ISSN NO. 2320-5407

Journal homepage:http://www.journalijar.com
Journal DOI: 10.21474/IJAR01

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH

RESEARCH ARTICLE

Performance Analysis of Heterogeneous Adaptive Throttling Algorithm with Varying Throttling Rates

***Sneha Soman[1] and Febin P Jacob[2].**

1. Department of Computer Science and Information Systems, Mahatma Gandhi University, India.
2. Department of Computer Science and Engineering, College of Engineering, Trikaripur, India.

Manuscript Info

Manuscript History:

Received: 16 May 2016
Final Accepted: 19 June 2016
Published Online: July 2016

Key words:
Flow Control,
Source throttling,
Adaptive throttling,
Heterogeneous adaptive throttling,
Network on chip.

****Corresponding Author***

Sneha Soman.

Abstract

Networks-on-chips (NoCs) is an advanced form of an interconnect network for a System-on-chip (SoC). This communication substrate is a promising solution for common issues like scalability, flexibility etc. faced by a typical bus based SoC. However, load balancing is one issue faced by NoC that would eventually lead to congestion. Hence congestion avoidance has become a primary challenge. Congestion avoidance schemes explore various routing algorithms, designs of routers etc. Another strategy for congestion avoidance is flow control. Various source throttling techniques are used to achieve flow control. This work proposes a scheme where the applications are classified into two major groups, network intensive and network non intensive. Further, network intensive group is throttled with a heterogeneous throttling rate that yields better performance by analysing the overall network. This work tries to impose varying throttling rates for network intensive applications and performcomparison with state-of-art method. This work is evaluated on different traffics and evaluations show a stable and consistent performance when compared to previous methods.

Introduction:-

Moore's law is the fundamental guiding principle of computer architecture which stated that the number of transistors on a chip would roughly double each year. This led to the emergence of System-on-chip (SoC), an integrated circuit (IC) that ideally integrates all components of a computer into a single chip. It evolved as an integrated solution to the challenging design issues in various fields of electronic domain. A multi-core chip comprises of multiple cores along with on-chipcache banks, DRAM memory controllers, accelerators, etc. In a multi-core chip, efficient communication between the various components on the chip is critical to exhibit good performance. So as to enhance communication between the components on the chip, interconnect serves as primary communication substrate for all cores, caches and memory controller.

Buses and crossbars served as interconnect substrate. However, these architectures could not adequately scale the ever growing demands and faced limitations such as long wire delays, restriction of communication between two cores at a time and other such design complexities. Hence a paradigm shift was required. "The scalability and success of switch-based networks and packet-based communication in parallel computing and internet has inspired the researchers to propose the Network-on-Chip (NoC) architecture as a viable solution to the complex on-chip communication problems" (Naveen Choudhary, 2012), (W. J. Dally et al., 2001). NoC primarily serves cache misses and memory requests. Packets are hence generated and switched across the network. When multiple cores inject at high rates and packets compete for channels, packets may deluge the network and would eventually lead to congestion. Congestion would lead to low system throughput and thereby degrade the overall system performance.
To handle congestion, basically the factors to be considered: a traffic pattern, NoC communication architecture, and an algorithm which best satisfies user's objectives. The traffic patterns known ahead of time can be dealt with a scheduling algorithm. On the other hand, dynamic traffic patterns rely on the use of a routing algorithm with a varying degree of adaptation to route packets (Wen-Chung Tsai et al., 2011). Often it is said that the performance of

NoC communication architecture is determined by its flow-control mechanism. Addition of buffers to networks would significantly improve the efficiency of a flow-control mechanism as a buffer can decouple the allocation of adjacent channels. In cases where there is no buffer, the two channels must be allocated to a packet (or flits) during consecutive cycles, or the packet must be dropped or misrouted which is again a serious issue (W. J. Dally et al., 2004). When two packets compete for same channel, flow control tries to answer critical questions like, whom the channel should be allocated and what the other packet would do (Kevin Kai-Wei Chang et al., 2012). Flow control is achieved through source throttling technique. Source throttling refers to a mechanism where core is refrained from injecting packets into the network. It makes the packets to wait a little longer.

