostly small as they are restricted to short distance on-chip communication. The small peak around 400 cycles on the right is traffic involving memory access. As there is only one memory controller on the CMP, all memory accesses have to go to the node where the memory controller is placed. The memory access time from the controller is a fixed number, 300 cycles, which makes the peaks way from the peak of coherence traffic. We need to distinguish the two types of traffic because the ideal RTTs for these two different types of traffics are different.

CONCLUSION: There are two types of traffics networks, cache coherence traffic and memory access traffic. They can be throttled together or separately. But as we can see, the amount of memory access traffic is very small compared with cache coherence traffic. Throttling separately will not have too much difference but will cost more in the overhead. So in this work the simulation will both types of traffic together.

The second simulation in this section is changing the ratio of core frequency over network frequency to obtain the more congested case. In this simulation, core frequency is fixed at 1GHz, while network frequency can be quite different from case to case. This simulation is run to find the relationship between injection rate and core versus uncore frequency ratio. The simulation results are listed below as Table 11, Table 12 and Table 13.

Table 11. Network results with different network frequency (Blackscholes)

ratio	net freq	RTT1	RTT2	Queue Time	LPH	inj avg	ROI length
0.25	4G	116.09	127.164	11.074	10.435	0.001485	91893444
2	500M	123.275	135.575	12.3	11.7952	0.001481	213263226
2.857	350M	117.144	133.141	15.997	11.9456	0.001474	311622993
3.125	320M	122.131	134.628	12.497	11.9556	0.001473	332950418
3.571	280M	123.036	135.395	12.359	11.9203	0.001431	362314078
4.545	220M	126.23	138.679	12.449	12.7141	0.001459	469233081
7.143	140M	120.913	133.401	12.488	11.9241	0.001541	752225026
12.5	80M	122.582	135.438	12.856	12.1196	0.001487	1225637913

Table 12. Network results with different network frequency (Swaption)

ratio	net freq	RTT1	RTT2	Queue Time	LPH	inj avg	ROI length
0.25	4G	109.71	120.278	10.568	10.1306	0.002286	199436796
1	1G	113.116	125.101	11.985	11.3965	0.001897	329279362
2	500M	116.832	129.506	12.674	11.4151	0.001867	490095276
2.857	350M	118.031	130.782	12.751	11.7322	0.001688	725837048
3.125	320M	119.321	131.865	12.544	11.8542	0.00174	767219653
3.571	280M	119.561	132.183	12.622	11.7135	0.001591	863039151
4.545	220M	124.573	137.305	12.732	12.509	0.001653	1099207696
7.143	140M	122.067	134.683	12.616	11.8743	0.001635	1625185527
12.5	80M	121.4	134.346	12.946	12.1006	0.001609	2720429021

Table 13. Network results with different network frequency (Streamcluster)

ratio	net freq	RTT1	RTT2	Queue Time	LPH	inj avg	ROI length
2	500M	128.336	138.554	10.218	11.1453	0.00319	1216566350
2.857	350M	128.145	139.114	10.969	11.2976	0.002644	1727283312
3.125	320M	131.503	142.515	11.012	11.3298	0.002607	1849381631
3.571	280M	132.728	144.381	11.653	11.4654	0.002454	2096949197
4.55	220M	134.463	146.218	11.755	11.9908	0.002221	2680568930
6.25	160M	126.991	139.478	12.487	11.7003	0.002119	3489948806
7.14	140M	127.803	139.908	12.105	11.7165	0.002103	3915168459
12.5	80M	127.52	140.743	13.223	11.9651	0.001864	6450388537

In this simulation, the second column in the table is the network frequency. RTT1 is the RTT without source queuing time. RTT2 includes the queuing time at the source. Standard deviation of latency per hop and average injection rate are also listed.

It can be found that when network frequency keeps increasing, the injection rate also increases, but not proportionally. When the network frequency doubles, the injection rate increases a little.

This phenomenon reveals an issue. That is, a core may have self-throttling. All the RTTs in the table above are based on network cycles. When they are converted to

CPU cycles, they can be as large as one thousand cycles. From CPU's point of view, such high RTTs are interpreted as network congestion. Then, a core naturally slows down issuing new requests into the network.

CONCLUSION: A core has implicit self-throttling that may overlap with the function of our explicit source throttling.

The third simulation is applying TCP-Vegas based source throttling on NoC. In this simulation is conducted on multiple benchmarks.

Table 14. Source throttling results vs. baseline results

	Blackscholes		Swaption		Streamcluster	
	baseline	ST	baseline	ST	baseline	ST
ROI start	5676227277	5676227277	5621312944	5621312944	5621207555	5621207555
ROI end	5805025918	5814526045	5960101586	5960880541	6199319536	6215786188
ROI length	128798641	138298768	338788642	339567597	578111981	594578633
Runtime	5853508340	5863069066	5969061237	5969608122	6207941902	6224694656
LPH var	87.7723	100.909	72.1854	75.9538	50.1769	37.2955
LPH avg	11.8174	11.9293	11.3854	11.4215	10.7793	10.0999
RTT1	144.764	148.503	131.942	130.829	142.621	143.04
RTT2	132.498	136.44	120.003	118.76	132.682	135.104
Queuing time	12.266	12.063	11.939	12.069	9.939	7.936

In Table 14, RTT1 and RTT2 still have the same meanings as precious table. It can be found that for all three benchmarks, runtime of application is increased with source throttling. Only Streamcluster shows reduction on LPH variation. This means TCP-Vegas based source throttling hardly have benefit on realistic benchmarks.