Previous works try to achieve source throttling through various strategies that usually take the factors like application awareness or network load awareness or both and further try to throttle application with constant throttling rate (Kevin Kai-Wei Chang et al., 2012), (George Nychis et al., 2010), (G. Nychis et al., 2001), (M. Thottethodi et al., 2012), (Eiman Ebrahimi et al., 2010). In heterogeneous environment applications vary and also differ from each other in behavior. Cases were observed where same application behaves differently in different epochs. In such environments, restricting heterogeneous application to a single global throttling rate seems to be a strict throttling strategy. Hence, the idea is to introduce heterogeneity in throttling rate as well. Every core could be provided with an individual throttling rate based on its behavior in every epoch. This work tries to provide a varying throttling rate to the heterogeneous application and thereby tries to analyze the application performance.

Background:-

An energy efficient NoC architecture that would exhibit remarkable performance has always been the focus of many researchers. Fine-tuning a system to maximize system performance and minimize energy consumption includes multiple trade-offs that have to be explored (J. Kim et al., 2005). Source throttling is one of the techniques that is used to limit the traffic and thereby control congestion. Many works (George Nychis et al., 2010), (G. Nychis et al., 2001), (M. Thottethodi et al., 2012), (Eiman Ebrahimi et al., 2010), have been proposed that adopts the source throttling mechanism. From the previous works it is evident that source throttling mechanismprimarily aims at reducing the injection of packets into the network. Throttling is implemented either by identifying the application behavior or by estimating the network state. All the mentioned works try to achieve source throttling by anyone of the method. However on analyzing the shortcomings it is clear that a good strategy should consider both application awareness and network load.

Heterogeneous adaptive throttling (HAT), is a novel method that tries to combine application-aware source throttling and dynamic network load-aware throttling adjustment to reduce network congestion (Kevin Kai-Wei Chang et al., 2012). It effectively resolves the conflicts of hard throttling schemes mentioned by the previous paper (M. Thottethodi et al., 2012). HAT considers the application characteristics and classifies the applications primarily into two groups, network intensive andnetwork non intensive group. Network intensive applications are further throttled with same throttling rate that may vary in every epoch. HAT claims the applications to be heterogeneous in nature. Even then the network intensive group of applications is throttled with a constant throttling rate. Application in every core tends to exhibit heterogeneous behavior. Applications are likely to generate different amount of packets. It is well evident that application has varying demands even whenthey fall under same category. Now since the applications have varying demands, applications expect highly dynamic strategies to meet the immediate requirements. Therefore it is ideal to set varying throttling rate in different workloads to throttle the applications.

Motivation:-

This work tries to understand how source throttling could effectively avoid congestion and thereby improve system performance. Past works on source throttling was key motivation factor (Kevin Kai-Wei Chang et al., 2012), (George Nychis et al., 2010), (G. Nychis et al., 2001). It is primarily motivated by the desire to extend the work heterogeneous adaptive throttling and evaluate its performance with varying throttling rate. It is a new method that primarily aims at setting a variable throttling rate rather than a strict global throttling rate for different workloads. Work is done with the objective that it should not degrade the system performance when compared to the paper HAT (Kevin Kai-Wei Chang et al., 2012) and should also exhibit significant performance improvement in other aspects as well.

Problem Statement:-
To effectively avoid congestion by studying the effect of throttling on “packet latency” by implementing Heterogeneous Adaptive Throttling Algorithm (HAT) with varying throttling rate.

Throttling At Varying Rates:-
This work is based on analyzing the heterogeneity of the applications, classifying the applications into two categories named, network intensive and network non intensive application and thereby throttle the applications whenever required, to reduce the congestion existing in the NoC. Hence, the entire work was divided into four primary stages, creation heterogeneous applications, and estimation of number of packets in each epoch, application classification algorithm and throttling rate adjustment for throttling applications. As an initial step heterogeneous traffic was created where a set of cores would generate less number of packets whencompared to other cores in the network. Every core would create and inject unique number of packets into the network. Once the traffic is created, the number of packets generated from each core has to be analyzed. Further, based on the number of packets generated, cores are sorted in the ascending order. For classification of applications a threshold is set based on which the applications would fall into categories, network intensive or network non intensive. The network intensive group is assigned with a throttling rate solely based on the entire network load and the individual application behavior. Once the throttling rate is determined, the flow of packets into the network has to be regulated. This is achieved by introducing a secondary buffer that would hold packets and would modulate the flow of packets based on the throttling rate.

Experimental Methodology:-
Booksim simulator was used for this work (Nan Jiang et al., 2002). A 64 core chip multi-processor (CMP) interconnected by using 2D mesh topology was modeled. Creation of a heterogeneous traffic was the initial step of the implementation as the simulator tends to create a homogeneous traffic. To achieve a mix of application characteristics where a set of cores would generate high number of packets than the remaining cores, the synthetic traffic generation algorithm was altered. An epoch of 1000 cycles was considered for effective study. This was done to know the rate at which every core tends to generate packets. Further this count is used to judge whether the application fallsin the network intensive category or network non-intensive category.

Now applications are to be analyzed and classified into two groups network intensive and network non intensive group.Initially the applications are sorted in ascending order based on the number of packets generated from each core. A counter is set that adds up the packet sum. An array (A), is used to store the number of packets that each core would generate during each epoch. The packet sum could be defined as, Sum [A(i), {A[0]+A[1]+A[2]+....+A[i-1]}]. The sum of the packets generated from every previous core and the numbers of packets generated by the current core. A threshold is set that is used to compare the value of the packet sum. The threshold setin this work is 9331. The criterion for the selection of threshold is detailed in 'section 6’. Once the packet sum is calculated it is compared with the pre-set threshold value. If the packet sum forCore-i is below the threshold the application at the core is considered as a network non intensive application, otherwise it is considered as a network intensive application. For ease of understanding a variable 'flag' is used. The flag is set to '0' when the value is below threshold and is set to '1' when it’s above the threshold. Hence applications were classified.

Deciding a throttling rate and thereby throttling the application are two crucial steps of this work.

Throttling rate adjustment:-
Once the application is classified, the network intensive application should be throttled. For throttling the application a varying throttling rate is used for applications with flag 1. Setting throttling threshold is explained in 'section 6'. Even though the application basically falls in either of two categories, network intensive and network non intensive, the applications are heterogeneous in nature. The network intensive group would again contain applications that exhibit heterogeneity in their behavior in every epoch. Rather than setting a constant throttling rate for all network intensive applications, it is ideal to set throttling rate based on the current scenario. Hence, the concept of varying throttling rate. Every application with flag 1 initially tries to set the throttling rate as 10%. Further it tries to increase the rate in accordance with the load with a step index of 10% till the throttling rate is below 70%. Once the throttling rate exceeds 70% the step index is 2% percentage and further above 90% it is 1% percentage increase (Kevin Kai-Wei Chang et al., 2012). Similarly, it would also decrease the throttling rate with the predefined step index while the

network load is below the threshold (Kevin Kai-Wei Chang et al., 2012). Now once the throttling rate is set, throttling is applied.

Throttling the application:-

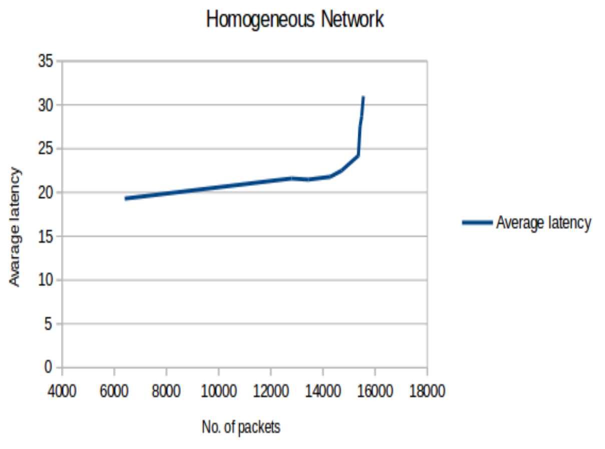
Once throttling rate is decided the applications classified as network intensive application should be throttled. Usually when a packet is generated from a core, the packet is injected into a buffer. Further the packets from the buffer are placed into the network by performingVC allocation and further get routed into the network. As mentioned in the paper (Kevin Kai-Wei Chang et al., 2012), HAT does not apply hard throttling. This means that when the core is being throttled, if the core is restricted to place packet into the network in the current cycle, then it should ideally retry for injection in subsequent cycle. To achieve the same, we tried to introduce a secondary buffer called "packet buffer". Now instead of injecting the packet into the primary buffer, the simulator is now modified to inject packets into the newly created "packet buffer". Further the flow from the packet buffer to the primary buffer is regulated. Apartfrom the normal injection rate (the rate at which the packets are transferred from core to packet buffer), a new injection rate is calculated that regulates flow from packet buffer to the primary buffer. The new injection rate is calculated based on the throttling rate. Additionally the size of the packet buffer is restricted to 8. If the size of the packet buffer happens to exceed the value 8, the core is temporarily restricted to generate packet.