To find out why there is no benefit on these benchmarks, a further experiment is done. It checks the number of outstanding requests issued by cores and the number of requests at the network interface, which is also MSHR occupation for last level cache.

Table 15. Some characteristics of PARSEC benchmarks

	Max outstanding request number	Max LLC MSHR occupancy	Average LLC MSHR occupancy
Blackscholes	16	1	0.0434
Freqmine	11	7	0.4166
Swaption	6	1	0.0178
Streamcluster	16	8	3.9252

According to the conclusion in section 5, source throttling is effective when the LLC MSHR occupancy is at least 6. PARSEC benchmarks do not meet the requirement as Table 15 shows. This is a main reason that source throttling does not show benefits in these cases.

CONCLUSION: PARSEC benchmarks usually do not have a large number of concurrent outstanding requests, which are necessary for the TCP Vegas based source throttling to be effective.

7. CONCLUSIONS

In this research work, it is found that there is potential benefit for source throttling in NoC under certain conditions. The number of concurrent outstanding requests needs to be large. But for practical benchmarks extracted from real life, it is hard to meet the requirement. So there is hardly any benefit for source throttling in full system simulation. In summary, source throttling can hardly help any system performance for practical benchmarks.

REFERENCES

[1] E.Baydal, P. Lopez, and J. Duato, “A Congestion Control Mechanism for Wormhole Networks,” *Euromicro Workshop on Parallel and Distributed Processing*, Mantova, Italy, page 19-26, Feb. 2001.

[2] E. Gran, M. Eimot, S. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, “First Experiences with Congestion Control in InfiniBand Hardware,” *Parallel & Distributed Processing Symposium*, Atlanta, Georgia, page 1-12, Apr. 2010.

[3] G. Nychis, C. Fallin, T. Moscibroda, and O. Mutlu, “Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need?” *ACM Workshop on Hot Topics in Networks*, Monterey, California, page 12, Oct. 2010.

[4] G. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan, “On-Chip Networks from A Networking Perspective: Congestion and Scalability in Many-core Interconnects,” *Special Interest Group on Data Communication*, vol. 42, no. 4 (2012), page 407-418, Aug. 2012.

[5] K. Chang, R. Ausavarungnirum, C. Fallin, and O. Mutlu, “HAT: Heterogeneous Adaptive Throttling for On-Chip Networks,” *International Symposium on Computer Architecture and High Performance Computing*, New York City, New York, page 9-18, Oct. 2012.

[6] W. Dally and B. Towles, “Principles and Practices of Interconnection Networks,” *Design Automation Conference*, Las Vegas, Nevada, Jun. 2001.

[7] S. Scott and G. Sohi, “The Use of Feedback in Multiprocessors and Its Application to Tree Saturation Control,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 4 (1990), page 385-398, Jan. 1990.

[8] M. Thottethodi, A. Lebeck, and S. Mukherjee, “Self-Tuned Congestion Control for Multiprocessor Networks,” *International Symposium on High Performance Computer Architecture*, Nuevo Leone, Mexico, page 107-118, Jan. 2001.

[9] Z. Lu and A. Jantsch, “TDM Virtual-Circuit Configuration for Network-on-Chip,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no.8 (2008), page 1021-1034, Aug. 2008.

[10] J. Lee, M. Ng, and K. Asanovic, “Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks,” *International Symposium on Computer Architecture*, Beijing, China, page 232-238, Jun. 2008.

[11] B. Grot, S. Keckler, and O. Mutlu, “Preemptive Virtual Clock: A Flexible, Efficient and Cost-effective QoS Scheme for Networks-on-Chip,” *IEEE/ACM International Symposium on Microarchitecture*, New York City, New York, page 268-279, Dec. 2009.

[12] D. Kroft, “Lockup-free Instruction Fetch/Prefetch Cache Organization,” *International Symposium of Computer Architecture*, Los Alamitos, California, page 81-87, May. 1981.

[13] S. Shakkottai, and R. Srikant, “Network Optimization and Control,” *Foundations and Trends in Networking*, vol. 2, no. 3 (2007), page 271 – 379, Jan. 2008.

[14] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, and D. Wood, “The GEM5 Simulator,” *ACM Computer Architecture News*, vol. 39, no. 2 (2011), page 1-7, May. 2011.

[15] C. Bienia and K. Li, “PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors,” *Workshop on Modeling, Benchmarking and Simulation, Association for Computer Machinery*, New York City, New York, Jun. 2009.

[16] M. Lee, D. Abts, and J. Lee, “Approximating Age-based Arbitration in On-Chip Networks,” *International Conference on Parallel Architectures and Compilation Techniques*, Vienna, Austria, page 575-576, Sep. 2010.

[17] S. Prabhu, B. Grot, P. Gratz, and J. Hu, “Ocin_tsim – DVFS Aware Simulator for NoCs,” *Workshop on SoC Architecture, Accelerators and Workloads*, Bangalore, India, Jan. 2010.