Network Capacity Estimation and Threshold Calculation:-

Estimating the capacity of the network and there by calculating the threshold value that decides the application nature and rate at which the applications should be throttled are important aspects.

Network capacity estimation:-

Network capacity primarily refers to the amount of packets that a NoC could handle irrespective of the fact that the network is homogeneous or heterogeneous. To estimate the total number of packets that could be generated seamlessly without resulting a congestion, experiments were conducted. On a homogeneous network starting from '0.1' the injection rate was varied and the latency was noted. A steady increase in packet latency was noted while increasing the injection rate till 0.243 beyond which it increased abruptly showing that the network saturated. It is evident that beyond this the network would be saturated.So the estimation of the packets when the packet latency is 30 was considered ideal since it represented the capacity of the network. It was seen that on an average 15,552 packets are generated. Figure 1 shows an abrupt rise in number of packets after increasing the latency beyond 30.



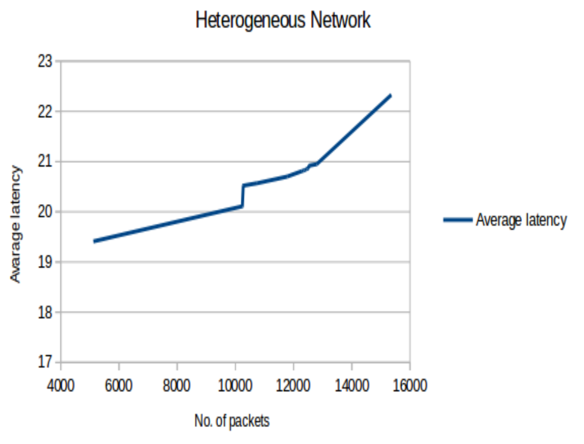


Figure 1: Capacity Estimate on Homogeneous TrafficFigure 2: Capacity Estimate on Heterogeneous Traffic

On a heterogeneous network similar procedure was followed. The injection rate was varied starting from '0.1' to '0.3'. Applications being heterogeneous in nature a steady rise in the latency were noticed till '0.31'. However beyond '0.3' there was an abrupt rise in the latency. Hence estimation of the packets when the packet latency is '27.44' was considered ideal and the total number of packets was estimated. It was seen that on an average 15,360 packets are generated. Figure2 shows an abrupt rise in number of packets after increasing the latency beyond 30. From the above two experiments it is observed that the network considered can handle up to 15,000 packets without causing a congestion. The graphs depict the convergence at approximately 15 k in either case.Therefore, the threshold was set as 60% of the total number of packets generated at the injection rate 0.243 in the homogeneous network. That is 60% of 15,552 which is approximately 9331.

Setting throttling threshold:-

To set a throttling threshold, the total packet sum calculated at each epoch is compared against the value 15,000. During this stage the system decides if the applications need to be throttled or not. If the packet sum is below 15000, the network is less congested and it indicates that either no throttling is required or in case if the network was previous throttled at a particular rate, then it is time to reduce the net throttling. However on contrast if thepacket sum exceeds 15000, throttling is required or the rate of throttling should again go up.

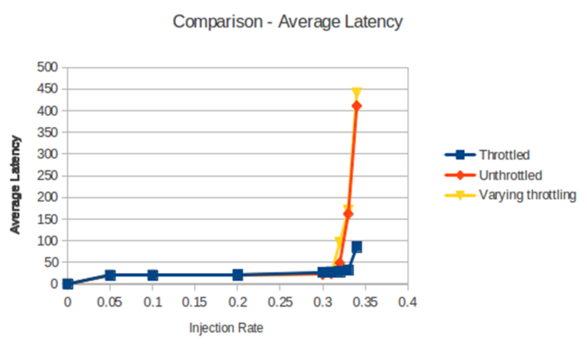
Experimental Analysis:-

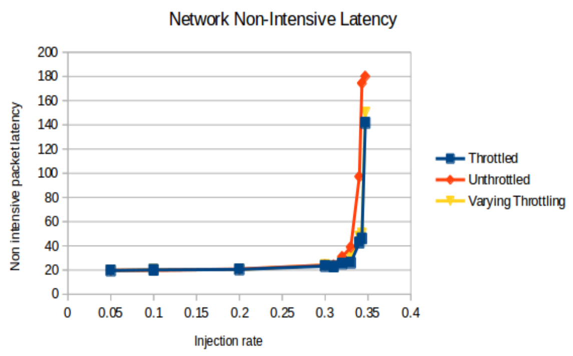
Traffic patterns like uniform, shuffle and tornado were considered for observation. Results were drawn for performance analysis. Performance analysis at different context is discussed below.

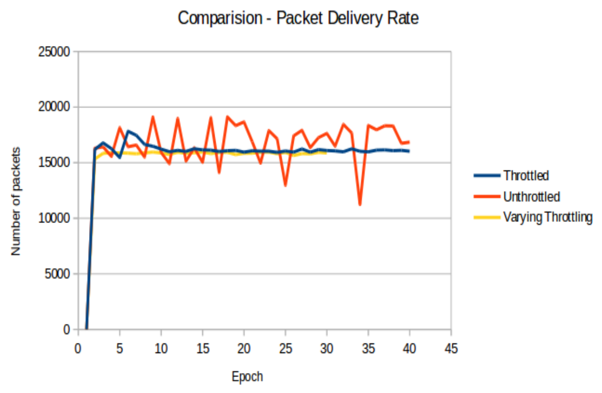
A. Uniform Traffic

1. Average Latency:-

Figure 3 depicts the average latency graph. The average latency of packets in an unthrottled, constant throttling rate and varying throttling rate environment are almost similar for lower injection rates. However, as the injection rates increases it is seen that throttled network exhibits better performance and has lower latency when compared to an unthrottled network.It is seen that constant throttling environment is much stable when compared to the varying throttling. However, varying throttling exhibits performance in midst of unthrottled and constant throttled environment.







Clockwise from left: Figure 3: Average latency - Uniform traffic
Figure 4: Network non intensive latency - Uniform traffic
Figure 5: Packet delivery rate – Uniform traffic

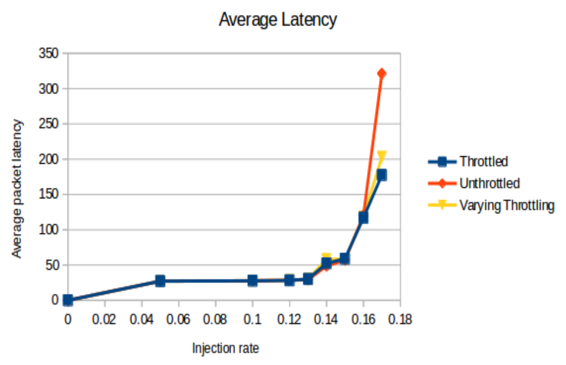
2. Packet Latency - Network Non Intensive applications:-
Throttling primary aims at reducing the packet latency of network non-intensive applications. Hence the average latency of network non-intensive applications were considered in an unthrottled environment, constant throttling rate and varying throttling rate. Figure 4 shows that the packet latency of non-intensive application in throttled network is lower and hence better when compared to the non-intensive applications in an unthrottled network.

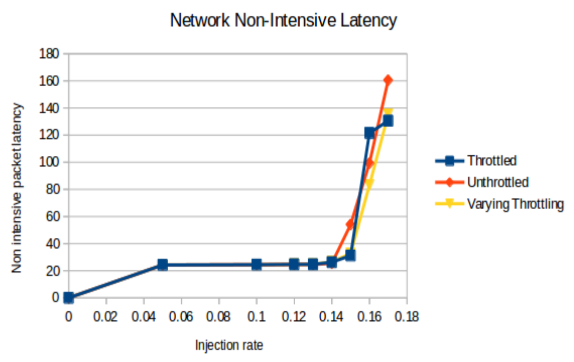
3. Packet delivery rate:-
The packet delivery rate refers to the number of packets delivered in each epoch. The packet delivery rate is consistent in the throttled network when compared to unthrottled network. Figure 5 depicts the packet delivery rate.

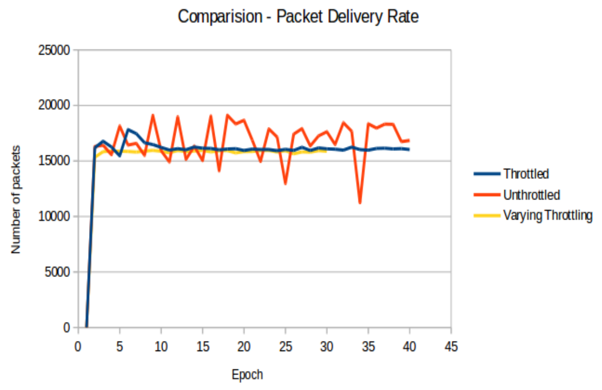
B. Tornado Traffic

1. Average Latency:-
Figure 6 shows that the average latency for a tornado traffic is highly consistent for a throttled network when compared to other network. As observed in case of an uniform traffic and the average latency of a varying throttled network lies between the throttled and unthrottled network. At lower injection rates the latency is similar in all the environments.

In varying environment average latency is stable and consistent. It also exhibits a performance which is better than an unthrottled network







Clockwise from left: Figure 6: Average latency - Tornado traffic
Figure 7: Network non intensive latency - Tornado traffic
Figure 8: Packet delivery rate – Tornado traffic

2. Packet Latency - Network Non Intensive applications:-

The network non intensive packet latency is consistent and better in throttled networks when compared to the unthrottled network as shown in figure 7. It is observed that for a tornado traffic varying throttling exhibits better performance at higher injection rate when compared to other environments. The network non intensive latency is consistent and stable for the varying throttling environment.

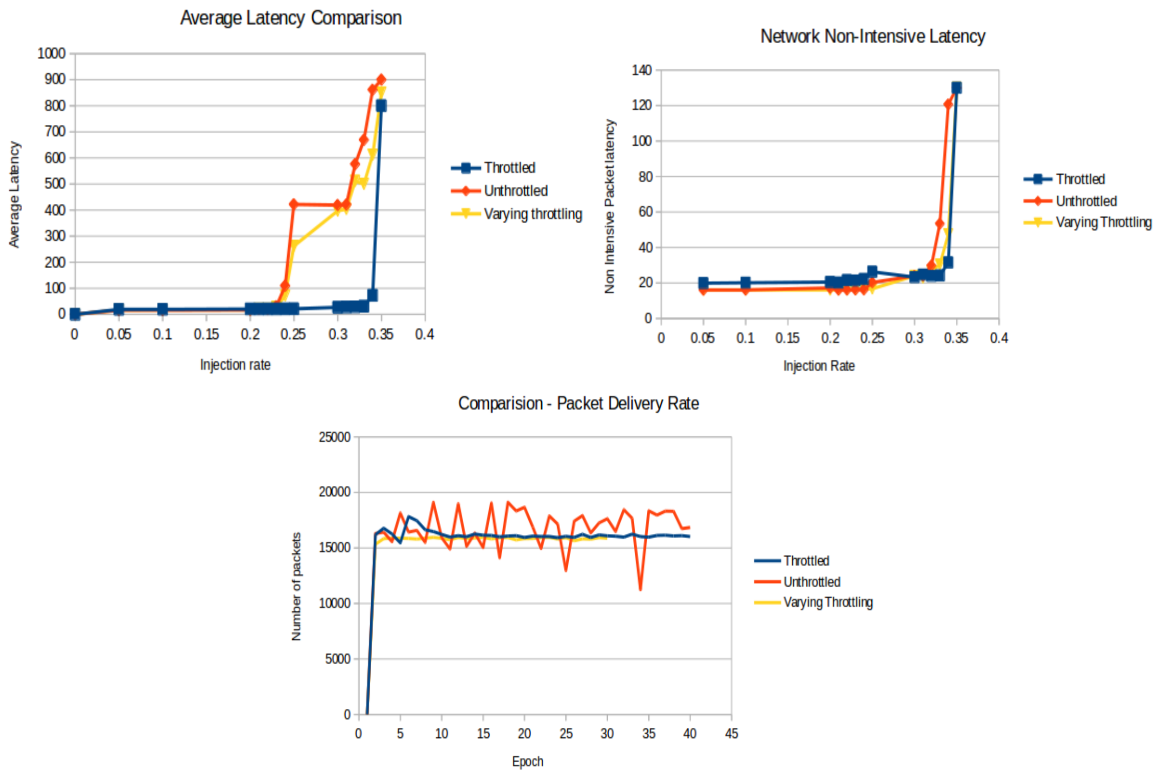
3. Packet delivery rate:-

As for other traffics, the packet delivery rate is consistent in the throttled network when compared to unthrottled network. Figure 8 depicts the packet delivery rate.

C. Shuffle traffic

1. Average Latency:-

The average latency for a shuffle traffic is highly consistent for a throttled network when compared to other network as shown in figure 9. As observed in the case of an uniform traffic, the average latency of a varying throttled network lies between the throttled and unthrottled network. In varying environment average latency could not be claimed consistent however it exhibits a performance which is better than an unthrottled network.



Clockwise from left: Figure 9: Average latency - Shuffle traffic
Figure 10: Network non intensive latency - Shuffle traffic
Figure 11: Packet delivery rate – Shuffle traffic

2. Packet Latency - Network Non Intensive applications:-

The network non intensive packet latency is consistent and better in throttled networks when compared to the unthrottled network. Figure 10 indicates that the network non intensive applications are benefited while throttling the network intensive applications.

3. Packet delivery rate:-

The packet delivery rate is consistent in the throttled network when compared to unthrottled network as shown in figure 11.

Conclusion:-

This work tries to achieve heterogeneous adaptive throttling with varying throttling rate. It is a mechanism that considers the application characteristics along with the network load and thereby decides to throttle the applications that generate high number of packets with heterogeneous throttling rates. This source throttling strategy aims at achieving a fair average latency along with overall improvement in the system performance. This work also proves that the applications thatfall under the network non intensive category are benefited by throttling the network intensive category. The results show scenarios where varying scheme exhibit consistency in latency along with high stability in packet delivery rate for different traffic types compared to state of art methods.

References:-

1. **Kevin Kai-Wei Chang,Chris Fallin, Rachata Asusavarungnirun and Onur Mutlu,** "HAT: Heterogeneous adaptive throttling for on-chip networks", Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), New York, NY,, Oct. 2012.

2. **George Nychis,Chris Fallin, Thomas Moscibroda and Onur Mutlu**, "Next generation on-chip networks: What kind of congestion control do we need?", in Montery, CA, USA at ACM, Oct. 2010.

3. **G. Nychis et al.,** "Self-tuned congestion control for multiprocessor networks.", HPCA-7,2001.

4. **M. Thottethodi et al.,** "On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects", SIGCOMM,Nov. 2012.

5. **J. Kim et al.,** "A low latency router supporting adaptivity for on-chip interconnects.", DAC-42, 2005.

6. **Eiman Ebrahimi,Chang Joo Lee, Onur Mutlu and Yale N. Patt,** "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems", , Pittsburgh, Pennsylvania,USA.,, March, 2010.

7. **Naveen Choudhary,** "Network-on-Chip: A New SoC Communication Infrastructure Paradigm", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue-6, , January 2012.

8. **W. J. Dally, B. Towles,** "Route packets, not wires: on-chip interconnectionne networks", in Proceedings DAC, pp. 684-689 , June 2001.

9. **Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen** "Networks on Chips: Structure and Design Methodologies", Journal of Electrical and Computer Engineering Volume 2012 (2012), Article ID 509465, 15 pages, October 2011.

10. **W. J. Dally and B. Towles,** "Principles and Practices of Interconnection Networks", Morgan Kaufmann, Waltham, Mass, USA, 2004.
11. **Nan Jiang, Daniel U. Becker, George Michelogiannakis, James Balfour, Brian Towles, John Kim, William J. Dally,** "A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator".