Balanced and efficient interconnects for Exascale supercomputers

Pablo Fuentes Sáez

supervised by
Dr. Ramón BEIVIDE and Dr. Enrique VALLEJO

Doctor of Philosophy
Departamento de Ingeniería Informática y Electrónica
Universidad de Cantabria
April 2017

Abstract

The need for more accurate or complete scientific results demands an increase in computing power to achieve Exascale machines, capable of performing 10^{18} floating point operations per second. One of the approaches to tackle Exascale computing is to increase the number of nodes in the machine, placing stronger demands in the system interconnect. The impact of the interconnection in HPC systems increases further with the surge of BigData applications, which have high network communication demands and different behavior than traditional HPC workloads, with a higher volume of communications and a more even distribution. Performance limitations can be reasonably expected to scale up with the network size, and some of them are likely to translate from system networks to the networks-on-chip within high-performance nodes.

This thesis introduces a synthetic traffic model of the communications in the Graph500 benchmark for BigData applications. The use of this traffic model simplifies the evaluation of data-intensive applications and their needs, and permits to predict the behavior in larger machines than currently available. An analysis of the benchmark communications shows a higher dependency with the network throughput than in traditional HPC applications.

Both BigData and HPC workloads can be significantly affected for the fairness in the network usage. This work conducts an analysis of the throughput fairness and evaluates the impact of different implicit and explicit fairness mechanisms. The fairness analysis and the evaluation of two proposed mechanisms have been performed through several synthetic traffic simulations in a 2-level hierarchical Dragonfly network with more than 15,000 nodes. Dragonflies are one of the high-radix, low-diameter network topologies proposed for Exascale system interconnects, and so far the only to have been implemented in a commercial system. A novel adversarial-consecutive traffic pattern is introduced for the evaluation of the throughput fairness, which particularly stresses the links in one of the routers of each group in the Dragonfly.

Results with the synthetic traffic model prove a significant constraint from the network throughput. They also evidence the existence of different communication distributions depending on the number of processes and their mapping to the network nodes. Throughput unfairness can limit average performance figures and even lead to starvation at those routers that become a bottleneck under adversarial traffic scenarios. Prioritizing in-transit traffic over new injections favours network drainage and reduces network congestion but is disadvantageous with adaptive routing, because it prevents injection from bottleneck routers and aggravates throughput unfairness.

Two mechanisms are proposed to improve network performance and simplify the router implementation. The first mechanism is the improvement in the detection of adversarial traffic patterns through contention information, using a metric based on contention counters. Four different implementations relying in this metric are evaluated. Evaluation results show that the use of contention counters provides competitive performance and much faster adaption to traffic changes, avoiding routing oscillations typical of congestion-based adaptive routing mechanisms.

The second proposal is a novel mechanism denoted *FlexVC*, which relaxes virtual channel restrictions in the deadlock avoidance mechanism. FlexVC reduces the number of buffers required, and provides a more balanced use. It also allows to employ more resources than strictly required by the routing and deadlock avoidance mechanisms, in order to provide higher performance. FlexVC improves performance with all routing mechanisms under each of the traffic patterns evaluated. Simulation results indicate that the performance benefits of FlexVC remain similar or improve when adaptive routing is used instead of oblivious routing, and FlexVC saves more resources with adaptive routing than with oblivious routing. FlexVC can be combined with contention counters to improve the identification of traffic scenarios with in-transit adaptive routing, achieving the best overall performance while halving the number of buffers required in the router.

Acknowledgment

This thesis represents a long journey, one that I could not have started were it not for my advisors, Mon and Enrique. Mon, you gave me the chance to enter this research group and over the years you have been supportive and friendly, always willing to orient me and to share your knowledge. Enrique, you have worked with me hand in hand every time, ensuring the works came to fruition; I hope your meticulousness and effort have paid off with this work. To both of you, a sincere thank you; this thesis is as yours as mine, and I am really lucky for having had you as advisors.

I also have to thank the help of Cyriel, José Luis and Mitch as mentors in some of the different works included here. Cyriel, many thanks for the incredible opportunity that you gave me (not only once, but twice) to come to Zurich and be a part of the amazing IBM ZRL family, where I have had some of the best moments of my life. José Luis, thank you for your patience and calm facing my countless questions and doubts, and your willingness to explore uncharted territories. Mitch, thank you for agreeing to mentor me and for your guidance and suggestions; unfortunately, some of them could not be included here, but I hope to explore them in a near future. I also have to thank Mateo for being the spark that ignited many of the works here included.

I want to thank German and Zuzana for opening us their doors and being the best hosts we could think of. Danke schön, ihr seid wunderschön und ich wünsche euch der Beste!

I would like to thank as well all the incredible people I have had the chance to meet across my time in the University of Cantabria and my stays in the Zürich Research Lab. Emilio, Marina, Cristóbal, Miguel, Iván, Borja, Mariano and Raúl, the talks in the office and the shared coffees made my day every single time. The same goes for Ana, Gilles, Hoisun, Rihards, Adela, Toke, Tobias, Hazar, Oliver, Benoit, Frank, Alessandro, Matteo, Radu, Stefan, Michael, Jonas, Angelo, and I am sure that I have missed someone (sorry!). Thanks to the rest of my great coworkers, Carmen, Esteban, Fernando, Rafa, Chus. Thanks to my former colleagues, Andreea, Bogdan, Georgios, Nikolaos, Freddy, and specially to Wolfgang for his always amusing after-lunch discussions. Thanks to Sandra and her cheerful welcomes. Thanks to the wonderful friends from ACACES. During the thesis I have not had many possibilities of meeting my old gang, but Eusebio, Alma, Fernando and Paula, you are always in my mind.

Quiero dar las gracias a mi familia, mis padres, mi hermana, mis abuelos, por haber estado siempre ahí, escuchándome incluso cuando os contaba mis batallitas de trabajo. También a ”mis niños”: Iker, Jana, os quiero inmensamente.

Y por último, pero no menos importante, quiero darles las gracias a Lorena, por apoyarme siempre, por estar dispuesta a hacer la maleta y acompañarme a donde ha hecho falta. Gracias, mi vida, porque a menudo has tenido más fé en mí que yo mismo. Sin ti esta tesis no habría sido posible.

To all those who are still around, I am glad to have you. And to all of you that are far away, as the song says, *I had to leave you and go away / but I think about you every day / ... / I wish that I could see you soon!*

This work has been supported by the Spanish Ministry of Education (FPU grant FPU13/00337), a Collaboration Grant from the HiPEAC Network of Excellence, the Spanish Science and Technology Commission (CICYT) under contracts TIN2010-21291-C02-02, TIN2012-34557 and TIN2013-46957-C2-2-P, the Spanish Ministry of Economy, Industry and Competitiveness under contract TIN2015-65316, the Spanish Research Agency (AEI/FEDER, UE - TIN2016-76635-C2-2-R), the JSA no. 2013-119 as part of the IBM/BSC Technology Center for Supercomputing agreement, the European Union FP7 programme (RoMoL ERC Advanced Grant GA 321253), and by the Mont-Blanc project. The Mont-Blanc project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671697.

Para Lorena

Resumen

El uso de máquinas con mayor capacidad computacional tales como los supercomputadores Exascale, capaces de realizar 10^{18} operaciones de punto flotante por segundo, es fundamental para obtener resultados científicos más precisos o completos. Una de las estrategias para desarrollar este tipo de máquinas es aumentar el número de nodos que las componen, incrementando los requisitos de la red de interconexión que los une. Para cumplir con dichas necesidades se apuesta por el uso de topologías de red de bajo diámetro empleando routers de alto grado.

El impacto de la red en sistemas de computación de altas prestaciones (HPC) crece con el aumento de la importancia de las aplicaciones BigData, cuyo patrón de comportamiento difiere de las cargas de trabajo tradicionales en HPC y presenta un mayor número de comunicaciones con una distribución más uniforme. Es razonable suponer que los límites impuestos por la red al rendimiento alcanzado aumenten de forma proporcional al tamaño de la red, y que algunos de estos límites también se propaguen a las redes-en-chip dentro de los nodos de alto rendimiento.

Esta tesis presenta un modelo de tráfico sintético que emula el patrón de comunicaciones de Graph500, un benchmark para evaluar el rendimiento de los sistemas bajo aplicaciones BigData. El uso de este modelo de tráfico simplifica la evaluación de aplicaciones intensivas en datos y de sus necesidades. Asimismo, permite predecir el comportamiento del benchmark en máquinas de mayor tamaño que las disponibles actualmente. Al analizar las comunicaciones del benchmark se observa que existe una mayor dependencia con la métrica de throughput que en aplicaciones propias de HPC, tradicionalmente limitadas por la latencia de la red. En ambos casos, existe un impacto significativo si el uso de los recursos de red no es equitativo entre los nodos.

Esta tesis realiza un análisis del troughput y la equitatividad entre nodos y evalúa el impacto de diversos mecanismos de equitatividad, implícitos o explícitos. También propone dos mecanismos para mejorar el rendimiento de la red y simplificar la implementación de los routers. El primero de ellos es una mejora en la detección de patrones de tráfico adversos mediante el uso de una métrica basada en contadores de contención de la cual se proponen cuatro implementaciones. La segunda propuesta es un mecanismo llamado *FlexVC* que relaja las restricciones en el uso de canales virtuales para evitar la aparición de deadlock. FlexVC reduce el número de buffers necesario para evitar deadlock, y los usa de forma más balanceada. También permite emplear canales virtuales adicionales más allá de los estrictamente requeridos, de modo que mejore el rendimiento.

El análisis de la equitatividad en el uso de la red y la evaluación de los mecanismos propuestos se han realizado mediante múltiples simulaciones con tráfico sintético. Se ha simulado una red Dragonfly con más de 15000 nodos. Las redes Dragonfly son una topología de red jerárquica con dos niveles, propuesta para los subsistemas de interconexión de máquinas Exascale; hasta la fecha, es la única topología propuesta para supercomputadores Exascale que ha sido implementada en un sistema comercial.

En el Capítulo 2 se detalla el desarrollo del modelo de tráfico sintético basado en las comunicaciones del benchmark Graph500. Debido al volumen de comunicaciones y a la cantidad de memoria empleada durante la ejecución del benchmark, no es factible emplear otros métodos de simulación tales como el modelado del sistema completo o el uso de trazas. El modelo de tráfico presentado proporciona un nivel de detalle sobre el impacto de la red que no puede alcanzarse con otros métodos de evaluación. A lo largo del capítulo se realiza un análisis en profundidad de las comunicaciones del benchmark, haciendo hincapié en la naturaleza asíncrona de las comunicaciones y la uniformidad entre procesos del número de mensajes enviados. Las comunicaciones están estructuradas en varios bloques durante los cuales el envío de mensajes es uniforme. Como parte del análisis también se evalúa el impacto de la agregación de mensajes sobre el tiempo de ejecución total de la aplicación. El modelo propuesto predice el número de mensajes que envía cada nodo de la red durante cada bloque de comunicaciones, a partir de los parámetros de entrada del benchmark y de una estimación de la capacidad de cómputo de los nodos de red.

El desarrollo del modelo incluye una implementación realizada sobre un simulador de red. Los resultados de la ejecución del modelo en una red Dragonfly evidencian varios escenarios en los que la red de interconexión es el cuello de botella. Estos escenarios están ligados al mecanismo de encaminamiento, y al número de procesos simulados y su mapeado sobre los nodos de red. Aunque colectivamente el tráfico se asemeja a un patrón uniforme aleatorio, la ejecución sobre un subconjunto del total de nodos de la red se aproxima más a tráfico de tipo adverso.

A partir de los resultados con el patrón de tráfico sintético se observa una degradación de rendimiento debida a desigualdades en el uso de la red. Como se explica en el Capítulo 3, las desigualdades de throughput occurren cuando los nodos de la red reciben distintas proporciones del uso de recursos. Esto puede limitar el rendimiento e incluso provocar casos de inanición en aquellos routers que suponen un cuello de botella bajo patrones de tráfico adversos. Para comprobar la equitatividad del uso de recursos se introduce un nuevo patrón de tráfico adverso-consecutivo (ADVc). ADVc sobrecarga de forma patológica los enlaces entre grupos de una red Dragonfly, provocando desigualdad con mecanismos de encaminamiento adaptativos.

El análisis determina el impacto de varios factores: prioridad de tráfico en tránsito, política de encaminamiento no mínimo global, y política de arbitrio. La red se drena más rápidamente si se prioriza el tráfico en tránsito sobre la inyección de nuevos paquetes a la red, y se reduce la congestión; no obstante, priorizar el tráfico en tránsito es perjudicial bajo encaminamiento adaptativo porque impide la inyección desde los routers que forman cuellos de botella y agudiza la desigualdad en el throughput. La imple-

mentación del encaminamiento adaptativo en origen presenta una incapacidad inherente para identificar correctamente el tráfico ADVc como adverso, que paradójicamente se agrava cuando se elimina la prioridad de tráfico en tránsito. Bajo el patrón ADVc se hace necesario un mecanismo explícito para garantizar la igualdad de uso de recursos; en esta tesis se ha empleado una política de arbitrio basada en la edad de los paquetes. No obstante, conlleva una implementación más compleja, por lo que en el resto de la tesis se ha empleado una política de arbitrio Round-Robin, más simple, sin priorizar el tráfico en tránsito.

En el Capítulo 4 se introduce el uso de información de contención en la decisión de encaminamiento no mínimo para mecanismos adaptativos. Dicha decisión, que elige entre una ruta mínima y otra no mínima, está típicamente basada en una estimación de la congestión del router vecino a partir del número de huecos disponible en las colas de entrada. Sin embargo, las métricas de congestión propician una dependencia con el tamaño de los buffers, oscilaciones en la decisión de encaminamiento, y una adaptación lenta a los cambios en el patrón de comunicaciones. La dependencia con el tamaño de las colas provoca un compromiso entre elegir colas de tamaño reducido, que reducen la granularidad de la decisión de encaminamiento, y buffers de mayor longitud, que tardan más tiempo en llenarse o vaciarse y aumentan el tiempo de respuesta a los cambios de tráfico. Las oscilaciones en el enrutamiento se deben a que el uso de caminos mínimos permite drenar la red, lo que lleva a la aparición de un lazo de realimentación entre la decisión y la métrica en que se sustenta.

Una métrica de contención, como la propuesta en el Capítulo 4, elimina estas limitaciones y monitoriza la causa de la congestión (contención entre recursos) en lugar de sus resultados (falta de huecos en las colas del router vecino). El mecanismo propuesto utiliza una métrica basada en un contador de contención para cada salida del router, registrando la demanda de los paquetes en cabeza de los buffers de las entradas. Sólo se contabiliza la salida correspondiente al camino mínimo de cada paquete, para favorecer la identificación del patrón de tráfico. Se emplean cuatro implementaciones diferentes del mecanismo, todas ellas con un rendimiento competitivo y una rápida adaptación a los cambios de tráfico. Las variantes *híbrida* y *ECtN* son las más atractivas porque superan el rendimiento del encaminamiento adaptativo en tránsito basado en métricas de congestión, pero conllevan un coste de implementación superior: la implementación *híbrida* combina las estadísticas de contención con información de congestión, y *ECtN* difunde dentro de cada grupo de la red las métricas de contención para los enlaces intergrupales.

La aparición de dependencias cíclicas puede provocar interbloqueos en la red (también llamados deadlocks), y para redes con bajo diámetro se suele prevenir mediante un mecanismo basado en canales virtuales (VCs) en el que cada salto del camino implica incrementar el índice del canal virtual. Cada canal virtual supone un buffer independiente por cada puerto de entrada e incrementa los costes de implementación del router. Este tipo de mecanismos impide un uso eficiente de las colas de entrada, y une el número de recursos necesarios a la longitud del camino no-mínimo más largo. Teóricamente, estas limitaciones se pueden evitar compartiendo la memoria

entre canales virtuales de un mismo puerto, mediante colas con asignación dinámica (DAMQs). Sin embargo, en redes de bajo diámetro se requiere que una parte significativa de la memoria se asigne de forma estática (negando sus beneficios), y su implementación es más costosa y con mayor retardo de acceso. En el Capítulo 5 se presenta una nueva gestión de los canales virtuales denominada *FlexVC*, que relaja las restricciones en el uso de los canales virtuales y depende sólo de colas particionadas estáticamente y de encaminamiento oportunista. La idea principal tras FlexVC es que para que un camino esté libre de deadlock basta con que exista una secuencia incremental de índices de canal virtual hasta el destino, aunque no se emplee. Por cada salto se pueden emplear todos aquellos canales virtuales con un índice inferior al correspondiente en el camino libre de deadlock. También se pueden añadir canales virtuales adicionales, que se pueden utilizar en todos los saltos de la ruta para mejorar el rendimiento. FlexVC es particularmente útil cuando se considera tráfico con dependencias entre clases de tráfico (por ejemplo, entre peticiones y respuestas), ya que se pueden reutilizar canales virtuales de las peticiones para las respuestas.

FlexVC aumenta el rendimiento bajo todos los mecanismos de enrutamiento con cada uno de los patrones de tráfico empleados. La mejora en rendimiento con FlexVC se mantiene o aumenta cuando se emplea encaminamiento adaptativo, así como el ahorro en el número de recursos necesarios. El reuso de canales virtuales en FlexVC complica la identificación del tráfico adverso en encaminamiento adaptativo en origen; para recuperar dicha capacidad se puede emplear un grupo adicional de contadores para registrar exclusivamente la congestión debida a paquetes que están avanzando por su ruta mínima. Esta estrategia no es suficiente para identificar el tráfico adverso con encaminamiento adaptativo en tránsito, pero se puede combinar FlexVC con el uso de los contadores de contención detallados en el Capítulo 4; este último caso proporciona el mejor rendimiento en general a la vez que reduce a la mitad el número de recursos necesarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The computational needs of high-demanding applications have exceeded for decades the capabilities of the most performant CPUs, requiring the aggregation of several computing nodes in order to achieve sufficient computational power to meet those needs. Typically, the communications between the application processes are performed through a message-passing library such as MPI [146]. One of the key components of large High-Performance Computing (HPC) systems is the interconnection subsystem; an interconnect is composed by a set of routers (which forward packets towards their destinations) and the cables to link them. The importance of the network role has increased with the next Exascale frontier targeted by forthcoming HPC systems (machines able to perform 10^{18} floating point operations per second), and by the surge of data-intensive applications in which the performance bottleneck is feeding data to the processors. These data feed bottlenecks come mainly from physical limits in memory bandwidth, but are also related to the network bandwidth of the interconnection. Furthermore, many of the design challenges of current system networks are expected to translate into the design of future processors as its interconnect evolves into networks-on-chip (NoCs).

The target of this work is to analyze the impact of the network interconnect under traction-gaining BigData applications, which are more limited by the network throughput than the typically latency-bounded HPC applications. To that effect, this work presents a synthetic traffic model of a representative BigData application and analyzes the fairness of the network and its impact in the applications performance. This work also introduces two architectural advantages that improve performance and reduce the number of resources needed, one to improve the detection of adversarial traffic scenarios and another to relax the routing and hardware restrictions of the network.

This introduction delves into the fundamentals of interconnection networks, going from the router components to the network topology. It also explores the use of simulation tools for the evaluation of the network performance, defining the performance metrics and focusing on the particular simulator employed in the rest of this work. Finally, it concludes itemizing the most significant contributions of this work.

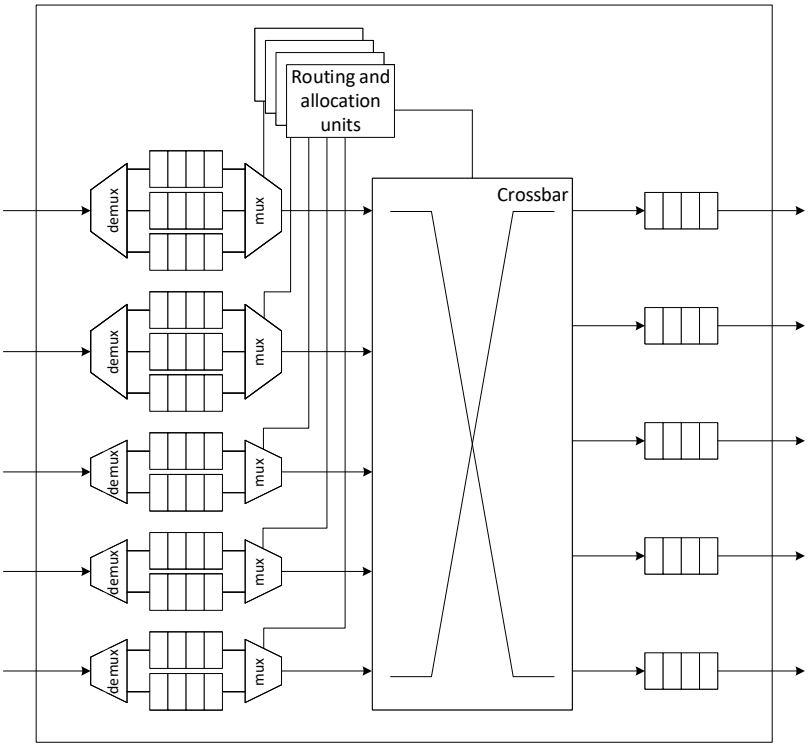
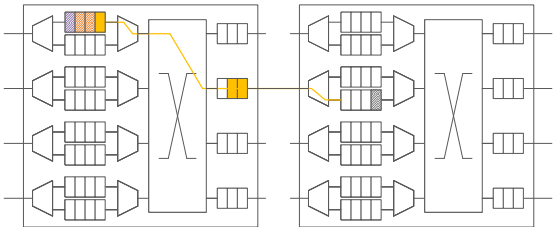


Figure 1.1: Architecture of a router with input and output buffering.

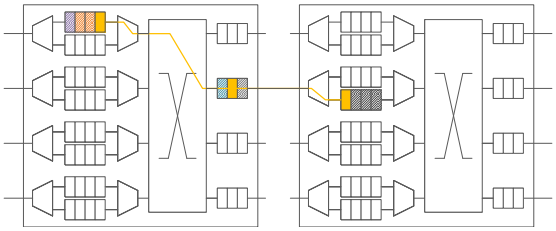
1.1 Router architecture

The architecture of a router is organized around the routing fabric, which handles the interconnection of the input and output ports. Router size is expressed as the *router radix*, the number of ports in the router. The routing fabric can be implemented with a monolithic crossbar [37, 105, 150, 26, 48] or through more complex solutions: for example, a buffered crossbar [127, 4, 83] or a multi-staged fabric composed of smaller crossbars [130, 1]. Figure 1.1 portrays an example of router architecture. It is composed by a series of buffers at the input and output ports, a switching fabric and the routing and allocation units that manage which packets are forwarded every cycle, and through which output port.

The architecture of the router is tied to the type of flow control employed. Flow control establishes how the network resources are allocated to the packets traversing the network, and can be buffered or bufferless. Bufferless flow control requires a path to be established between a pair of nodes before the traffic flow between them can be injected into the network. Buffered flow control strategies can allocate the use of the switching fabric on demand, at the expense of devoting some area to memories that store incoming data when a connection between a pair of router input and output is not available, before it can be forwarded to another router or its destination. Buffered flow control mechanisms differ in the granularity at which they perform resource allocation; most common mechanisms are Virtual Cut-Through (VCT, [85]) and WormHole



(a) Virtual Cut-Through switching.



(b) WormHole switching.

Figure 1.2: Buffered flow control mechanisms. Orange line highlights the traversal of the marked packet (also in orange). Note that in VCT the packet can only advance if there is enough space in the next buffer to wholly host it, whereas in WH the packet can be spread across multiple buffers in different routers.

(WH, [45]).

VCT manages the buffering in units of packets, requesting space in the next buffer to host a whole packet before forwarding it. In memory-constrained networks such as Networks-on-Chip (NoCs) wormhole flow control is employed, where buffering is performed at a finer granularity of *flits*, smaller than a packet. WH reduces the buffer size but allows packets to be spread across multiple buffers at any given time, increasing the complexity of the flow control mechanism and/or reducing the flexibility of the routing function. In recent design proposals of optical interconnect routers flow control is performed at a *virtual circuit* level, using a given path for several packets before reconfiguring it [128, 94, 119]. This design alleviates the impact of reconfiguration delays, which are orders of magnitude higher than commutation times for optical circuitry, and can either be bufferless or employ much smaller buffers. In traditional electronic interconnects, however, the penalty for establishing a new path is much more negligible and a buffered flow control is used. This work will hereon focus on the analysis of networks with VCT flow control, although some of the proposals can also be applied to WH flow control.

Buffered flow control mechanisms require updates of the buffer occupancy from neighboring routers to ensure the availability of space to host the packet at the input buffer of the next router. This can be performed through different strategies; the most common are credit-, grant-, or rate-based. A detailed description of different flow control mechanisms and their implementation can be found in [107].

Flow control based on credits sends updates to the neighbor router at each input port to acknowledge the removal of a packet from the input queue. This increases a counter

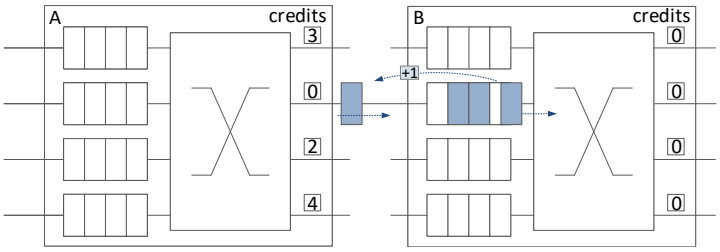


Figure 1.3: Credit-based flow control. Each output has a credit counter tracking the number of available slots in the next buffer: a packet dispatch from router A decrements the credit counter for the output linking to B, and a dispatch from the input queue in B triggers the delivery of a credit back to A, reporting the availability of one additional slot.

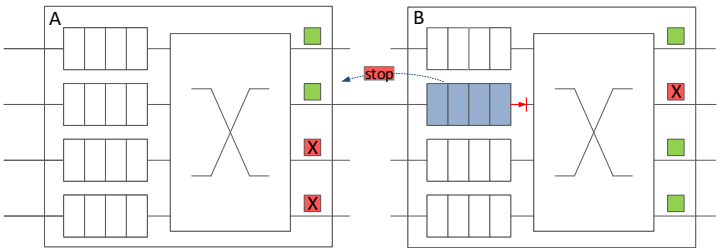
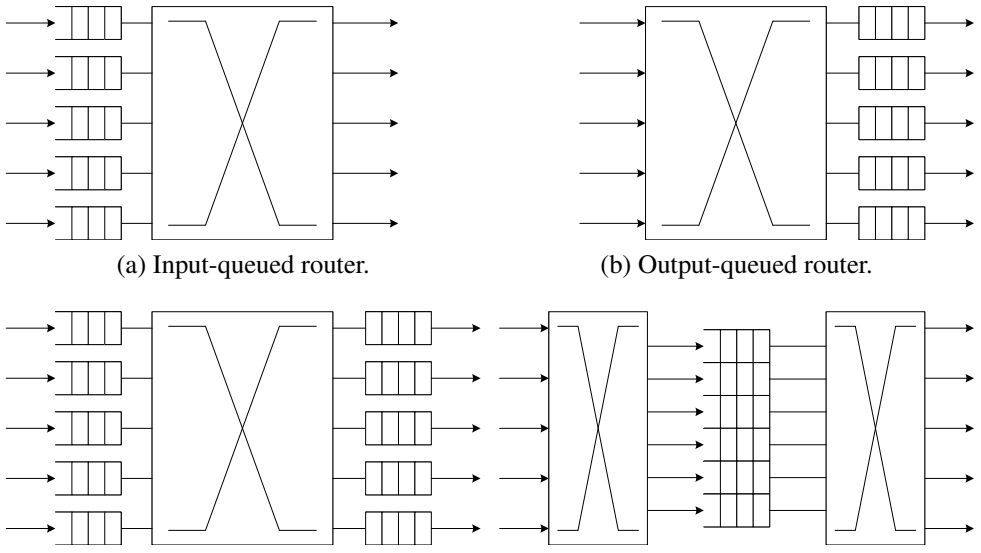


Figure 1.4: Grant-based flow control. Output ports with available space in the neighbor router are marked in green, and the red mark implies the lack of available space. When the input buffer in router B becomes full, it sends a signal to router A to stop the transmission through that link. When packets are advanced and new slots become available, another signal is transmitted to notify that the communication can be restarted.

of *credits* (available slots on the input buffer); whenever a packet is sent towards that input buffer, the credit count is decreased. Figure 1.3 illustrates the functioning of credits on a transmission between two routers, A and B. Once the credit count reaches zero, transmission towards that queue is halted until new credits are received. Credits can be either absolute or relative; absolute credits report the total number of free slots in the neighbor queue, whereas relative credits are incremental and represent the number of slots that have become free. The latter is more common at a link level, because absolute credits need the packet sequence to be numbered.

Grant-based flow control substitutes credit updates for two signals to inform when the router can start or must stop the communication. This diminishes the amount of control messages to be sent (particularly for large buffers) but significantly reduces the granularity of buffer management, only being able to ascertain the lack of available space in the next queue after it is full. This constitutes a significant restriction for adaptive routing mechanisms where the route selection typically depends on buffer occupancy. Figure 1.4 represents the delivery of a stop signal because the input buffer in router B has filled up. Finally, rate-based flow control throttles the transmission: the neighbor router hosting the input queue sends messages to increase or decrease the transfer speed of the link. This work focuses on credit-based flow control, since it is



(a) Input-queued router. (b) Output-queued router.

(c) CIOQ router. (d) Central-buffered router.

Figure 1.5: Different buffering configurations in a router.

the most usual implementation in system networks such as Infiniband [76], Intel OmniPath [26], Bull BXI [48], Cray Cascade [56], Cray Seastar [6] or IBM PERCS [16]. The flow control credits employed are relative, only sending incremental updates to the credit list.

In routers with buffered flow control, the buffers are typically First-In, First-Out (FIFO) queues where only the packet at the head of the buffer can advance out of the queue. Multiple buffering strategies exist: a central pool of buffers (linked both to input and output ports), input-connected buffers, output-placed buffers and buffers both at input and outputs (known as Combined Input/Output-Queued, CIOQ [121]). Figure 1.5 shows a scheme of routers with different configurations. Each buffering strategy has associated shortcomings [75, 70, 112]. Routers with input-buffering suffer from Head-of-Line Blocking (HoLB), where a packet at the head of the buffer cannot advance and blocks the forwarding of another packet further in the queue. Furthermore, they cannot exploit the benefits of crossbar frequency speedups, where the crossbar commutes at a faster rate than the network links. Output-buffering removes HoLB but reduces the potential for adaptive routing, since the routing decision cannot be re-evaluated. Central buffering and CIOQ keep routing flexibility while mitigating HoLB, but require more buffer area and power consumption. This work considers networks with CIOQ routers as the one depicted in Figure 1.1.

The routing units determine which output will be used for any given packet. Routing mechanisms can be oblivious or adaptive. Oblivious routing assigns a fixed path based on the destination of the packet, whereas adaptive routing takes into account the status of the network and selects between multiple paths.

The allocation units assign the router resources, matching input buffers to the out-

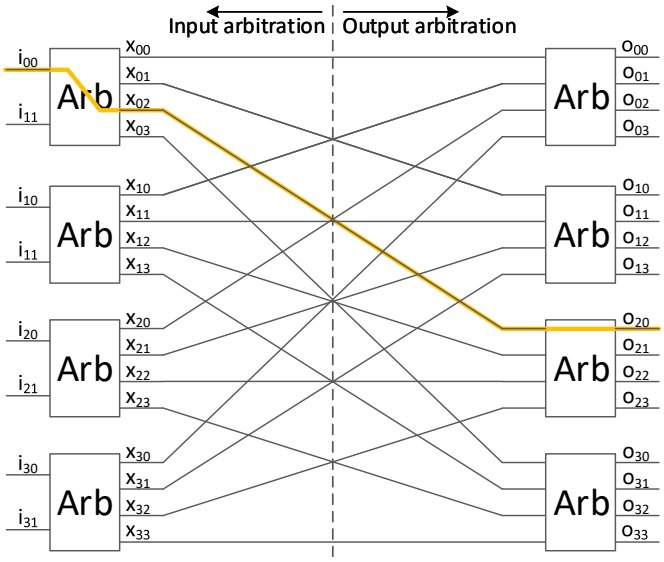


Figure 1.6: Input-first separable allocator. The allocation is performed in two stages: first the input arbiters select one virtual channel i_{ij} to access the crossbar, triggering the x_{ik} signal for the k port. Next, the output arbiters grant the output port to one of the contending inputs. The figure shows a successful request i_{00} at the first input, which wins at the input arbiter (triggering the x_{02} signal) and at the output arbiter, being granted access to the crossbar and to the output port 2.

puts requested after the routing is performed. There are several allocation algorithms; this work considers solely *separable allocators* [46, 20], which consist of two stages of arbitration, across the input and the output ports. In each input port there are several buffers, one per every *virtual channel* (VC); the purpose of VCs will be introduced in Section 1.3. Input arbitration assigns the use of the input port to the crossbar to one of the VCs, and places a request for the output that the packet at the selected VC will employ. Output arbitration matches the output port to an input that has requested it. Each arbiter follows an arbitration policy to establish the order in which the demanding ports or buffers will be attended. Figure 1.6 illustrates a sample 2x4 input-first separable allocator. The arbitration policy needs to ensure a fair attendance to avoid starvation at certain inputs or VCs. A well-known policy is Round-Robin (RR), in which the list of priorities for the ports is rotated every time the arbitration is performed. Separable allocators do not guarantee an optimal arrangement but are easier to implement and perform a faster matching, specially for large radix routers. The modelled router in this work employs an input-first separable allocator, employing RR in each arbiter but only shifting the priority list of ports when the resource is granted.

Most modern routers split their functionality into multiple pipelined steps that can be performed concurrently for different packets [46], in order to reduce the duration of the clock cycle and increase performance. In a typical virtual channel router, these steps are the computation of the routing, the allocation of a virtual channel for every input port, the allocation of an input for every output port, and the traversal of the

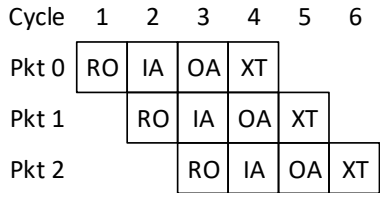
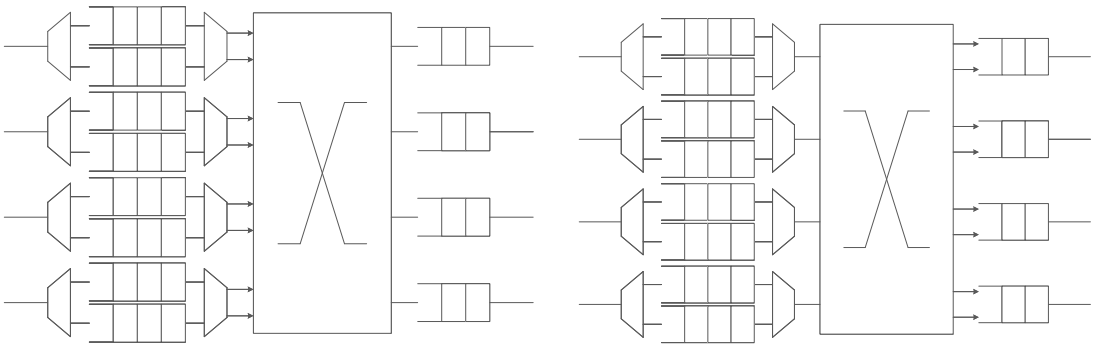


Figure 1.7: Pipeline of a router. Each packet goes through 4 steps: routing (RO), input allocation (IA), output allocation (OA) and crossbar traversal (XT). At each cycle, the router can be processing multiple packets, each at a different stage.



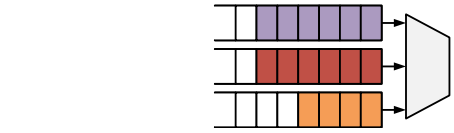
(a) Input speedup. (b) Output speedup.

Figure 1.8: Router with an input/output speedup of 2. Note that the router with input speedup has 2 ingress points to the crossbar for every input port, whereas the router with output speedup has two egress points from the crossbar to each output port.

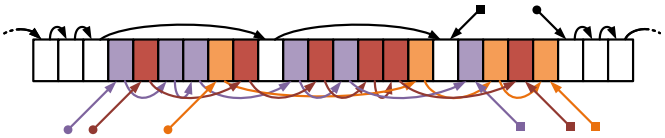
router crossbar. Figure 1.7 shows the pipelining of those functions for three different packets during 6 cycles of the router.

In order to increase the router performance and improve the allocation of the resources the router may have *speedup*, provisioning more resources than ideally required. Different implementations can be considered, such as input, output and internal speedup. In input speedup, the router crossbar has several input ports for each router ingress point, allowing more than VC of a router input port to advance concurrently. Similarly, in output speedup the router crossbar has multiple output ports for each router egress point. Figure 1.8 illustrates routers with input and output speedup. Both input and output speedup increase the effective crossbar transfer rate (multiple packets can advance from/to the same port) and help to mitigate the impact of suboptimal allocations. Their implementation is necessarily costly, because it requires a switching fabric with at least twice the input/output ports of the router, and scales poorly. This work will focus in an internal speedup where the router stages (routing, input and output arbitration, and crossbar traversal) work at a higher frequency than the network links.

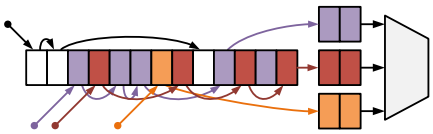
Different buffer implementations can be employed depending on the buffer allocation, as displayed in Figure 1.9. Statically partitioned buffers assign a fixed amount of



(a) Statically partitioned buffers.



(b) Dynamically Allocated Multi-Queue (DAMQ).



(c) DAMQ with statically allocated buffering for each VC.

Figure 1.9: Logical organization of static vs dynamic buffer management with three VCs. In DAMQs a buffer pool is shared between VCs. Head and tail pointers are used to access data from each VC, and allocate free memory (black pointers).

memory per VC, whereas dynamically allocated buffers (such as Dynamically Allocated Multi-Queues, DAMQs) [138] share a single memory buffer across all the VCs within a port and allocate memory dynamically to each VC on demand. Intermediate approaches employ a shared pool combined with statically allocated buffering for each VC.

1.2 Topologies for Exascale system networks

Besides the router architecture, a key characteristic of system networks is the topology in which they are arranged. Ideally, the topology shall meet the performance needs of the applications at the minimum attainable cost. Some of the most common topologies exploited in HPC systems have been k-ary n-cubes [43] and folded-Clos networks [42] (also called *fat trees* [99]). These topologies have high path diversity between pairs of nodes, which allows them to be non-blocking and to cope with faulty cables without isolating nodes. k-ary n-cubes have been traditionally based on low-radix routers with a reduced number of ports and low bandwidth, and benefit the execution of applications with near-neighbor communications. As the technology has improved, routers have achieved higher bandwidths, which are better exploited through a higher router radix [92] [130]. This has fueled the popularity of fat-trees [130, 152] which have high radix and achieve better performance at a lower cost, when compared to k-ary n-cubes.

However, they face packaging issues when larger network sizes are considered in order to achieve Exascale machines. In this context, highly-scalable high-radix topolo-

Table 1.1: List of abbreviations for the Dragonfly network.

Abbr.	Parameter
N	Number of nodes in the network.
p	Number of nodes per router.
h	Number of inter-group (global) links per router.
a	Number of routers per group.
r	Router radix (number of ports).
d	Network diameter.
l	Local hop (traversal of a local link).
g	Global hop (traversal of a global link).

gies have been proposed, such as the Flattened Butterfly [90], the Dragonfly [91], the SlimFly [24] and the Projective Networks [32]. One of the features of these topologies is that they are *direct* networks, where all the routers are directly connected to one or more nodes; this decreases the switch to node ratio and helps to diminish costs, as well as reaching higher network sizes for the same router radix. Out of these topologies, the Dragonfly is particularly interesting because it has been the only to have been implemented in an HPC system so far.

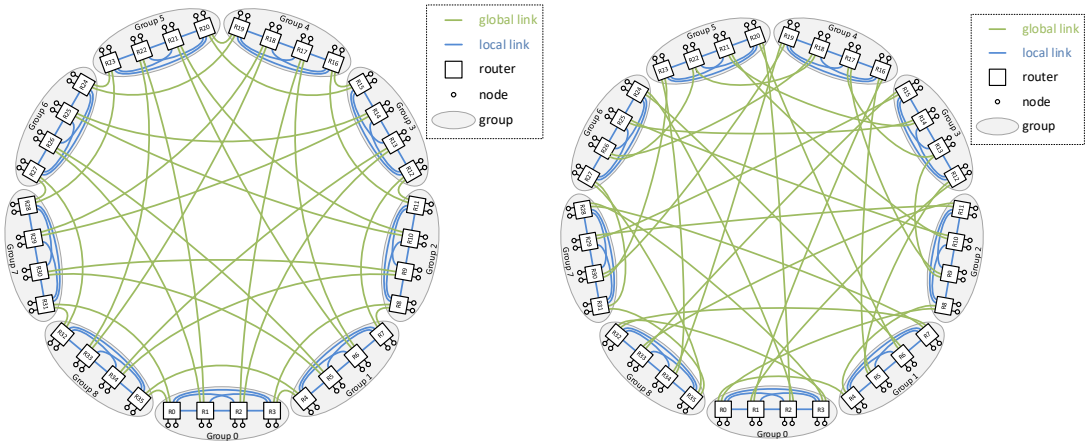
1.2.1 Dragonfly networks

The Dragonfly topology was first introduced by Kim *et al.* in [91]. It is a low-diameter cost-efficient network topology based on high-radix routers and suitable for Exascale systems. This topology has been used in the Cray Cascade [56] and IBM Power 775 [16].

In a Dragonfly, network routers are arranged into groups following a 2-level hierarchy. It is a direct network with one or more compute nodes connected to every router. The size of the network is described through three parameters: the number of compute nodes per router (p), the number of routers per group (a) and the number of inter-group links per router (h). Thus, the number of ports per router (*router radix*) is $r = p + a - 1 + h$. Table 1.1 summarizes the abbreviations for the network parameters and variables.

The maximum size that can be achieved in a Dragonfly is $N_{max} = ap\,(ah+1)$, when there is only one inter-group link between every pair of groups. To ensure a balanced use of the links under a uniform workload, the p, a and h parameters follow the relation $a = 2p = 2h$ [91]. These guarantees the number of global links departing from any group to be the same as the number of injectors in the group, and the number of local links to be twice that amount, because the longest path in a minimal route between two nodes employs two local links but only one global link. Figure 1.10 displays a Dragonfly network of size $h = 2$ with 72 compute nodes and 36 8-port switches. The size of this Dragonfly is maximum for the router radix $k = 8$ of the routers.

In the base version of the Dragonfly, both the topology within each group and



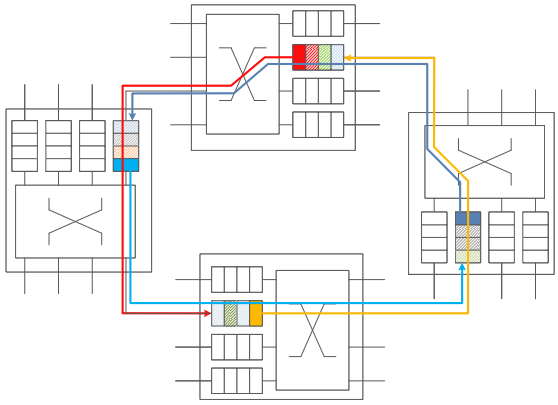
(a) Palm-tree arrangement.

(b) Consecutive arrangement.

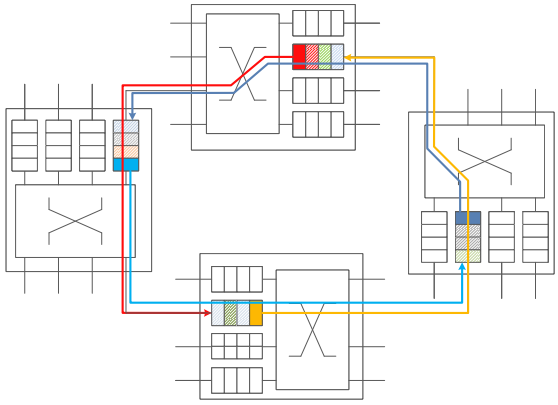
Figure 1.10: Sample Dragonfly network with $h = 2$ global links per router, $p = 2$ compute nodes per router and $a = 4$ routers per group. Two different arrangements of the global links are considered, palm-tree and consecutive.

between groups are complete graphs; if the size of the network is maximum ($N = ap(ah+1)$), only one minimal path connects any pair of given nodes. The diameter of the network is then $d = 3$, corresponding to the longest path between two given routers, which occurs when the routers are in different groups and none of them is directly attached to the link joining the groups. Other variations of the Dragonfly increase the connectivity between neighboring routers at the expense of losing full connectivity between routers of the same group, and increasing the network diameter. Such variations are considered in this work.

Typically the links between groups (also called *global* links) are much longer than those within a group (or *local* links) and employ optical cables instead of electrical wires. The arrangement of the global links among the routers in a group can follow different layouts [33]. As will be explained in further sections, certain combinations of link layout and traffic workload can lead to an unbalanced use of the network and translate into undesired behaviors. However, the layout only changes the specifics of the workload needed to present such behavior and does not undermine the need for mechanisms to alleviate it. The arrangement implicitly shown in [91] is consecutive, mapping the global links in the group to the groups in the network in an incremental sequence, starting always from group 0 and omitting those links that would connect to the current group. The Dragonfly in Figure 1.10b employs a consecutive arrangement of global links, with the first global link in every group connecting to group 0 (except at group 0). The Dragonfly networks employed in this work follow a *palm tree* layout in which the links in the group connect to precedent groups following a modulo sequence that joins to other groups consecutively, starting by the h previous groups that are always reached through the global links of the first router of the group, and ending with the h following groups connected to the last router of the group. The global links



(a) With deadlock.



(b) Without deadlock.

Figure 1.11: Cyclic dependency between packets at the head of the buffer in different routers. Each arrow represents the intended path of a packet at the head of the queue. In the upper figure, none of the packets can advance because there are not any available slots in the next queue, and none of the queues can drain, leading to a deadlock. In the lower figure, virtual channels are employed to break the cyclic dependency and allow queues to drain eventually.

in the Dragonfly of Figure 1.10a follow a *palm tree* layout.

1.3 Deadlock-avoidance mechanisms

Almost all network topologies are susceptible to present cyclic dependencies between packets and are prone to inter-lock those packets. Lossless networks such as those used in HPC systems do not drop packets in such scenarios, and can eventually halt the whole network in what is known as a *deadlock*; Figure 1.11a shows an example of cyclic dependency.

To prevent the network halt, either a deadlock avoidance or a deadlock recovery mechanism is used. Deadlock avoidance guarantees that deadlock conditions are not

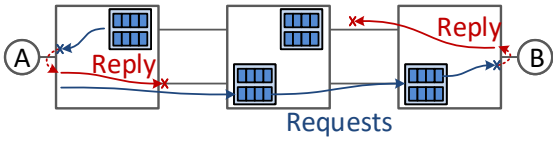


Figure 1.12: Example of protocol deadlock in a generic diameter-2 network with 2 VCs. There is a cyclic dependency between the replies from A, the requests from A, the replies from B and the request from B. Since the buffers are filled up with requests, there is not sufficient space to inject replies, and the consumption of requests stalls.

met, whereas with deadlock recovery a deadlock can potentially happen, and the mechanism is responsible of detecting and correcting it [120].

There are multiple techniques to avoid deadlock. One is to impose routing restrictions that ensure the absence of cyclic dependencies, as is done in DOR routing [46], O1TURN [132], or in different proposals for routing in Dragonflies [64, 33]. Another method is to apply injection or flow control restrictions, as is done in the Bubble router [123], the BlueGene/L [27] and the Flit Bubble flow control [102]. A third technique to break the cycles is to split a physical channel into multiple *Virtual Channels* (VCs), placing multiple buffers for a single link [44, 52, 69]; this work only focuses on avoiding deadlock through the use of VCs. Certain deadlock avoidance mechanisms rely on virtual networks which combine certain routing restrictions with the use of VCs, such as SSSP [72], DF-SSSP [50] or NUE routing [49].

Figure 1.11b shows the same head-of-buffer packets as in Figure 1.11a, but the use of multiple buffers in each port breaks the cyclic dependency and allows network drainage. VC buffers are connected to the same link port through a demultiplexer, and to a crossbar input through a multiplexer. This is the case of the router in Figure 1.1; notice that the output ports employ a single buffer instead of one per VC because all the packets at a given output require the same outgoing link and do not suffer HoLB, since the crossbar is only traversed if there is enough space at the buffer in the next router.

There are multiple deadlock avoidance mechanisms based on the use of VCs. In the mechanism proposed by Günther [69], packets at each hop of the path are hosted in a different VC (buffer) and do not block other packets at a different stage of their path, breaking the cyclic dependency and allowing network drainage. The original proposal for deadlock avoidance in the Dragonfly [91] uses a similar approach and increases the VC index for each hop of the path, but distinguishes between local and global links. As shown in Section 5.1.1 the local and global hops can only follow a given sequence, and the number of VCs needed is not determined by the highest index value but by the amount of indices within each set. VCs are also used to mitigate HoLB, what increases the router performance.

Deadlock can also appear at the protocol level [46], when there is a cyclic dependency between different packet classes that reuse the same channels. Figure 1.12 presents a sample cyclic dependency between two nodes, where the delivery of a response is halted for the sending of a request. The typical solution to prevent protocol

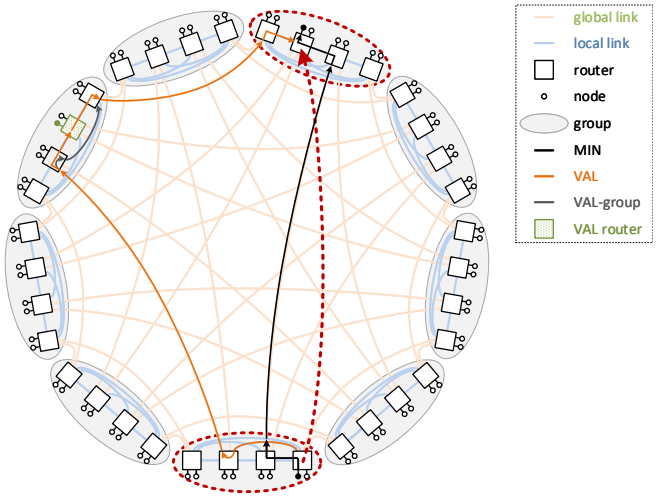


Figure 1.13: Example of oblivious routing mechanisms in a Dragonfly network. Dashed red line signals the source and destination of a packet. MIN routing is shown in black, VAL routing in orange, and VAL-group in grey. VAL and VAL-node paths overlap except for the extra jump to the VAL node.

deadlock is to employ separate virtual networks to handle each class of traffic, as in Cascade [56].

1.4 Routing mechanisms

The routing function determines the path that a packet will follow to reach its destination. These functions are typically classified into *oblivious* or *adaptive* depending on whether they set the path independently of the state of the network or not.

1.4.1 Oblivious routing

1.4.1.1 Minimal routing (MIN)

Minimal routing (MIN) employs the shortest path between the source and destination of a packet. In a balanced Dragonfly network with $a = 2p = 2h$ this results in only one possible path of 3 or less hops: up to one local hop in the source group, one global hop between source and destination groups, and one local hop in the destination group ($l - g - l$). The local hop in the source group is employed when the destination is in a different router in the same group, or when the destination is in a group connected to a different router of the group. The local hop in the destination group occurs when the global link is not connected to the same router as the destination node. An example of MIN routing is presented in Figure 1.13.

MIN provides sufficient throughput and minimal latency under balanced traffic loads with spatial and temporal uniformity, where it represents an ideal target. How-

ever, other workloads stress non-uniformly the network links and reduce performance, enforcing the need for alternative routing mechanisms. A nonminimal routing mechanism such as a Valiant routing exploits longer random paths to regain uniformity in the use of the network resources.

1.4.1.2 Valiant routing (VAL)

Valiant routing (VAL) is an oblivious nonminimal routing mechanism proposed by Valiant in [143]. Under VAL, packets are first minimally routed to a randomly selected router and then minimally towards the destination. This effectively makes the traffic workload uniform and balances the use of the links at the cost of doubling the length of the path. Giving the balance between links and nodes, the maximum performance that can be achieved with VAL is 50% of the maximum allowed by the network. This gives an advantage over MIN under non-uniform traffic loads, but trades off performance when the communications are uniformly distributed.

In [91], Kim *et al.* employ a variant of VAL which selects a random intermediate group instead of a node, to save a local hop in the intermediate group and reduce the path length. This allows a performance increase under adversarial traffic patterns such as those described in Section 1.6.1.2; however, it provokes a pathological bottleneck in the local links of the intermediate group under other adversarial patterns, as it was first observed by García *et al.* in [63]. In this work, we will refer such variant of VAL routing as *VAL-group*. A sample of the paths in both versions of VAL routing is drawn in Figure 1.13. It must be noted that, if VCs are employed as deadlock avoidance mechanism, VAL requires an additional VC over VAL-group, corresponding with the additional local hop in the path. The relation between the length of the longest possible path and the number of VCs required to avoid deadlock is described in more depth in Chapter 5.

1.4.2 Adaptive routing

The previous oblivious routing mechanisms constitute a good point of reference for the performance targets under concrete scenarios (i.e., MIN when the communications are uniform) but are not well suited to match the needs of ongoing transmissions with other communication patterns. Adaptive routing takes into account the state and usage of the links to choose between multiple paths in order to provide a better service. In the context of Dragonfly networks (particularly if they are of maximum size) adaptive routing switches between MIN and VAL routing for every packet. The election between minimal and nonminimal paths relies on a *misrouting decision* which can be based on different metrics of the links and occur at different points of the route. Adaptive routing mechanisms can be split into two categories: source-adaptive and in-transit adaptive routing.

1.4.2.1 Source-adaptive routing

Source-adaptive routing mechanisms perform the misrouting decision only at the injection of a new packet into the network. They can be simpler to implement than in-transit adaptive routing, since the misrouting decision is only performed once per packet instead of at multiple points of the path. The misrouting decision typically relies on information about the saturation of the queues and is performed through a comparison of available credits (empty space in the next buffer) per output. The selection of the intermediate node is similar to VAL, and a VAL-group variant can be employed.

In [82], Jiang *et al.* describe a simplified implementation of the Universal Globally-Adaptive Load-balanced routing (UGAL, [134]) for Dragonfly networks. It tracks the occupancy of the router buffers, and routes minimally if the following relation is accomplished:

$$Q_{MIN} \leq 2Q_{VAL} + T \tag{1.1}$$

where Q_{MIN} and Q_{VAL} represent the queue occupancy in the first hop of the MIN and VAL paths, and T is a routing threshold that can be adjusted to prevent excessive misrouting under low traffic loads. The $2\times$ factor represents that, in average, VAL paths are twice as long as MIN paths, and therefore if the occupancy in the VAL queue is more than half of the MIN buffer, the nonminimal route is counterproductive. In this work, a generalized version of this decision is employed, where the $2\times$ value is replaced by a tunable factor F; Equation 1.2 defines this generalized version.

$$Q_{MIN} \leq F \times Q_{VAL} + T \tag{1.2}$$

The original proposal does not specify if the decision is based on credits per-port or per-VC. In the first case, credit count gathers the credits for all the buffers (VCs) in each port, whereas in the comparison per VC the credit count is the number of empty slots in the buffer to be used. Unless otherwise stated, in this work the comparison is performed per VC. Section 5.2.4 discusses the impact of each approach and justifies the use of a per-VC comparison.

Since the misrouting decision is only performed at injection and the path cannot be altered afterwards, local saturation information may be insufficient under adversarial traffic. This can be circumvented with the exchange of link saturation status between routers, as done in PiggyBack routing.

1.4.2.1.1 PiggyBack PiggyBack routing (PB) is a source-adaptive routing mechanism proposed by Jiang *et al.* in [82] for Dragonfly networks, extending UGAL with a mechanism to detect saturated global channels. All the routers perform periodically the misrouting decision on each of its global links compared against their average global link saturation, and mark as saturated those that exceed it. This is decision is summarized in Equation 1.3, where S_{GC} is the bit that signals wether a global channel is saturated or not, Q_{GC} is the occupancy of the global channel, $\bar{Q}$ is the average occupancy and T is a threshold to avoid excessive misrouting under low traffic loads; the

original version of PB in [82] employs a fixed $2\times$ factor but here it is again generalized through a tunable misrouting factor F.

$$S_{GC} = Q_{GC} > F \times \bar{Q} + T \tag{1.3}$$

The saturation status of the global links is then broadcast within the group through piggybacking, attaching it to the header of regular packets exchanged in the group. Packets attempting to take a route with a global port marked as saturated (bit $S = 1$) or with a queue occupancy in the next hop that does not fulfill Equation 1.2 are re-routed nonminimally through a different path.

1.4.2.2 In-transit adaptive routing

In-transit adaptive routing mechanisms reevaluate the misrouting decision at multiple hops of the path, deciding between the minimal path towards the destination and a nonminimal path through an intermediate node. This gives more flexibility in path usage and allows to react during the network traversal to local situations of congestion, reducing latency in exchange for higher router complexity. However, being able to perform the misrouting decision permits to adapt to adversarial traffic relying only on local saturation information and erases the need to broadcast the saturation status. The misrouting decision is typically based on the occupancy of the queues, as in source-adaptive routing mechanisms. Note that adaptive routing focuses on avoiding in-network congestion, where traffic flows compete for the same network resources across their paths. Adaptive routing can not avert end-point congestion, where network destinations suffer contention from multiple traffic flows, what propagates saturation back to the network. This work does not evaluate end-point congestion avoidance.

Figure 1.14 depicts a case of in-transit adaptive routing in a Dragonfly network. A nonminimal path can be selected at injection, after a minimal hop in the source group and within the intermediate group. The only case where a nonminimal route is enforced regardless of link saturation status is when two local hops have been made at the source group. Such situation can only occur when the minimal is selected at injection, and a nonminimal local hop is performed because the minimal global link is saturated. In that case, a nonminimal global hop is performed to avoid a packet getting stuck at its source group and originating a livelock in which the packet never stops transiting but never reaches its destination.

Progressive Adaptive Routing (PAR) was proposed by Jiang *et al.* in [82] as an in-transit adaptive routing for Dragonfly networks. The misrouting decision is performed as described for UGAL in Equation 1.1. The main drawbacks of PAR in comparison to a source-adaptive routing as PB are the higher implementation complexity and the higher number of VCs required to prevent deadlock, which implies an extra buffer for every router port. To alleviate the latter constraint, García *et al.* proposed in [64] a new in-transit adaptive routing called *Opportunistic Local Misrouting*.

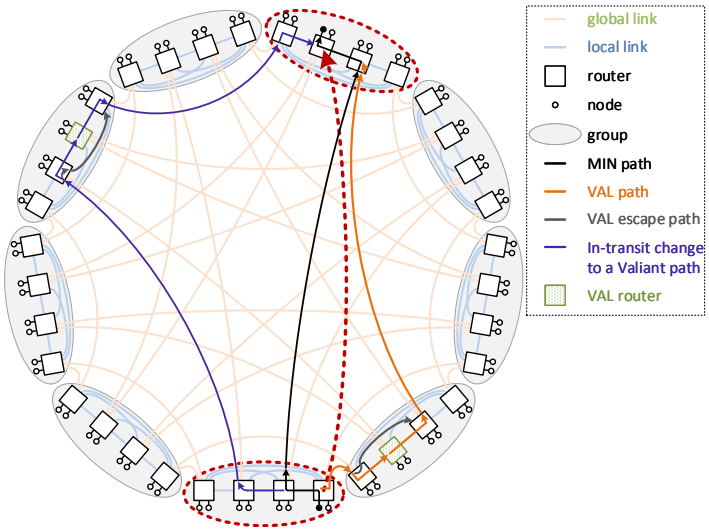


Figure 1.14: Example of in-transit adaptive routing in a Dragonfly network. Dashed red line signals the source and destination of a packet. MIN routing is shown in black, VAL paths in orange (misrouting at injection) and blue (misrouting in-transit). Escape paths in dark grey revert to MIN paths from the current hop of the path. When the VAL router is reached, a minimal route is followed towards the destiantion.

$$\begin{aligned} PAR\ VC\ set: &\quad l_0 - l_1 - g_2 \quad - \quad l_3 - l_4 - g_5 \quad - \quad l_6 \\ OLM\ VC\ set: &\quad l_0 - l_0 - g_1 \quad - \quad l_0 - l_2 - g_3 \quad - \quad l_4 \end{aligned}$$

Figure 1.15: VC index set used in PAR and OLM routing. Shorter paths follow the required subset of hops, in the same order. Hops are gathered by the group in which they occur (source, intermediate, destination).

1.4.2.2.1 Opportunistic Local Misrouting A safe *escape path* is a deadlock-free route, where cyclic dependencies are prevented.

The Opportunistic Local Misrouting (OLM) mechanism [64] performs a similar misrouting decision as in PAR, but saves resources through the use of *opportunistic hops* which do not require to reserve an additional VC. A hop is considered as opportunistic when it is open to cyclic dependencies, reusing the VCs from the previous path hop. In order to ensure deadlock freedom, opportunistic hops require the existence of a safe *escape path* from the next buffer. Escape paths are deadlock-free routes; in the case of OLM, the escape paths resort to an increasing VC index along the route to prevent cyclic dependencies. OLM exploits opportunistic routing for the nonminimal local hops, allowing paths of equal length to PAR (and similar performance) with two fewer buffers in each local port, significantly reducing allocation complexity and the power and area associated to the buffers. For the sake of clarity, a comparison of the selected VC in each hop of the path in OLM vs PAR is provided in Figure 1.15.

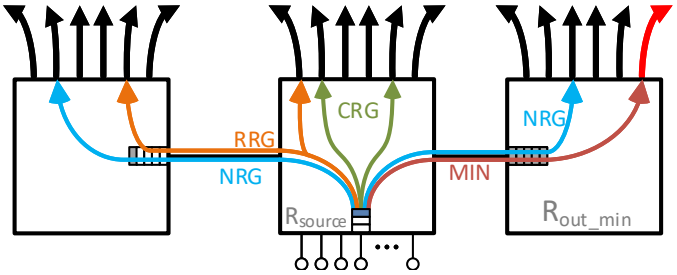


Figure 1.16: Global misrouting policies for source routing. The red arrow represents the global link in the minimal path.

1.4.3 Global misrouting policy

Nonminimal routing in Dragonfly networks implies the traversal of an intermediate group to balance the use of global links, which are more prone to become a bottleneck; this is discussed more comprehensively in Section 1.6.1 and Chapter 3. The *global misrouting policy* defines the set of inter-group links that can be used to send nonminimal traffic to avoid a congested link. This policy determines the intermediate group that will be traversed in a nonminimal path, depending whether it is directly or indirectly connected from the current router. When the remote group is directly linked to the current router, only one global link needs to be traversed to reach it. Arriving to an indirectly linked group implies traversing another router in the current group, requiring two hops: one local link from the current router to the neighbor router which is connected to the destination group, and one global link between the two groups (lg). An in-depth analysis of different global misrouting policies and their impact can be found in [60].

The global misrouting policy is independent of the implementation of the misrouting decision, but depends on whether it is only performed at injection (source-adaptive routing) or at any hop of the path (in-transit adaptive routing). In general, three different global misrouting policies can be considered for source-based adaptive routing:

- ***Random-router Global, (RRG)***: the intermediate group is selected randomly across the network, regardless of its distance from the current router. This is the policy that matches the original description of Valiant routing [143].

- ***Current-router Global, (CRG)***: only those groups that are directly linked to the current (source) router are candidates for the nonminimal path. In this case, there is always a 1 hop distance towards the intermediate group, skipping the first nonminimal local hop.

- ***Neighbor-router Global, (NRG)***: in nonminimal paths, traffic is diverted to a group connected to a different router in the source group. Packets traverse 2 links (lg) before reaching the intermediate group.

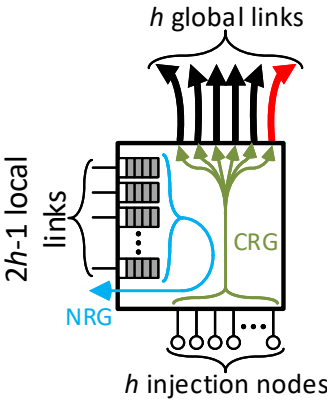


Figure 1.17: MM global misrouting policy for in-transit adaptive routing.

Any of these policies can be used on each hop with in-transit adaptive routing. Alternatively, different policies can be applied depending on whether the misrouting decision is taken at injection or after one or more local hops. The ***Mixed-mode (MM)*** mechanism implements such differentiated policy for in-transit adaptive routing:

- ***MM*** employs a *CRG* policy when attempting misrouting at the source router, and a *NRG* policy for traffic which is in-transit.

Figure 1.16 depicts these three global misrouting policies. *RRG* balances evenly the nonminimal traffic load between all the global links in the network, whereas *CRG* reduces the length of nonminimal paths. *NRG* has the longest average nonminimal path and reduces performance under uniform traffic. The *MM* policy is depicted in Figure 1.17. It tries to balance traffic at injection evenly across all the global links in the network and mitigate unfairness compared to the *RRG* and *CRG* policies which stress more heavily the minimal links in routers directly linked to the destination group. The impact of the global misrouting policy in throughput unfairness is analyzed in Section 3.1.1.

1.5 Performance metrics

To determine the fitness of the network to the system needs, a set of goals needs to be established. In terms of performance, the typical goal is to reduce the total execution time of the applications. However, in many cases it is of high interest to evaluate the network performance in a steady state where there is a constant flow of messages. This allows to characterize the network in a typical scenario where multiple applications are running concurrently. The most common metrics in a steady-state characterization are the *latency* and the *throughput*.

The latency measures the amount of time passed between the generation of a packet and its delivery to its destination. This provides a reference of how fast or slow the

network is in dispatching messages. Traditional workloads in high-performance computing consist of long phases of computation interleaved with bursts of communication during which the execution is halted. These workloads are therefore limited by the latency of the network, because a high latency value prevents them from resuming their computation.

Throughput, also known as effective bandwidth, is the sustained data transfer rate that is effectively achieved. Under a synthetic traffic load with a sustained injection rate, the ideal throughput target is to match the injection rate. That is an unfeasible scenario in any realistic network without full all-to-all connectivity across the whole network, due to contention for the network resources. Certain real-world applications are more constrained for the network throughput than for its latency, as it will demonstrated in Section 2.

Reaction time measures the amount of time required to adapt to a change in the traffic pattern, when an adaptive routing mechanism is employed. In the case of Dragonfly networks, this can generally be observed as the interval for a change in the proportion of minimal and nonminimal path usage, after the (non-)uniformity of the traffic alters.

1.5.1 Unfairness metrics

Systems can suffer different types of throughput unfairness, such as per-node throughput unfairness, where the source nodes receive a different service level from the network, being able to send different amounts of traffic. There exist multiple indicators to quantify the presence of throughput unfairness in a system. Some of the most frequently used are:

- Number of injected packets per router (or number of bytes per router, if packets with variable size are considered). This metric gives the *traffic load* at each router of the network, calculated as the number of packets injected by nodes directly linked to it. This allows to determine the difference in network resources allocation to the nodes at each different router, and detect the existence of a router whose nodes receive lower throughput or even suffer starvation.

- Minimal injected load (*Min inj. load*). It states the lowest number of packets injected per router in the network. This allows to detect a case of unfairness across the whole network. This value represents a combined metric of performance and fairness, since it can be constructed as the product of average throughput and the quotient between lowest injected throughput and average throughput. As it is discussed in Section 1.5, the average throughput is a performance metric; the quotient constitutes a metric of fairness itself.

$$min\ inj. = \frac{thput_{min}}{thput_{avg}} \times thput_{avg}$$

For a case of throughput unfairness, the minimal injected load does not identify if it is an isolated anomaly or a common behavior for multiple routers in the network.

- Max-to-min ratio (*Max/Min*). It is the quotient between the highest and lowest number of injections per router in the network. This metric highlights both the cases of allocation of too many or too few resources to a given router compared to the rest of the network.

- Coefficient of variation (CoV). This metric equals the quotient between the variance and the average number of injections per router:

$$COV = \frac{\sigma}{\mu}$$

It allows to discriminate between two different cases which may perform similarly in every other stated metric:

One router has an isolated situation of starvation and another router is given an abnormally high number of resources.

Half of the routers starve and the other half benefit from an unfairly high number of allocated resources. From the point of view of the applications, both of these situations are obviously undesirable, but the latter will arguably have a more negative impact on application performance, or affect a larger number of system users.

1.6 Network simulation tools

Network architects need tools for the design and evaluation of new systems. The development and manufacturing of router prototypes requires time and is costly. Network simulators constitute one of the most useful tools, specially at early design stages. They fall commonly on one of three different categories: full-system simulators, trace-driven execution, and synthetic traffic patterns-based.

Full system simulators are commonly used because they give a picture of the whole system execution, and generally allow to tune the detail granularity of each subsystem, providing a reliable analysis of the system behavior. Some of the best known examples include the Gem5 [25] and Simics [103] simulators. However, full system simulators are computing and memory intensive and require long execution times to complete a simulation. An alternative to diminish such requirements is to replace the application running over the simulated system by a skeleton restricted to the most relevant sections for the network evaluation. This approach has been employed in the SST simulator [126]. However, it is not a feasible option when the region of interest of the application presents memory and/or execution time constraints.

Trace-driven simulators replace the system simulation by a trace with the communications of an application executed on a real system. The events in the trace are

translated into communications in the simulated network. These simulators constitute a mid-way to evaluate the performance with real application workloads with constrained computation and memory requirements closer to synthetic traffic patterns. However, they require the obtention of a trace over a real system. These traces have a substantial size, specially for network-intensive applications, and become unmanageable when more than a few hundreds of processes are used. The number of nodes employed in the trace must match (or be similar to) the number of nodes in the simulated system. Scaling the communications for a different size broadens the usage of the trace but reduces the accuracy of the results. Finally, for systems with nodes dissimilar to those simulated, traces fail to accurately represent the execution dependencies.

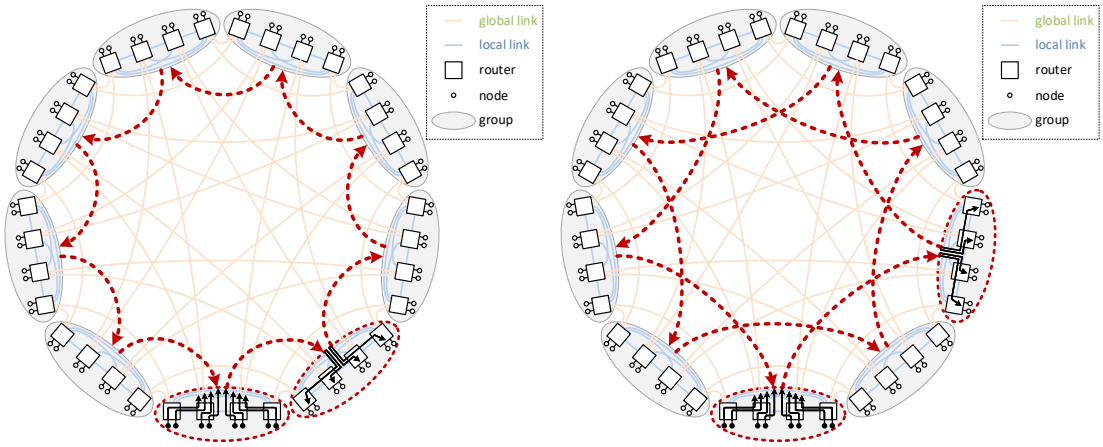
Synthetic traffic models offer less granularity and accuracy than full system simulations in exchange for lower computational and memory requirements. For many evaluations, they provide enough insight about the network performance and the most likely use cases. They typically consist of permutations to determine the destination for a message from a given node, as described in Section 1.6.1. This is the case for the network-only simulation mode of the Garnet module [8] from the Gem5 simulator, or the BookSim simulator [79]. There are nevertheless simulators that employ synthetic traffic patterns which preserve the characteristics (temporality, destinations, volume, etc) from executions of a real application, such as SynFull [17]. SynFull employs Markov Chains to model the memory accesses and their associated coherence traffic for the different phases of an application, and is focused on Networks-on-Chip (NoCs).

Simulation can be either time-driven or event-driven. Time-driven simulators conduct the procedures of the simulation actors (in this case, the network nodes and routers and their submodules) following the increases in the time cycle counter, regardless any given actor may not have any effective outcome at certain cycles. In event-driven simulations the events spawn further events to be performed, and the steps of an actor are only performed when there is an event requiring them instead of every cycle. For example, the routing and arbitration of a packet at the header of a queue is only simulated when the queue has finished transmitting any previous packets. Time-driven simulations are typically easier to model, and event-driven simulators have faster execution times, particularly under low level workloads.

1.6.1 Synthetic traffic patterns

The performance of a system network varies significantly depending on the workload characteristics. In the analysis and development of a system network, a network architect needs to ensure an adequate level of performance in a wide range of traffic workloads. A typical approach is to characterize the performance for an array of *synthetic traffic patterns* in which each node sends packets at a given injection rate and the destination of the packets follows a spatial distribution (either uniform, or a function of the source node ID) [57].

This work focuses on three different traffic patterns. Under two of them (random uniform and adversarial), nodes generate packets at a steady injection rate following



(a) ADV+1. (b) ADV+h.

Figure 1.18: Example of adversarial traffic patterns in a Dragonfly network of size $h = 2$. Highlighted local links correspond to the outgoing traffic from one of the groups.

a Bernoulli process. The other pattern models the existence of traffic bursts. The uniform and bursty uniform patterns represent a general case of use and characterize the behavior in a best case scenario for a Dragonfly network, whereas the adversarial traffic tests the behavior in a particular corner case for the Dragonfly topology.

1.6.1.1 Random Uniform Traffic (UN)

In the random uniform (UN) traffic, nodes send packets following a Bernoulli process in which the probability of generating a new message depends on the injection rate. The destination of each message is randomly selected across all the nodes in the network. This traffic pattern is benign for the network and is commonly employed in the evaluation of interconnects to provide a performance estimation in a best-case scenario, since it balances the use of the network links [46]. It also appears in real-world applications, as will be demonstrated in Chapter 2. Under UN traffic, MIN represents the optimal routing mechanism because the workload already has spatial and temporal uniformity, and using the shortest path achieves optimal latency.

1.6.1.2 Adversarial Traffic (ADV)

The adversarial traffic pattern constitutes the worst-case scenario for a network, stressing certain links to become bottlenecks and providing a lower bound for the network performance. In the particular case of a Dragonfly network, all nodes in a given group send their traffic to nodes in consecutive groups. Which consecutive group is selected depends on the concrete variant of adversarial traffic, ranging from ADV+1 (the traffic goes to the following group) to ADV+h (the traffic goes to the h groups away from current). The main consequence is that all minimally routed traffic between two

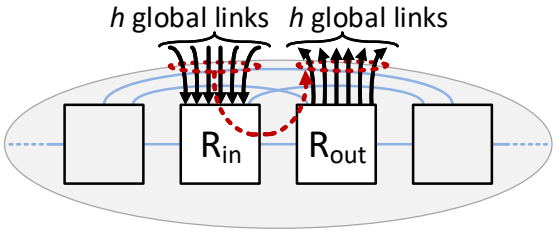


Figure 1.19: Bottleneck at the intermediate group in ADV+h traffic, where the h global input and output links are joined by only one local link.

groups employs a single global link, which becomes a bottleneck and hinders performance restricting throughput to $\frac{1}{ap}$ $\left(=\frac{1}{2h^2}\right)$ if the network is balanced. This limit is extremely low and lowers linearly with the network size, enforcing the need for alternative routing, such as the oblivious VAL and the different adaptive routing mechanisms. Figure 1.18 illustrates the case of ADV+1 and ADV+h traffic patterns in a sample Dragonfly network of size $h = 2$.

Adversarial traffic in Dragonfly networks has been widely used in previous works such as [91, 88, 82, 122, 61, 87]. ADV+1 is the most common variant employed; however, ADV+h adds further knowledge about unfairness and performance degradation due to an unbalanced use of the local links which is particularly harmful in many non-minimal routing mechanisms, as was first observed in [63]. It is frequent that the minimal local hop in the intermediate group can be avoided with VAL-group routing (skipping the second local link from the $l-g$ $-$ $l-g-l$ route proposed in [91]), because the global links are connected to the same router in the intermediate group. However, in ADV+h the incoming and outgoing global links are in two separate routers only linked by one local link which saturates and becomes a bottleneck, as it is highlighted in Figure 1.19. This traffic pattern requires a nonminimal local hop in the intermediate group to improve performance. Figure 1.19 highlights the bottleneck at the local minimal hop in the intermediate group.

In this work, an additional variant of the adversarial traffic pattern is proposed, named *adversarial consecutive* (ADVc). This pattern sends its traffic not only to a single group but to multiple consecutive groups attached to the same router in the current group. As it will be comprehensively explained in Section 3, this traffic presents a challenge in terms of throughput unfairness and can lead to starvation at certain nodes.

1.6.1.3 Bursty Uniform Traffic (Bursty-UN)

UN traffic provides an scenario where the communications display spatial and temporal uniformity, but it does not take into consideration the temporal non-uniformities that can appear as a result of traffic bursts in the applications. The bursty uniform traffic emulates such case, presenting temporal uniformity only if a long enough period of time is considered. This traffic is modelled through a Markov chain with two *ON* and *OFF* states; nodes only attempt message generation when they are at the *ON* state.

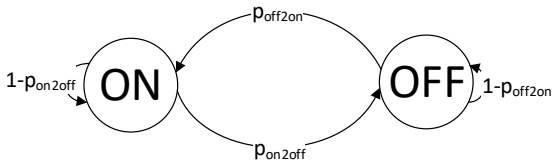


Figure 1.20: State diagram of the Bursty-UN traffic pattern.

Figure 1.20 shows the state diagram of the model. The destination of the messages is randomly selected across the network nodes when transiting to the *ON* state, and remains unchanged while the node stays at the *ON* state. In the employed model, nodes can transit from *ON* to *OFF* state and back to *ON* within the same cycle, to allow non-fixed destination traffic bursts while keeping full injection rate. This pattern is statistically similar to the traffic bursts approach for the evaluation of LAN switching networks described in the Requests For Comments 2544 [29] and 2889 [104].

The probabilities of being at a certain state (P_{ON} and P_{OFF}) are defined in Equations 1.4- 1.5 and depend on the injection rate and the size of the messages. The probability of transiting from the *ON* to the *OFF* state (p_{on2off}) is the inverse of the average length of the bursts (in messages), expressed in Equation 1.6. The probability of transiting from the *OFF* to the *ON* state, p_{off2on}, is defined in steady state by Equation 1.7 as a function of the probability of being at either state. Steady state implies that the probability of being in either state is independent of the time, and requires the Markov chain to be regular.

$$P_{ON} = \frac{injectionrate}{packetsize} \tag{1.4}$$

$$P_{OFF} = 1 - P_{ON} \tag{1.5}$$

$$p_{on2off} = \frac{1}{B} \tag{1.6}$$

$$p_{off2on} = \frac{P_{ON}}{B + P_{ON}(1-B)} \tag{1.7}$$

1.6.1.3.1 Obtention of the BURSTY-UN model The probability of being at state *ON* is set by the average injection rate at which nodes operate, because a node at state *ON* tries to inject one packet per cycle. In a Markov chain, the sum of state probabilities needs to be 1, and therefore establishes the probability of the *OFF* state. Transiting from *ON* to *OFF* state depends only on the average length of a burst.

The probability of being in state *ON* is a function of the transit probabilities:

$$\begin{aligned} P_{ON}(t=i) = &\left(1 - p_{on2off}\right) \cdot P_{ON}(t=i-1) \\ &+ p_{on2off} \cdot p_{off2on} \cdot P_{ON}(t=i-1) \\ &+ p_{off2on} \cdot P_{OFF}(t=i-1) \end{aligned}$$

If the Markov chain is regular, the probability of being in either state is independent of the time and the equation can be solved as follows:

$$\begin{aligned} P_{ON} &= \left(1 - p_{on2off} + p_{on2off} \cdot p_{off2on}\right) \cdot P_{ON} \\ &\quad + p_{off2on} \cdot P_{OFF} \\ &= \frac{p_{off2on}}{p_{on2off} + p_{off2on} - p_{on2off} \cdot p_{off2on}} \end{aligned}$$

which, replacing P_{ON} and p_{on2off} by their definitions in Equations 1.4 and 1.6, leads to the definition of p_{off2on} in Equation 1.7.

1.6.1.3.2 Proof of Markov chain regularity A Markov chain is said to be *regular* if its transition matrix A has a power A^i for which all transition probabilities are strictly greater than 0. The transition model for the Bursty-UN traffic is defined by the matrix in Equation 1.8, where first column and row correspond with state *ON* and the second row and column to *OFF*. Transitions from *ON* state (first column) take into consideration that a transition from *ON* to *OFF* can trigger another transition within the same cycle.

$$\begin{aligned} A &= \begin{bmatrix} P(ON \to ON) & P(OFF \to ON) \\ P(ON \to OFF) & P(OFF \to OFF) \end{bmatrix} \\ &= \begin{bmatrix} 1 - p_{on2off} + p_{on2off} \cdot p_{off2on} & p_{off2on} \\ p_{on2off} \cdot \left(1 - p_{off2on}\right) & \left(1 - p_{off2on}\right) \end{bmatrix} \end{aligned} \tag{1.8}$$

Replacing the transition probabilities with Equations 1.6 and 1.7 verifies that for any finite burst size (B),

$$B \geq 1, B \neq \infty \Rightarrow p_{on2off} > 0$$

As for p_{off2on}, it is strictly greater than zero except for the extreme P_{ON} values:

$$0 < P_{ON} < 1 \Rightarrow p_{off2on} > 0$$

In the extreme values ($P_{ON} = 0$, $P_{ON} = 1$) the Markov chain is *absorbing*, as it immediately reaches an absorbing state which never leaves (*ON* state if $P_{ON} = 1$, *OFF* if $P_{ON} = 0$):

$$P_{ON} = 1 \to p_{off2on} = 1 \to A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

$$P_{ON} = 0 \to p_{off2on} = 0 \to A = \begin{bmatrix} 1 - p_{on2off} & 0 \\ p_{on2off} & 1 \end{bmatrix}$$

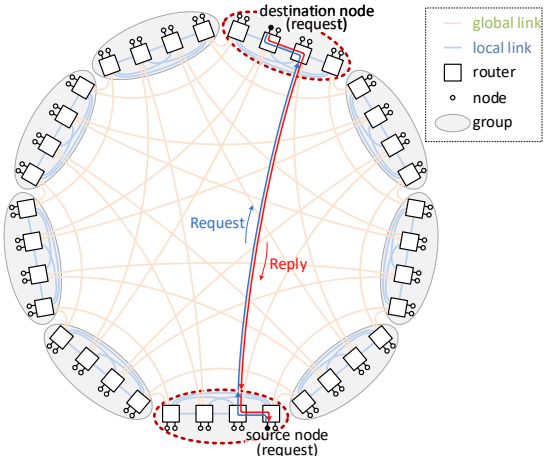


Figure 1.21: Example of request-reply traffic in a Dragonfly network of size $h = 2$. When the upper highlighted node receives a request message, it generates a reply towards the source of the petition.

1.6.2 Request-reply Traffic

The previous traffic patterns consider an asymmetrical traffic exchange where message generation is independent of message reception. However, many applications have a reactive nature where the reception of a message triggers a response towards the source of the request. Figure 1.21 shows an example of this exchange in a Dragonfly network. Applications with such reactive nature can lead to protocol deadlock, as described in Section 1.3.

This work employs request-reply traffic where request messages are generated following one of the previously described traffic patterns: random uniform, bursty uniform and adversarial. Upon the reception of a request, the destination sends a reply message back to the source of the request. Since the injection of a request eventually generates two messages (request and reply) the injection rate of the request messages needs to be half of the injection rate in non request-reply patterns. Request-reply traffic allows to evaluate the behavior and performance with a protocol deadlock avoidance mechanism.

1.6.3 FOGSim Network Simulator

The results presented in this work have been obtained through the execution of the FOGSim network simulator. FOGSim [61] is a modular time-driven, synthetic traffic pattern-based network simulator developed by the Department of Computer Science and Electronics at the University of Cantabria. It has been employed in previous works by García [63, 64, 60, 62] and Benito [22, 23].

FOGSim supports Dragonfly network topologies of different size under a range of varied synthetic traffic patterns, plus traces of parallel applications following the format of Dimemas [95]. Several routing mechanisms are implemented, either oblivious

or adaptive (relying on information about the status of the network), according to the descriptions in Section 1.4. Deadlock is prevented following a distance-based deadlock avoidance mechanism that uses different VCs for packets at different steps of their paths, in a similar fashion as the proposal from Günther [69].

Simulated network routers employ VCT switching and can either be input-buffered or input-output-buffered; the latter is required when the router crossbar operates at faster paces than the network links, because the output buffer decouples the crossbar and link speeds. Router resources are assigned through an input-first separable allocator following a two-stage arbitration, one stage for the competing VCs at every input and one for the competing inputs at every output. Output arbiters only receive up to one request per input, corresponding to the VC granted by the input arbiter. At each stage, multiple arbitration policies can be implemented; default behavior is a Round Robin policy in which the first port to serve is the follow-up to the last port to receive a grant. The simulation is cycle-accurate and is performed at a phit-level, modelling the advance of the data transfer unit per clock cycle of the router.

1.6.3.1 Input parameters

Typical input of the simulator includes the size of the network and router characteristics. Input parameters can either be specified through a mandatory parameters file or via the command prompt as execution arguments. The sole exception is the name of the parameters file, which must always be input as the first argument of the simulator executable. Some of the input parameters are:

- Size of the network (h, a and p).
- Buffer sizing. Routing can be input-buffered or input-output-queued. User can set the size of the buffers, distinguishing between injection, local link inputs, global link inputs and outputs.
- Internal speedup, sets the crossbar traversing rate compared to link data rate. Speedups greater than 1x require output buffering to act as a cache for the link. The router can also employ input speedup, specifying the number of concurrent accesses to the crossbar per input port of the router.
- Link delay. Number of cycles employed to traverse a link, distinguishing between local (intra-group) and global (inter-group) links.
- Crossbar delay. Number of cycles employed to traverse the crossbar.
- Number of VCs, distinguishing between injection, local and global input ports. The number of VCs for local and global ports is normally tied to the deadlock avoidance mechanism; in this work different combinations of routing and deadlock prevention mechanism are used, with each one having different requirements for the number of VCs. Injection VCs mitigate HoLB and increase performance, and they are completely unrelated to the deadlock avoidance purpose

of local and global VCs. Users can configure the VC selection policy used at injection, choosing between random, destination-dependent and JSQ (Jump to the Shortest Queue) policies.

- Global links arrangement, can either follow a palm tree configuration or a consecutive order starting by the first group connected to the first global link of the group.

- Arbitration policy. This is separately defined for the input and output arbitration phases of the allocation procedure. Arbitration policy can be Round-Robin (RR), Least-Recently Served (LRS) or Age-based. Round-robin serves first the port following the last one that has been granted the resource. LRS always checks first those port that have spent longest time without being granted the resource. It must be noted that the input and output arbitrations are handled separately: for a VC to be considered as attended, it only needs to place a request for an output, regardless it is granted or not. Age-based gives priority to oldest packets. Output ports arbitration can be selected to give priority to packets coming from transit inputs over new injections, in a similar procedure to the in-network priority fraction of the BlueGene/L interconnection network [7]. The in-transit priority allows a higher network drainage and greater performance at a cost of unfairness and node starvation, as is analyzed in Chapter 3.

- Traffic pattern, can either be one of several synthetic traffic patterns or a trace. In the former case, it can be random uniform, adversarial, or a mix of them. The mix can either be concurrent, where nodes follow a uniform or adversarial pattern depending on the specified proportion between them; or consecutive. Consecutive mixes permit to observe the transition between patterns and the velocity of the network to adapt to them. Uniform and adversarial traffic patterns can also be of request-reply type, where request messages are generated at a constant rate through a stochastic process, and the arrival of a request triggers the generation of a reply towards the source of the request.

- Routing mechanism, which can be oblivious (MIN, VAL) or adaptive. Adaptive routing mechanisms receive as input parameters the threshold values for the misrouting decision.

- Number of simulated cycles. This consists of two parameters, one for the total duration of the simulation, and one for the number of cycles devoted to warming up the network. This increases the statistics accuracy, since it captures the network behavior in a steady use. If warmup is not specified, it is assigned the same duration as the statistics-tracking execution, effectively doubling the simulation time. Statistics are only obtained once the warmup has finished and up to the end of the execution. The length of the simulation can be overridden for certain configurations, such as when a fixed number of sent packets is set and the simulation finishes once they are all delivered to their destinations.

- Injection probability, sets the injection rate of the network nodes in percentage of maximum theoretical injection rate allowed by the network; an injection probability of 100% corresponds with a rate of 1 injected phit per node and cycle.

- Seed for the pseudo-random values, favours reproducibility of the experiments. For a given seed and network configuration, the output remains unchanged. Averaging multiple executions with different seeds is necessary to eliminate the inherent variability between simulations of the same configuration.

1.6.3.2 Simulator output

During the execution, the simulation prints temporal statistics of the number of injected and delivered packets at regular intervals. Statistics obtained during network warmup are discarded and reset after the warmup is completed. When the execution is finalized, the simulator prints a set of statistics and the configuration parameters to an output file. Some of the most relevant statistics are:

- Throughput, defined as the number of received phits divided by the number of cycles and the number of nodes. Total number of packets received (and sent) is additionally provided. In the case of request-reply traffic, throughput is also separately reported by traffic flow (request or reply).

- Latency, given as a total, minimum, maximum and average per packet. The latter is broken down to network and injection latency.

- Number of misrouted packets received. These are also discriminated by the misrouting category they fall into (through a local link, through a global link, through a global link after a nonminimal local hop) and the group at which the misrouting is performed (source, destination or intermediate group).

- Number of performed hops (links traversed). This figure is given as a total and as a per-packet average distance. In the latter case, it is broken into local and global hops.

- Port and VC usage, averaged from all the network routers.

- Average input and output buffer occupancy, segregating per input port and VC (in the case of input port buffers; at the output ports, packets are not stored in different queues per VC).

- Average allocation rate, providing petitions performed from the input to the output arbiters and their attendance rate.

- Unfairness metrics: minimum and maximum injections from any network router, unbalance quotient and coefficient of variation (CoV).

Table 1.2: List of parameters employed in the simulations.

Parameter	Value
Router size	31 ports (h=8 global, p=8 injection, 15 local)
Group size	16 routers, 128 computing nodes
System size	129 groups, 16,512 computing nodes
Router buffering	Combined Input-Output-Queued (CIOQ)
Router latency	5 cycles
Crossbar frequency speedup	2×
Crossbar latency	3 cycles
Global link arrangement	Palm tree [33]
Link latency	10 (local), 100 (global) cycles
Switching	Virtual Cut-Through
Packet size	8 phits
Buffer size (phits)	32 (output buffer, local input buffer per VC), 256 (global input buffer per VC) 256 (injection input buffer per VC)
Virtual Channels (VCs)	2 (global ports), 3 (local and injection ports), 4 (local ports when VAL or PB routing is used)
Simulation length	60,000 cycles (warmup) + 60,000 cycles (tracking statistics)
Number of simulations	5 per point

1.6.4 Simulation configuration

Unless otherwise stated, the results presented in this work have been obtained through the FOGSim simulator employing the configuration detailed in Table 1.2, with a network of 2064 routers and 16512 compute nodes.

This configuration emulates the behavior of 2- and 20-meter wires for the local and global cables, as in the evaluation performed by Jiang *et al.* in [82]. This corresponds to a router working at 1 GHz and a transmission speed of 10GB/s (which translates into a phit size of 10 bytes). A packet size of 8 phits equals 80 bytes and should allow for a 64-byte payload as occurs in the Cray Cascade network [56].

1.7 Contributions

The main contributions of this work are:

- The development and implementation of a synthetic traffic model of the communications in the Graph500 benchmark. This model allows to predict the performance of a system running this data-intensive benchmark without the memory constraints associated with the use of traces or full system simulations, and is the first of its kind as far as the author is concerned. Its development includes

a thorough analysis of the communications, which exposes the significance of message aggregation in the execution of the benchmark.

- An analysis of the communications behavior over a large Dragonfly network when only a reduced set of nodes is employed, leading to a pathological unfairness effect. This analysis rejects the use of an in-transit traffic priority for fairness reasons.

- The proposal of a synthetic traffic pattern that reproduces such pathological behavior, and an analysis of the explicit mechanisms required to prevent the associated throughput unfairness.

- A novel misrouting decision based on *contention counters* that decouples misrouting from buffer occupancy and provides faster adaption to traffic changes. This misrouting decision improves performance by reacting to the roots of network congestion rather than its consequences, and does not entail a high implementation cost. Multiple variants of the mechanism are also proposed, evaluating their performance benefits and the trade-off in implementation complexity.

- A new buffer management mechanism denoted *FlexVC* that permits a more flexible use of VCs through opportunistic routing and a relaxed distance-based deadlock avoidance policy. It partially decouples the amount of VCs from the deadlock avoidance mechanism and provides finer granularity in the selection of the number of buffers to be used. Furthermore, FlexVC allows to reduce up to 50% the memory requirements of the router while preserving or improving the performance of the network, at a minimal implementation expense which is more than compensated by the reduction in the number of buffers needed.

This work starts evaluating the impact of the system network on the performance of a data intensive application in Chapter 2. This analysis is conducted through a novel synthetic traffic model of the Graph500 benchmark, which is intended to constitute a representative workload of BigData applications. Chapter 2 also provides an implementation of the traffic model in the FOGSim simulator. Results upon a Dragonfly network validate the importance of the network and unveil a pathological case of unfairness when less than the whole system is devoted to the execution of a single application. Chapter 3 explores the severity of this unfairness in resource allocation depending on the routing mechanism and the traffic pattern, and the solutions needed to mitigate the associated performance degradation. It also presents a new adversarial traffic pattern linked to a certain distribution of concurrent applications, and links it to the Graph500 synthetic traffic pattern. This adversarial traffic motivates a pathological case of unfairness and requires specific fairness mechanisms to mitigate it.

After assessing the need for high-performant interconnects, this work proposes two novel paradigms to improve the proficiency of the network. Chapter 4 introduces the use of contention information to perform the misrouting decision in nonminimal

adaptive routing, instead of relying on traditional credit-based queue occupancy status. This allows to decouple the misrouting decision from the size of the buffers, and overcome the limitations of congestion-based adaptive routing: slow adaption to traffic changes, dependency on buffer sizes and oscillations in routing. This proposal is especially beneficial in high-radix routers like those targeted for Exascale system networks. Chapter 5 presents FlexVC, an innovative buffer management mechanism that allows a more flexible use of the virtual channels and reduces the number of resources needed in adaptive routing. FlexVC outperforms traditional buffer management when an otherwise identical router is considered. FlexVC also achieves similar performance to traditional buffer management with a smaller set of buffers. Furthermore, it provides higher granularity in the trade-off between performance and the number of resources employed.

Chapter 6 outlines related works to the proposals presented on this work and discusses their merits. Finally, Chapter 7 summarizes the contributions of this work and provides a set of conclusions and a series of future research lines.

Chapter 2

Graph500 Synthetic Traffic Model

This chapter presents a synthetic traffic model of the communications in the Graph500 benchmark, and provides an implementation in an open-source network simulator. This model eases the evaluation of the network interconnect and its impact under BigData applications.

Network architects employ simulators as a tool to analyze and predict the behavior and performance of a given system under different scenarios, as it has been discussed in Section 1.6. Synthetic traffic patterns constitute the fastest method to perform a characterization and allow to focus on different cases or certain network features. However, they sometimes fail to accurately represent the workloads the system will be subject to.

Benchmarks attempt to characterize a system under a workload representative of a set of real world applications. For decades, High Performance Computing (HPC) systems have been ranked through the Linpack benchmark [51]. This benchmark performs like an archetypical HPC application following the Bulk-Synchronous Parallel execution model (BSP) [144], [145], where long bursts of computation are interleaved with phases of network communication to synchronize the processes.

In the last few years, the preponderance of HPC applications have been overshadowed by the need for BigData applications that analyze and try to extract knowledge from large sets of data, such as the MapReduce [47] or Spark [154] frameworks. BigData applications differ significantly from typical HPC workloads and are mainly constrained by memory and IO requirements. The Graph500 benchmark [2] appears in 2010 with the aim to reproduce the characteristics of BigData applications and steer the design of new systems to match their needs.

Unfortunately, the intensive use of memory and network of BigData applications restricts significantly its evaluation on network simulators, both with full-system simulations and traces. In order to alleviate such limitations, this work provides a synthetic traffic pattern that exemplifies the network usage of the Graph500 benchmark but has low memory requirements. In-depth analyses of the benchmark communications can be found in the works of Anghel *et al.* in [14] and [15].

In this chapter a comprehensive description of the proposed traffic pattern is provided, with an outline of the process followed for its development. A sketch of its

implementation on the FOGSim simulator is also included; this implementation is publicly available and can be handily adapted to other simulators.

2.1 Analysis of the communications in the Graph500 benchmark

The Graph500 [109] benchmark is based on the execution of a Breadth-First Search (BFS) over an undirected graph. BFS is a strategy to traverse a graph organizing all its nodes (or *vertices*) in a tree, starting by a given root vertex. Vertices are connected through links denoted as *edges*, and the number of edges per vertex is referred as *vertex degree*. The search of the tree is conducted in multiple stages or 'graph levels', traversing all the edges connected to vertices visited in the previous level. The vertices are analyzed in a FIFO-queue fashion. All the vertices in the graph are therefore visited in order of their distance from the root vertex. One of BFS uses is to find the path which traverses the lowest number of edges between two specific vertices in a graph. More details about BFS and some alternative implementations are given in [131].

The benchmark consists of two kernels: one to generate the graph and the BFS itself. Both kernels are timed, but the benchmark metrics only account for the time invested in the BFS execution. The benchmark provides four different implementations of the BFS. Both the analysis of the communications in the benchmark and the proposed traffic model focus only on the BFS in its most scalable *simple* implementation. A thorough comparison of the different implementations can be found in [141].

The benchmark receives two parameters, *scale* and *edgefactor*. Scale is the base two logarithm of the number of vertices in the graph. Edgefactor is the proportion of edges to vertices in the graph, which equals the half of the average number of edges per graph vertex. The size of the graph is fully determined with the values of scale and edgefactor. The benchmark returns the time employed for the BFS execution and a performance metric called Traversed Edges Per Second (TEPS). The number of TEPS is calculated as the quotient between of the number of edges traversed in the graph (which depends on the input parameters) and the execution time for the BFS.

A pseudocode of the BFS is given in Figure 2.1. The benchmark is run with several processes executing in multiple nodes and communicating through MPI routines. The graph is evenly split into the processes, with each process hosting a fair subset of the vertices. Coupled with the random nature of the graph, this provides a more or less random distribution of the communications across the benchmark execution, similar between different processes. Communications along the BFS execution are tied to the levels of the resulting tree. In each level, processes search the neighbors of the current frontier of the tree and mark them to visit in the following level. If the neighbor vertex is allocated to a different process a query is generated, corresponding to a petition to determine if the vertex has already been visited or not. These queries translate into point-to-point messages.

From the pseudocode it can be observed that the communications consist of two

```
1:  visited = current = ∅
2:  next = {root}
3:  repeat
4:     current = next
5:     for all vertex in current do
6:        for all neigh in Neighbors(vertex) do
7:           if neigh hosted in another process then
8:              MPI_Isend [vertex,neigh] to host process
9:              while not MPI_Test (outgoing messages completed) do
10:                if outgoing message completed then
11:                   clear sending buffer
12:                if MPI_Test (incoming messages) then
13:                   [vertex,neigh] = receive()
14:                   if neigh not in visited then
15:                      visit neigh and add it to visited and next
16:          else
17:             if neigh not in visited then
18:                visit neigh and add it to visited and next
19:    for all process do
20:       MPI_Isend (this process has stopped sending)
21:    repeat
22:       if MPI_Test (incoming messages) then
23:          [vertex,neigh] = receive()
24:          if neigh not in visited then
25:             visit neigh and add it to visited and next
26:    until rest of processes have stopped sending
27:    MPI_Allreduce(next)
28: until next is empty
```

Figure 2.1: Pseudocode of the BFS, pointing the placement of the communications.

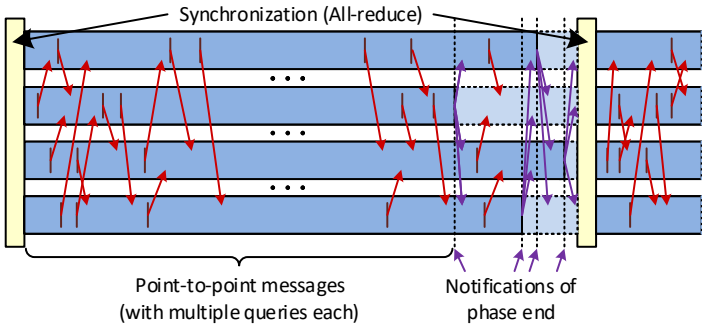


Figure 2.2: Outline of the communications in each BFS phase.

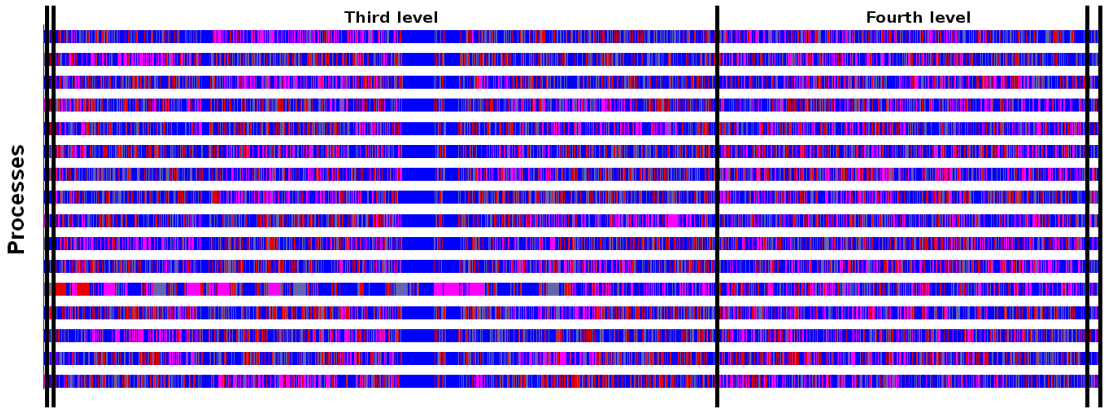


Figure 2.3: Trace of an actual execution with 16 processes. Scale $s = 22$ and edgefactor $f_e = 16$. Blue blocks represent computation, pink blocks represent the dispatch of a message, and black vertical lines mark the all-reduce collectives.

asynchronous send calls (lines 8 and 20) whose reception is polled at the receiver (lines 9,12 and 22), and an all-reduce collective operation (line 27) which acts as a synchronization point and also distributes the number of new visited vertices in the stage. Figure 2.2 portrays an outline of the communications within a single phase of the BFS, depicting the dispatch of messages through arrows. From a network perspective, each tree level consists of a batch of point-to-point messages succeeded by a final notification to all other processes, to signal the end of the level within the process. Processes deliver a number of point-to-point messages through the *send* call in line 8. Once a process has finished dispatching messages at the current phase, it broadcasts a phase end notification (line 20 of the pseudocode). The end of each phase is marked by the all-reduce (line 27), which comprises a reduction and a broadcast. Between the dispatch of the notification of phase end and the beginning of the all-reduce (shown in light blue), processes only perform the associated computation of the incoming messages. The length of this block is determined by the slowest process to enter the all-reduce.

Figure 2.3 presents a trace of a BFS taken from the execution of the Graph500 benchmark over a graph of *scale* 22 and *edgefactor* 16, employing 16 processes. Blue blocks represent computation, and pink blocks correspond to the dispatch of a message. Black vertical lines point the all-reduce collectives. It can be appreciated that communications are interspersed with computation along the execution, with message shipment uniformly distributed across the execution.

The number of messages is evenly distributed across each stage but varies significantly between tree levels, with two big levels typically comprising the majority. The impact of the point-to-point messages to notify the end of a tree level is negligible, because the number of messages sent is several orders of magnitude lower than the point-to-point queries between processes. Figure 2.4 displays the total number of point-to-point messages sent during an execution of the BFS, with a number of end-of-message-dispatching notifications 30 times lower than the point-to-point query

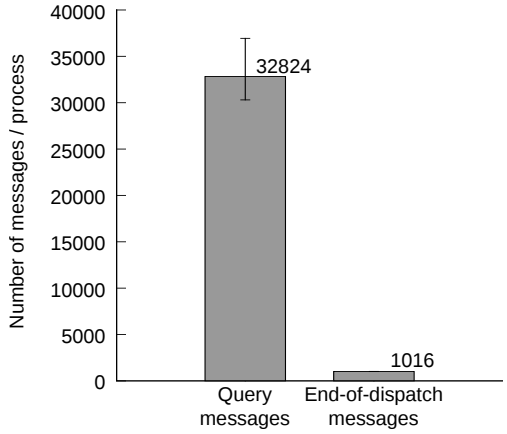


Figure 2.4: Number of messages sent per process during BFS, for an execution with 128 processes of a graph with scale $s = 25$ and edgefactor $f_e = 16$. Bar value represents average value, errorbar shows standard deviation between processes.

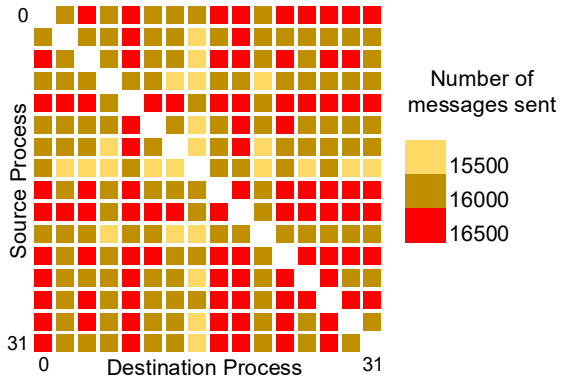


Figure 2.5: Communication matrix for point-to-point exchanges across processes, for a graph of scale $s = 20$ and edgefactor $f_e = 16$. Three ranges of values are distinguished. Spaces in white correspond to the absence of self-messages.

exchanges. The all-reduce call is also infrequent, and is only important because it synchronizes the generation of new messages.

From a network perspective the major communications are the point-to-point exchanges, which are highly homogenous spatially following the even distribution of the graph across the processes. Figure 2.5 represents the communication matrix for said messages for a sample BFS execution with 16 processes. Rows correspond to the message source processes, and columns to destinations. The values are very similar among all the source-destination process tuples, with less than 8% difference between maximum and minimum values. Moreover, there is a clear symmetry in the matrix, with a similar amount of messages sent in both directions in any process tuple. This symmetry comes as a result of the undirect nature of the graph, which forces all edges to be traversed twice during the search - once per each way.

Graph500 exploits message aggregation in these point-to-point messages to reduce

Table 2.1: List of abbreviations employed in the equations.

Abbr.	Parameter	Description
s	Scale	Base 2 log of the number of vertices in the graph.
f_e	Edgefactor	Half of the average vertex degree.
c_s	Coalescing size	Amount of explored edges aggregated per message.
n	Number of nodes	Number of nodes (processes) employed in the benchmark execution.
n'	Number of destination nodes	Number of nodes (processes) which actually receive a message in a given level.
$\sum q$	Number of queries	Number of queries sent from each process across the whole BFS execution.
q	Number of queries per process	Number of queries sent from each process per tree level.
m	Messages per process	Number of messages sent from each process per tree level.
E_l	Edges per tree level	Total number of edges explored within each stage of the BFS.
d_r	Degree of the root	Number of edges connected to the root vertex.
l	Tree level	Stage of the tree in the BFS execution, starting at 0.

network traffic, with every message grouping multiple queries up to a value named *coalescing size*. Incomplete messages smaller than the coalescing size correspond to the last queries from a process that has completed the traversal of a tree level. The default coalescing size value is 256 queries per message, which compels 4KB messages (each query has a size of 16 bytes). The coalescing size is not an input parameter to the benchmark, but it can be easily tuned to better fit the network characteristics and improve performance.

Table 2.1 provides a summary of the abbreviations that are employed in the equations of the model, with a brief description of each parameter. The *scale*, *edgefactor* and *coalescing size* are input parameters of the benchmark, and the number of nodes n is directly tied to the benchmark execution; the rest of the variables in the table are employed internally in the benchmark.

The total number of queries sent by a process during the BFS, described in Equation 2.1, is determined by the number of explored edges connecting to vertices in another process. The equation considers the total number of edges in the graph ($2^s \cdot f_e$) traversed in both ways, uniformly distributed between processes ($1/n$), subtracting those edges hosted within the sender process because they do not generate a query ($(n-1)/n$). It must be noted that applying this equation to the parameters of Figure 2.4 and considering a coalescing size of 256 results in a total of 32512 messages per process, which is only 1% less than the observed value. The divergence is a result of incomplete messages, as the total number of queries to be generated is divided into

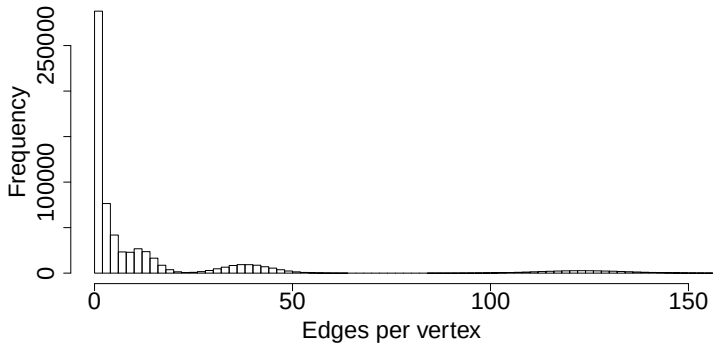


Figure 2.6: Histogram of the vertex degree of the graph, truncated to 150. Graph scale is $s = 17$, edgefactor is $f_e = 16$.

multiple tree levels and destinations.

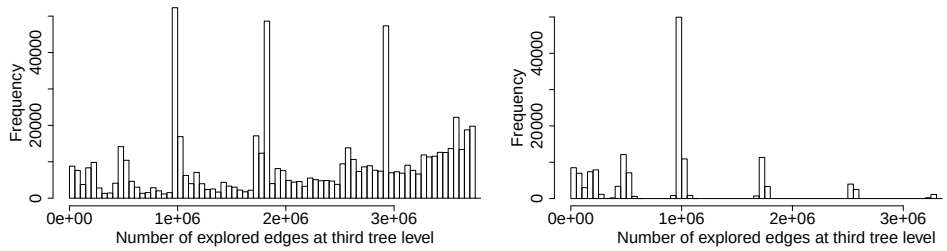
$$\sum q = 2^{s+1} \cdot f_e \cdot \frac{n-1}{n^2}. \tag{2.1}$$

Graphs in the benchmark are generated through a Kronecker matrix product similar to the Recursive MATrix (R-MAT) scale-free graph generation algorithm [34]. Graphs generated by R-MAT emulate the behavior of small-world phenomena, where a small fraction of the vertices have a significantly large number of direct connections with other vertices, and a large proportion of the vertices have a low vertex degree [149].

Figure 2.6 displays a histogram of the vertex degree measured for a graph generated in the benchmark. A reduced set of sparse higher degree values (up to around 30,000 edges) has been truncated from the figure for ease of visualization. It can be appreciated that the distribution is heavily condensed in low degree values, with a trend of tails that favor some high degree values over others. The distribution of the vertex degree in Kronecker matrix product-generated graphs such as the one used in the Graph500 benchmark is most accurately described as a series expansion from normal distributions [68], but it can be characterized through a combination of a lognormal and an exponential [133] or simply with a lognormal [93] without losing significant accuracy.

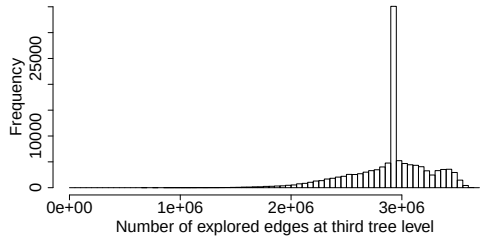
2.2 Synthetic Traffic Model

The Graph500 synthetic traffic model replicates the staged structure of the benchmark introduced in the previous section, with large batches of uniformly distributed point-to-point messages ending in a collective all-reduce operation. The number of messages for each stage cannot be defined as a fixed value, because it is highly dependent on the size of the graph in the benchmark, and it varies significantly depending on the degree of the vertex selected as root. Instead, the number of messages per stage is generated through a Gaussian distribution whose mean and standard deviation are a function of the input parameters of Graph500. The equations for the mean and standard deviation

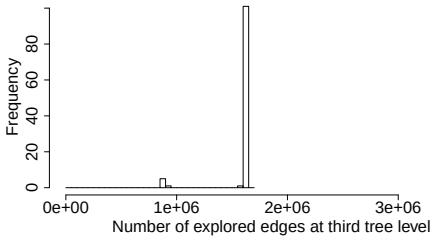


(a) Aggregated for all possible d_r.

(b) Roots with $d_r = 1$.



(c) Roots with $10 \leq d_r \leq 20$.



(d) Roots with $d_r \geq 10^4$.

Figure 2.7: Histogram of the number of explored edges in the third tree level, with different root degree d_r. Graph with scale $s = 17$ and edgefactor $f_e = 16$.

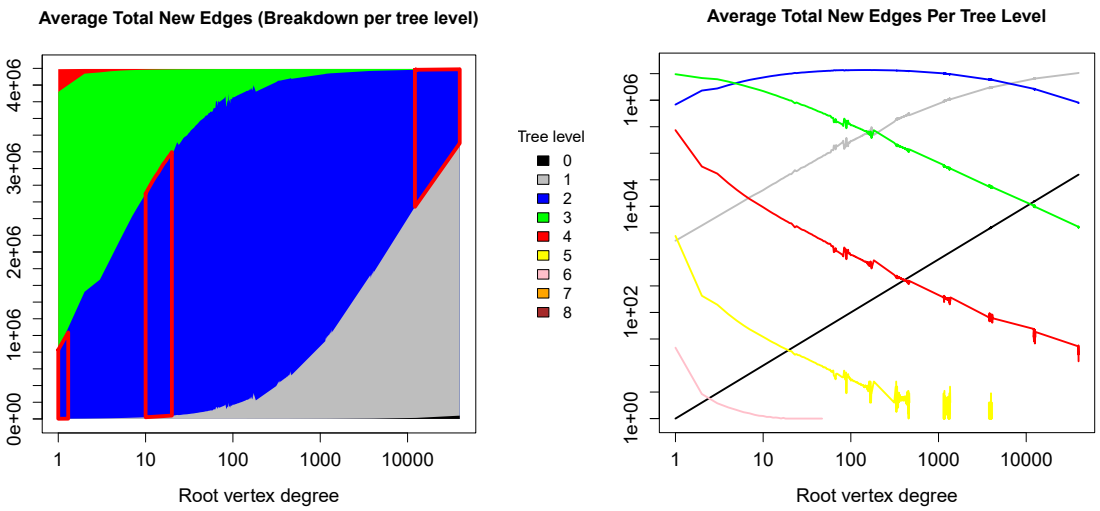
of these distributions are obtained empirically in Section 2.2.1 from a characterization of several benchmark executions with different input parameters.

From the network perspective, the most significant part of the benchmark execution for any nontrivial graph size will be the point-to-point communications, since the messages will significantly outnumber those during the all-reduce. Point-to-point message generation rate depends fundamentally on the node computation capabilities, and constitutes an input parameter to the model. Heterogenous systems can be modelled by tuning specific sets of nodes to under- or out-perform the rest of the system.

The number of messages depends directly on the amount of queries sent by each process, which varies by tree level. Message destination is selected randomly among all the other processes in the application, following the uniform distribution of the vertices observed in the analysis of the communications. The allocation of the number of explored edges per process among tree levels varies heavily, even though the total across the whole BFS execution is almost constant for graphs of the same size.

Figure 2.7a depicts the histogram of the number of new edges traversed in a graph during the third tree level ($l = 2$), which gathers most of the point-to-point communications across the whole execution. This histogram has been obtained by running a BFS for every root in the graph, masking the actual nature of the distribution by averaging multiple executions. It shows multiple peaks of similar impact, not presenting a distinctively clear distribution.

This is not the case if the histogram is restricted to only BFS executions for roots with the same or similar vertex degree. Figures 2.7b-2.7d display the average distribution of the number of explored edges in the third tree level for the same graph in



(a) Stacked values. (b) Curves.

Figure 2.8: Average number of explored edges per root degree, broken down per tree level. Values come from the same graph as in Figure 2.7. Note that the Y-axis is in logarithmic scale for the right figure.

Figure 2.7a, with three different ranges of root degree: roots with only one neighbour (Figure 2.7b), roots with 10 to 20 neighbors (Figure 2.7c) and roots with a high root degree of 10,000 neighbors or more (Figure 2.7d). It can be observed that for each histogram there is only one predominant peak, as there is a heavy dependence of the distribution on the root degree. Roots with low vertex degree originate a low amount of communications at the first tree levels, shifting the biggest part of the graph traversal further into the BFS execution, whereas roots with high degree will rapidly explore the majority of the graph and present low (or nonexistent) communication at higher levels. Therefore, the traffic model depends on the vertex degree of the root, what conditions the equations for each case.

The approach followed is to characterize empirically the mean and standard deviation of the number of edges explored in each tree level for each possible vertex degree of the root. Figures 2.8 and 2.9 depict the mean and standard deviation per tree level upon the root vertex degree. Results come from the same graph used to obtain the histograms in Figure 2.7. X-axis is displayed in logarithmic scale. Note that Y-axis in Figure 2.8b is also in logarithmic scale.

In Figure 2.8a, the three blocks circled in red correspond with the average number of explored edges in the third tree level whose distribution was presented in Figures 2.7b-2.7d. Some values in Figure 2.8 are interpolated, as not all the vertex degrees are present in a graph. This translates into the gaps in the curves of the standard deviation (Figure 2.9). The aggregated amount of edges in Figure 2.8a remains almost constant; since the size of the graph is independent of the vertex selected as root, the amount of edges to traverse during the BFS is similar. However, the distribution of

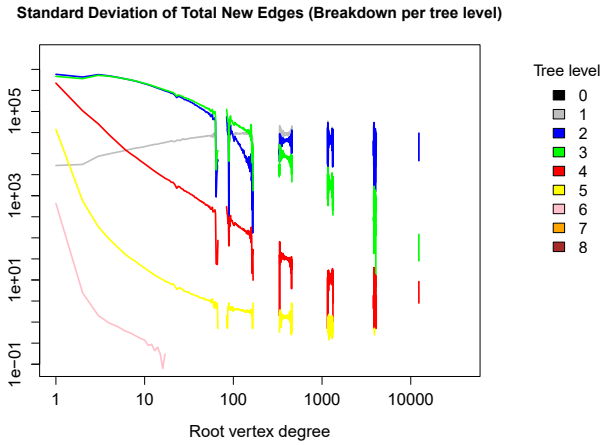


Figure 2.9: Standard deviation of the number of explored edges per root degree for each tree level. Values come from the same graph as in Figure 2.7.

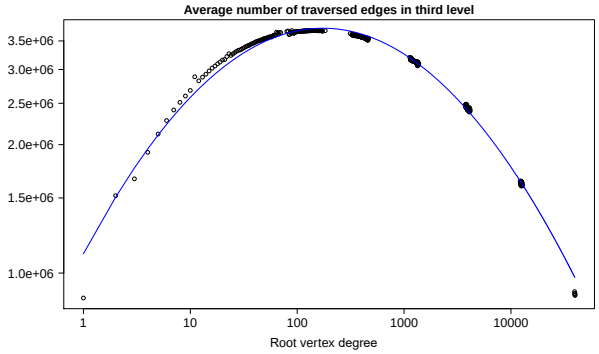


Figure 2.10: Example of a fitting curve for the third tree level upon graphs of scale $s = 17$ and edgefactor $f_e = 16$. Points correspond with the average number of new edges explored per degree at the root. Line represents the fitting curve responding to a linear combination of the natural logarithm of the root vertex degree.

those edges among the tree levels varies significantly depending of the vertex degree at the tree root.

The number of edges in the first tree level ($l = 0$) is the vertex degree of the tree root. The distribution of the mean for other levels in Figure 2.8b can be approximated through a polynomial of degree 2, where the variable is the natural logarithm of the vertex degree of the tree root. For each tree level, the coefficients of the polynomial in the fitting curve have been obtained through a model fitting tool based on the function described by Chambers in [35]. Figure 2.10 shows an example of a fitting curve for the mean of the number of explored edges in the third tree level in graphs of scale $s = 17$ and edgefactor $f_e = 16$.

A similar analysis is performed for the standard deviation in Figure 2.9, although in this case the approximation function changes significantly between tree levels. Lower

levels can be described through an exponential of a polynomial like the one used for the average, whereas higher levels fit better into an exponential of an inverse function. Interestingly, the second and third tree levels present a dual nature; in the second level there are two zones with different trends, and the second zone follows the curve from the first level. Something similar happens for the third tree level, in which the first points equal those of the fourth level and the remaining match the values of the second level; this third case cannot be properly spotted in this figure, but has been observed with higher scale sizes. In both cases, this is equivalent to taking the maximum between the crossing curves.

The analyses of the fitting curves of mean and standard deviation have been performed for several combinations of *scales* $s = 16, 17, 18, 19, 20, 23, 25$ and *edgefactors* $f_e = 16, 20, 32, 45, 64$. The empirical values come from the measurement of the number of explored edges per level across all processes in the BFS, running a tree search for every vertex in the graph, and multiple graphs for each graph size. These measurements are oblivious to the infrastructure employed, so they can be extrapolated to any other system. The collected coefficients of those linear combinations are generalized to a set of equations that follows their variations with the input parameters in the next section.

2.2.1 Equations of the model

In every level the number of messages sent for each process is determined following Equations 2.2- 2.4.

$$q = \frac{E_l}{n} \cdot \frac{n-1}{n} \tag{2.2}$$

$$n' = n \cdot \left(1 - \left(1 - \frac{1}{n}\right)^q\right) \tag{2.3}$$

$$m = \left\lceil \frac{q}{c_s} \cdot \frac{1}{n'} \right\rceil \cdot n' \tag{2.4}$$

Equation 2.2 determines the number of queries q per node and tree level, which equals the number of explored edges minus those edges hosted within the sender process, which do not span any network communication.

Equation 2.3 defines the actual number of destinations (n') as a Bernoulli experiment [114] with as many trials as queries sent and a uniform probability of selecting a given node as destination for a query. If $q \to \infty$, the number of effective destination equals the number of nodes in the network, n.

The number of messages given in 2.4 equals the quotient of the number of queries sent per tuple of source-destination processes and the coalescing size (c_s), times the number of destinations. The quotient is rounded up to account for incomplete messages that carry less than c_s queries.

The number of explored edges in the first tree level is trivially determined as the root degree. All the point-to-point communications in this first level are originated at only one node, the one hosting the tree root.

The number of queries per node and tree level is modelled through a Gaussian distribution; this method has the benefit of adjusting reasonably well to the observed behavior while remaining low-demanding computer-wise. The Gaussian distribution is defined by the mean and standard deviation obtained in the characterization described at the beginning of this section. Both mean and standard deviation are calculated as a function of the root degree d_r, the graph size (scale s and edgefactor f_e) and the tree level l. The notation for the tree level l spans from $l = 0$.

The evolution of the mean per tree level (as shown in Figure 2.8b) is estimated through a polynomial of degree 2 of the natural logarithm of the root degree, described in Equations 2.5- 2.8. The mean of the Gaussian distribution is truncated when the number of edges explored through the whole execution reaches the limit of twice the number of edges in the graph.

$$\ln\left(\overline{E_l}\right) \approx A \cdot \ln^2(d_r) + B \cdot \ln(d_r) + C, \qquad l \geq 1 \tag{2.5}$$

$$A = -0.133 + 0.0046 \cdot s + e^{0.01257 \cdot f_e - 0.1829 \cdot s - e^{1.75 - 0.7 \cdot l}} \tag{2.6}$$

$$B = 2 - l \cdot (0.91 + 0.002 \cdot f_e - 0.012 \cdot s) \tag{2.7}$$

$$C = e^{1 + (1 + 0.004 \cdot f_e) \cdot (-0.81 + 0.8411 \cdot \ln(s)) \cdot e^{-\frac{(l-2.8)^2}{30}}} \tag{2.8}$$

Equation 2.17 refers the model for the standard deviation, where tree levels with index $l > 3$ are approximated through the exponential of an inverse and second and third levels are the maximum of two curves, as described in the model approach. Functions $f_1(l, d_r)$ and $f_2(l, d_r)$ are determined by the formulas in Equations 2.9 - 2.16. These equations reflect a high divergence for roots with low degree, which have a much larger number of samples in the graph. High degree roots are much sparser, reducing the deviation between samples.

$$f_1(l, d_r) = D \cdot \ln(d_r)^2 + F \cdot \ln(d_r) + G \tag{2.9}$$

$$f_2(l, d_r) = \frac{K}{{d_r}^H} + J \tag{2.10}$$

$$D = 0.002 \cdot s - 0.14 - (0.015 \cdot s - 0.285)(0.56 + 0.033 \cdot f_e)(l - 1) \tag{2.11}$$

$$F = 0.97 - (l - 1)(4.438 - 0.168 \cdot s)(0.83 + 0.01 \cdot f_e) \tag{2.12}$$

$$G = ((2.1 + l)(5.35 + 0.093 \cdot s) - 13.625)\left(1.23 - \frac{2.9 + 0.69 \cdot l}{f_e}\right) \tag{2.13}$$

$$H = 2^l \cdot (0.011 + 0.00012 \cdot f_e) \cdot e^{1.7 - \frac{s}{10}} \tag{2.14}$$

$$J = 1 - e^{(1 - 0.012 \cdot f_e) \cdot (9.55 - 1.3 \cdot l - s \cdot (0.6 - 0.1 \cdot l))} \tag{2.15}$$

$$K = 13 \cdot (0.062 \cdot s - 0.046) \cdot (2 - 0.24 \cdot l) - J \tag{2.16}$$

Table 2.2: List of abbreviations employed in the model implementation.

Abbr.	Parameter	Description
p_{inj}	Injection probability	Probability of injecting a phit in a given cycle.
p_s	Packet size	Size of the packet, in phits. A phit is the number of bytes sent per simulation cycle.
t_q	Query time	Number of cycles of computation associated with processing a query.
t_c	Cycle time	Length of a simulation cycle, in seconds.

$$\ln(\sigma_{E_l}) \approx \begin{cases} f_1(l,d_r) & l=1, \\ \max(f_1(1,d_r), f_1(2,d_r)) & l=2, \\ \max(f_1(2,d_r), f_2(4,d_r)) & l=3, \\ f_2(l,d_r) & l>3. \end{cases} \tag{2.17}$$

The vertex degree of the tree root depends on the scale of the graph and is approximated via a lognormal distribution characterized by its mean and standard deviation in Equations 2.18-2.19, per Kim and Leskovec [93].

$$\overline{d} = \ln((0.3604)^s) + 1.0661704 \cdot s \tag{2.18}$$

$$\sigma_d = 0.079313065 \cdot s \tag{2.19}$$

2.2.2 Implementation

The proposed Graph500 synthetic traffic model has been implemented in the FOGSim network simulator. This implementation is publicly available and can be adapted to other network simulators in order to evaluate the impact of the Graph500 traffic workload on different networks. The model implementation receives as input parameters the scale of the graph s, the edgefactor f_e and the coalescing size c_s to set the amount of point-to-point messages that will be transmitted during the execution. It also requires the computation time associated to an incoming query (or *query computation time*) t_q to compute the injection rate of the point-to-point messages. Table 2.2 presents a list of abbreviations used in the implementation of the model.

As opposed to the regular fixed-length executions of FOGSim described in Section 1.6.3, the Graph500 synthetic traffic model finishes only when one BFS is completed. The most significant performance metric in this case is therefore the execution time, which omits graph generation as it is not evaluated in the performance figures of the benchmark.

The time invested in the computation of a query must be input in numbers of cycles. The injection probability of the point-to-point messages is expressed as a percentage of the maximum traffic load that can be delivered by the network. For a given node

Table 2.3: List of query computation time for different node architectures.

Node	t_q
Altamira supercomputer - IBM iDataplex dx360m4, Intel Xeon E5-2670 @2.6GHz, 64GB RAM @1.6GHz	1.5ns
Intel Core i5-5200U @2.2GHz, 8 GB RAM @1.6GHz	2.25ns
Intel Xeon E5-2620 @2GHz	2.4ns
Mont-Blanc prototype [124] - Samsung Exynos 5, ARM Cortex A15 @ 1.7GHz	15ns

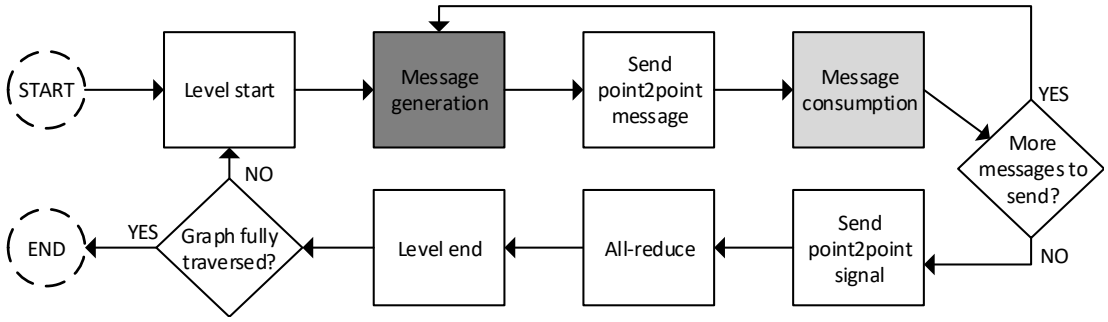


Figure 2.11: Flowchart describing the behavior of the Graph500 simulation model.

architecture, Equation 2.20 determines the corresponding injection probability at the simulator as a function of the query execution time and the length of a simulation cycle.

$$p_{inj} = \frac{p_s \cdot t_c}{t_q \cdot c_s} \tag{2.20}$$

Query computation time is highly related to the performance of the simulated machine. In order to ease model usability, Table 2.3 provides a reference of the per-query computation times measured under different node architectures. Provided values correspond to the characterization of the computation time in a conventional HPC cluster, a regular desktop CPU and a novel ARM-based cluster prototype. Should a user demand more precise values, an instrumentalized version of the Graph500 benchmark can be run to quantify t_q.

The variation between t_q in different architectures is not very significant in absolute terms; query processing is not compute-intensive and is mainly bounded by memory accesses. The best performer in the list is Altamira, an HPC cluster located at the University of Cantabria. However, a laptop chip such as the Core i5 performs only a 30% slower. The highest computation time corresponds to the Mont-Blanc prototype [124], an ARM low-power cluster. This prototype has been built as part of the Mont-Blanc project, an EU-funded program that targets energy-efficient computation through large ARM-based systems. A high execution time is consequently expected due to the low-power nature of the processors, and acts as a lower bound for the performance across the list.

Figure 2.11 portrays a flowchart with the states followed by the Graph500 synthetic traffic generator. Simulation starts by determining the number of point-to-point com-

munications that will be delivered within the current stage or tree level. Each node of the network corresponds to a process in the benchmark; for the first stage, one node is selected as host of the root vertex and begins the generation and dispatch of the point-to-point messages, being the lone sender for this level.

The amount of messages responds to the Gaussian distribution defined by the equations in Section 2.2.1, and depends on the number of edges explored per tree level E_l. In the first phase or tree level the Gaussian distribution is not used, as E_l trivially equals the degree of the root vertex and is defined by a lognormal distribution described in the same section.

Whenever a node generates a point-to-point message, it attempts to insert it in the injection buffer of the directly-linked router. Following the *simple* implementation of the benchmark, the node does not generate new messages until all received messages have been consumed. New message generation is halted while a incoming message is being consumed or an outgoing message cannot be injected into the network (due to lack of space, or buffer in use).

After a node has delivered all the messages for the current stage, it dispatches a signal to all other nodes in the network. When all nodes have finished sending and receiving messages, they enter an all-reduce phase comprising a reduce (all nodes send to one) and a broadcast (one node sends to the rest). The host node for the root vertex acts as receiver for the messages in the reduce and as transmitter during the broadcast.

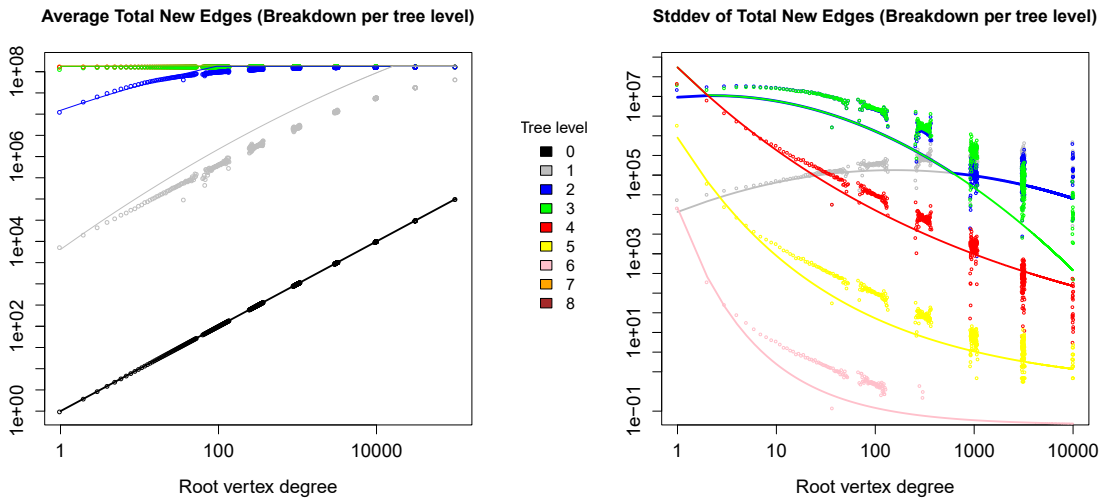
Following the end of the all-reduce, the simulator determines if the graph has been wholly traversed. Fully traversal of the graph implies that all the queries have been sent, upper bounded by twice the number of edges in the graph. If the condition is not met, nodes re-enter the level start phase and proceed with another stage; otherwise, the execution ends.

2.3 Validation

The validity of the Graph500 synthetic traffic model is confirmed in a two-phase approach. First, the prediction from the equations in Section 2.2.1 is crosschecked against empirical values for different graph sizes from those employed for their obtention. Then, an analysis of the results from FOGSim with the synthetic traffic model is performed with different input configurations.

2.3.1 Validation of the model equations

This validation concentrates on the accuracy of the prediction for the mean $\overline{E_l}$ and standard deviation σ_{E_l} that characterize the distribution of the number of explored edges per tree level upon the root degree, since the impact of message aggregation is clearly defined by Equation 2.4. Both predictions for mean and standard deviation are compared against measured values from a set of BFS executions with parameters different than those employed to obtain the equations.



(a) Mean. (b) Standard Deviation.

Figure 2.12: Validation of the model. Points correspond to measured average and standard deviation values from a real execution, averaging multiple graphs with scale $s = 22$ and edgefactor $f_e = 16$. Lines correspond to the fittings from the model.

Figure 2.12a displays the average number of explored edges $\overline{E_l}$ for all possible root vertices in a set of graphs with *scale* $s = 22$ and *edgefactor* $f_e = 16$. Points correspond to the measured values, whereas lines are the fittings obtained through the model. The fitting curves approximate clearly the observed behavior, following the same trends as the measurements for every stage of the execution. The model reproduces the staged behavior and replicates the dependence on the root degree, observing a similar proportion between the impact of each stage in the total amount of explored edges. Model results from the second level $l = 1$ and upwards are truncated for large root degrees when the maximum number of edges in the graph are explored.

The dynamic range of the observed values is very large due to its logarithmic nature; this implies that any deviation in the prediction will incur in a very large absolute error. Still, the relative error of the model for this second tree level is lower than 18% in more than 90% of the cases. For the third tree level, which amounts the largest amount of communications for most root degrees, the model is still able to reproduce the same behavior with an average relative error of 12.5%. Total number of explored edges across the graph presents a relative error below 12%, which is corrected when the maximum value is reached and the edges in the last levels are truncated.

Figure 2.12b depicts in a similar fashion the measured points and fittings for the standard deviation. Deviation between the measured points and the model prediction is larger in this case: the dispersion for large root degrees is so large that any fitting must necessarily incur in substantial errors. The model curves nevertheless resemble the trend of the samples.

A similar analysis has been conducted for the mean and standard deviation upon graphs of *scale* $s = 18$ and *edgefactor* $f_e = 40$, with analogous results.

Table 2.4: Simulation parameters for the Graph500 synthetic traffic model evaluation.

Parameter	Value
Router size	7 ports (h=2 global, p=2 injection, 3 local)
Group size	4 routers, 8 computing nodes
System size	9 groups, 72 computing nodes
Latency	50/500 cycles (local/global links), 200 cycles (router)
Buffer size	16kB (local input per VC, output), 128kB (injection and global input per VC)
Packet size	4kB

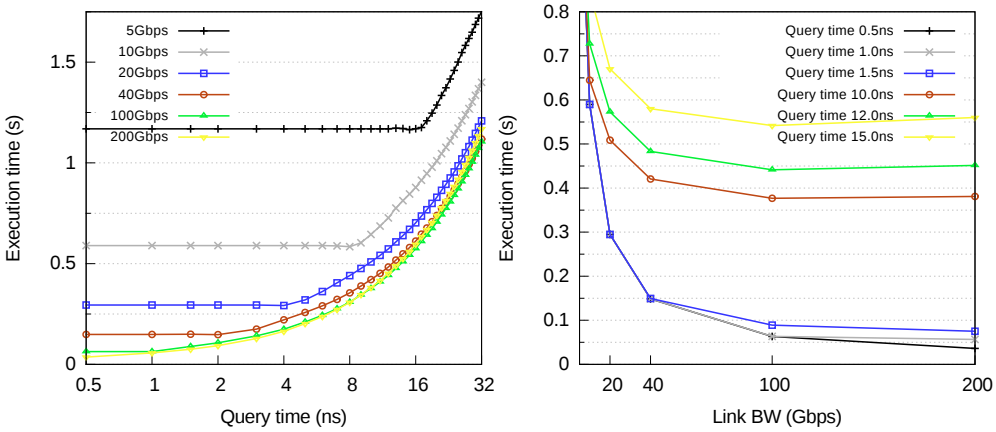
2.3.2 Simulation results

The accuracy of the model as a whole and its implementation is evaluated through a battery of simulations with FOGSim. The configuration is similar to that described in Section 1.6.4 but with a smaller network size to reduce execution time and considering the specifications of the 40Gbps QDR/FDR10 InfiniBand (IB) switch in the Altamira supercomputer. Table 2.4 describes the values of the redefined parameters. All results represent the average of 5 different executions with different root degree.

Each execution of the simulator requires less than 100MB of memory in a single process; as a reference point, the capture of the behavior for a scale $s = 23$ (8 times smaller than the scale $s = 26$ employed for these results) and 64 processes involves a trace of 600MB. Furthermore, trace size scales almost linearly with the size of the graph and the number of processes, making unfeasible to evaluate larger cases. By contrast, the synthetic traffic model only requires a longer execution to perform the analysis of greater graph sizes.

Figure 2.13 illustrates the impact of the node computation capabilities (which determine the processing time associated to every query) and the link bandwidth (BW) on the execution time of one BFS execution, in a graph of scale $s = 26$ and edgefactor $f_e = 16$, with a coalescing size of $c_s = 256$. The simulation considers an execution with 72 processes sequentially distributed across a Dragonfly network of size $h = 2$, and minimal (MIN) routing.

Figure 2.13a depicts the execution time in seconds for a sweep in the query computation time (t_q), for different link bandwidths. The curves present two distinctively different behaviors. For long query times, the execution time grows with the query computation time. This zone corresponds to CPU-bounded execution, where the node computational capabilities act as the bottleneck for the performance of the benchmark. However, for short query computation times the execution time stales, and the network becomes a bottleneck for the system performance. The frontier between these zones represents a balanced system. Interestingly, for a system with the characteristics of the Altamira supercomputer (query time $t_q = 1.5ns$ and 40Gbps of link bandwidth) the execution time is already hindered by the network limitations. Network impact is restricted to bandwidth (and not latency) characteristics; simulations employing links



(a) Node capability sweep. (b) BW sweep.

Figure 2.13: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Dragonfly network of size $h = 2$ employing MIN routing.

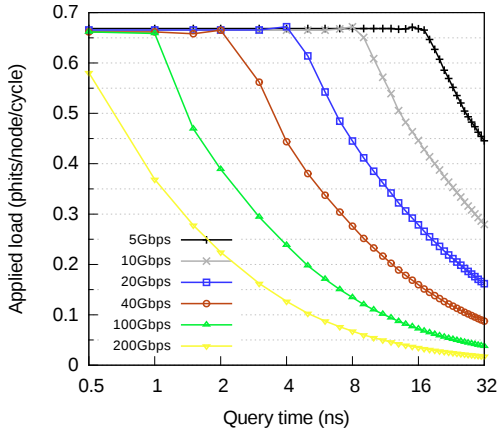
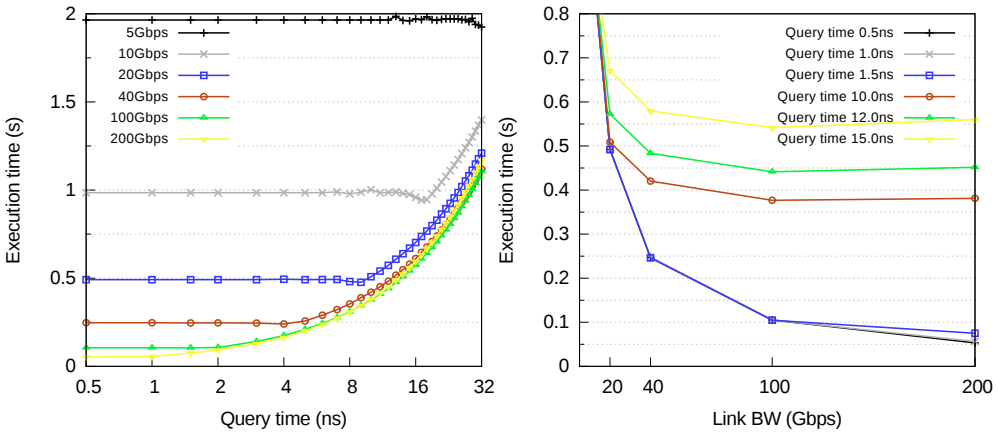


Figure 2.14: Network use per node computation capability, under different link bandwidths, for a graph of scale 26 and edgefactor 16. Dragonfly network of size $h = 2$ with MIN routing.

with a 10x increase in delay (omitted for the sake of brevity) rendered negligible differences from the displayed curves. This behavior is related to the nature of the communications of the benchmark, which do not present any interdependencies between messages within the same tree level and therefore favour higher message dispatch rates over lower latencies.

Figure 2.13b represents the same conduct from the opposite perspective: it renders the execution time in seconds upon the link bandwidth, for different query computation times. For a node with a query computation time $t_q = 15ns$ (similar to the ARM CPU of the Mont-Blanc prototype) an increase in link BW beyond 40Gbps is close to ineffective, but for a query time below 1.5ns an improvement from 40Gbps to 100Gbps halves



(a) Node capability sweep. (b) BW sweep.

Figure 2.15: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Dragonfly network of size $h = 2$ employing VAL routing.

the execution time. Performance in the zone limited by the network is not completely proportional to the link BW, because the model does not only consider the point-to-point communications but also the synchronization phases (through the end-of-level signals and the all-reduce operation).

Figure 2.14 portrays the average network usage for the same curves in Figure 2.13. For all link bandwidths the curves saturate below 0.7 phits/(node·cycle) (70% of the maximum) due to performance limitations of the Dragonfly network and the half-duplex nature of the program, where nodes cannot generate new messages when they are consuming an incoming packet (however, they may take longer to inject than to consume due to performance limitations of the router). Out of the saturation region, the query time determines the message injection rate; for a given query computation time, a higher bandwidth gives a lower execution time and less utilization of the network bandwidth. Although the most important target from a performance perspective is a lower execution time of the application, working under the bandwidth limits of the network implies an under-use of resources or that the application experiments frequent peaks of high-utilization interleaved with low network usage.

Figures 2.15 and 2.16 repeat the same performance and network usage metrics for the same network and graph parameters combinations, but employing oblivious nonminimal routing (VAL). Performance in this case is significatively worse, with execution times almost doubling those for minimal routing. This occurs because VAL routing caps sustained network performance at 0.5 phits/(node·cycle), 50% of the maximum network performance, as presented in Section 1.4.1.2. The network usage graph in Figure 2.16 is below that limit, at 0.4 phits/(node·cycle) because part of the execution time is dedicated to process synchronization and network is not fully exploited in such phases.

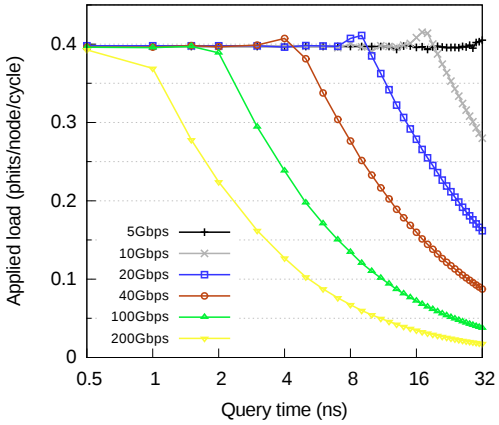
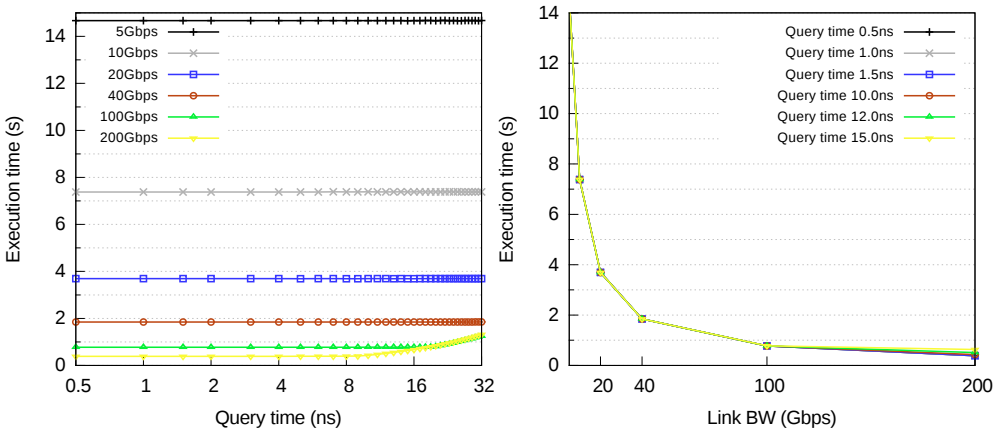


Figure 2.16: Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16, running over a Dragonfly network of size $h = 2$ with VAL routing.

Previous results have considered fully exploiting a small system of 72 computing nodes. However, in HPC systems shared among several users applications are usually assigned a subset of the system nodes, and the simplest approach for the job scheduler is a consecutive allocation of groups. Figures 2.17 and 2.18 repeat the same evaluation of Figures 2.13 and 2.14 with a similar number of processes (64 processes instead of 72) but using two groups of a larger Dragonfly network of size $h = 4$. To simplify the evaluation, the rest of the network sits idle during the execution; in a typical system these nodes would run applications from other users, interfering in the network access and degrading the performance.

The use of MIN routing is detrimental for the observed performance; since the spatial distribution of the model communications is uniform, half of the communications must advance through a single global link between the two groups and are capped to the $\frac{1}{2h^2}$ explained in Section 1.6.1.2. The theoretical maximum throughput of the network is then $2\frac{1}{2h^2} = 0.0625$ phits/(node·cycle). This constitutes an adversarial case similar to the ADV+1 traffic described in Section 1.6.1.2, and increases the execution time an order of magnitude compared to the uniform layout in a Dragonfly of size $h = 2$ observed in Figure 2.13. Moreover, the impact of the query time in the duration of the execution becomes negligible, negating the benefits of a node with greater computation capabilities.

VAL routing allows to exploit link diversity and improve performance significantly. Figures 2.19 and 2.20 refer the execution time and network usage of the Graph500 traffic model running over 2 groups of a Dragonfly network of size $h = 4$ with oblivious nonminimal VAL routing. Both the execution time and the network usage are at an intermediate point between those of MIN and VAL routing with the whole $h = 2$ Dragonfly. These results are optimistic since the rest of the network is idle and the nonminimal traffic does not contend with traffic from other applications in the system, but provide a reference for comparison against MIN routing.



(a) Node capability sweep. (b) BW sweep.

Figure 2.17: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Execution with 64 processes spread in 2 groups of a Dragonfly network of size $h = 4$, employing MIN routing.

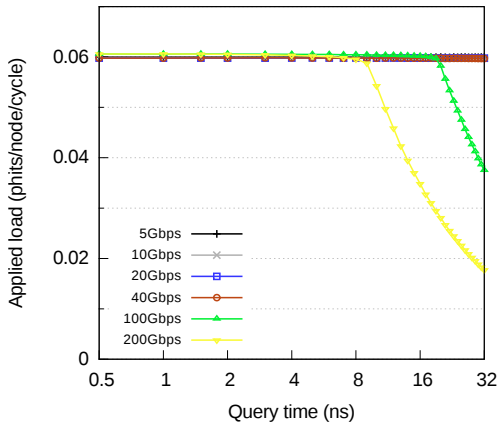
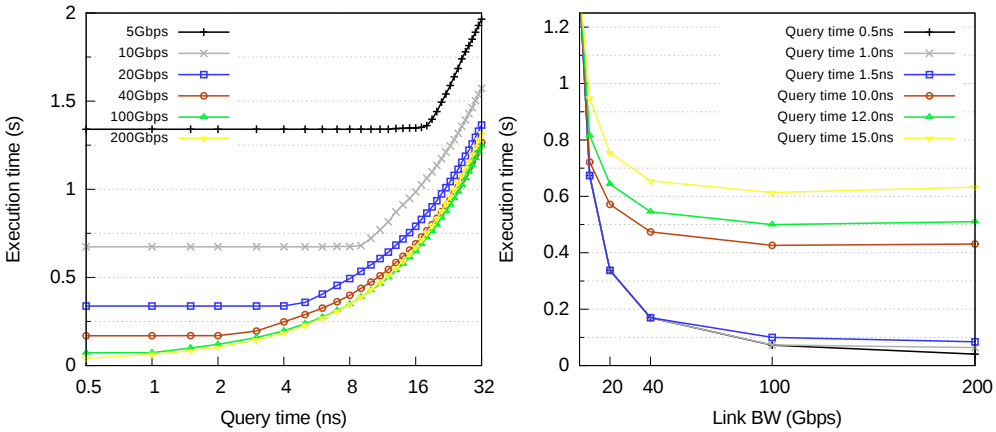


Figure 2.18: Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16, running over 64 processes spread in 2 groups of a Dragonfly network of size $h = 4$ with MIN routing.



(a) Node capability sweep. (b) BW sweep.

Figure 2.19: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Execution with 64 processes spread in 2 groups of a Dragonfly network of size $h = 4$, employing VAL routing.

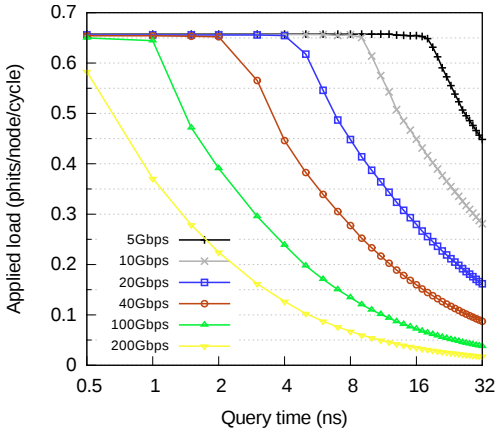
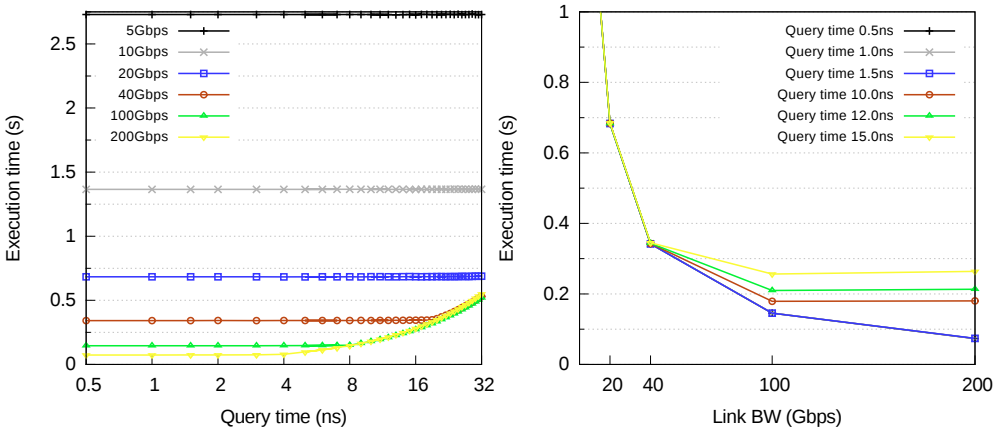


Figure 2.20: Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16, running over 64 processes spread in 2 groups of a Dragonfly network of size $h = 4$ with VAL routing.



(a) Node capability sweep. (b) BW sweep.

Figure 2.21: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Execution with 160 processes spread in 5 groups of a Dragonfly network of size $h = 4$, employing MIN routing.

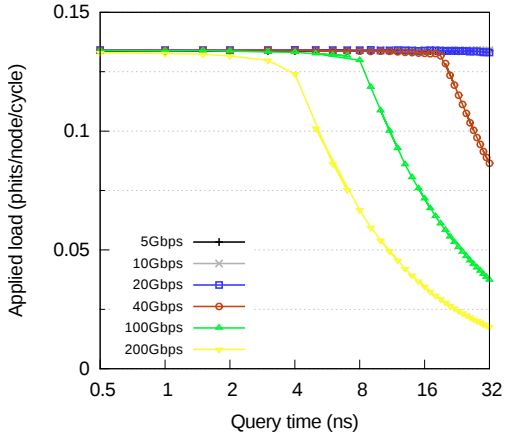
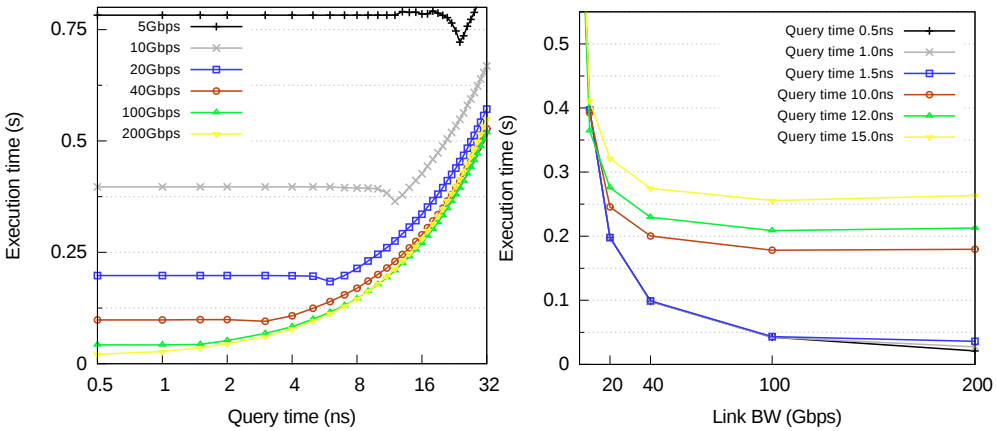


Figure 2.22: Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16, running over 160 processes spread in 5 groups of a Dragonfly network of size $h = 4$ with MIN routing.



(a) Node capability sweep. (b) BW sweep.

Figure 2.23: Execution time for a sweep in node computation capability and link bandwidth (BW), with a graph of scale 26 and edgefactor 16. Execution with 160 processes spread in 5 groups of a Dragonfly network of size $h = 4$, employing VAL routing.

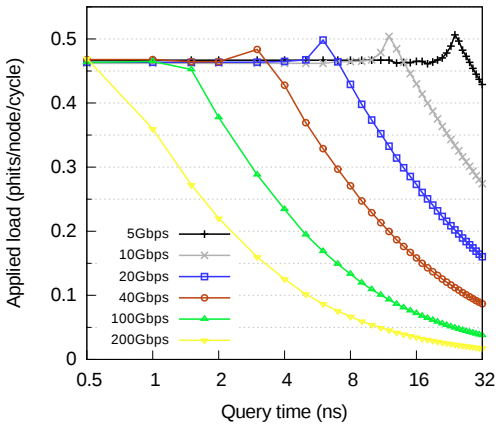


Figure 2.24: Network usage per node computation capability, under different link bandwidths. Results for a graph of scale 26 and edgefactor 16, running over 160 processes spread in 5 groups of a Dragonfly network of size $h = 4$ with VAL routing.

Figures 2.21- 2.24 expand the synthetic model to 5 consecutive groups, increasing the number of processes up to 160. Using only MIN routing, it can be observed that 80% of the traffic in every group is concentrated in 4 global links. In the case of the first group, those global links are attached to the same router; this traffic introduces a problem of unfairness that is analyzed in Chapter 3. Execution times are now comparable to the figures from the first case, where the number of processes was only 72 (less than a half of the nodes used here). With VAL routing, results are better than their counterpart in the $h = 2$-sized network because more than 80% of the nodes are idle; in a real system the network usage would be likely upper bounded around 0.5 phits/(node·cycle) and the execution would last approximately 0.25 seconds for a 40Gb network BW and a query time below 4ns.

2.4 Conclusions

Previous evaluations of BigData workloads consist mostly of full-system simulations of the real applications, or rely on the use of traces. Both options limit severely the size and detail of the network that can be investigated via simulation - which confers observability otherwise unattainable. This is a strong limitation for network architects, since data-intensive applications gather a strong interest and place a particular stress in the network subsystem of large HPC supercomputers.

This chapter introduces a novel computationally non-intensive synthetic model of the communications of Graph500. This traffic model emulates the communications workload of the most scalable implementation of the benchmark. In order to develop this model, an analysis of the communications is performed; messages are distributed in stages, with the volume per stage being related to the number of explored edges per tree level. Furthermore, it has been identified a strong connection to the degree of the vertex selected as root of the tree.

This synthetic traffic models the benchmark behavior as a set of stages of point-to-point messages separated by all-reduce collective operations. The number of messages is defined as a function of the benchmark parameters (*scale* and *edgefactor*) for each stage within the BFS computation (which equals the level of the outcoming tree). This function is characterized as a normal distribution which calculates the number of edges per tree level for any given degree of the root vertex. Both metrics are defined through a set of equations inferred from an empirical characterization of actual benchmark executions for different graph parameters. The degree of the root vertex used in the model equations is randomly decided following a lognormal distribution, which constitutes a simple yet accurate approximation of the degree distribution in Kronecker matrix-product graph generators.

This work also provides an implementation of the model for the FOGSim network simulator, which can be handily adapted for other network simulators. This implementation features high scalability with the size of the graph and the number of nodes employed. Node computation and memory characteristics are assessed in the model through a query computation time parameter; its value can be measured at a target

architecture, and short list of the associated computation time for different system architectures is provided as a reference value.

Following the network parameters in the simulator that correspond to a reference InfiniBand router architecture, a set of simulations is performed and a figure is provided with the model execution time for different node capabilities and link bandwidths. Results demonstrate that the execution time of the model presents two distinct behaviors: a CPU-bounded region where execution time grows with query computation time, and a flat region when the node computation capability exceeds the maximum injection rate of the network. Maximum performance when the network becomes the bottleneck does not only account for the point-to-point communications but also the synchronization phases.

Results also account for the task mapping over the simulated system, considering the use of a whole dedicated system and two comparable subsets of a larger network. Interestingly, the near-uniform nature of the benchmark communications mimics the behavior of adversarial traffic in a Dragonfly network when a subset with multiple groups is employed.

Chapter 3

Throughput unfairness

The previous chapter has presented a synthetic traffic model of the communications in the Graph500 benchmark, and the results from its implementation in the FOGSim network simulator prove the existence of adverse traffic patterns for executions allocated to multiple groups of the Dragonfly network. Adverse traffic patterns require nonminimal adaptive routing to exploit nonminimal paths and increase the performance. This chapter delves into the unfairness issues on the use of the network resources due to adverse traffic patterns combined with certain global misrouting and traffic prioritization policies.

In a fair situation, all processes running in the system receive the same service level regardless of their location. The lower performance (longer execution time) observed with the Graph500 traffic model for certain task mappings corresponds to per-node throughput unfairness, where the source nodes receive a different service level from the network, being able to send different amounts of traffic. The causes of throughput unfairness are diverse, and can depend among others on the network topology (if it presents asymmetries), the traffic pattern, or network faults. The impact of unfairness depends on the structure of the application; in the worst case of typical HPC applications that follow a fork-join nature, the system can slow to the speed determined by the worst-serviced node. If the unfairness leads to a quasi-starvation situation, performance of the applications running on affected nodes will degrade severely.

Although throughput unfairness may lead to significant performance degradation in real applications, considering the whole system performance under synthetic traffic patterns can mask the effects of the unfairness and even lead to better average throughput and latency results. If the traffic load is not penalized for slowing a given node the unfairness can improve performance, as it benefits the injection rate of outperforming nodes more than it slows down that of starving nodes. Synthetic traffic notwithstanding, if several applications are running concurrently on the system, throughput unfairness will be detrimental to users who are allocated the affected nodes. Unfairness metrics are therefore required to estimate the degree of throughput unfairness and establish the effectiveness of each mechanism at mitigating it.

This chapter analyzes some of the causes of throughput unfairness in Dragonfly networks, and evaluates the impact of different implicit and explicit fairness mecha-

nisms for different traffic patterns. It also introduces the adversarial-consecutive traffic, a synthetic traffic pattern that is particularly harmful for the throughput fairness, and under which a pathological case of degraded performance can be observed with source-adaptive routing.

3.1 Throughput unfairness in Dragonflies

Dragonfly networks presents a balanced use of resources under uniform traffic, but suffer from throughput unfairness under nonuniform traffic patterns. This work reviews the impact of three aspects which have been identified to decrease fairness in Dragonfly networks: global misrouting policy, prioritization to in-transit traffic, and adversarial traffic patterns.

3.1.1 Global misrouting policy

It has been discussed in Section 1.4 that systems require adaptive routing mechanisms to achieve good performance under uniform and nonuniform traffic patterns, the latter of which do not fully exploit link diversity through the use of minimal paths. Nonminimal adaptive routing in Dragonfly networks typically implies the traversal of an intermediate group to balance the use of global links, which constitute the bottleneck in most adversarial traffic patterns. The global misrouting policy, which determines which inter-group links to employ in nonminimal routing, has been introduced in Section 1.4.3.

This work only considers the *RRG* and *CRG* policies for oblivious and source-adaptive routing. *NRG* has the longest average nonminimal path and reduces performance under uniform traffic, and it is not evaluated in this work. Under adversarial traffic, the *MM* policy attempts to reduce the impact of nonminimal traffic over those global links which are heavily congested due to minimally routed traffic. This improves performance and mitigates throughput unfairness compared to the *RRG* and *CRG* policies which stress more heavily the minimal links in routers directly linked to the destination group.

In spite of the benefits of the *MM* policy, it is yet unable to avoid unfairness under certain adversarial traffic patterns, as it is observed in Section 3.3.1. In such case, explicit fairness mechanisms are required to ensure a balanced assignment of the resources to each router.

3.1.2 In-transit traffic priority

Many systems employ prioritization of in-transit traffic, a switch arbitration policy which always selects an in-transit packet rather than one in the injection queues when both compete for an output port. A well-known example of system with this policy is the whole line of BlueGene supercomputers [7, 39]. In-transit traffic priority

favours draining the network rather than injecting more traffic, reducing network congestion and eventually leading to better performance. This policy has a downside, it significantly aggravates unfairness for nonuniform traffic patterns. This effect is later evaluated in Section 3.3.

3.1.3 Traffic pattern

The traffic pattern is a significant contributor to throughput unfairness in the network. Nonuniform traffic patterns introduce an uneven distribution of traffic on the network, generating unfairness in the areas which process more traffic.

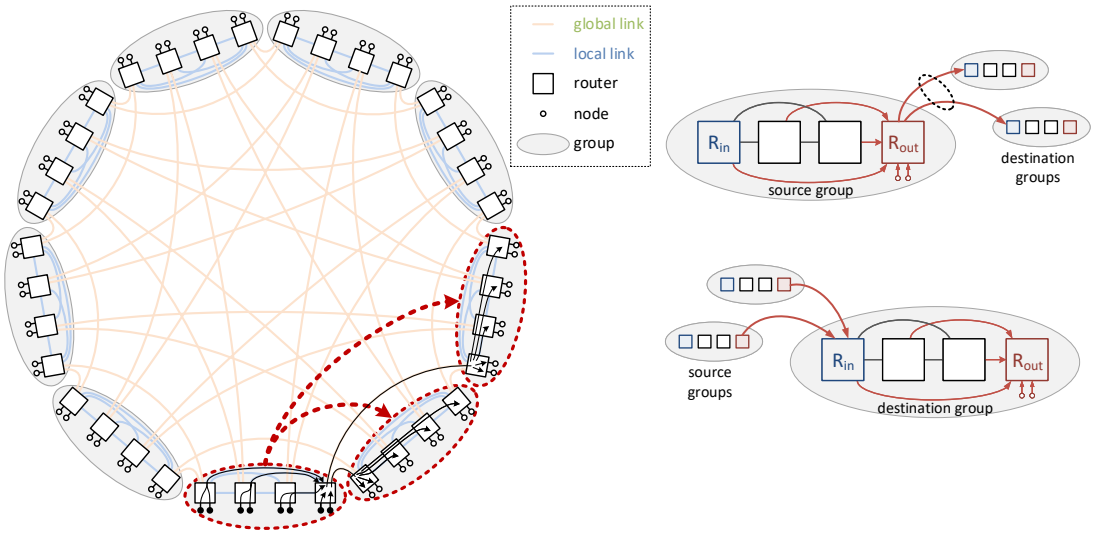
Under adversarial traffic, one single router in each group of the Dragonfly network concentrates all the ongoing traffic originated in the group if the minimal path is employed. This router, denoted R_{out} in Figure 3.1, hosts the minimal outputs for all the traffic originated in the group. Nodes connected to this router will have more difficulty to inject traffic and, therefore, receive a worse service from the network. Under adaptive nonminimal routing, the situation is not completely fair: the routing mechanism actually requires a certain level of congestion in R_{out} in order to select a nonminimal path.

Multiple variants of adversarial traffic can be defined. Section 1.6.1.2 introduced the ADV+1 and ADV+h traffic patterns, which represent the worst case scenario in terms of performance. Out of these two, this chapter employs solely the ADV+1 traffic in the evaluation. This section presents another variant named *adversarial-consecutive* which, although not as restrictive in terms of performance, generates more throughput unfairness. An *adversarial-consecutive* communications pattern can easily appear during the execution of an application when the network is partitioned in a natural, consecutive sequence.

3.1.3.1 ADVc traffic

In the *adversarial-consecutive (ADVc)* traffic pattern, messages are sent randomly to destinations in the h groups connected to the R_{out} router. With the *palmtree* arrangement of the global links discussed in Section , these destinations are the h consecutive groups $(+1, +2, ..., +h)$ after the source group. Figure 3.1 illustrates this traffic pattern with a Dragonfly network with $h = 2$, highlighting the R_{out} which hosts all the minimal global outputs and the R_{in} router hosting all the minimal global inputs. For other arrangements, the ADVc pattern is determined by selecting all the destination groups which are connected to a given router R_{out}.

As briefly observed in Section 2.3.2, this traffic pattern is similar to the workload observed when an application with a uniform spatial distribution of the communications is spread over not the whole system but a subset with multiple groups (in this case, $(h+1)$ groups), which is a typical scenario in an HPC system. If an application is allocated $(h+1)$ groups, even uniform traffic between its processes translates into ADVc traffic in the network, at least for one of the involved groups. Alternative allocation schemes which avoid consecutive group allocation can also inadvertently generate



(a) ADVc traffic.

(b) View of a source and a destination group.

Figure 3.1: Adversarial-consecutive (*ADVc*) traffic pattern in a Dragonfly with $h = 2$. Traffic from each source group i targets the next $h = 2$ consecutive groups $(i+1, i+2)$. Right picture shows a closer view of the source and destination groups. Highlighted router R_{out} connects to the minimal global link towards those destination groups.

this traffic pattern, with a different bottleneck router in one of the groups of the system, especially for large h or for different global link arrangements.

ADVc traffic is not as adversarial in terms of throughput as ADV, since the traffic going minimally from any group is distributed across h global links instead of one. However, it constitutes a challenge for throughput fairness, because the global links of the bottleneck router at each group are likely to get congested due to minimally-routed traffic from neighbor routers within the group. This is analyzed in further detail in Section 3.3.

Throughput unfairness under ADVc traffic exacerbates with the *CRG* global misrouting policy, since the allowed nonminimal global paths for nodes in the bottleneck router R_{out} coincide with the minimal global links for traffic flows from other routers, which are probably congested.

Additionally, ADVc traffic saturates all the output links in R_{out}, what interferes with the congestion notification mechanism employed in PiggyBack. As it was observed in Section 1.4.2.1.1, PB is one of the best performing source adaptive routing mechanisms proposed for Dragonfly networks.

A topology-aware system batch scheduler could take into account this scenario, and consider the layout of the network nodes for their allocation to system jobs. Such scheduler should check all pre-selected groups to validate that they are not connected via a single output router at any of them. However, since allocation is a dynamic process which occurs as new jobs arrive, a practical policy which completely avoids

this traffic would be complex and probably would reduce the throughput of the system. An alternative approach is to modify the network topology through global trunking, which employs two or more global links between pairs of groups in smaller networks to provide full bisection bandwidth. Networks with global trunking can employ disjoint pairs of routers for each parallel link between groups, avoiding the concentration of traffic in a single output router. Such design would divide the output load between two or more output routers; to minimize the concentration of traffic, parallel global links to different destination groups should connect to different routers. As far as the author is aware, avoiding the collision of sets of parallel links in the same routers has not been considered before for the design of a Dragonfly topology.

3.2 Fairness mechanisms

This chapter evaluates different mechanisms to mitigate or remove throughput unfairness under adversarial traffic patterns in a Dragonfly network. Fairness mechanisms can be either explicit (such as Age Arbitration [46] or SAT [77]) or implicitly coupled with another network mechanism like the congestion control (as done in TCP [13]), the routing mechanism (such as the use of the *MM* global misrouting policy under in-transit adaptive routing evaluated in Section 3.1.3) or the the arbiter policy (such as the removal of in-transit traffic priority, also evaluated in Section 3.1.3). The fairness mechanism can act locally in the router (as is the case for Age Arbitration) or through injection throttling in the nodes (e.g., in SAT and TCP).

The implicit fairness mechanisms evaluated in this work are insufficient to prevent throughput unfairness under ADVc traffic, as will be demonstrated with the results in Section 3.3.1. To deal with the challenges imposed by ADVc it is required an explicit fairness mechanism. Section 3.3.4 evaluates the effectiveness of age arbitration which is, as far as the author knows, the only fairness mechanism that has been implemented in commercial HPC systems [5]. This work does not consider the use of any global congestion mechanisms (e.g., the injection throttling employed by TCP).

3.2.1 Age Arbitration

Age-based arbitration [5] is a variant of the arbitration policy which takes packet age into account. Each packet has an associated age, defined as the elapsed time since it was generated. When two packets contend for the same output port, the arbiter compares their age and always selects the oldest. This policy favors latency fairness by equalizing the delay of competing flows, and also provides throughput fairness.

The complexity of this mechanism relies on tracking the age of the network packets. A perfect globally synchronized network clock is not feasible, so actual implementations rely on increasing the packet age by the amount of time travelling through network links and waiting in buffers. The age of a packet is stored in a field of the packet header. Different implementations of age-based arbitration mechanisms are discussed in Section 6.3.

3.3 Results

This section presents a series of performance and throughput unfairness metrics for different arbitration policies. Unless otherwise noticed, the network configuration corresponds to the description in Section 1.6.4. Four different routing mechanisms have been employed: MIN and VAL represent the reference oblivious routing under UN and adversarial traffic patterns, respectively. Source-adaptive routing (Src) selects between minimal and nonminimal paths at injection and broadcasts the congestion status of the global links within each group following the implementation of PB described in Section 1.4.2.1. In-transit adaptive routing adheres to the implementation of OLM detailed in Section 1.4.2.2; the terms source-adaptive and in-transit adaptive are employed instead of PB and OLM because their original definition only considers one concrete global misrouting policy, *RRG* for PB and *MM* for OLM. This section considers the combination with the *RRG*, *CRG* and, only in the case of in-transit adaptive routing, the *MM* global misrouting policies.

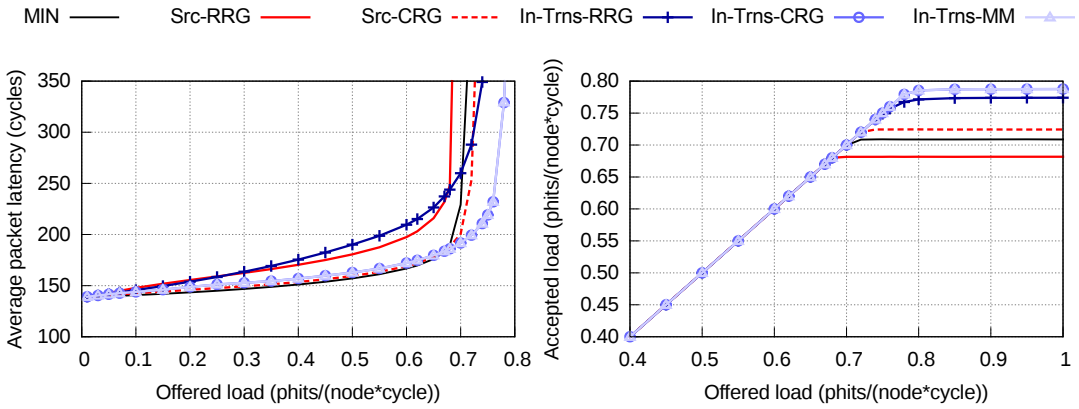
Sections 3.3.1 and 3.3.3 analyze the behavior using the Round-Robin (RR) arbitration described in Section 1.1, first giving priority to transit over injection (Section 3.3.1), and later without prioritizing any traffic flow (Section 3.3.3). Removing transit priority over new injections mitigates significantly the throughput unfairness (specially for in-transit adaptive routing) at the expense of a drop in performance; however, it is insufficient in the case of ADVc traffic. Section 3.3.4 studies the capability of Age-based arbitration to avoid throughput unfairness under all traffic scenarios.

3.3.1 Results with Round-Robin arbitration and in-transit priority

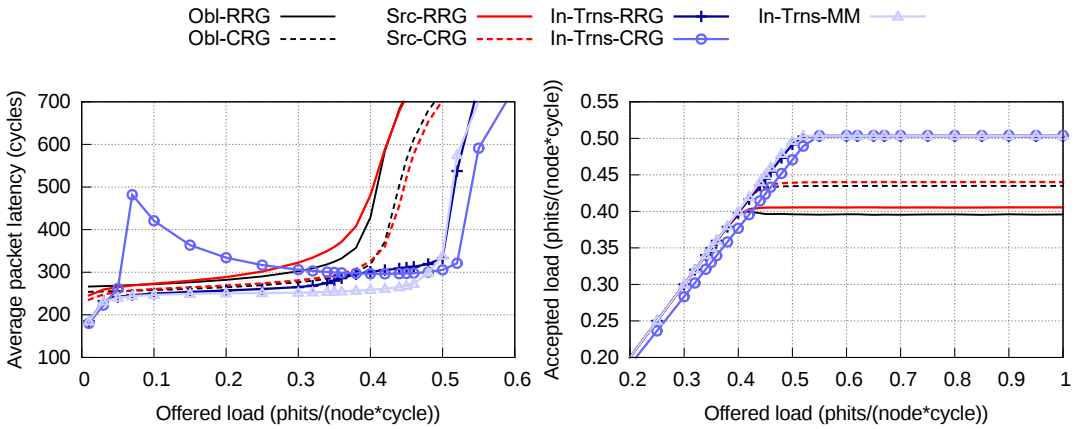
In order to distinguish the impact of the different routing mechanisms, the evaluation of the results with in-transit priority is split in two sections; first, the performance metrics (throughput and latency) are analyzed, and then the throughput fairness is scrutinized.

3.3.1.1 Network performance with RR arbitration and in-transit priority

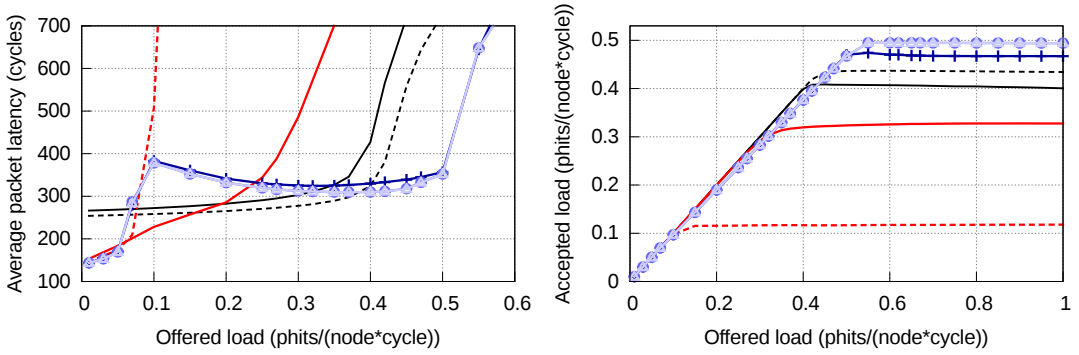
Figure 3.2 shows average throughput and latency under UN and adversarial traffic employing RR arbitration with priority of transit over injection traffic. Performance under UN traffic (Figure 3.2a) is good for all the routing mechanisms. Latency with the CRG and MM global misrouting policies for source-adaptive and in-transit adaptive routing is close to the minimal marked by MIN routing. Since MIN routing does not employ nonminimal paths, the impact of the global misrouting policy with oblivious routing under UN traffic is not explored. Source adaptive routing mechanisms perform misrouting only at injection. Since the traffic pattern is already uniform, ideally the percentage of misrouted traffic should be close to 0% and a higher percentage should degrade performance. However, the traffic pattern may present temporal nonuniformities during the execution, and the higher number of VCs employed by adaptive routing mechanisms alleviates HoLB and outperforms the degradation associated to misrouting in the case of Src-CRG and all the in-transit routing mechanisms. The



(a) UN traffic.



(b) ADV+1 traffic.



(c) ADVc traffic.

Figure 3.2: Latency and throughput under uniform (UN) and adversarial (ADV+1, ADVc) traffic patterns, using Round-Robin arbitration and prioritizing transit over injection.

amount of misrouting decreases slightly at high traffic loads, because the minimally routed traffic already exploits most of the maximum utilization of the network links. The use of a RRG policy is detrimental in the case of UN traffic because it increases latency and has a negative effect in throughput.

The impact of the global misrouting policy is more significant under adversarial traffic patterns ADV+1 (Figure 3.2b) and ADVc (Figure 3.2c) where oblivious Valiant routing constitutes the reference. All the adaptive routing variants enroute more than 97% of the packets nonminimally to prevent the performance bottleneck of the minimal global links. Under ADV+1 traffic, CRG achieves the highest saturation point within each routing mechanism and also provides better throughput under oblivious and source-adaptive routing; the impact on throughput for in-transit adaptive routing is hidden by the theoretical performance bound of 0.5 phits/(node·cycle) discussed in Section 1.4.1.2. The spike in latency for In-Trns-CRG is discussed in Section 3.3.1.2.

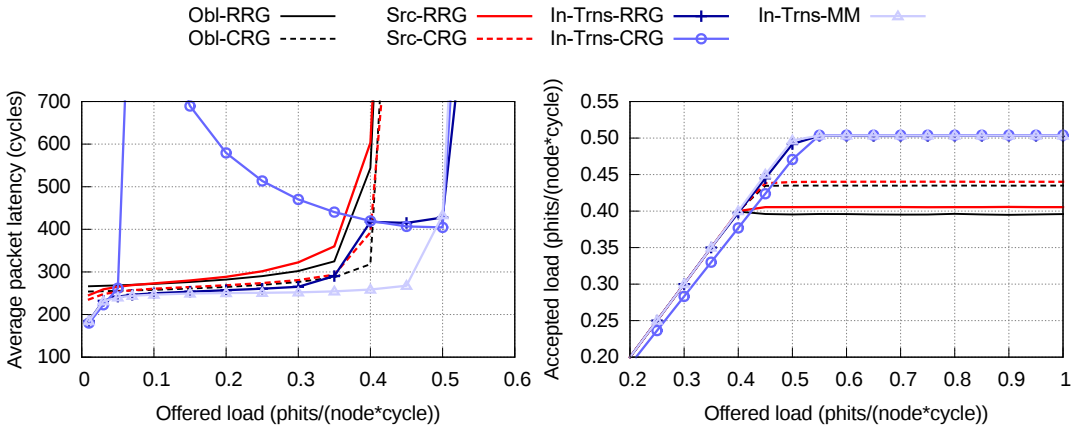
RRG reduces performance under adversarial traffic, because it employs longer paths than CRG in average due to the extra local hop in the source group. Moreover, it does not alleviate the unbalance because it exploits uniformly the global links for the nonminimal communications. This implies fewer available resources for injection at the bottleneck router of the group which, combined with the priority of transit over injection, prevent those nodes in the bottleneck router from injecting at the same rate as the rest. The highest throughput under UN and ADV traffic is achieved by in-transit adaptive routing with the MM policy, since it combines the most beneficial selection at injection (CRG) and during network traversal (NRG).

All the routing mechanisms fail to perform well in both performance metrics under ADVc traffic. Oblivious and source-adaptive routing mechanisms have lower latency below the saturation point than in-transit adaptive routing and do not display latency peaks due to throughput unfairness, but achieve lower throughput. The implementation of source-adaptive routing is unable to properly identify ADVc as an adversarial traffic pattern and degrades significantly the performance; this effect is analyzed in Section 3.3.2. In-transit adaptive routing achieves the highest throughput but clearly suffers from throughput unfairness, what can be appreciated in a lower throughput than the offered load before reaching the saturation point. This is delved into in Section 3.3.1.2.

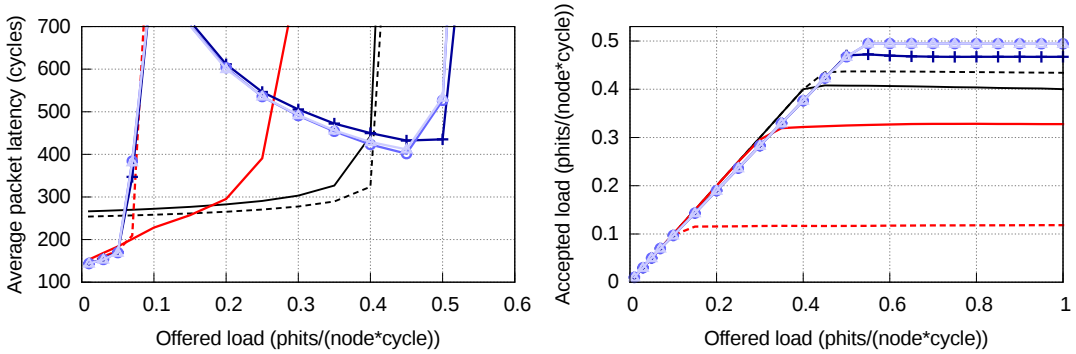
Overall, CRG is the most suitable policy for oblivious nonminimal routing under adversarial traffic. However, it performs poorly in source-adaptive routing with ADVc traffic. This effect is discussed in Section 3.3.2.

3.3.1.2 Throughput unfairness with RR arbitration and in-transit priority

The severity of latency peaks due to throughput unfairness aggravates with longer injection queues. Figure 3.3 displays the throughput and latency under adversarial traffic patterns when the length of injection buffers is increased to 1000 phits from the default value of 256 phits presented in Table 1.2. Throughput remains unchanged, but latency peaks of in-transit adaptive routing below the saturation point are more pronounced. The peak in latency occurs when the bottleneck router in each group starts to



(a) ADV+1 traffic.



(b) ADVc traffic.

Figure 3.3: Latency and throughput under adversarial traffic patterns, using Round-Robin arbitration and prioritizing transit over injection. Injection queues of 1000 phits.

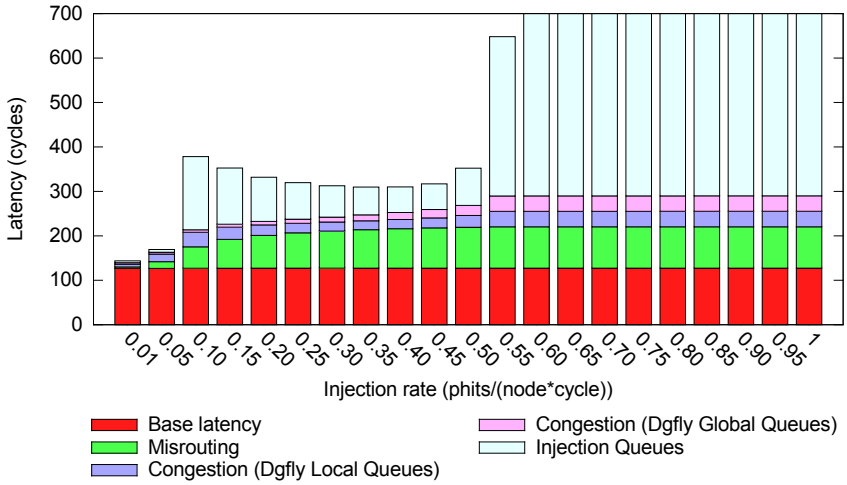
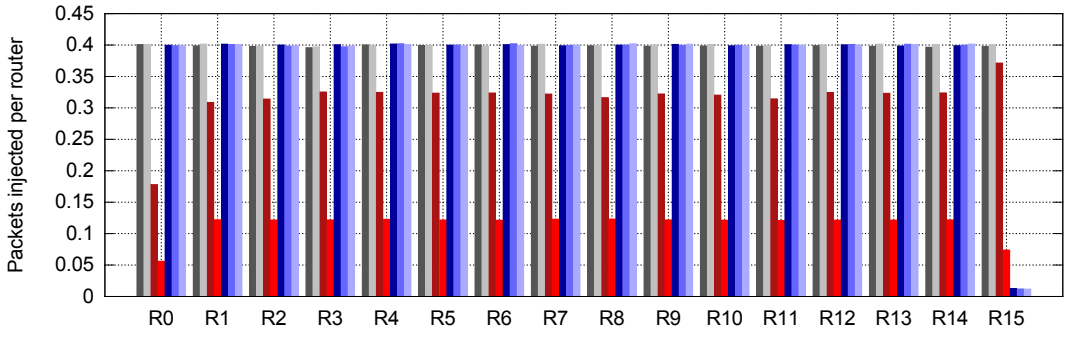


Figure 3.4: Breakdown of latency components for in-transit adaptive routing with MM global misrouting policy under ADVc traffic. Round-robin arbitration policy. Transit is prioritized over injection.

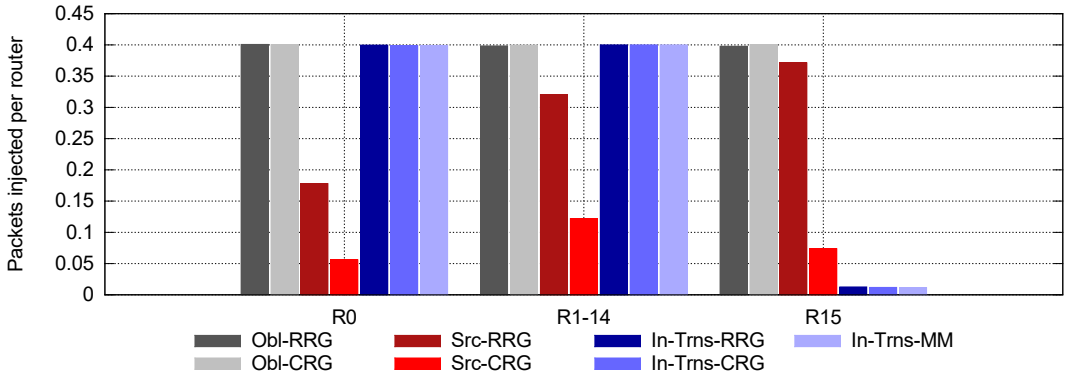
suffer starvation at low loads, because in-transit traffic received from neighbor routers is given precedence in the arbitration. The accepted load of starved routers remains low thereafter, forcing the accepted load to be lower than the offered traffic load before reaching saturation. However, the high latency of those packets coming from the starved router is hidden when averaging with the remaining routers in the group, which are not saturated and inject a higher load. When the starvation becomes severe, the number of delivered packets from the starved routers decreases significantly and the average latency gets lower, although the maximal latency is very high. With the CRG global misrouting policy, this behavior occurs at an extremely low load. The impact with RRG and MM is more subtle and at higher traffic loads, but it can be observed with longer injection queues.

Figure 3.4 displays the latency breakdown for in-transit adaptive routing with the MM policy under ADVc traffic. Five different components are considered: link traversal through the minimal and nonminimal paths, waiting time in local and global link queues, and waiting time at injection. Misrouting latency is caused by the traversal of additional nonminimal links and increases with the injection rate until it stabilizes at the saturation point, at 0.5 phits/(node·cycle). Congestion, both in local and global links, also rises until reaching the saturation point and has a relatively low impact on the total latency under all traffic loads.

The latency component which increases the most with the offered load is the waiting time at injection queues. It is also the source of the latency peak below saturation point: it grows before reaching a peak around 0.1 phits/(node·cycle) and then steadily diminishes until reaching saturation. As it has been discussed, this latency comes mostly from starved packets at the bottleneck routers. The use of deeper buffers (Figure 3.3) aggravates the effect, because a larger set of packets is waiting to be sent and



(a) Injection per router.



(b) Injection per router, grouping into 3 sets.

Figure 3.5: Injected load per router in group 0, under ADVc traffic with applied load 0.4 phits/node/cycle. Round-robin arbitration with transit-over-injection priority.

accumulating more delays.

Figure 3.5 helps to quantify the unfairness in the network, portraying the injected load for nodes connected to every router of one group under a 0.4 phits/(node·cycle) ADVc traffic load. Under this traffic pattern, R_0 and R_{15} behave respectively as the R_{in} and R_{out} routers depicted in Figure 3.1. As it can be seen in Figure 3.5a, for every combination of routing and global misrouting policy the values for middle routers 1-14 are very similar. Figure 3.5b replicates the same data as in Figure 3.5a but averaging the results for routers 1-14 for the sake of clarity.

Oblivious nonminimal routing does not suffer from throughput unfairness, and all the routers of the groups inject a similar amount of traffic regardless of the global misrouting policy. Adaptive routing mechanisms have nevertheless a completely different conduct. Source-adaptive routing tends to favor some routers in detriment of others: with the RRG policy, router R_0 injects significantly less packets than the rest of the group, whereas router R_{15} injects more than the average; with the CRG policy, those two routers inject less than the others. In-transit adaptive routing is particularly harmful for the injection at the bottleneck router R_{15}, which is several orders of magnitude lower than in the other routers of the group with any of the three global misrouting

Table 3.1: Fairness metrics for all routing and global misrouting policy combinations under ADVc traffic, with traffic in the transit queues given priority over traffic in the injection queues. Values are specified for two different traffic loads per combination, one below and one above the average saturation point.

	Avg sat. load	**Offered load**	**Min inj. load**	**Max/Min**	**COV**
MIN	0.05	0.03	0.0275 (91.7%)	1.180	0.0236
		0.55	0.0100 (1.82%)	43.81	0.4106
Obl-RRG	0.40	0.35	0.3416 (97.6%)	1.048	0.0069
		0.55	0.3522 (64.0%)	1.309	0.0345
Obl-CRG	0.43	0.40	0.3913 (97.8%)	1.045	0.0063
		0.55	0.3833 (69.7%)	1.321	0.0533
Src-RRG	0.33	0.30	0.1589 (52.9%)	1.925	0.0483
		0.55	0.1739 (31.6%)	3.138	0.1569
Src-CRG	0.12	0.10	0.0571 (57.1%)	1.829	0.0857
		0.55	0.0528 (9.59%)	8.832	0.2251
In-Trns-RRG	0.47	0.40	0.0066 (1.64%)	62.79	0.2435
		0.55	0.0039 (0.71%)	145.29	0.2820
In-Trns-CRG	0.49	0.40	0.0042 (1.06%)	97.90	0.2450
		0.55	0.0032 (0.58%)	172.89	0.2525
In-Trns-MM	0.49	0.40	0.0053 (1.32%)	79.74	0.2442
		0.55	0.0051 (0.92%)	112.56	0.2519

policies despite achieving the best throughput and latency results in almost all cases in Figure 3.2.

Table 3.1 presents the fairness results from the metrics described in Section 1.5.1 (minimum injection, max/min ratio, and coefficient of variation) under ADVc traffic. Since the level of unfairness typically increases after saturation, two values of each metric are given for every routing mechanism: one for a traffic load below the saturation point, and one above. Results with MIN routing are included as a reference. MIN achieves extremely low throughput under all adversarial traffic patterns and for ADVc it saturates at 0.05 phits/(node·cycle). However, it achieves reasonably good fairness metrics before reaching saturation, with a lower Max/Min than all adaptive routing mechanisms. It presents higher unfairness than nonminimal oblivious routing, specially for traffic loads above the saturation point, because the severity of the congestion is much higher and thus limits the amount of injection that can be achieved at the router directly connected to the destination group.

The second column presents the average injected traffic load from the router with lowest injection in the network. For fair mechanisms such as Obl-RRG it corresponds before saturation roughly to 97% of the offered load. This percentage obviously reduces after saturation, but the Max/Min ratio also typically increases, evidencing that some nodes inject more than others. All in-transit adaptive configurations are significantly less fair than oblivious and source-adaptive routing, with at least one router injecting much less traffic than the rest. The *Max/Min* metric adds further information, with all the routing mechanisms achieving the same order of magnitude before saturation for the different global misrouting policies: around 1.1 for oblivious, around 2 for

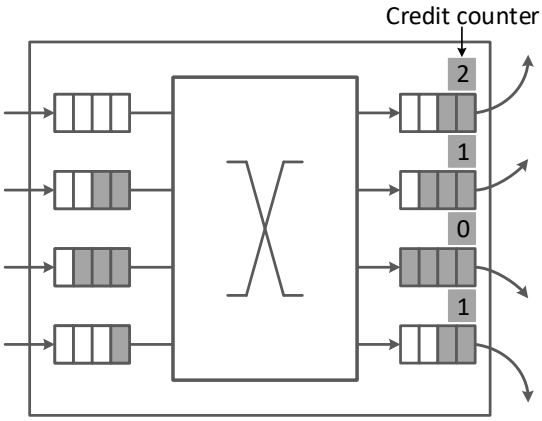


Figure 3.6: Example of router with similar congestion in all the queues. The value in the credit counter is similarly low for all the output ports, making difficult to discern if it is a general case of congestion (e.g., high UN traffic load) or pathological saturation in the router links (e.g., ADVc in the bottleneck router).

source-adaptive, and around 60-100 for in-transit adaptive. CoV is also high, implying that the unfairness is not constrained to a few isolated cases.

3.3.2 Performance issues with source-adaptive routing

Results in Section 3.3.1.1 have shown that source-adaptive routing is not competitive under ADVc traffic due to unfairness and degraded average performance. This is specially concerning as source adaptive routing is considered one of the most suitable mechanisms due to its performance and easiness of implementation [82]. The use of PB [82] for source-adaptive routing relies on the saturation status of the global link in the minimal path to choose between minimal and nonminimal routes at injection. Global links are marked as saturated when their occupancy exceeds twice the average of the global links within the same router; information about the status of other links in the group is broadcasted through Explicit-Congestion Notification (ECN) messages. Under ADVc traffic all the minimal global links at every group are connected to the same bottleneck router R_{out} and have a similar level of congestion, preventing the router from detecting the saturation properly. In particular, the R_{out} router cannot discern between a high load uniform case and a scenario in which all its output links are congested, as is the case with ADVc. Figure 3.6 shows an example of said uncertainty.

The remaining routers in the network revert consequently to employing solely the credits at their own ports, incurring in an excessively low load of misrouted traffic. When the priority of in-transit traffic over new injections is employed, Src-RRG sends minimally between 15 and 20% of the traffic, whereas the desired behavior would be to employ mostly nonminimal paths. With the CRG policy the number of available misrouting ports is much lower (h versus ah), and the amount of minimally sent traffic rises to 40%. In a Dragonfly of size $h = 8$ (16,512 computing nodes), only $1/(2h) =$

0.0625 phits/(node·cycle) can be delivered minimally.

As it is observed later in Section 3.3.4.2, unfairness under ADVc traffic with source-adaptive routing remains even employing an explicit fairness mechanism such as age-based arbitration, because the imbalance is originated due to an excessively low amount of misrouting at injection and the inability to divert in-transit packets to nonminimal paths. Moreover, since the amount of contending injection ports in R_{out} is higher than the amount of incoming traffic from any other single router, and age-based arbitration ensures that every input port receives a fair amount of local resources, injection at R_{out} is higher compared to other routers in the group. Relying on absolute saturation levels as the congestion metric would help in this case but trigger misrouting in excess under high loads of UN traffic and degrade performance. To adequately address this issue, a different congestion sensing mechanism is required.

3.3.3 Results with Round-Robin arbitration without in-transit priority

Results in Section 3.3.1 show sizable unfairness and poor performance with adaptive routing mechanisms. The results in this section evaluate the impact of removing the priority of in-transit traffic over new injections, to diminish the starvation of those nodes linked to the bottleneck routers.

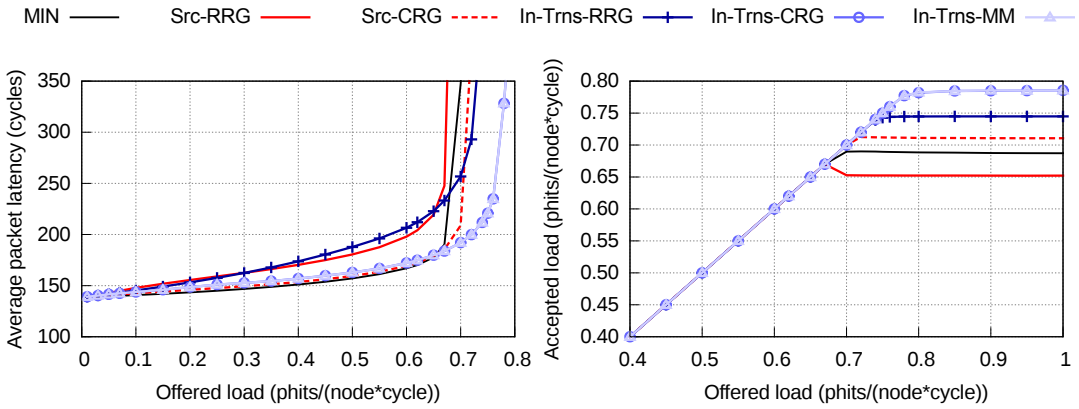
3.3.3.1 Network performance with RR arbitration, without in-transit priority

Figure 3.7 replicates the same results from Figure 3.2 but removing the priority of transit traffic respect to new injections. This increments the congestion level of the network, and can reduce throughput. However, in this evaluation the reduction is minimal: throughput with MIN routing under UN traffic decreases around a 3% and the behavior with source-adaptive routing is likewise similar as with priority. Latency peaks due to unfairness seemingly disappear; although they are heavily reduced, repeating the evaluation with 1000-phit injection buffers allows to still appreciate their presence at higher traffic loads, as evidenced in Figure 3.8.

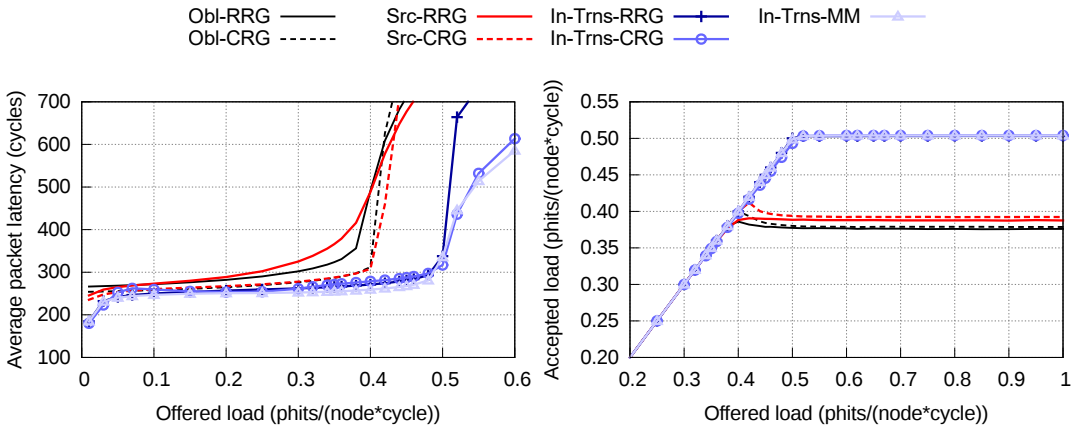
3.3.3.2 Throughput unfairness with RR arbitration, without in-transit priority

Figure 3.9 presents the injected load at the nodes of each router in a group under ADVc traffic with a load of 0.4 phits/(node·cycle), without in-transit-over-injection priority. Compared to Figure 3.5, oblivious routing mechanisms maintain the same throughput fairness between routers. Src-CRG displays a significatively higher load at the bottleneck router R_{15}, not only higher than with priority but also around $3\times$ higher than the load in other routers of the same group. This occurs because the bottleneck router is directly attached to the minimal global links and detects their availability earlier than the rest of the group, taking more advantage of the shorter minimal paths.

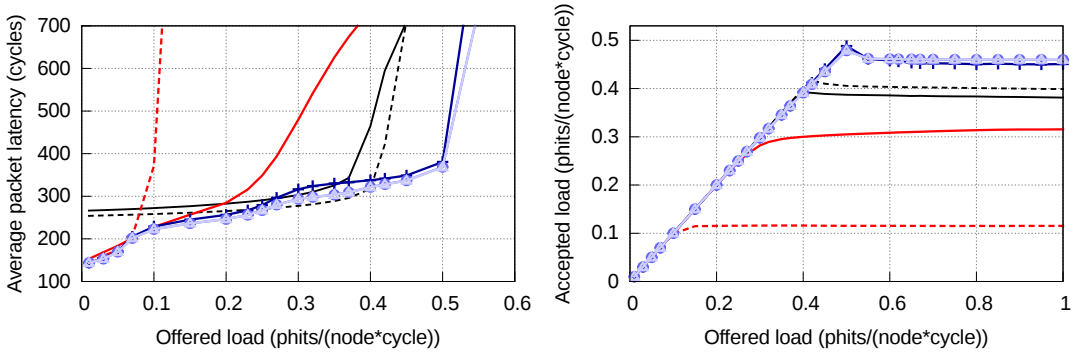
Injection at the bottleneck router with in-transit adaptive routing improves significantly with the absence of in-transit priority. The improvement is similar with any of



(a) UN traffic.

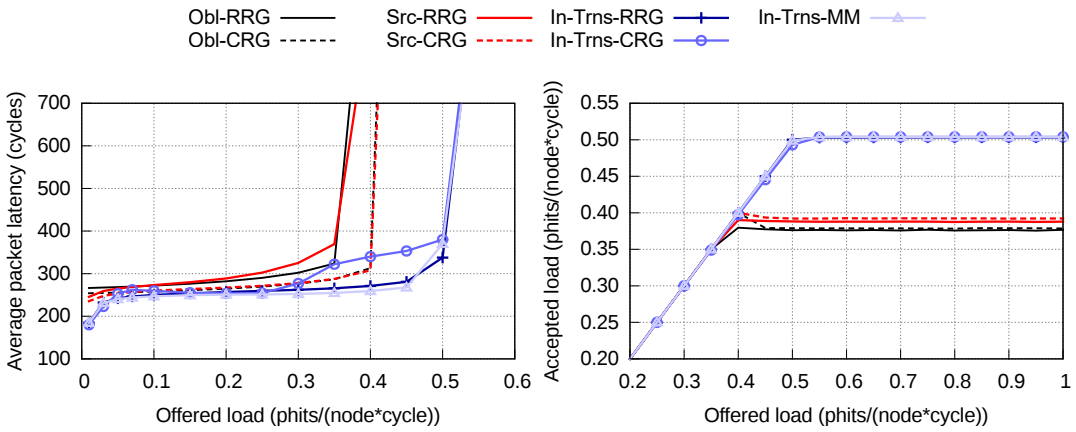


(b) ADV+1 traffic.

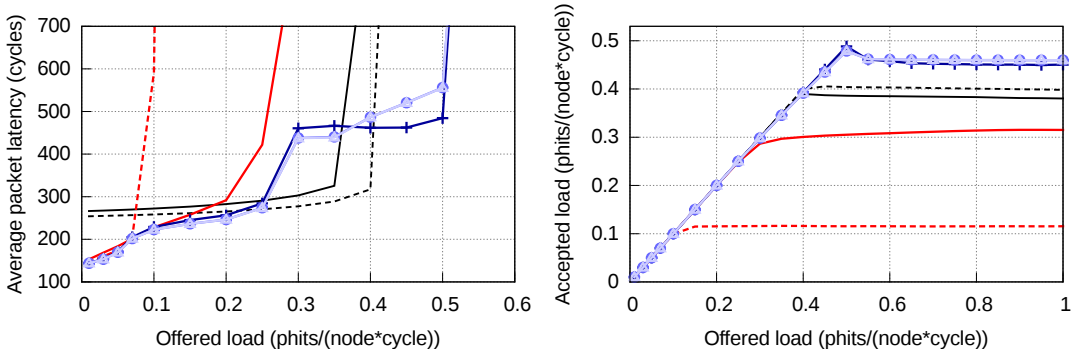


(c) ADVc traffic.

Figure 3.7: Latency and throughput under uniform (UN) and adversarial (ADV+1, ADVc) traffic patterns, using Round-Robin arbitration, without prioritizing transit over injection.

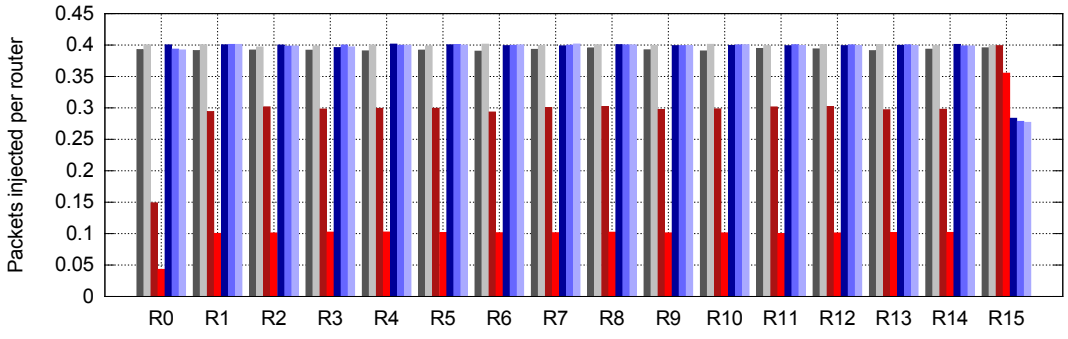


(a) ADV+1 traffic.

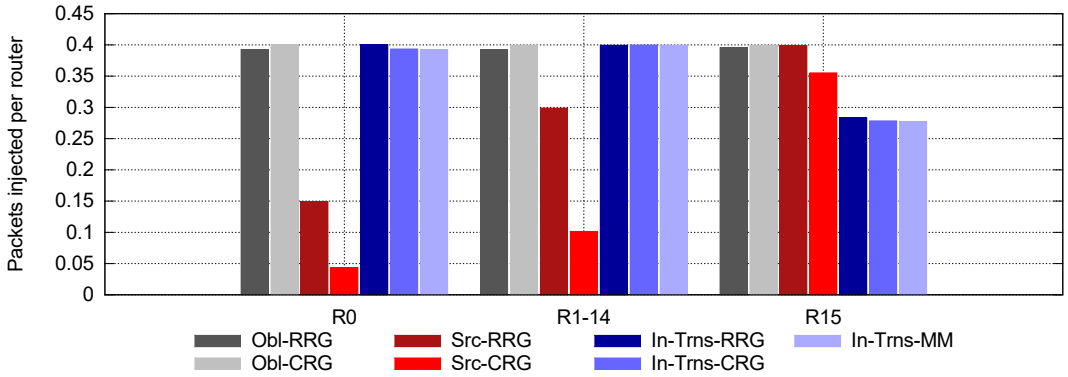


(b) ADVc traffic.

Figure 3.8: Latency and throughput under adversarial (ADV+1, ADVc) traffic patterns, using Round-Robin arbitration without priority of transit over injection. Longer injection queues of 1000phits.



(a) Injection per router.



(b) Injection per router, grouping into 3 sets.

Figure 3.9: Injected load per router in group 0, under ADVc traffic with applied load 0.4 phits/node/cycle. Round-robin arbitration without transit-over-injection priority.

the policies but, unfortunately, not sufficient to consider the use of the global links as fair, since the injection at R_{15} is approximately 30% less than at the other routers.

Values in Table 3.2 quantify the unfairness level with this configuration. Interestingly, MIN achieves the same results as with priority before reaching saturation, but for higher traffic loads the unfairness aggravates with respect to Table 3.1. This occurs because, by removing the priority of in-transit over injection, each node at the bottleneck router of each group has the same share of the global output links as all the nodes in a different router combined. Nonetheless, MIN achieves extremely low performance under adversarial traffic and is not representative.

The most significant change relative to the numbers in Table 3.1 occurs for in-transit adaptive routing. The starvation problem in R_{out} is avoided, so their Max/Min ratio is reduced to reasonable values around 1.5 before saturation. According to this metric, in-transit adaptive mechanisms present now better fairness than source-adaptive after saturation. In any case, all the adaptive routing mechanisms are unfair compared to their oblivious counterparts.

Figure 3.10 studies the evolution of these performance and fairness metrics with the network size, ranging up to more than 40,000 nodes and using routers of up to

Table 3.2: Fairness metrics for all routing and global misrouting policy combinations under ADVc traffic, without transit-over-injection priority. Values are specified for two different traffic loads per combination, one below and one above the average saturation point.

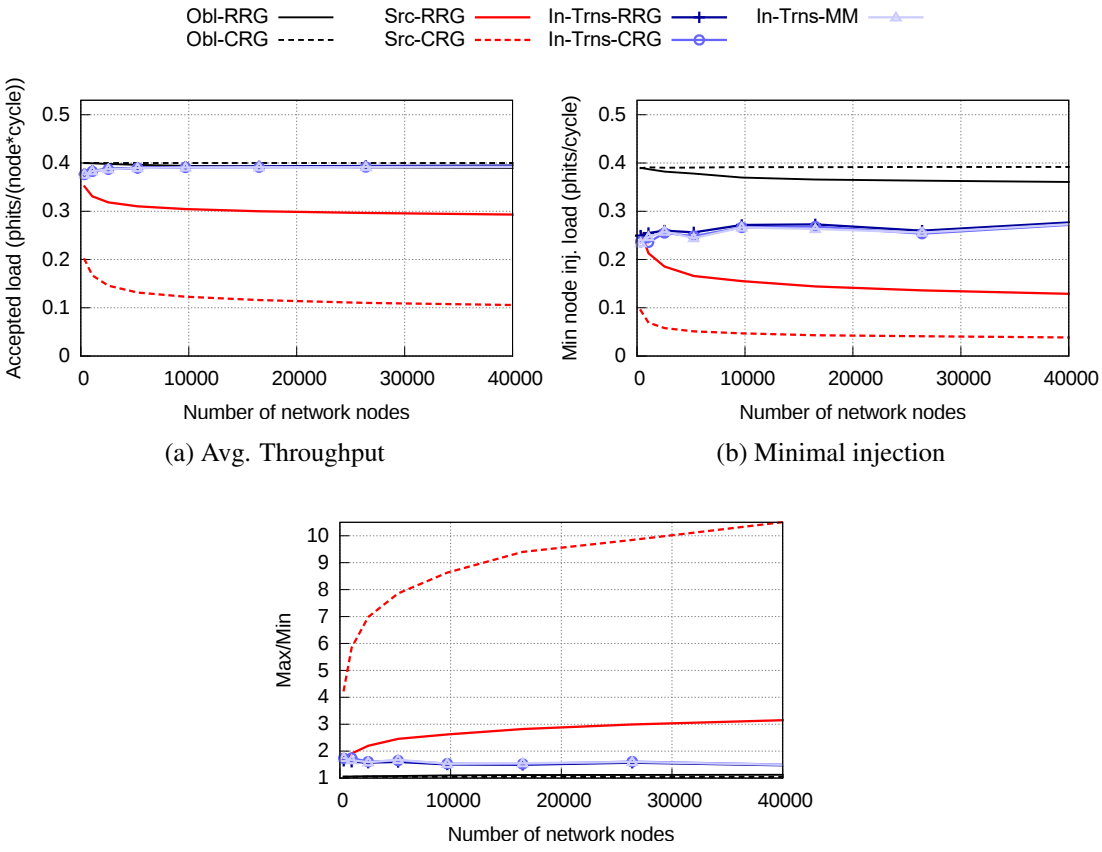
	Avg sat. load	Offered load	Min inj. load	Max/Min	COV
MIN	0.05	0.03	0.0275 (91.7%)	1.180	0.0236
		0.55	0.0056 (1.02%)	91.16	1.3382
Obl-RRG	0.38	0.35	0.3424 (97.8%)	1.047	0.0068
		0.55	0.3237 (58.9%)	1.364	0.0427
Obl-CRG	0.40	0.35	0.3421 (97.7%)	1.049	0.0068
		0.55	0.3477 (63.2%)	1.354	0.0394
Src-RRG	0.32	0.30	0.1388 (46.3%)	2.207	0.0668
		0.55	0.1409 (25.6%)	3.954	0.2216
Src-CRG	0.12	0.10	0.0560 (56.0%)	1.868	0.0339
		0.55	0.0419 (7.62%)	13.14	0.5937
In-Trns-RRG	0.45	0.40	0.2729 (68.2%)	1.496	0.0701
		0.55	0.3092 (56.2%)	1.781	0.1032
In-Trns-CRG	0.46	0.40	0.2683 (67.1%)	1.523	0.0740
		0.55	0.2814 (51.2%)	1.874	0.1080
In-Trns-MM	0.46	0.40	0.2634 (65.8%)	1.551	0.0741
		0.55	0.2829 (51.4%)	1.857	0.1078

40 ports ($h = 10$). Values for the network size employed so far appear in the middle of the graphs, at 16,512 nodes. In general, the problems of unfairness remain similar or become more critical as the network size grows. Average throughput remains similar with oblivious and in-transit adaptive routing, whereas it decreases slightly when using source adaptive routing. This confirms that source adaptive routing mechanisms become less able to accurately determine the presence of congestion under bigger network sizes, reducing the achieved performance. Likewise, the severity of the unfairness observed in Figures 3.10b and 3.10c remains unchanged with bigger network sizes for both oblivious and in-transit adaptive routing mechanisms regardless of the global misrouting policy, whereas it exacerbates with source-adaptive routing (specially combined with the CRG policy).

These results evidence that the prioritization of in-transit traffic for adaptive routing is disadvantageous in general: source adaptive routing presents relative low throughput, and in-transit adaptive routing suffers severe starvation under adversarial traffic patterns. Removing the in-transit priority does not alter significantly the performance metrics, but mitigates the throughput unfairness and its consequences (latency peaks and average throughput below offered load before saturation).

3.3.4 Results with Age arbitration

Results in Section 3.3.3 have demonstrated the inability of implicit fairness mechanisms (such as removing the in-transit-over-injection priority, or the use of the MM global misrouting policy with in-transit adaptive routing) to avoid unfairness under



(a) Avg. Throughput

(b) Minimal injection

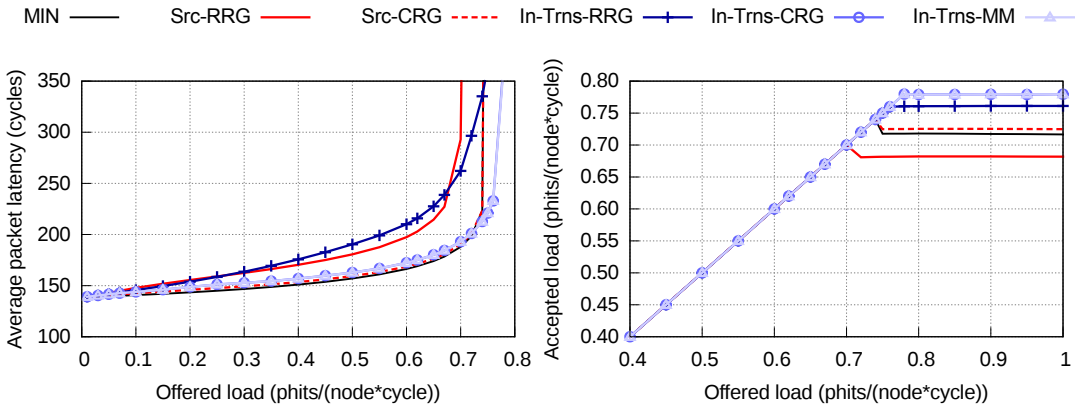
(c) Max/Min throughput ratio

Figure 3.10: Evolution of the fairness metrics with the network size, under an ADVc traffic load of 0.4 phits/(node·cycle), with RR arbitration without in-transit-over-injection priority.

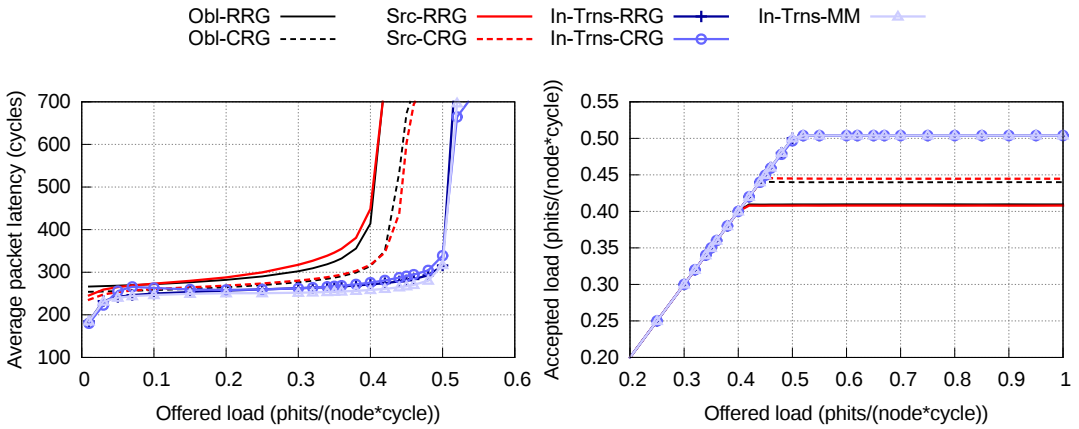
adversarial traffic patterns, specially ADVc. This section analyzes the impact of an explicit fairness mechanism, repeating the same results using age-based arbitration in the allocation process instead of the default RR arbitration. Early results combining age-based arbitration with the priority of transit traffic over injection present similar pathologies to those observed in Section 3.3.1, and are omitted for brevity.

3.3.4.1 Network performance with Age arbitration

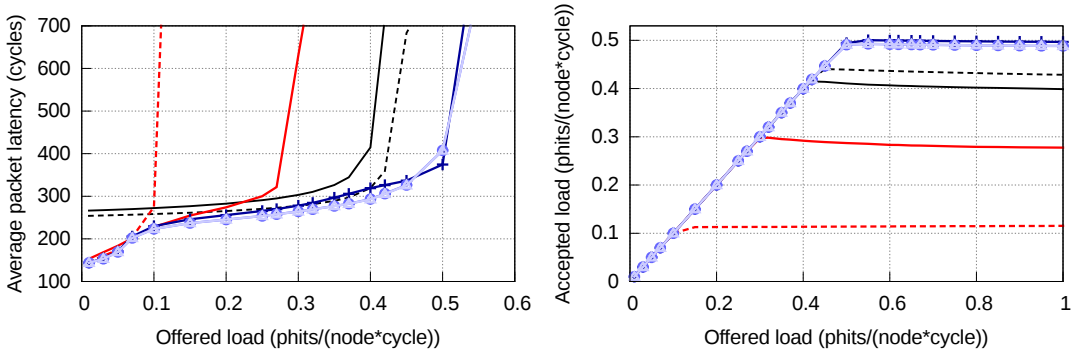
Figure 3.11 features the latency and throughput results for all the combinations of routing and global misrouting policy, using age-based arbitration. The most remarkable characteristic is the absence of fairness pathologies such as the latency spikes observed in the previous configurations. Under UN traffic, the RRG policy is detrimental because it employs longer paths, increasing average latency and diminishing average throughput. CRG and MM save the first local hop in most cases and both have similar performance with in-transit adaptive routing. Source-based adaptive mecha-



(a) UN traffic.

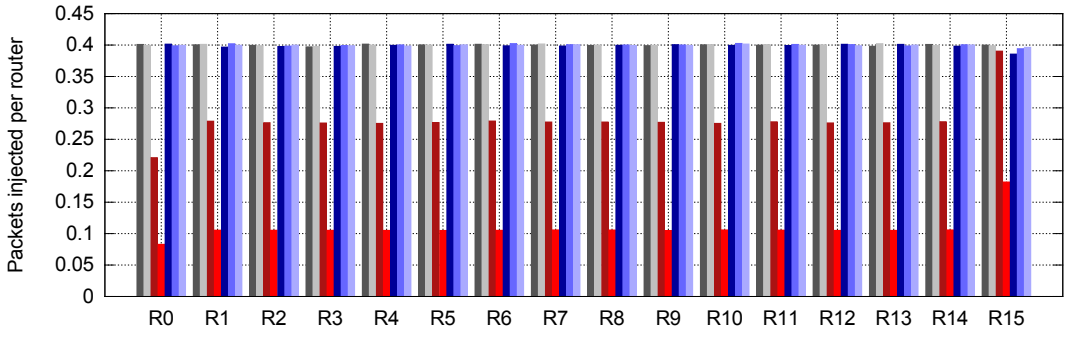


(b) ADV+1 traffic.

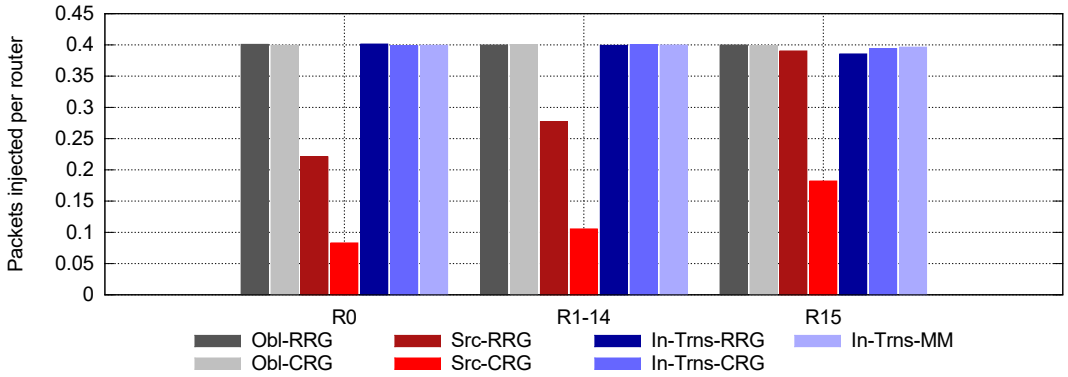


(c) ADVc traffic.

Figure 3.11: Latency and throughput under uniform (UN) and adversarial (ADV+1, ADVc) traffic patterns, employing age-based arbitration.



(a) Injection per router.



(b) Injection per router, grouping into 3 sets.

Figure 3.12: Injected load per router in group 0, under ADVc traffic with 0.4 phits/node/cycle of applied load. Age-based arbitration.

nisms obtain remarkably low throughput under adversarial traffic, particularly ADVc. This limitation comes from an inability to properly detect adversarial traffic, which is discussed in Section 3.3.2 and cannot be avoided through the use of age-based arbitration.

Under adversarial consecutive traffic the use of age-based arbitration introduces a problem of congestion, which slightly reduces throughput after saturation; this effect is less noticeable with in-transit adaptive routing. Interestingly, under ADVc traffic the throughput results for source adaptive are inverted, and RRG clearly presents the best result.

3.3.4.2 Throughput unfairness with Age arbitration

The injection per router of a group is reported in Figure 3.12, with router R_0 again receiving the traffic from other groups and R_{15} processing all the outgoing traffic from the group under minimal routing. Figure 3.12b collapses the results of the rest of the routers in the group into a single set of bars for the sake of simplicity. There are two very significant changes from the results without age-based arbitration. First, all the in-transit adaptive routing mechanisms obtain a fair result, which is a expected

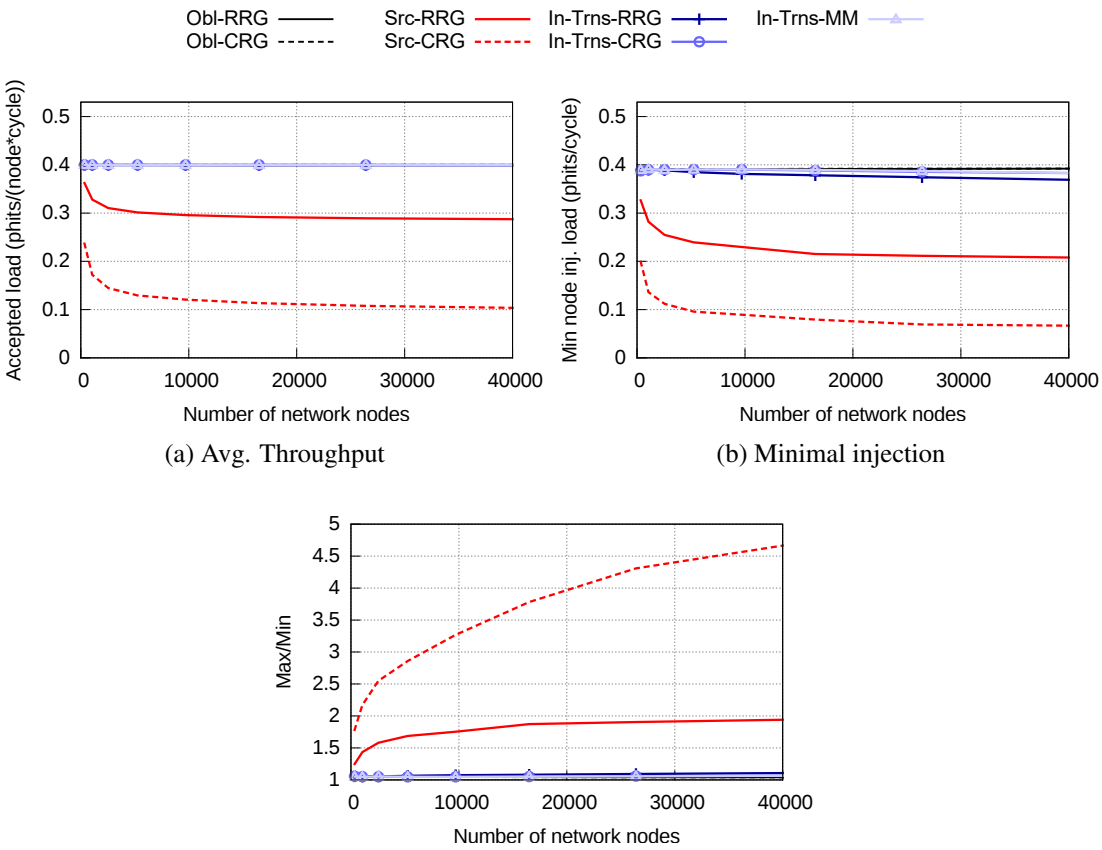
Table 3.3: Fairness metrics for all routing and global misrouting policy combinations under ADVc traffic with age-based arbitration. Values are specified for two different traffic loads per combination, one below and one above the average saturation point.

	Avg sat. load	Offered load	Min inj. load	Max/Min	COV
MIN	0.05	0.03	0.0275 (91.7%)	1.180	0.0236
		0.55	0.0156 (1.56%)	29.54	0.5676
Obl-RRG	0.40	0.35	0.3422 (97.8%)	1.046	0.0068
		0.55	0.3794 (68.9%)	1.212	0.0317
Obl-CRG	0.43	0.40	0.3919 (97.9%)	1.043	0.0063
		0.55	0.3797 (69.0%)	1.359	0.0632
Src-RRG	0.28	0.25	0.2428 (97.1%)	1.060	0.0081
		0.55	0.1961 (35.6%)	2.802	0.2190
Src-CRG	0.12	0.10	0.0956 (95.6%)	1.092	0.0128
		0.55	0.0782 (14.2%)	4.167	0.1830
In-Trns-RRG	0.50	0.40	0.3784 (94.6%)	1.082	0.0103
		0.55	0.2582 (46.9%)	2.069	0.1033
In-Trns-CRG	0.49	0.40	0.3883 (97.1%)	1.052	0.0067
		0.55	0.2701 (49.1%)	2.044	0.0979
In-Trns-MM	0.49	0.40	0.3896 (97.4%)	1.050	0.0068
		0.55	0.2735 (49.7%)	2.020	0.0973

behavior because age-based arbitration provides global fairness between all competing flows. The other interesting part is that source-adaptive routing fails to obtain fairness even with age-based arbitration, as analyzed in Section 3.3.2. For Src-RRG there is a significant variation between traffic in R_0, R_{15} and the rest of the routers, with R_{15} (the congested router) receiving the best injection rate. With Src-CRG the variation is reduced, but the throughput is so low that the mechanism is not competitive.

Table 3.3 quantifies the fairness results of each configuration with age-based routing. Even with the explicit fairness mechanism of age-based arbitration, all the configurations present a given level of unfairness after saturation, specially Src-CRG. Before saturation, in-transit adaptive mechanisms perform as fair as the oblivious ones using any of the global misrouting policies, as observed in the injection results of Figure 3.12. Results with source-adaptive routing are again constrained by their poor average throughput, particularly with CRG. The reference MIN performs notably worse than the other mechanisms since the amount of congestion is extremely high due to a poor balance of the link usage.

Finally, Figure 3.13 reproduces the evolution of the fairness with the network size when age-based arbitration is employed. Figures 3.13b and 3.13c demonstrate the inability of age arbitration to avoid the pathological unfairness effect with source-adaptive routing (and its growth with the number of network nodes), although it diminishes significantly compared to the use of a RR arbitration policy in Figure 3.10. Moreover, the results validate the efficacy of age arbitration to eradicate unfairness with in-transit adaptive routing.



(a) Avg. Throughput

(b) Minimal injection

(c) Max/Min throughput ratio

Figure 3.13: Evolution of the fairness metrics with the network size, under an ADVc traffic load of 0.4 phits/(node·cycle), with age-based arbitration.

3.4 Conclusions

This chapter has evaluated the throughput unfairness in Dragonfly networks due to certain routing/traffic pattern combinations. This includes the *adversarial-consecutive* traffic pattern, which is particularly harmful for throughput fairness. Throughput unfairness can severely limit the execution performance of an application, since synchronous communications are bounded by the last message to be received.

Prioritization of in-transit traffic provides minimal benefits in terms of average throughput, but presents a high amount of throughput unfairness. Under ADVc traffic this unfairness can lead to starvation of nodes linked to the bottleneck router of every group when in-transit adaptive routing is used. The consequences of this effect can be better appreciated with longer injection queues. Without prioritizing in-transit traffic, in-transit adaptive routing provides the best results, particularly with the MM global misrouting policy which avoids concentrating traffic in the congested minimal router. However, throughput unfairness is not completely eradicated.

Age-based arbitration is evaluated as an explicit fairness mechanism to avoid throughput unfairness. With age-based arbitration, in-transit adaptive routing provides the best performance among all the routing mechanisms and achieves complete fairness, at the expense of a more complex arbitration policy. It requires a comparator of the age of the packets in the router arbiters, plus additional router hardware to increase the age of the packets. Combined with this policy, source adaptive routing provides relatively poor performance and provides a less fair distribution of the network resources. These fail to properly detect congested links under ADVc traffic, regardless of the arbitration policy.

A pathological case of unfairness and degraded performance is observed with source-adaptive routing under ADVc traffic, independently from the global misrouting policy employed. This effect occurs because the adaptive decision is only taken at injection, and the availability of saturation information is dissimilar for the bottleneck router, as analyzed in Section 3.3.2.

In the rest of the thesis only one global misrouting policy per nonminimal routing mechanism is considered, the one that provides the best overall performance and lowest throughput unfairness: RRG with oblivious nonminimal and source-adaptive routing, and MM with in-transit adaptive routing. In each case, the global misrouting policy selected also matches the closest behavior to the original definition of the routing in which their implementation is based; Valiant routing in the case of oblivious, PB for source-adaptive, and OLM for in-transit adaptive. For the sake of brevity, in the following chapters only the VAL, PB and OLM notations will be used. It has been observed that the use of age arbitration can be beneficial in certain adversarial traffic patterns such as ADVc; the evaluations in Chapters 4 and 5 do not consider ADVc traffic and employ Round-Robin arbitration to analyze each proposal in isolation, without interference from an explicit fairness mechanism.

Chapter 4

Contention counters

The adaptive routing mechanisms employed for the evaluation of the throughput unfairness in the previous chapter employ a misrouting decision to select between the preferred minimal path and one longer nonminimal path. This misrouting decision, also called *misrouting trigger*, is typically based on the congestion status of the network links. The congestion of a link is estimated through the occupancy of the associated router buffers, which is measured through the credits from the link-level flow control. Different variants of such mechanisms are used or have been proposed in Cray Cascade[56], UGAL [134], OFAR [63] and many other works.

Congestion-based misrouting triggers have significant shortcomings that limit their effectiveness; among those can be included a strong dependency on the buffer size, uncertainty in the estimation, a slow adaption to traffic changes, and the appearance of traffic oscillations due to feedback loops.

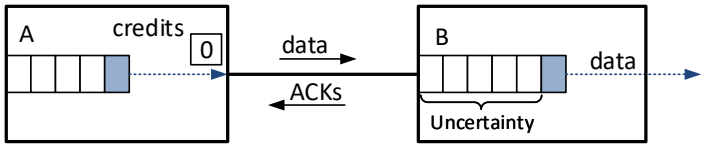
One fundamental flaw of a congestion-based misrouting decision is that the adaptivity in the routing comes when the network has already started to saturate; this erroneously identifies congestion as a *reason* to trigger the use of alternative paths, instead of the *consequence* of previous suboptimal routing decisions.

4.1 Limitations of credit-based congestion decision.

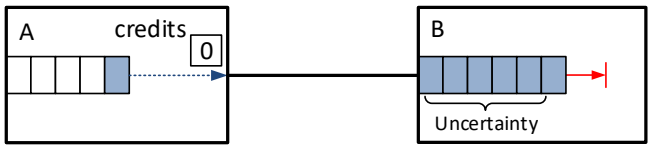
This work identifies some of the limitations of congestion-based misrouting triggers. It must be noted that, whereas some of these limitations can be individually mitigated, such mitigation is counter-productive for the others. The most clear case is the buffer length: employing small buffers affects the granularity of the decision, but large buffers exacerbate potential routing oscillations.

4.1.1 Granularity of the congestion detection

The granularity at which the level of queue occupancy can be measured is imposed by the size of the packets, the phits and the router buffers, along with the credit management mechanism. Virtual Cut-Through (VCT) switching with fixed-size packets



(a) Continuous transmission.



(b) Full queue.

Figure 4.1: Uncertainty in the use of credits with small buffers. The continuous transmission in the upper figure is indistinguishable from a full queue in the lower one, because all packets and credits are in-flight.

exhibiting coarser granularity, or routers with small buffers, can compromise the effectiveness of the detection mechanism.

4.1.2 Oscillations of routing

Occupancy-based congestion detection is prone to oscillations between different paths (in the case of the Dragonfly, minimal and nonminimal) due to the existence of a feedback loop. When the buffers in the minimal path saturate, traffic is diverted to the nonminimal routes; reducing the minimally-sent load lets buffers in the minimal route drain their packets, reverting traffic dispatch back to the minimal paths. This cycle provokes an oscillation of the network throughput, which aggravates with longer buffer sizes due to a longer time required to fill up and empty the buffers in the minimal path.

Such oscillations are especially important when the routing decision is not taken using local information, but rather relies on remote notifications of congestion (similar to ECN messages); this is the case of PB, as discussed in Section 1.4.2.1.1.

4.1.3 Uncertainty when using output credits

Credit-based flow control mechanisms employ a counter of the estimated empty space at the buffer in the neighbor router. The initial value of the counter is the buffer size of the receiver, of which the sender is aware. The credit count is decremented when a packet is sent, an incremented upon the reception of an acknowledgement packet (ACK). ACKs are sent from the receivers when the packets are forwarded from their input buffers. The bandwidth-delay product sets the minimum buffer length needed at the receiver to establish a reliable continuous transmission.

Estimations of remaining empty buffer space in neighbor nodes upon the credit count carry an inherent uncertainty due to in-flight data packets and ACKs on the link. Figure 4.1 illustrates two different situations for a pair of consecutive routers A and B

which have almost the minimum capacity dictated by the link round-trip time (RTT). The credit count for the output port in router A is 0 in both cases. In the upper case there is no network congestion, and router B forwards all packets as soon as they arrive; credit count in A still reaches zero because packets and credits are in flight. In the lower figure, credit count in A is also zero because the buffer in router B is full due to congestion.

A sender cannot distinguish between both situations (which demand a completely different response from the routing function) because it is not aware of in-flight packets and credits. To support a credit-based misrouting trigger, buffer sizes need to be significantly larger than the lower limit imposed by the RTT. An alternative is to track the rate at which credits are returned, but it would be accepted likewise from changes in the traffic pattern. Any of these solutions increases the complexity of router implementation, and its associated area and power requirements.

4.1.4 Reaction time on traffic changes and slow-lane traffic

Credit-based congestion detection mechanisms require a high occupancy in the buffers of the preferred path to trigger the use of alternative routes. Transitions of the traffic pattern to an adversarial case generate network hotspots and should prompt an immediate turn to alternative nonminimal paths, but take a significant amount of time to fill the buffers and trigger the congestion detection. Furthermore, traffic in congested paths is condemned to suffer a high latency before reaching its destination.

Figure 4.2 showcases this problem. After a change in the traffic pattern, packets from input ports $P_1 - P_4$ in router A target P_9 as their minimal output port, but might advance nonminimally through one of $P_5 - P_8$ output ports. Since multiple input ports are competing for the same output, nonminimal routing is desirable to sustain performance. However, nonminimal routing is not triggered until the input queue in router B reaches a significant population count, as is the case in the lower figure. At this point the input queues in router A will typically be quite populated, forcing packets in the minimal path to endure high latency due to queue drainage on top of the high latency required to detect the adversarial traffic scenario. Such overhead is unavoidable because congestion can only be detected if part of the traffic goes through the slow, congested path. Furthermore, as it occurs with the oscillations of routing, this effect magnifies with large buffers.

4.2 Contention counters

In order to prevent the associated shortcomings of relying on congestion indicators (such as the buffer occupancy) to trigger adaptive routing, this work explores the use of a *network contention* metric. A contention-based misrouting trigger chooses between a preferred (minimal in the case of the Dragonfly) path and one or more alternative paths based on the contention level of each port.

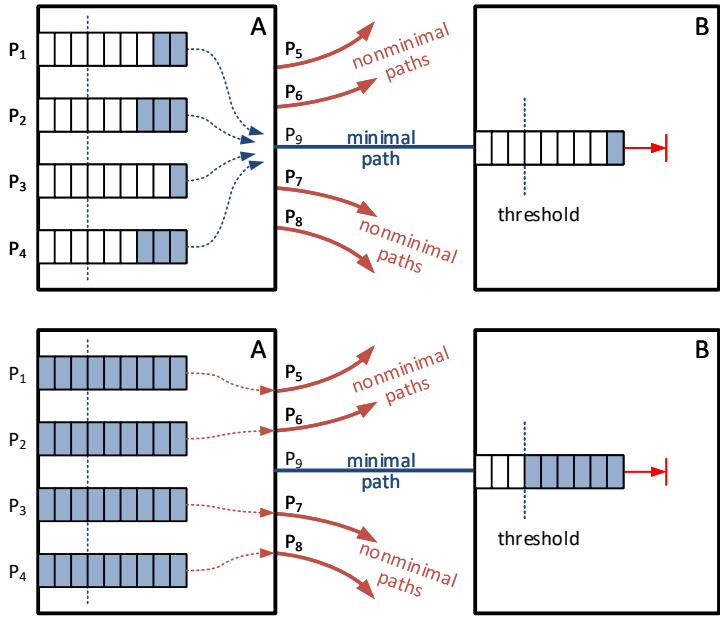


Figure 4.2: Reaction time on traffic changes and slow-lane traffic. In the upper figure, the traffic pattern changes and multiple input ports compete for the same minimal output with low occupancy. In the lower figure, the queue in the minimal output has got full enough and traffic is diverted nonminimally, but all the input queues have become full and need a long time to drain.

In this work, a set of *contention counters* estimates the contention of each output port, based on the demand of each output port as the egress in the minimal path of the packets at the input buffers. The idea behind this mechanism is that an output port suffers contention when multiple packets want to use it simultaneously. In such case, packets will be diverted to alternative paths using nonminimal routing without requiring the queues to be full. Hence, the mechanism decouples the buffer capacity from the misrouting trigger mechanism and permits an early detection of adverse network situations before performance degrades due to fully populated buffers.

4.2.1 Implementations

Multiple variations of the general idea of contention counters can be devised. This work evaluates four different variants; three of them are topology-agnostic and rely on local information, whereas the fourth is specifically designed for a Dragonfly network and distributes contention information across the network.

4.2.1.1 Base

The *Base* implementation of the mechanism employs one contention counter per output, as shown in Figure 4.3. When the header of a packet reaches the head of an input

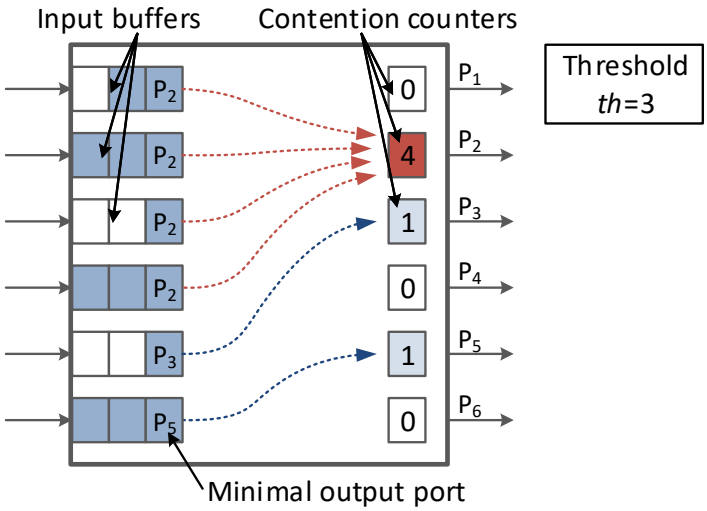


Figure 4.3: *Base* contention-detection mechanism. Output port P_2 is marked as having contention, since its counter exceeds the threshold $th = 3$.

buffer, the routing function determines its minimal path and increases the contention counter for the corresponding output port. Alternative routing is only triggered when the counter for the minimal output port has exceeded a given threshold th. Counters are only decreased when a packet is completely forwarded, i.e., when its tail is removed from the input buffer. Counters only track the packets for which they are the minimal egress, even if those packets are diverted to other paths and sent through a different port.

Only the packets at the head of input buffers are accounted to prevent a single flow from one input port to trigger misrouting when long buffers are used; this decouples the mechanism from the buffer length. Similarly, counters are not decreased until the packet tail has been forwarded through the output port, which may be different from the preferred port. Doing otherwise would lead counter values to be excessively low to provide statistical significance, since packet headers would likely be received in different cycles.

Base is adequate for high-radix routers where several input ports contribute to the contention detection, and the counters provide enough statistical significance. It must be noted that virtual channels from a single input port can concurrently increment associated contention counters, even if they cannot advance concurrently to the crossbar.

4.2.1.2 Filtered

Base is prone to suffering high variability in contention estimation when multiple short flows cross the same router. Moreover, considering contention only from head-of-buffer packets arguably fails to properly reflect the contention of all short flows in the router.

The *Filtered* mechanism is a refinement over the *Base* implementation, employ-

ing an exponential averaging to smooth variations in the contention counters. Instead of the current contention counter value, the misrouting trigger relies in an estimation $contention|_t$ that depends on the estimation in previous cycles, following the exponential average in Equation 4.1. Other averages can be used instead, but they would require more storage. $Counter|_t$ represents the counter value as calculated in *Base*, initial contention is $contention|_{t=0} = 0$ and α is a tunable parameter that regulates the importance of the historic vs. the current value of the contention counter. As in *Base*, misrouting is triggered when the metric exceeds a fixes threshold.

$$contention|_t = \alpha \cdot contention|_{t-1} + (1-\alpha) \cdot counter|_t \tag{4.1}$$

4.2.1.3 Hybrid

The *Hybrid* implementations combines an estimation of the contention (same as in *Base*) with the traditional evaluation of the congestion, measuring queue occupancy. Two separate thresholds are considered, one for the contention counters and one for the number of output credits. Traffic is nonminimally routed whenever any of these two thresholds is exceeded. Since any individual threshold can act as trigger, both thresholds can be higher to prevent excessive misrouting without losing accuracy. This mechanism is designed to be effective to detect adversarial traffic patterns when the routers radix is too small and the counters fail to achieve enough significance.

4.2.1.4 Explicit Contention Notification (ECtN)

The *Explicit Contention Notification* mechanism (ECtN) applies to contention counters the idea of explicit congestion notifications employed in PB that was discussed in Section 1.4.2.1.1. ECtN distributes contention information among the routers in the same group of a Dragonfly network to ensure a more accurate routing decision, in addition to the per-port counters. Every router maintains two arrays of contention counters for the global links, denoted as *partial* and *combined*. Each of these arrays has as many counters as remote groups are in the network, considering that only one link joins every pair of groups. Counters in the *partial* array track information per destination group instead of per minimal output port. They compute only those packets that target a remote group and are being injected into the current group, coming from either a compute node or a global link. As with regular per-port counters, they are updated from the packets at the head of the router input queues and decremented only when packets leave the input queues.

Routers in each group broadcast periodically their *partial* arrays. The *combined* array is calculated adding the values from the counters of all the *partial* arrays within the same group. Upon reception of a *partial* array, routers update their *combined* arrays, as depicted in Figure 4.4. When traffic is injected to a group and the *combined* counter associated to the destination group of the packet exceeds a threshold, it is misrouted.

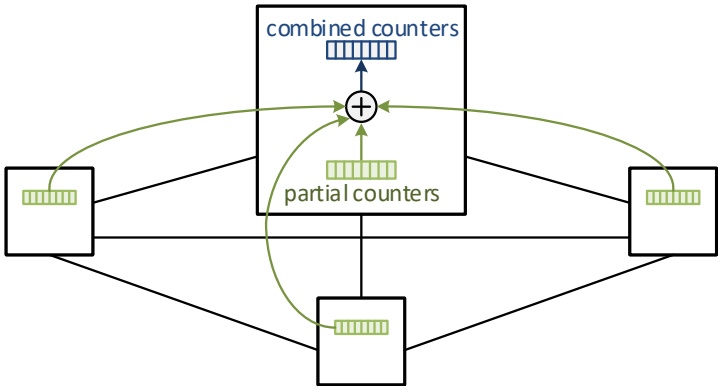


Figure 4.4: Obtention of a *combined* counter as a sum of received and own *partial* counters in the ECtN implementation.

Additionally, routers also maintain one *local* counter per output port as in the *Base* or *Hybrid* implementations. These counters provide contention information for its own outputs and allow for in-transit hop-by-hop routing decisions.

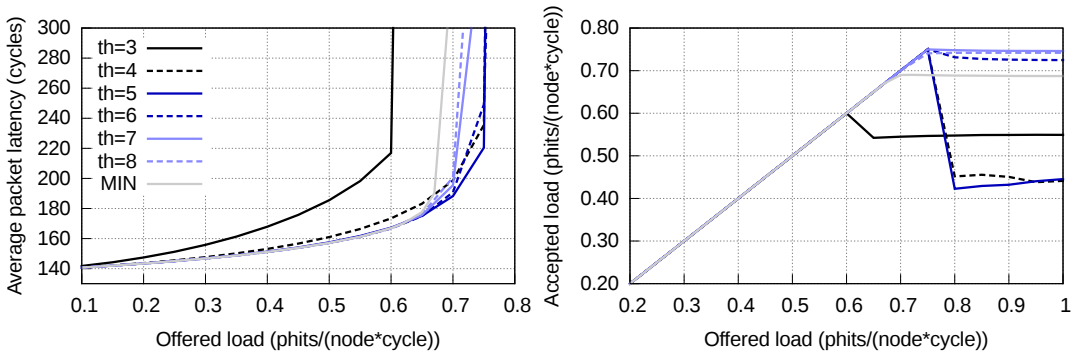
ECtN allows routers to have contention information for all the global ports in the group. This idea can be extended to other low-diameter networks where the broadcast of contention information is feasible in a timely manner.

4.2.2 Threshold selection

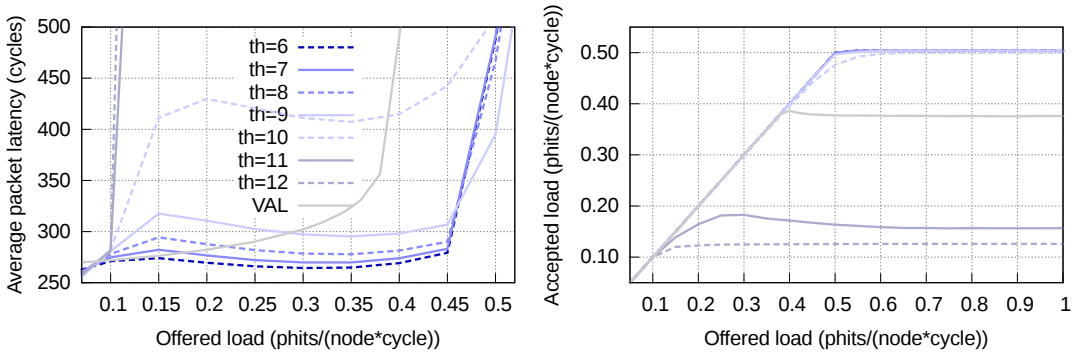
All the discussed implementations of contention counters trigger misrouting when the employed contention metric exceeds a threshold. As it occurs with congestion-based misrouting triggers, threshold selection imposes a tradeoff between performance under uniform and adversarial traffic patterns. Low threshold values penalize UN traffic, enforcing an unnecessarily high level of misrouting and increasing packet latency. The threshold needs to be high enough to prevent false triggers under saturation, to reduce the frequency at which misrouting appears. Under saturation it is safe to assume that all input VCs will have at least one packet, increasing the value of a given counter; assuming uniform distribution, the average value of the counters will equal the average number of VCs per input port. However, this is an average and there will be a number of ports with higher contention. A threshold doubling that average establishes a safeguard range to avoid the feedback loop that can originate with frequent misrouting under UN traffic; such a threshold is sufficient to prevent a performance decrease.

On the other hand, high threshold values promote minimal routing and penalize ADV traffic. In this case, packets in all p injection ports of every router target the same minimal output port, typically a local link to other neighbor router directly connected to the destination group. A threshold $th \leq p$ ensures that misrouting is applied at the injection ports and improves performance under adversarial workloads.

Figure 4.5 provides an evaluation of the impact of the threshold value over the performance under UN and ADV traffic for the network size presented in Section 1.6.4



(a) UN



(b) ADV+1

Figure 4.5: Sensitivity of *Base* to the misrouting threshold.

that is used in the performance evaluation of the mechanisms. As expected, higher threshold values provide better response under uniform traffic, and lower values improve throughput and latency under adversarial traffic. In this case, the average number of VCs per port with in-transit adaptive routing is 2.75, and the lower threshold bound accomplishes $th \geq 6$, as can be observed in Figure 4.5a. Theoretically, the threshold needs to be lower than the number of injection ports per router $th \leq p = 8$ to guarantee misrouting under adversarial traffic. In practice, the bottleneck router of every group easily detects the adversarial pattern and performs misrouting. This increases the average traffic in local and global input ports, and softens the change in performance with higher thresholds, as can be seen in Figure 4.5b.

Within the valid range, $6 \leq th \leq 8$ in this case, a low threshold should be selected to favor low latency under adversarial traffic. Therefore a threshold value $th = 6$ has been selected. Interestingly, routers with a larger radix expand the range of threshold values that do not compromise performance under uniform or adversarial traffic.

A similar study was applied to select the *combined* threshold $th_c = 10$ in *ECtN*. In the case of the *hybrid* variant, the contention threshold is raised (to $th = 8$) to prevent excessive misrouting, since the use of nonminimal paths can be triggered either by congestion or contention information.

4.2.3 Implementation costs

All the implemented models employ a fixed misrouting threshold. Under heavy adversarial traffic, this might lead to all of the traffic being diverted nonminimally because the contention counters are high, while the minimal path remains completely empty. In a real system this would typically not happen because not all traffic can be sent adaptively (e.g. in Cascade [56] minimal routing is used for packets that need to preserve in-order delivery). A statistical misrouting trigger can be alternatively considered, where a contention counter exceeding a threshold increases the probability of routing nonminimally. This allows the minimal path to still be used in a certain proportion. The use of statistical misrouting trigger has not been explored in this work.

The complexity of the use of contention counters in most of the implementations is very low, requiring a set of parallel counters [137] updated and compared with every packet sent in a similar fashion to credit-based routing mechanisms. The *filtered* implementation increases the costs by requiring an additional set of registers to store the previous value of the counters, plus the arithmetical logic to compute the contention metric. These registers can be implemented through latches, as the value is directly taken from the contention counters before they are updated. The update of the contention counters can be placed outside of the critical path if needed, at the expense of slightly less accurate misrouting decisions.

By contrast, the cost of *ECtN* is not negligible: it requires two additional large sets of counters, *partial* and *combined*, plus the required memory to hold the *partial* values received from other routers. This implementation also adds further stress to the network due to the delivery of the full *partial* counters. The evaluations in this work assume a spread of these counters every 100 cycles, without simulating the corresponding overhead. For a Dragonfly network of size $h = 8$, the *partial* arrays contain 128 counters (one for every global link in the group). Each counter has 4 bits to host the misrouting threshold ($10 \leq 2^4$). With the 10-byte phits discussed in Section 1.6.4, it would require around 6 phits to broadcast the *partial* counters, incurring in a 6% overhead in the communications. This traffic overhead can be diminished by simplifying the counter information exchanged. Simply limiting communications to only nonempty values provides a very similar overhead: the updated counters require a 7-bit identifier ($128 \leq 2^7$) and the number of active counters at a time can be as high as 40, if all the injection and global queues in the router target different outputs. A better approach is to combine it with incremental updates added to the current array values, reducing the size of each counter update. Another option is the use of asynchronous updates with a larger dissemination period, only sending immediately those counters that change abruptly.

4.3 Results

This section lays out the simulation results with the evaluation of the contention counters. Unless otherwise noticed, the network configuration follows the one described in

Section 1.6.4. Results include steady-state and transient experiments, where the traffic pattern changes during the simulation to observe the routing ability to recognize the new workload and adapt to it. In both cases, the measurements are performed after 60,000 cycles of network warm-up, as described in Table 1.2.

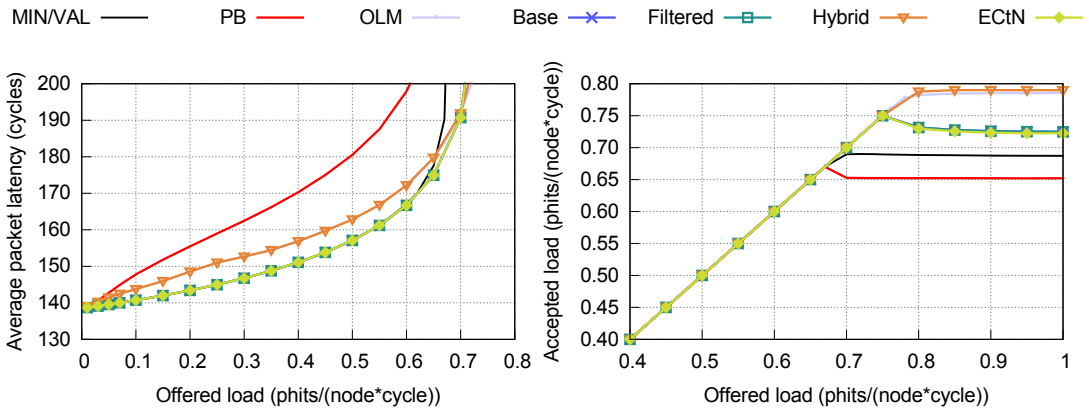
4.3.1 Steady-state results

Figure 4.6 displays the latency and throughput obtained under steady state experiments. In Figure 4.6a, the oblivious MIN routing sets the optimal latency achievable under UN traffic, because it never misroutes traffic. Both adaptive mechanisms based on credits (PB and OLM) exhibit higher latency, since they occasionally send traffic nonminimally based on the measured buffer occupancy. By contrast, the *base*, *filtered* and *ECtN* variants of the contention-based in-transit adaptive routing match perfectly the latency of MIN before congestion, which is arguably the most frequent region of operation of the network. The *hybrid* version can send traffic nonminimally based on the credit count and occasionally does under low loads, pushing its latency to identical values as OLM.

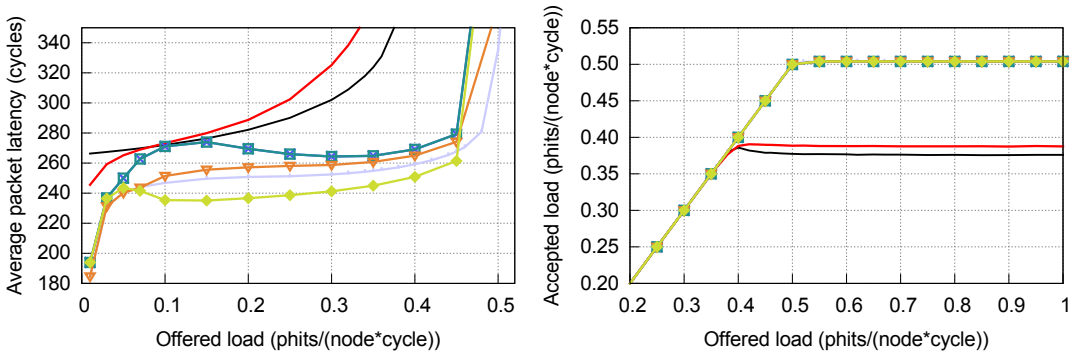
Throughput under UN traffic in Figure 4.6a shows that in-transit adaptive mechanisms perform a better job than MIN, occasionally sending traffic nonminimally under high loads to exploit all available outputs. Credit-based OLM is the second best performer in terms of throughput. Contention-based implementations fall slightly below OLM because they attempt less misrouting, since network contention is far less frequent than network congestion under UN traffic; they only exception is the *hybrid* variant, that thanks to the combination of network congestion and contention information is able to match (and slighlty outperform) credit-based OLM. None of the mechanisms obtains a throughput higher than 0.8 phits/(node·cycle).

Figure 4.6b depicts the behavior under adversarial ADV+1 traffic. Adversarial traffic patterns require global misrouting in order to achieve acceptable performance; VAL routing constitutes the reference because it always sends packets nonminimally. PB performs slightly better in throughput, at a cost of higher latency. Credit-based OLM obtains better latency and throughput than VAL and PB, since it avoids local misrouting and it sends part of its traffic minimally when possible. The throughput of all contention counters mechanisms is identical to credit-based OLM, slightly exceeding the Valiant theoretical limit of 0.5 phits/(node·cycle) because part of the traffic is sent minimally.

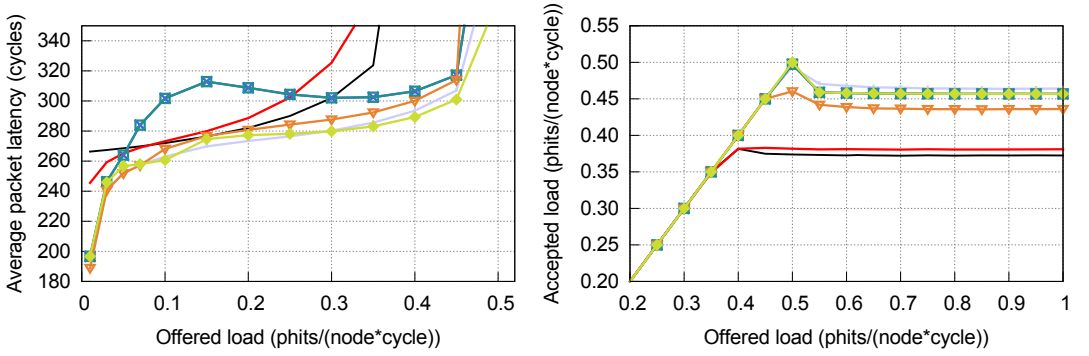
By contrast, their latency shows a particular behavior split in three different zones. Under very low loads (0.01 phits/(node·cycle) their latency is relatively low, because most of the traffic is sent through the noncongested minimal path. With low loads (0.05-0.15 phits/(node·cycle)) the latency using contention counters grows higher than OLM. Under these traffic loads there are not enough packets in the input queues to increase the contention counters and provide an accurate estimation of contention. This leads to minimal routing of the traffic, with packets accumulating in the heads of the queues until the counter eventually reaches the threshold and traffic is diverted nonmin-



(a) UN traffic.



(b) ADV+1 traffic.



(c) ADV+h traffic.

Figure 4.6: Latency and throughput under UN and adversarial traffic (ADV+1, ADV+h).

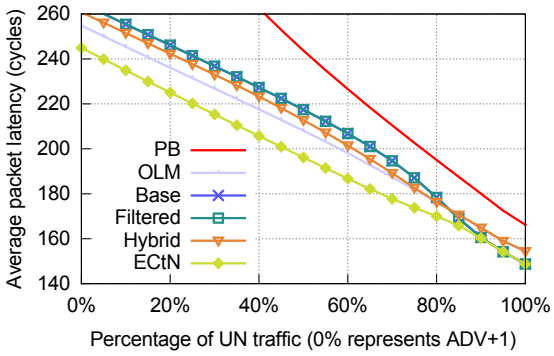


Figure 4.7: Latency under a load of 0.35 phits/(node·cycle) with a mixed traffic pattern, split between ADV+1 (left) and UN (right).

imally. The increase in latency equals the cycles required for the traffic to accumulate and trigger misrouting. Alternative implementations might consider a variable threshold based on the amount of traffic in the input queues; however, it would worsen the performance under UN traffic due to an increase in the rate of unnecessary misrouting. Finally, under loads below the saturation point, the latency of contention-based mechanisms is competitive with OLM. *ECtN* is the best performer under ADV+1 traffic, since the distribution of contention information among all the routers in the group increases the statistical significance of the measurement, allowing misrouting at injection whenever it is required.

Figure 4.6c shows the results under ADV+h traffic, which requires local misrouting in the intermediate group to prevent a pathological case of local link contention described by Prisacari *et al.* in [122]. This local misrouting makes the latency of VAL and PB (which force misrouted traffic to reach an intermediate node before traversing minimally to the destination) more competitive than under ADV+1. Results are similar to those under ADV+1 traffic, except for a drop in throughput after the saturation point. This behavior occurs under all in-transit adaptive mechanisms (congestion- or contention-based) and VAL. Additionally, now *ECtN* is slightly outperformed by OLM for traffic loads between 0.1 and 0.25 phits/(node·cycle).

Figure 4.7 represents the average latency obtained when the traffic pattern is a combination of ADV+1 and UN in different rates, with a load of 0.35 phits/(node·cycle). Even in intermediate cases in which the traffic pattern is not clearly shaped, contention counters are competitive with OLM; most notably, *ECtN* clearly outperforms the other mechanisms, including credit-based OLM routing.

4.3.2 Transient results

One of the most relevant advantages of a contention-based misrouting trigger is the adaptability to traffic changes, since it does not need to wait for the router buffers to fill up or drain in order to determine the nature of the traffic. Figure 4.8 shows the evolution of the average per-packet latency and the percentage of misrouted packets

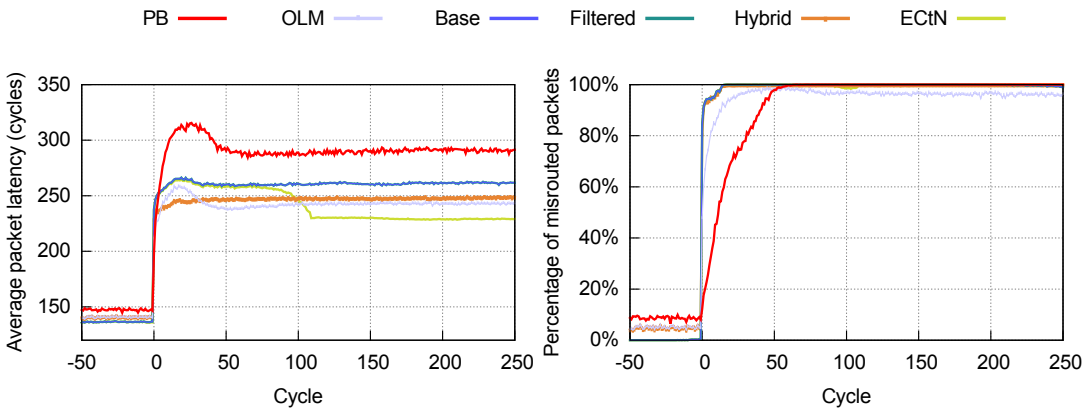


Figure 4.8: Evolution of latency and misrouting when the traffic pattern changes from UN to ADV+1, with a load of 0.2 phits/(node·cycle).

when the traffic pattern makes a transition. After a warmup with UN traffic, at cycle 0 the traffic shifts to ADV+1. Other transitions have also been evaluated, with similar results.

The congestion-based adaptive mechanisms (PB and OLM) show a transient period of around 100 cycles while routing is adapting to the new traffic pattern; this transient consists of a "bump" where the average latency grows (waiting for queues to fill up and trigger misrouting) and then lowers thanks to misrouted traffic, and a phase where the latency raises very slightly before reaching convergence, where the number of packets arriving through the largest nonminimal paths grows until it stabilizes. Both phases last 50 cycles, and the second phase is more distinctive with OLM.

By contrast, contention-based implementations react almost immediately to the pattern shift, with a reaction time of around 10 cycles. *Filtered* has a very small lapse behind *base* due to the filtering effect; increasing the importance of the contention history over current values reduces flickering of the misrouting decision under high loads of uniform traffic at a cost of larger reaction times. Interestingly, *ECtN* follows the behavior of *base* and *filtered* for the first 100 cycles, because pattern shift occurs immediately after the *partial* counters have been broadcasted with the values sensed under UN traffic. For that period, routers rely on local information to detect the presence of an adversarial traffic workload; when the remote information is updated, all routers are able to acknowledge the new traffic pattern and divert traffic nonminimally at injection. From this moment, minimal local hops in the source group are avoided and latency decreases.

The amount of misrouted packets follows the same trend as per-packet latency. It is notable that the amount of misrouted packets when using counters is very close to 0% or 100% when the routing stabilizes; the slowest mechanisms to stabilize are PB and OLM, which are based on credits. Furthermore, OLM always sends minimally a small percentage of its traffic under the adversarial pattern. Contention-based implementations send almost all their traffic nonminimally to avoid congesting the minimal links in adversarial traffic, and almost always minimally with a uniform traffic pattern.

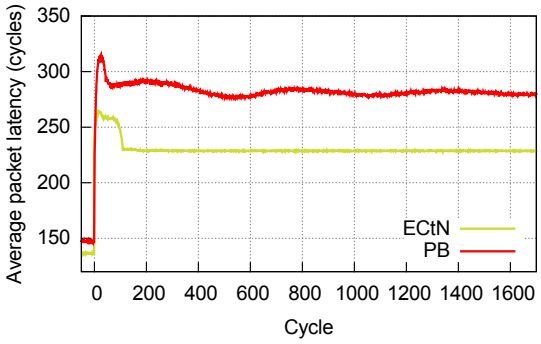


Figure 4.9: Zoom-out of the evolution of latency when the traffic pattern changes from UN to ADV+1, with a load of 0.2 phits/(node·cycle), considering only PB and *ECtN*.

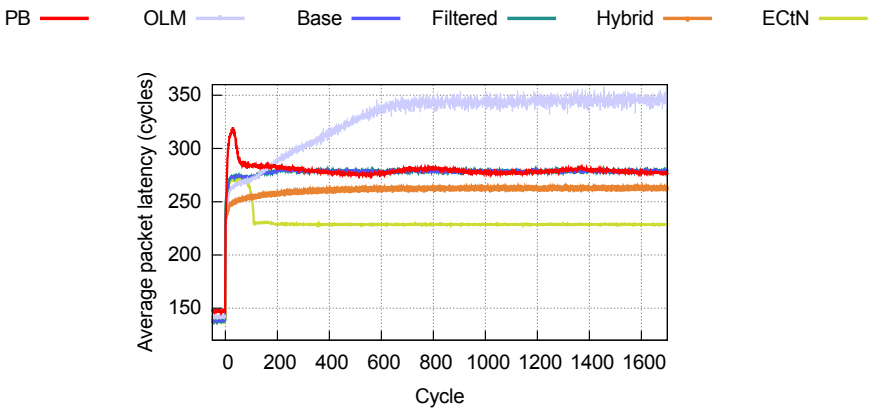


Figure 4.10: Evolution of latency when the traffic pattern changes from UN to ADV+1, with a load of 0.2 phits/(node·cycle), employing large buffers of 256 and 2048 phits per VC for local and global ports, respectively.

Routing mechanisms that react to congestion are prone to oscillations because the congestion status acts as misrouting trigger (thus influencing routing) but also depends on the previous routing decisions. When congestion status is received via Explicit Congestion Notifications from a remote router, the effect is amplified due to a longer control loop. This is the case of PB, which performs the misrouting decision based on the saturation status broadcast by the neighbor routers from the same group. Figure 4.9 shows the average per-packet latency for the same case as Figure 4.8 but with a larger timescale, focusing only in PB and ECtN. PB shows oscillations around every 600 cycles, which get progressively smaller as the queue occupancy converges to a steady state but do not completely disappear. In contrast, the behavior of ECtN is completely flat and does not alter once the *partial* counters are first updated after the pattern shift.

The last limitation of congestion-based adaptive routing to consider from those discussed in Section 4.1 is the delay in the detection of traffic changes with larger buffers. Figure 4.10 displays the reaction time as traffic changes from UN to ADV+1 when buffers are eight times bigger: 256/2048 phits per VC for input local/global

ports instead of 32/256. Output and injection buffers maintain their previous size. The reaction time of the two credit-based mechanisms (PB and OLM) is larger with deeper buffers; OLM now requires 700 cycles to reach its steady state, as opposed to 100 cycles in Figure 4.8. By contrast, mechanisms based on contention present the same reaction time as before.

Latency with PB does not increase significantly with smaller buffers. The behavior of OLM diverges significantly to the one observed with shorter buffers in Figure 4.8: following the first 100 cycles after the traffic changes, latency starts to grow until it converges to 350 cycles per packet, almost 100 cycles more than with the smaller buffers. This effect is due to the larger buffer sizes and the dependency of the misrouting decision on the buffer occupancy (relative to other queues in the same router). Since the comparison is relative, most packets are misrouted in-transit to avoid the congested global link (hence the rise of latency at cycle 0); when the transit queues start to fill up, congestion and latency increase. This growth stops when the amount of congestion at the local queues is enough to guarantee misrouting at injection. It must also be noted that the lowest latency achieved with OLM in the first cycles after the pattern shift is comparable or higher than the latency of all contention-based mechanisms.

4.4 Conclusions

This chapter has introduced the idea of a contention-based misrouting decision for adaptive routing. The use of contention information mitigates many of the shortcomings of adaptive routing dependent on congestion metrics such as credits: dependency on buffer size, oscillations in routing, and slow adaptation to changes in the traffic pattern.

Contention-based adaptive routing can be implemented in high-radix routers through a low-cost set of contention counters. In this work, four different variants are evaluated: *base*, *filtered*, *hybrid* and *ECtN*. The *base* implementation uses solely the values of the contention counters within the router to perform the misrouting decision. *Base* obtains optimal latency under UN traffic, and competitive throughput compared to the best state-of-the-art adaptive routing mechanisms. It also adapts immediately to traffic changes and mitigates the impact from congestion at in-transit queues when deep buffers are used.

The *filtered* mechanism uses a regressive metric which combines its previous value with the current information from the contention counters. This mechanism trades-off the reaction time to traffic changes for a lower impact of sudden spikes of traffic under uniform patterns, such as those that can appear with traffic bursts. However, its performance is identical to *base* and increases slightly the implementation complexity.

The *hybrid* version combines the contention statistics from the contention counters with congestion information through the use of credits. Under UN traffic, it improves throughput (outperforming credit-based OLM) but provides worse latency; with adversarial traffic, throughput is competitive and latency is better than the *base* variant.

ECtN disseminates contention information within the group in the same fashion as

PB to trigger misrouting at injection under adversarial patterns. This mechanism provides the best latency (or close to) in all scenarios, but entails a higher implementation cost, both in area and communication requirements.

Chapter 5

Flexible VC management

The routing mechanisms employed in the previous chapters rely on an increasing VC index order across the packet path as a deadlock avoidance mechanism, as introduced in Section 1.3. The only exception is the in-transit adaptive OLM routing, which relaxes these limitations to reuse VC indices for opportunistic nonminimal local hops provided that those hops can be avoided in case of congestion.

The most widespread deadlock avoidance mechanism in lossless low-distance networks is distance-based, relying on a fixed order in the use of VCs. Günther [69] proposed a simple mechanism in which packets follow a strictly increasing VC index sequence with each hop of their paths, assigning the VC index i for each hop i in the path. Figure 5.1 presents a sample case of this deadlock avoidance in a generic network with diameter 2 using MIN and VAL routing, where path length is up to 2 and 4 hops, respectively. MIN routing can be supported with only 2 VCs, but 4 are required for VAL routing. The VC index in each hop is fixed, and MIN is unable to exploit half of the VCs.

This deadlock avoidance policy restricts the number of buffer resources depending on the routing mechanism. Supporting long nonminimal paths as is the case in VAL and the adaptive routing mechanisms requires a larger number of VCs compared to

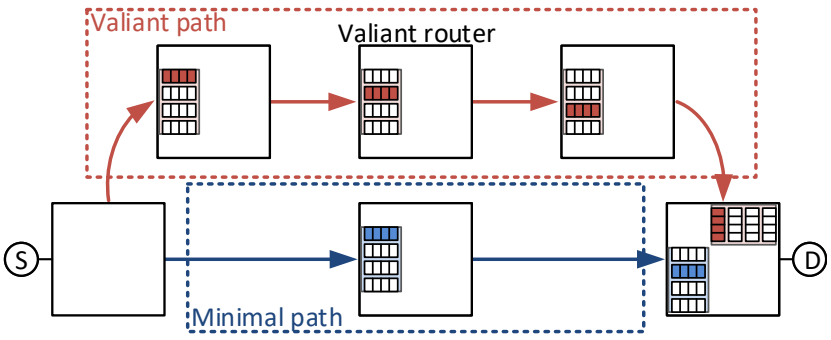


Figure 5.1: Distance-based deadlock avoidance with MIN/VAL routing in a generic diameter-2 network with 4 VCs. Traffic is sent from source S to destination D. Only the shaded buffers at each router of the paths are allowed for those hops.

the base minimal routing, as many as the length of the longest allowed path. VC requirements increase likewise when multiple QoS traffic classes are supported, or when protocol deadlock avoidance is considered, as is the case for the Dragonfly network in Cray Cascade [56], which employs 8 VCs. Unfortunately, an increasing VC-index deadlock avoidance policy prevents a free use of the VC buffers and enforces an inefficient use of the resources, increasing Head-of-Line Blocking (HoLB) and limiting performance. A larger number of VCs reduces HoLB but comes at a cost of increasing buffer area and complexity of the router implementation. Furthermore, unbalance of VC usage exacerbates with deep buffers such as those required to handle bursts of traffic and ease congestion detection under adaptive routing. Therefore, it is desirable to decouple the number and usage of VCs from the deadlock avoidance mechanism.

This chapter introduces *FlexVC*, a simple buffer management mechanism which permits to use the maximum amount of VCs in every hop of the path, mitigating HoLB and reducing up to 50% the memory requirements. FlexVC can be applied to any network topology with a distance-based deadlock avoidance mechanism that relies on VCs.

5.1 Limitations of deadlock avoidance mechanisms based on VCs

Distance-based deadlock avoidance policies that rely on VCs present significant limitations. Supporting long paths implies a large number of resources which scales with the network size. More significantly, the additional VCs required are only used when the longest paths are exploited and under-exploited in uniform traffic communications.

These limitations exacerbate when protocol deadlock is considered, with request-reply communications where destination nodes receive requests and generate replies back to the original sources. As discussed in Section 1.3, deadlock in request-reply traffic scenarios can be avoided employing separate virtual networks. Separate virtual networks provide isolation between different classes of messages but double the amount of buffers required, since for each hop of the path a *request VC* and a *reply VC* are needed.

5.1.1 Routing or link-type restrictions

In certain networks, the network links can be classified into different disjoint sets, which are traversed in a fixed order. This is the case of the X/Y/Z links in a 3D Flattened Butterfly (traversed following Dimension-Order Routing), the local/global links in Dragonflies (where the minimal paths are *l-g-l*), or the upward/downward links in a Fat Tree (with routing going first up and then down the tree). In such networks, the number of VCs required to avoid deadlock is lower than the maximum path length, and for each set of links it depends only on the amount of hops within the set.

In some cases, the VC requirements are imposed by the nature of the topology.

This is the case for Dragonflies and Orthogonal Fat Trees with adaptive routing. In other cases, the number of VCs is a trade off for minimal path diversity: in the FB, adaptive routing can be supported at the cost of a larger set of VCs.

Each hop of the path is assigned a given VC (specifying type of link and VC index), and the VC order sequence (denoted as the *reference path*) needs to be preserved to avoid deadlock. In the case of the diameter-3 Dragonfly the minimal reference path is $l_0 - g_1 - l_2$, where l/g denotes the type of link, local or global, and the subscript represents the VC index. To simplify the explanation, hops are assigned consecutive indices regardless of the link type. The amount of VCs is therefore not determined by the highest index value but by the amount of indices within each set; in this case, 2 VCs in input local ports and 1 VC in global ports are enough. This VC arrangement is denoted 2/1. Shorter paths with missing hops from the reference path ($l_0 - g_1$, $g_1 - l_2$) are possible when certain hops are not needed to reach the destination. However, a different order of hops ($l_0 - l_2 - g_1$) is never allowed. Valiant routing (as defined in Section 1.4.1.2) requires 4/2 VCs: $l_0 - g_1 - l_2$ to reach the Valiant router and $l_3 - g_4 - l_5$ to travel from there towards the destination. In-transit adaptive routing without opportunistic hops, as is the case of PAR (see Section 1.4.2.2), adds an additional local VC up to 5/2 VCs: $l_0 - l_1 - g_2 - l_3 - l_4 - g_5 - l_6$. As discussed at the beginning of this section, considering protocol deadlock avoidance doubles the path length and the number of resources for any of these routing mechanisms.

5.1.2 Buffer organization and cost

Augmenting the number of VCs has a significant penalty in the implementation costs of the router. A constraint of current ASICs is to use limited buffers [130]; trading off the buffer size for a larger number of buffers reduces the capability to handle large bursts of traffic and to detect congestion under adaptive routing. Moreover, small sets of VCs increase HoLB and limit performance. VC-index restrictions imposed by the deadlock avoidance mechanism are specially pervasive, because they prevent full usage of all VC buffers.

An alternative to improve the balance of usage between buffers is to switch to a different buffer allocation strategy. In Section 1.1 different implementations are discussed, such as Dynamically Allocated Multi-Queues (DAMQs [138]) in which a buffer pool is shared between VCs, and dynamically allocated buffers combined with smaller statically allocated per-VC buffers. Dynamic buffer organizations increase the implementation complexity of the router. FIFO buffers are typically implemented through circular buffers using SRAM [140]. DAMQs also rely on SRAM to store data, but need to store pointers for linked lists [138] or alternative control structures. The memory overhead for DAMQs is small but not negligible: for a 4KB DAMQ with 8-byte phits (512 phits per DAMQ), pointers need to be 9 bits long and the overhead is roughly 576 bytes (14% increase). Considering per-packet pointers (as in [40]) with 8-phit packets shrinks this overhead to 1.6%, but reduces flexibility with variable packet sizes. The added complexity of DAMQs also implies a penalty in memory accesses.

The implementation in [59] adds three cycles to read or write access latency. Choi *et al.* measure in [40] slowdowns in packet access time ranging 59-77% for different DAMQ implementations.

Furthermore, in the context of Dragonfly networks under adversarial traffic patterns they only provide small performance gains against statically partitioned memories because a significant amount of buffering needs to be statically allocated to avoid congestion. An evaluation of the impact of the amount of static buffering on the network performance is given in Section 5.3.1.

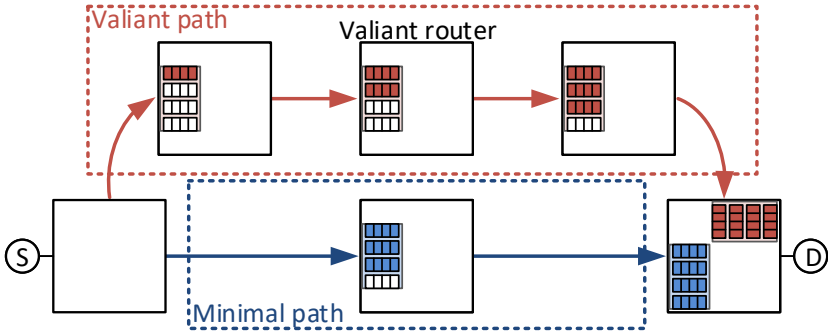
5.2 FlexVC mechanism

This section introduces a novel buffer management mechanism called *FlexVC*, which permits a more flexible use of the VCs. FlexVC combines statically partitioned buffers, opportunistic routing and a relaxed distance-based deadlock avoidance policy. FlexVC permits packet forwarding to several VCs, providing similar or better performance than shared buffer implementations as DAMQs. It also reduces the minimal number of buffers needed, because it can be implemented with less VCs than hops in the longest allowed nonminimal path.

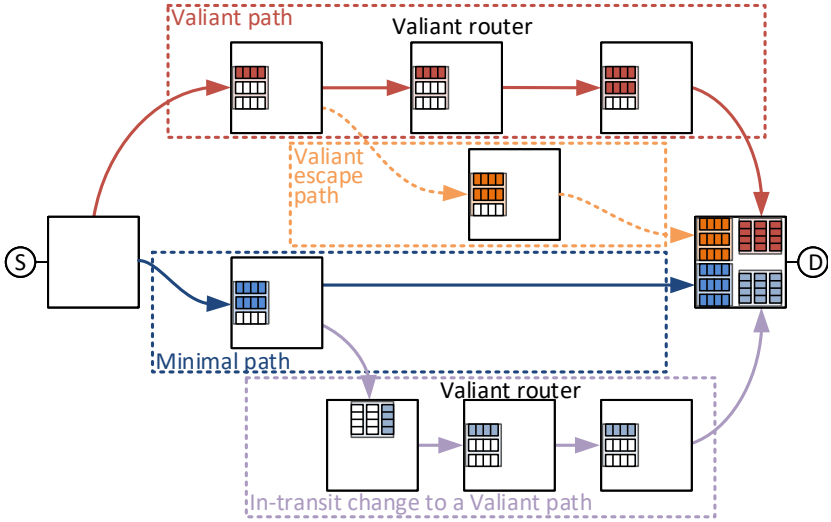
5.2.1 Base FlexVC

The key idea of FlexVC is that packets do not need to *follow* a strictly increasing order of VCs $\{c_0, c_1, c_2, \cdots\}$ to guarantee deadlock freedom. To avoid deadlock, it is only required that *such increasing path exists* for every hop in the path of a packet, from the VC currently hosting the packet upwards to the final destination. This increasing index path is denoted *escape path* as in the notation from [53], because it is deadlock free due to the absence of cyclic dependencies and provides an alternative route to other paths which might present cycles. Note that, contrary to Duato's protocol [53], escape paths in FlexVC share resources with other paths. For a VC index to be used in a given hop, it is only compulsory to have such increasing VC-index path towards the destination. Observer that all the implementations of FlexVC are applied to Virtual Cut-Through (VCT) switching, which is typical in system networks since the buffer requirements from link-level flow control already guarantee enough space to hold a complete packet.

Using the notation in [46], the set of channels is denoted by C and the set of network nodes by N. The incremental routing function $R : C \times N \mapsto C$ specifies an output VC, c_k, for each path determined by the routing protocol. A routing protocol R based on FlexVC specifies the *highest* VC c_k allowed in each hop, and considers *safe* and *opportunistic* hops. The router allocator and forwarding unit employ a certain *VC selection function* to select any VC, c_j, with available credits in the output port, such that $0 \leq j \leq k$. This work considers four different VC selection functions: *highest-index*, *lowest-index*, *JSQ* (Join the Shortest Queue) and *random*.



(a) Safe MIN/VAL paths with 4 VCs.



(b) Opportunistic VAL and in-transit adaptive paths with 3 VCs.

Figure 5.2: Sample FlexVC usage in a generic diameter-2 network. Allowed VCs in each hop are shaded.

Definition 1 Safe hops *require that from the input channel* c_{j_0} *there exists a* safe path $\{c_{j_0}, c_{j_1}, \cdots, c_{j_n}\}$ *to the destination with increasing vc index* $c_{j_k} > c_{j_l}$ $\forall k > l \geq 0$.

Definition 2 Opportunistic hops *require that regardless of the input channel* c_{j_0} *there exists a safe path* $\{c_{j_1}, c_{j_2}, \cdots, c_{j_n}\}$ *from the next buffer* c_{j_1} *to the destination node, with increasing vc index* $c_{j_k} > c_{j_l}$ $\forall k > l \geq 0$. Opportunistic paths *contain safe hops and one or more opportunistic hops, each of them with their associated safe path as escape.*

Every connected routing protocol R must provide at least one safe path for each possible destination to guarantee deadlock-freedom. *Opportunistic* paths can be used as long as the buffers after the next hop hold enough space for the complete packet; otherwise, packets revert to the corresponding safe path. This routing restriction avoids the appearance of cyclic dependencies in the extended resource dependency graph [53] and can be applied to wormhole networks as long as input buffers can store a complete

Table 5.1: Allowed paths using FlexVC in a generic diameter-2 network.

	VCs			
Routing	2	3	4	5
MIN	safe	safe	safe	safe
VAL	X	opp.	safe	safe
In-Trns	X	opp.	opp.	safe

packet. Several *opportunistic* hops can be performed in the same path as long as there is a safe escape path from each of them. The longest allowed safe path is a design parameter and can vary, as shown in Figure 5.2; routers with more VCs can reduce HoLB and implement more safe paths.

For each safe hop, the VC index determined by R equals the maximum amount of VCs minus the remaining hops to the destination router. For each opportunistic hop, the VC index determined by R equals the index set by the shortest safe escape path associated to the opportunistic hop.

Theorem 1 *A routing protocol R using FlexVC is deadlock-free with as many VCs as the longest safe path allowed in the network.*

Note that Definition 1 requires as many VCs as the path length to consider such path as safe. With consumption assumption [71], safe paths are deadlock free by induction on the VC indices. Opportunistic paths are deadlock-free since they necessarily have a safe escape path for each opportunistic hop, similar to Duato's protocol [53]. Observe that, as opposed to Duato's mechanism, FlexVC does not require dedicate resources for the escape path; VCs in the safe path can also be used for opportunistic hops as long as an increasing VC index order can be guaranteed in the safe path. FlexVC therefore reduces the minimum number of buffers required. It also allows to exploit *additional* VCs above the minimum required by the longest path, in order to reduce HoLB.

Figure 5.2 presents two examples of paths allowed by FlexVC in a generic diameter-2 network from a given source node S to the destination D. This network might represent a 2D Flattened Butterfly, a SlimFly, or a Demi-Projective network. The situation in Figure 5.2a employs routers with 4 VCs per input port. With 4 VCs, both minimal and Valiant paths (whose length is 2 and 4, respectively) are safe by Theorem 1 and the amount of allowed VCs on each hop only depends on the remaining distance to the destination. It is remarkable that in most hops FlexVC allows to choose between several VCs, compared to the base deadlock avoidance protocol which specifies a single one. This is particularly useful to absorb transient bursts of traffic because the effective buffer space per hop increases, without requiring dynamic buffer configurations such as DAMQs. Additionally, relegating higher-index VCs to latter steps in the path makes FlexVC immune to congestion caused by excessive occupancy of a single buffer.

The example in Figure 5.2b employs routers with 3 VCs per input port. With this configuration minimal paths are safe (they only require 2 VCs for a diameter-2 network) but Valiant paths are not with less than 4 VCs. However, for each opportunistic

hop of the path there is a safe escape path available, so opportunistic Valiant paths can be implemented. The first two hops of the upper Valiant path are opportunistic, and the escape path for the second is depicted; the escape path for the first opportunistic hop is the original minimal path. Changing from minimal to Valiant paths with in-transit adaptive routing is also supported with two opportunistic hops in the path. The escape path for the first hop of the Valiant path is the continuation of the minimal path, and the escape path for the second opportunistic hop is omitted for simplicity reasons. Note that VAL and in-transit adaptive routing would not be deadlock-free with the 2 VCs required for MIN because their opportunistic hops would not have an associated safe escape path. Table 5.1 summarizes the allowed paths depending on the amount of VCs for a generic diameter-2 network.

5.2.2 FlexVC considering protocol deadlock

At the beginning of the chapter it has been discussed how considering protocol deadlock doubles the VC requirements, considering separate sets of *request* and *reply* VCs. FlexVC concatenates both paths in a single unified sequence, joining all VCs in a single set rather than considering them as distinct virtual networks in order to increase the flexibility in buffer management. With this approach, request packets can only employ their associated request VCs in the manner presented in the previous section, but reply packets can employ both reply and request VCs. This increases flexibility and further reduces HoLB.

Theorem 2 *FlexVC is deadlock-free in presence of request-reply traffic.*

Both request and reply paths are deadlock-free considering Theorem 1 in the subsequence of request and reply VCs. Reply messages can also employ request VCs since there exists a safe path to the destination considering the complete sequence of VCs.

FlexVC can be further exploited to reduce the number of VCs required to support long paths. The set of reply VCs only needs to be dimensioned for safe minimal paths (by Theorem 1) since opportunistic reply hops in nonminimal paths can leverage lower-index request VCs. Therefore, a larger amount of VCs can be employed to either support longer safe paths (e.g. 4+4 for safe VAL paths in a diameter-2 network) or as *additional* VCs at the start of the request sequence. The second option reduces HoLB in request and reply subpaths, because they both are allowed to use them. Both alternatives are later evaluated in Section 5.3, observing that the use of additional VCs is beneficial compared to supporting longer safe routes.

Table 5.2 summarizes an example for a diameter-2 network. Distance-based deadlock avoidance requires 5+5=10 VCs to support safe VAL and PAR paths in both request and reply virtual networks. FlexVC can support the same paths with just 3+3=6 VCs, as illustrated in Figure 5.2b. Moreover, if reply VCs are dimensioned to only support MIN routing (leveraging request VCs for opportunistic VAL and in-transit adaptive paths) the number of VCs further diminishes to 3+2=5 VCs, as depicted in

Table 5.2: Allowed paths using FlexVC and considering protocol deadlock in a generic diameter-2 network.

	VCs (Request + Reply = overall)				
Routing	2+2=4	3+2=5	3+3=6	4+4=8	5+5=10
MIN	safe	safe	safe	safe	safe
VAL	X	opp.	opp.	safe	safe
In-Trns	X	opp.	opp.	opp.	safe

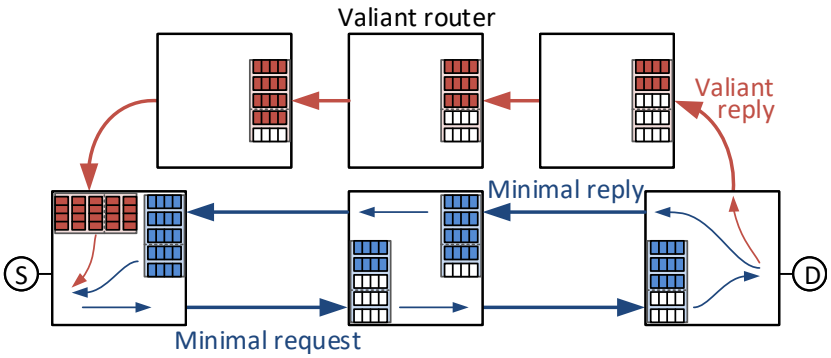


Figure 5.3: Example of protocol deadlock avoidance in a generic diameter-2 network with $3+2=5$ VCs using FlexVC.

Figure 5.3. This represents a reduction of 50% compared to the baseline reference. With 8 VCs as in Cray Cascade [56] safe routing can be applied to both request and reply routes, distributing 4+4 VCs; they can also be combined with opportunistic routing in the request (2+4+2 VCs) or reply (4+2+2 VCs) subpaths, with the two additional VCs available for each hop.

5.2.3 FlexVC with link restrictions

Section 5.1.1 discussed the particularities of networks with routing or topology-induced path restrictions, which lowered the number of VCs required below the length of the longest path. In such networks, FlexVC needs to also consider the specific sequence of hops in the reference paths, in addition to the distance to the destination.

Table 5.3: Allowed paths using FlexVC in a diameter-3 Dragonfly network following local/global links in topology-determined order.

	VCs (Local/Global)				
Routing	2/1	3/1 or 2/2	3/2	4/2	5/2
MIN	safe	safe	safe	safe	safe
VAL	X	X	opp.	safe	safe
In-Trns	X	X	opp.	opp.	safe

Table 5.4: Allowed paths with FlexVC considering protocol deadlock in a diameter-3 Dragonfly network.

	VCs (Request + Reply = overall)			
Routing	$2\times(2/1)=4/2$	$3/2+2/1=5/3$	$2\times(4/2)=8/4$	$2\times(5/2)=10/4$
MIN	safe	safe	safe	safe
VAL	X / opp.	opp.	safe	safe
In-Trns	X / opp.	opp.	opp.	safe

In the case of the Dragonfly network, 2/1 VCs support minimal paths $l_0-g_1-l_2$ but adding a single VC to either global (2/2 VCs) or local ports (3/1 VCs) would not support opportunistic VAL. In the first case, there would be no safe path after the first opportunistic local hop l_0 because there would not be two additional local hops for MIN routing to the destination. In the second case, there is no possible safe escape path after the first opportunistic global hop. To support opportunistic VAL and in-transit adaptive paths, one VC needs to be added to both local and global ports (up to 3/2 VCs) using the sequence $l_0-g_1-l_2-g_3-l_4$. To support *safe* VAL paths an additional local VC is needed, following the sequence $l_0-g_1-l_2-l_3-g_4-l_5$. Another extra local VC is required to make safe in-transit adaptive paths, employing the sequence $l_0-l_1-g_2-l_3-l_4-g_5-l_6$ and pushing the total to 5/2 VCs. This behavior is summarized in Table 5.3. *Additional* VCs of any type can be inserted at the start of the reference path.

Protocol deadlock avoidance requires in each sub-path a longer reference path taking into account the link type. The reply sub-sequence can be dimensioned for MIN paths as in the base case described in Section 5.2.1, and use opportunistic nonminimal paths that exploit the request subsequence of VCs. Table 5.4 summarizes the possible configurations, with 5/3 overall VCs required for opportunistic VAL and in-transit adaptive paths in the request and reply subpaths. Note that 4/2 VCs allows for opportunistic hops in the reply path under VAL and in-transit routing, but not for requests since there are no safe escape paths using request VCs.

5.2.4 Detection of adversarial patterns in source-adaptive routing with FlexVC

In the results from previous chapters, the implementation of source-adaptive routing has performed the misrouting decision marking global links as saturated when they exceeded a threshold over the average occupancy of the global links in the router. This comparison has been performed *per-VC*, observing the occupancy of the buffers in a given VC in other ports. In this chapter, it is also considered a variant observing the saturation *per-port*, aggregating all the buffers for the different VCs in the same port. The behavior of both implementations is similar under uniform traffic, but diverges significantly under ADV patterns, as is shown later in Section 5.3.

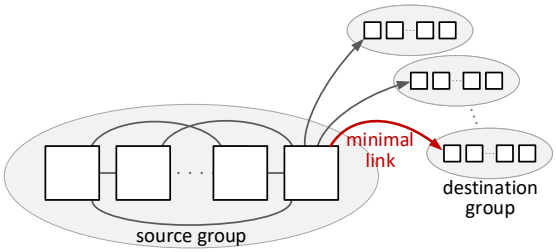


Figure 5.4: Zoom of a source group under ADV+1 traffic. The minimal global link is highlighted in red.

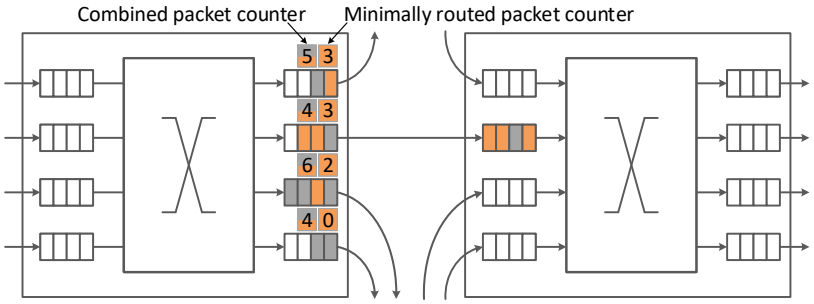


Figure 5.5: Router with *minCred* misrouting decision. Each output port has a regular counter tracking the total number of occupied slots in the next buffer, and a counter to track only the occupancy for minimally routed packets. Link-level flow control relies on the first counter, but the misrouting decision is taken based on the second counter.

Figure 5.4 shows a source group sending ADV+1 traffic, highlighting the minimal global link. Minimal global links connecting to destination groups are only used to deliver traffic minimally using the first VC in the path (or the first VC in the request or reply subpath, correspondingly, when request-reply traffic is considered). The rest of the global links forward nonminimal Valiant traffic, employing two global VCs in a balanced way: one for traffic coming out of the same group, and another for traffic that employs the current group as intermediate. Under a high traffic load all global links forward a significant amount of traffic and render *per-port* sensing inefficient. However, the *per-VC* variant identifies implicitly the traffic pattern by analysing the amount of traffic being routed minimally, because only the first VC is used for minimally routed traffic in the global links. Since this port is only used for minimal traffic, and nonminimal traffic in the other global links is balanced across two VCs, the first VC in the minimal global link receives twice as much traffic as other global link buffers and triggers misrouting even when a high threshold is employed.

FlexVC allows minimally and nonminimally routed traffic to share the same first VC in the global links. Therefore, both *per-port* and *per-VC* variants are unable to properly identify adversarial traffic and provide poorer performance, as will be observed in Section 5.3. An alternative version is provided to regain the capability to identify the traffic pattern with FlexVC, tracking separately the buffer occupation as-

sociated to minimally-sent packets to identify global link buffers as saturated. This version is denoted *minCred*, and can be combined with any of the previous strategies (*per-VC* and *per-port*); however, a per-port comparison performs significantly better under request-reply traffic or when a VC set larger than minimally required is used, because it accounts all minimally-routed packets spread across multiple VCs. The implementation cost of this strategy only implies an additional credit counter per output port and a flag per credit packet, to distinguish the credits for minimally and nonminimally routed packets, as can be observed in Figure 5.5. Regular data packets do not need any additional flags, since they already carried a field in the header specifying the type of routing.

5.2.5 Implementation costs

One of the main reasons for the use of FlexVC is to alleviate the hardware costs associated to the use of dynamic memory management configurations (such as DAMQs) while keeping or improving their performance. From the point of view of the mechanism, FlexVC is particularly frugal in its implementation. Part of the additional hardware required is a VC selection function that selects which VC to employ from the range available. As it will be demonstrated in Section 5.3.6, the main contributor to the achieved performance is the number of VCs employed, and the particular VC selection function employed does not have an appreciable impact. A policy like JSQ can be therefore considered competitive, and implemented easily with a stage of comparators which select the VC with the highest credit count. Regarding flow control, FlexVC relies on the original and simpler per-VC credit management rather than tracking both per-port and per-VC occupancy as is required in DAMQs.

The complexity and latency of the (de)muxers and allocators grows with the number of buffers implemented per port; nevertheless, FlexVC can exploit buffers which would be required anyhow for deadlock avoidance when using long paths, so in practice it does not need to increase the number of buffers. Conversely, it also allows to reduce the number of buffers required for nonminimal paths and to avoid protocol deadlock, simplifying the design of the router elements.

5.3 Simulation results

This section presents the results of the evaluation of FlexVC in combination with oblivious and adaptive routing mechanisms, observing the impact of router speedup and protocol deadlock. The routing mechanisms and the network configuration employed follow the descriptions in Sections 1.4 and 1.6.4. The uniform pattern with bursts of traffic BURSTY-UN employs a burst size of 5 packets; this implies that once a destination of a traffic burst is selected, in average 5 packets are sent towards said destination. The impact of FlexVC is compared against the baseline VC management with traditional FIFO and DAMQ buffers. Before evaluating the performance of DAMQs, a configuration for the DAMQ memory allocation needs to be selected in Section 5.3.1.

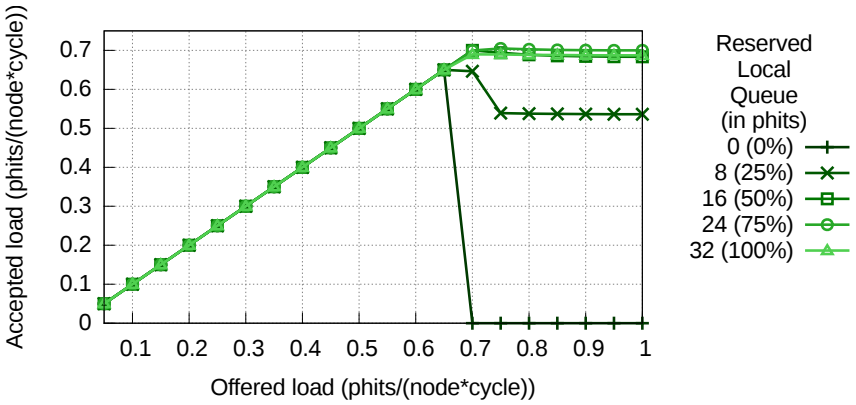


Figure 5.6: Throughput under UN traffic with MIN routing, using DAMQ buffers with different buffer reservation per VC.

Results are split into three subsections, one for oblivious routing, one with source-adaptive routing, and another with in-transit adaptive routing. Section 5.3.5 evaluates the performance without router speedup and the impact of the VC allocation policy in FlexVC.

5.3.1 Impact of reserved space in DAMQs

Figure 5.6 portrays the achieved throughput with MIN routing under UN traffic with DAMQ buffers, varying the size of the reserved buffers for each VC. This system employs 2/1 VCs with 128 phits in local ports and 512 phits shared among the VCs in input global ports. Each line represents a different amount of reserved statically allocated buffering per VC, in packets; possible values range from 0 to 8 packets. A reserve of 0 packets means that the whole DAMQ buffer is shared, whereas a reservation of 8 packets is equivalent to statically partitioned buffers (64 phits are statically allocated for each VC, and there is no shared buffering).

Without reserved statically allocated memory, the system presents deadlock: when VC c_0 is assigned all the memory in several ports, packets cannot advance to c_1 in the next buffer and create a cyclic dependency between VCs. Deadlock is only observed at saturation loads but may occur for any traffic load.

With 16 reserved phits per local port (25% of the overall space is statically allocated) the results present congestion. An analysis of the simulation data shows that most of the buffering space is again dynamically assigned to c_0, not leaving enough amount of buffer in c_1 to cover link round trip time. The optimal result is obtained when 75% of the buffering is reserved per VC and statically assigned, and the improvement over statically allocated buffers (represented by the curve with 100% reserved local queue) is negligible.

5.3.2 Results with oblivious routing

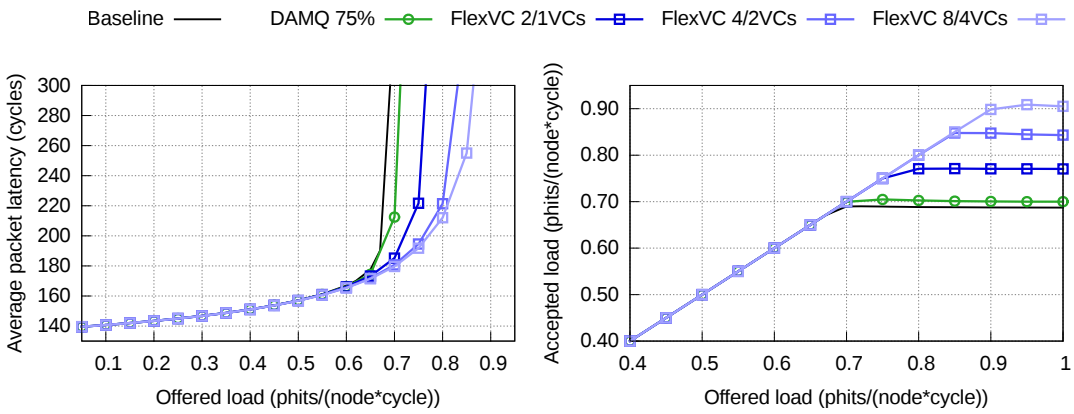
The performance evaluation is split into two separate cases: with oblivious traffic patterns, in which no cyclic dependencies at a protocol level are analyzed, and with request-reply traffic. Note that instead of the Valiant routing implementation described in Section 1.4.1.2 and employed in previous chapters, results under ADV traffic correspond to a VAL variant where packets at injection can recalculate the Valiant intermediate router. This allows a fair comparison against the similarly updated source-adaptive variant used in Section 5.3.3 to avoid a pathological case of HoLB; this effect is explained in more detail in Section 5.3.3.3.

5.3.2.1 Results with oblivious routing under traffic without dependencies

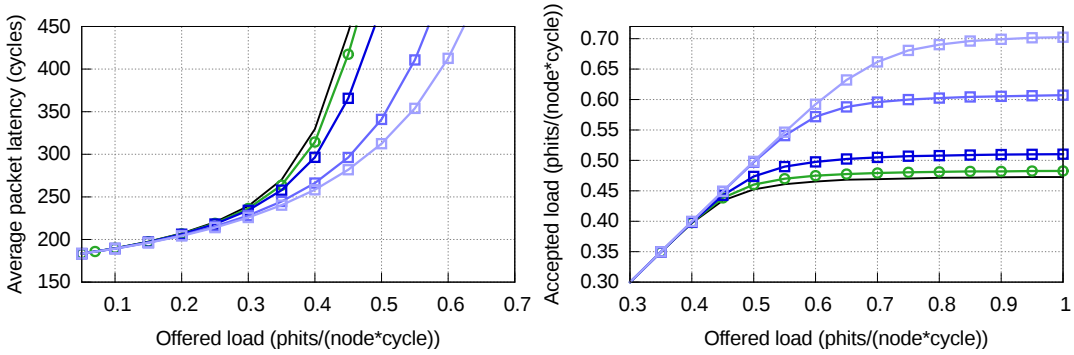
Figure 5.7 displays the latency and throughput under uniform and adversarial traffic patterns, employing the internal speedup described in Section 1.6.4 and keeping constant the amount of memory per VC (32 phits per VC at local input ports, and 256 phits per VC at global ports). Oblivious routing is employed and constitutes the baseline reference for each pattern, MIN under uniform traffic and VAL for adversarial. In all cases, latency remains similar before reaching saturation, regardless of the buffer management. FlexVC outperforms the baseline and the DAMQ implementation, significatively when the number of VCs exceeds the restricted set that can be used by the baseline and DAMQ-based mechanisms. In particular, under UN traffic FlexVC with 8/4VCs is able to reach a throughput of 0.9 phits/(node·cycle), approaching the maximal performance of the network. Interestingly, under BURSTY-UN traffic the saturation throughput decreases with respect to UN, and the average latency curves diverge slightly before saturation: for a load of 0.4 phits/(node·cycle) DAMQ reduces average latency in 4,8% over baseline, while FlexVC reduces 11,2%, 23,8% and 27,6% with 2/1, 4/2 and 8/4 VCs respectively. Under ADV traffic with VAL routing, the network links represent a significant bottleneck; nevertheless, FlexVC achieves a throughput of up to 0.49 phits/node/cycle in saturation, approaching the theoretical limit for VAL routing described in Section 1.4.1.2. Latency-wise all implementations perform similarly.

However, the comparison is not entirely fair for the case of FlexVC with larger VC sets than strictly required (4/2 VCs and 8/4 VCs), where not only the number of VCs per port is higher than in the baseline and DAMQ cases but also the amount of memory (the amount of memory per-VC is kept fixed). Figure 5.8 compares the throughput achieved in saturation with the same mechanisms when the memory per port is constant. Four total buffer capacities have been considered: 64/256, 128/512, 192/768 and 256/1024 phits per local/global port. Left charts refer the absolute throughput in phits/(node·cycle), whereas right charts display the relative increase over the baseline with the same total buffer capacity.

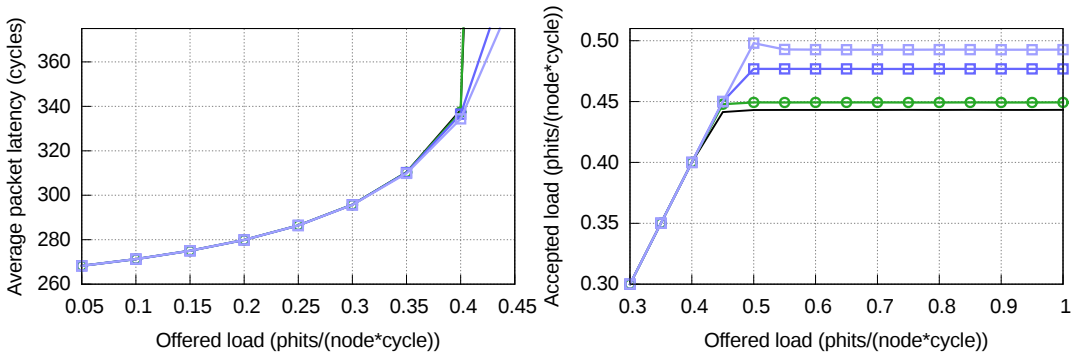
FlexVC is beneficial under uniform and adversarial traffic patterns for all buffer



(a) UN traffic with MIN routing.

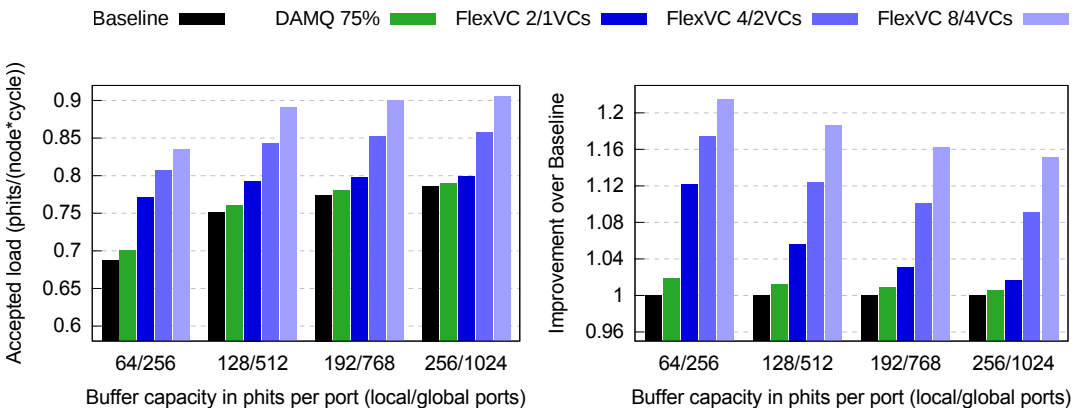


(b) BURSTY-UN traffic with MIN routing.

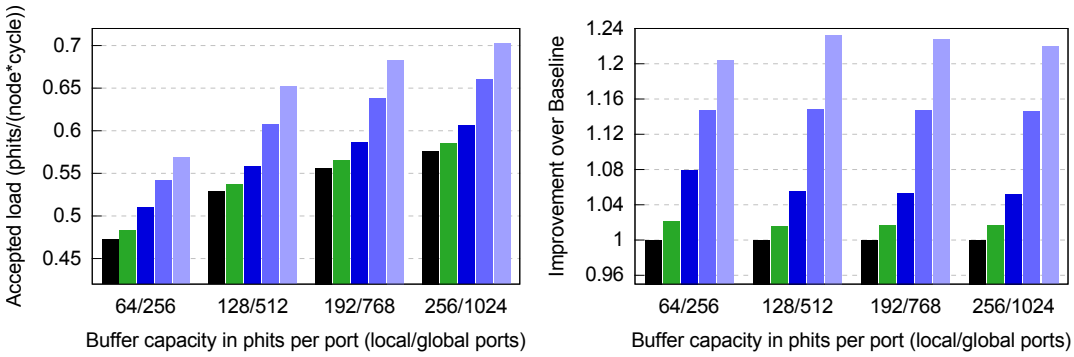


(c) ADV+1 traffic with VAL routing.

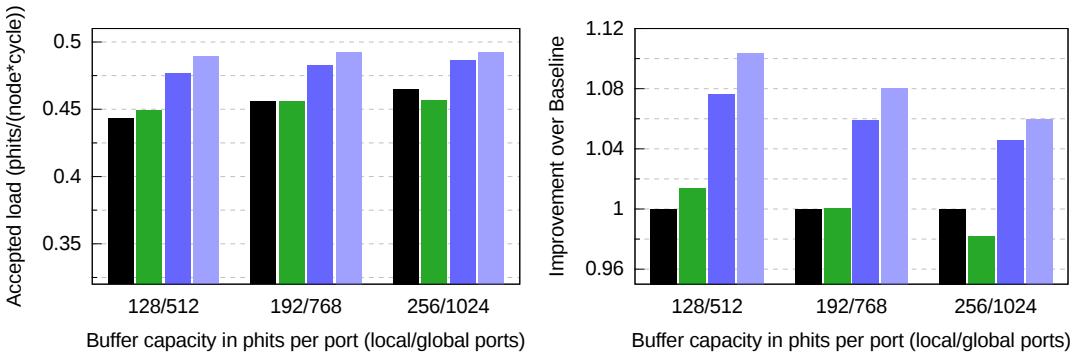
Figure 5.7: Latency and throughput under uniform (UN), uniform with bursts of traffic (BURSTY-UN) and adversarial (ADV+1) traffic with oblivious routing.



(a) UN traffic with MIN routing.



(b) BURSTY-UN traffic with MIN routing.



(c) ADV+1 traffic with VAL routing.

Figure 5.8: Absolute and relative maximum throughput under uniform and adversarial traffic with oblivious routing.

sizes, increasing throughput between 8% and 12% with the initial amount of memory (64/256 phits per port). Under UN traffic, FlexVC with 2/1 VCs and the smallest memory configuration achieves the same throughput as the baseline and DAMQ implementations with three times more memory per port. Since the number of VCs is the same, the improvement comes from a more flexible usage of the VCs that helps to mitigate HoLB. Furthermore, FlexVC with four times more VCs but only 64/256 phits per port clearly outperforms baseline and DAMQ with four times more memory. Largest buffer sizes reduce the improvement that can be achieved with FlexVC and bigger VC sets, but FlexVC is beneficial in every case compared to the baseline and more efficient than DAMQ buffers.

5.3.2.2 Results with oblivious routing under request-reply traffic

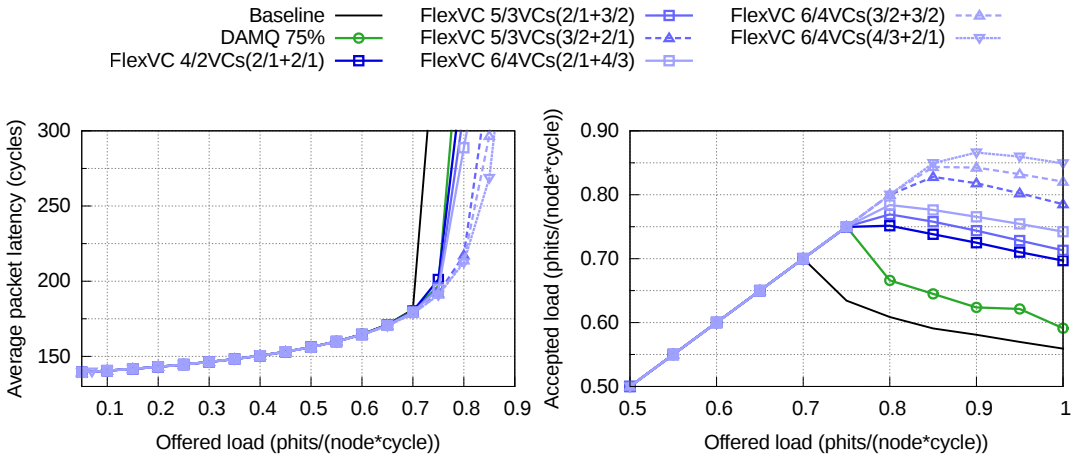
Figure 5.9 displays latency and throughput modeling request and reply messages as described in Section 1.6.2. In this case, the minimum number of VCs that can be used is 4/2 (2/1+2/1) for MIN and 8/4 for VAL. Latency curves are similar below the saturation point; however, under UN the base implementations (MIN and DAMQ) present congestion after reaching the saturation point, with DAMQ providing slightly better response. FlexVC with the same 4/2 VCs presents both higher peak throughput and a less pronounced congestion effect. Throughput at maximum load increases 24.6% from MIN and 17.9% from DAMQ.

The use of more VCs increases peak throughput and mitigates congestion. FlexVC with 6/4 VCs arranged in 4/3+2/1 reduces congestion significantly (less than a 2% drop from peak to highest offered load) and reaches a 23.7% increase over MIN; other configurations present intermediate results. It is noteworthy that throughput under request-reply traffic is not sorted by the overall amount of VCs, but by the amount of VCs in the request subpath. The three FlexVC configurations with 2/1 VCs in the request subpath are at the bottom, two configurations with 3/2 are in the middle, and the best configuration employs 4/3 VCs for requests. While only 2/1 VCs are required in each subpath for MIN routing, the allocation of *additional VCs* at the start of the request subpath makes them available for both requests and replies, making a more efficient use of them.

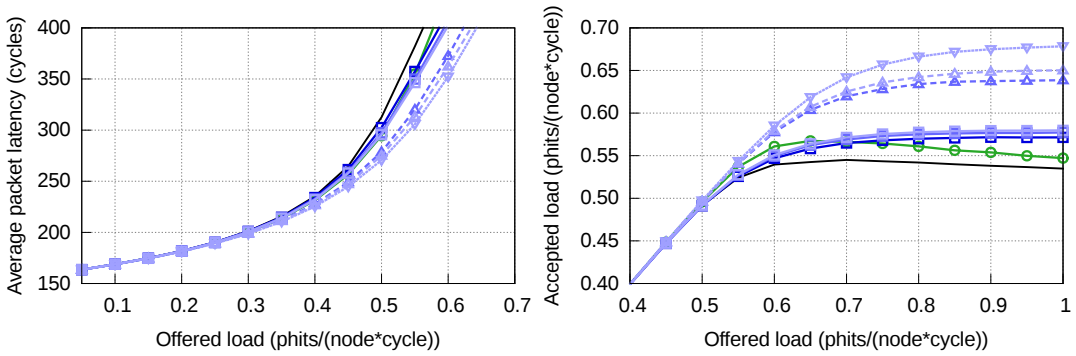
BURSTY-UN and ADV+1 only present congestion with the DAMQ implementation, and with a much smaller impact. The behavior under BURSTY-UN traffic is analogous to that of UN; in the case of adversarial traffic, the impact of the distribution of additional VCs is less noticeable because the maximal theoretical limit of 0.5 phits/(node·cycle) explained in Section 1.4.1.2 is reached.

5.3.3 Results with source-adaptive routing

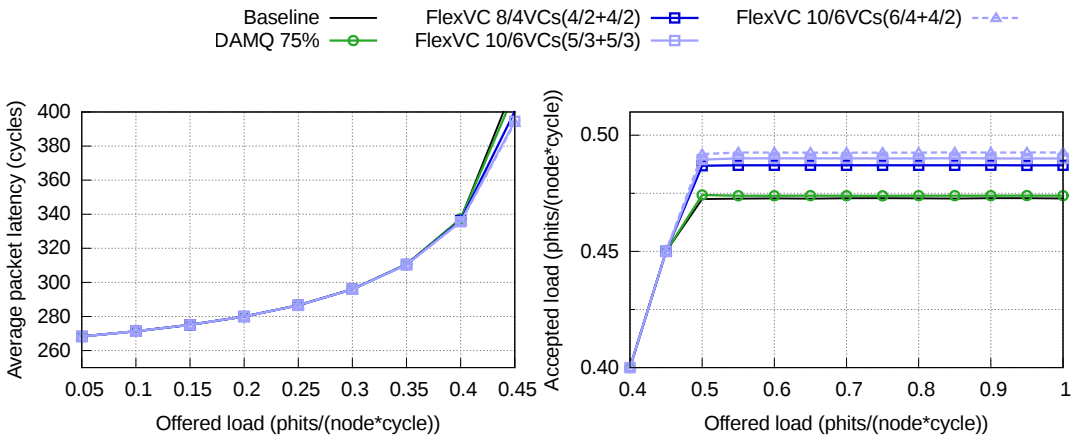
This section evaluates the performance of FlexVC when combined with source-adaptive routing, implemented following the description of PB in Section 1.4.2.1.



(a) UN traffic with MIN routing.



(b) BURSTY-UN traffic with MIN routing.



(c) ADV+1 traffic with VAL routing.

Figure 5.9: Latency and throughput under uniform (UN, BURSTY-UN) and adversarial (ADV+1) traffic patterns with oblivious routing, modeling request-reply dependencies.

5.3.3.1 Results with source-adaptive routing under traffic without dependencies

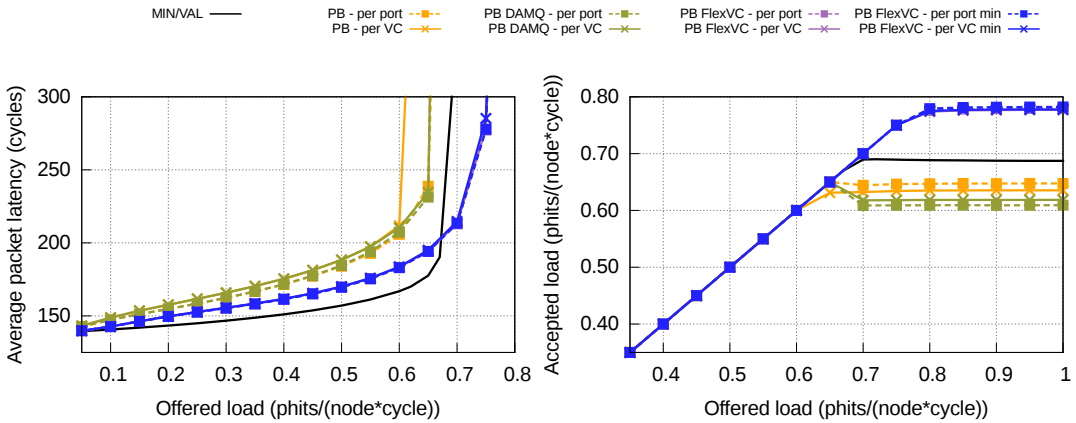
Figure 5.10 shows the results of source-adaptive routing under oblivious traffic workloads without request-reply dependencies, as in Figure 5.7. The two variants of buffer occupancy comparison described in Section 5.2.4, *-per-port* and *-per-VC*, are considered for the baseline and DAMQ implementations. All configurations require 4/2 VCs except the MIN oblivious reference under uniform patterns, which only uses 2/1 VCs.

FlexVC performs significantly better than the base and DAMQ implementations, with lower latency under uniform traffic and higher throughput in all cases; DAMQ only gives a small improvement over baseline under adversarial and bursty uniform traffic. The impact of the congestion-sensing implementation with source-adaptive routing and oblivious traffic workloads is almost negligible. In the baseline PB, *per-Port* sensing reduces the amount of misrouting performed and under UN traffic it gives closer throughput to the oblivious MIN reference, but requires a high offered load to properly detect adversarial traffic patterns. With DAMQ, the only difference occurs with UN traffic, where *per-VC* sensing indirectly achieves a better result as a consequence of not accounting for the uneven distribution of memory among VCs, enforcing less misrouting. FlexVC shows no difference in throughput whatsoever across its implementations, and only slightly lower latency when the distinction between minimal and nonminimal traffic is applied to the misrouting decision (*PB-minCred-per-VC* and *PB-minCred-per-port*). It outperforms by a wide margin both the base and DAMQ implementations of PB and also the oblivious routing reference under all traffic patterns, with the exception of marginally higher latency under intermediate traffic loads of uniform patterns.

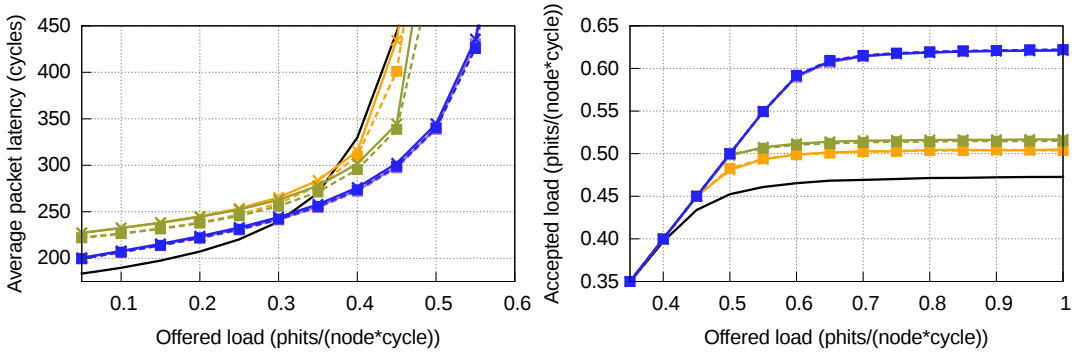
5.3.3.2 Results with source-adaptive routing under request-reply traffic

Figure 5.11 presents results of source-adaptive routing with request-reply traffic. Base configurations require 4/2+4/2=8/4 VCs, while FlexVC variants employ 6/3 VCs arranged as 4/2+2/1, according to the findings in the analysis of the behavior with oblivious routing. Note that the drop in performance under UN traffic (Figure 5.11a) when the network is saturated is less pronounced than with MIN routing.

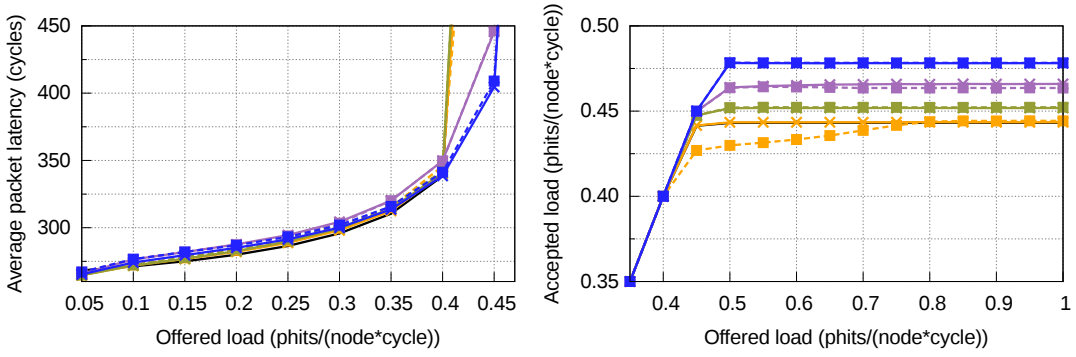
Figure 5.11c shows that, for the baseline PB, *per-port* sensing performs worse than *per-VC* sensing under ADV traffic. With the base VC management, *per-VC* sensing implicitly identifies the traffic pattern by analysing the amount of traffic routed minimally, as discussed in Section 5.2.4. The figure is truncated to distinguish better the performance of each sensing; the full Y-axis is shown in Figure 5.13b. The DAMQ implementation suffers a significant case of congestion under ADV that drops the throughput below 0.15 and 0.05 phits/(node·cycle) with *per-VC* and *per-port* sensing, respectively. Congestion appears regardless of the sensing employed, because it is due to an interference of the memory allocation with the misrouting decision in PB: if



(a) UN traffic.

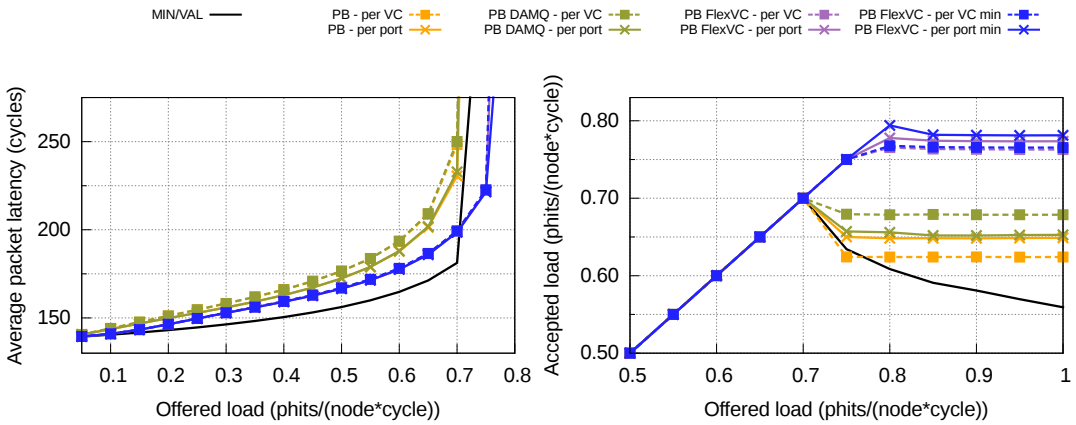


(b) BURSTY-UN traffic.

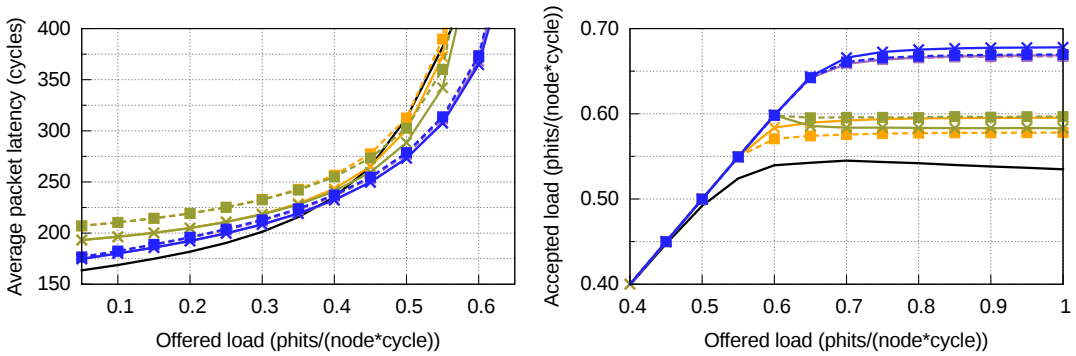


(c) ADV+1 traffic.

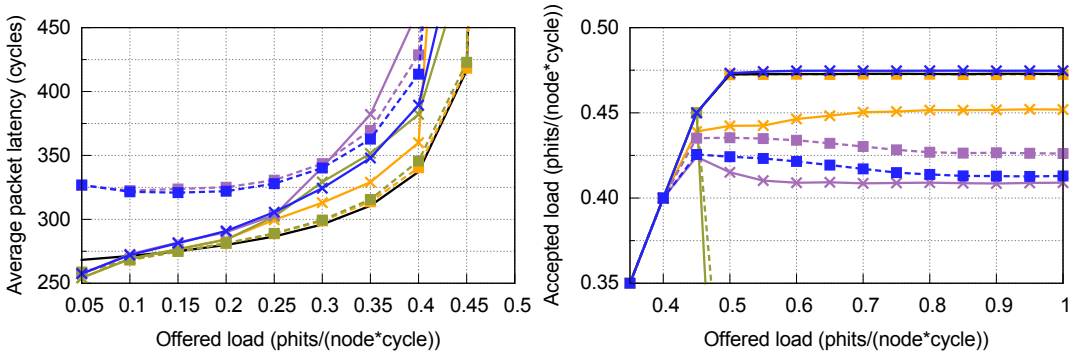
Figure 5.10: Latency and throughput under uniform (UN, BURSTY-UN) and adversarial (ADV+1) traffic patterns with source-adaptive routing. MIN is the oblivious routing reference for uniform traffic, and VAL for the adversarial pattern.



(a) UN traffic.



(b) BURSTY-UN traffic.



(c) ADV+1 traffic.

Figure 5.11: Latency and throughput under request-reply uniform (UN, BURSTY-UN) and adversarial (ADV+1) traffic patterns with source-adaptive routing. MIN and VAL are the oblivious routing reference for uniform and adversarial traffic, respectively. 4/2+4/2 VCs are used in baseline PB and VAL, 4/2+2/1 in FlexVC PB and 2/1+2/1 in MIN.

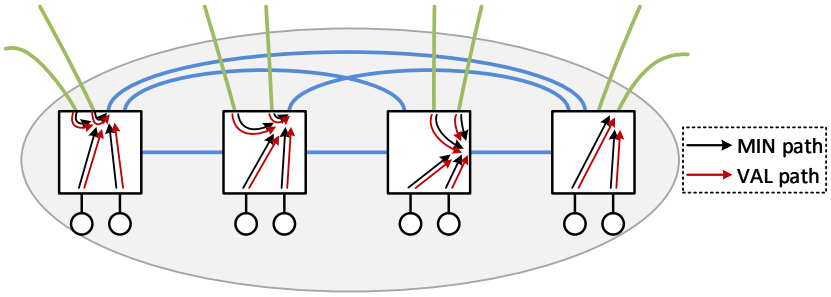


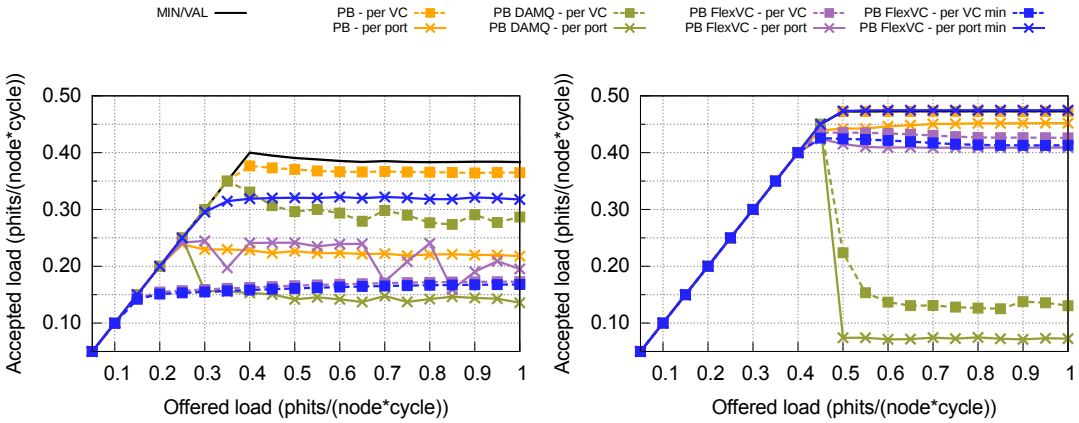
Figure 5.12: Sample group of a Dragonfly network, suffering a pathological case of congestion with oblivious/source-adaptive nonminimal routing under ADV traffic. Black lines represent MIN paths, red lines correspond to VAL paths. Note that the path overlap prevents any of the routes from being followed except at a very slow pace.

a given VC receives a larger segment of the shared buffer, a fair comparison between port/VC tuples cannot be performed.

Under uniform traffic, all four FlexVC variants clearly outperform the baseline PB, with a throughput increase of up to 20.4% in saturation under UN and overall reduced latency. However, under ADV traffic the two variants of FlexVC without differentiated credit tracking (*FlexVC-per-VC* and *FlexVC-per-port*) perform worse than the base VC management. FlexVC allows different flows to employ a common set of VCs, so *per-VC* sensing provides less accuracy and lower throughput even with separate credit tracking for minimally-routed packets. In such case, only *FlexVC-minCred* using *per-port* sensing is competitive with the baseline. The *FlexVC-minCred-per-port* variant is preferable over the baseline PB as it achieves better latency and significantly better throughput under uniform patterns, and competitive throughput and latency in ADV, while requiring 25% less buffers.

5.3.3.3 Behavior with source-adaptive routing

The implementation of source-adaptive Piggyback routing described in Section 1.4.2.1 does not recompute the intermediate Valiant node in case its associated path is not available. Since a Valiant node is only restricted to be located outside the destination group of the packet, the first global link in the nonminimal path can be connected to the same router as the global link in the MIN path. In such case, the local link at the source group is the same for both the minimal and nonminimal paths. Under ADV traffic, this implies that the nonminimal path traverses through the bottleneck router of the group. Similarly, misrouted packets may choose a Valiant node linked to the bottleneck router of their group, forcing incoming global traffic to compete with the injection ports for the already congested global link. When an adversarial request-reply traffic pattern is considered, the congestion due to this effect can also propagate from replies to requests. Since the nonminimal path cannot be recomputed at any point of the route, a pathological case of HoLB can appear where most of the packets at the injection and global buffers are stalled because the head-of-buffer packets contend for



(a) Without recomputing VAL router.

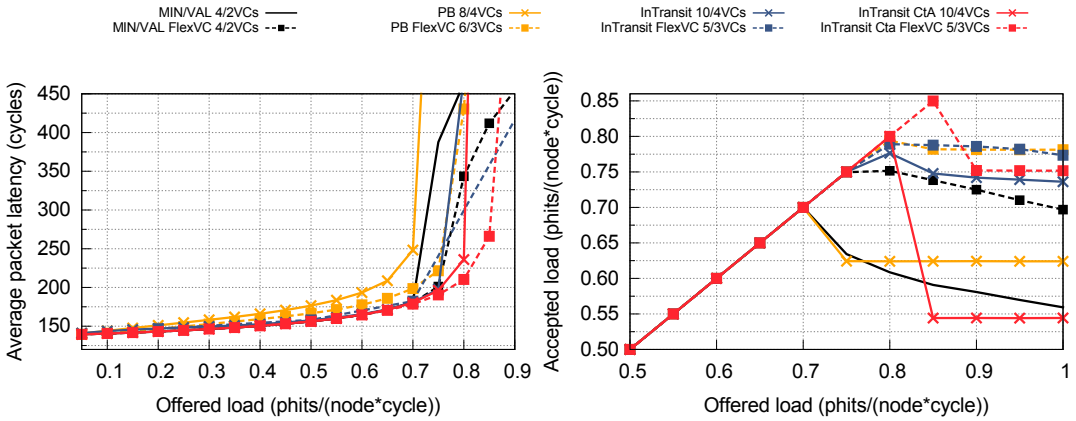
(b) Recomputing VAL router.

Figure 5.13: Throughput under request-reply ADV+1 traffic with VAL and source-adaptive routing, with/without recalculation of the VAL router.

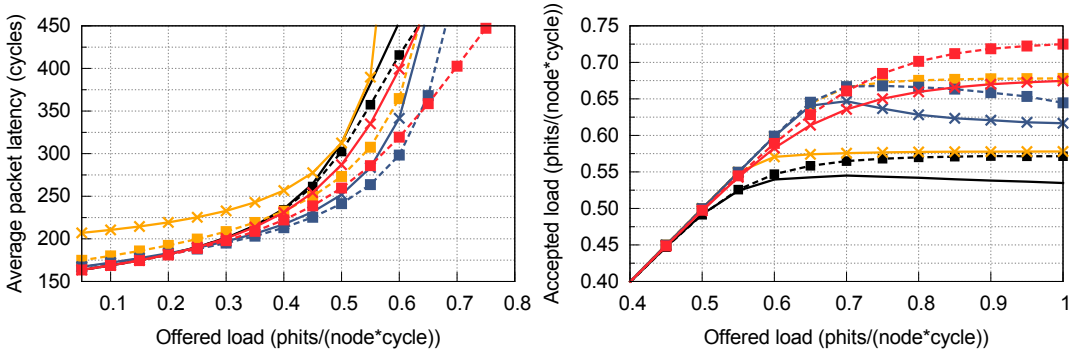
the same output port, and performance drops significantly. This situation can originate with the base VC management employed in previous chapters, but it exacerbates with FlexVC because minimally- and nonminimally-routed packets share the same buffers. Figure 5.12 shows this behavior in one group of the network.

To avoid this scenario, a different implementation of PB can be employed. One variant is to recompute the Valiant intermediate router for those packets at the head of the buffer that are not assigned an output in the allocation phase. Another implementation is to consider multiple Valiant destinations and choose the path with the least congested output port, as is done in Cray Cascade [116]. This exploits more efficiently the available resources and avoids stalls from HoLB in the majority of queues. This section employs the first solution, exploiting more efficiently the available resources and avoiding stalls from HoLB in the majority of queues. In order to perform a fair comparison, a similar variant of VAL routing where packets at injection can rerun the selection of the Valiant node is used as a reference. The previous non-reconfigurable implementation of VAL presented in Section 1.4.1.2 can also suffer from this pathological case of congestion, although it is less likely because all packets are diverted to the longer nonminimal route at injection.

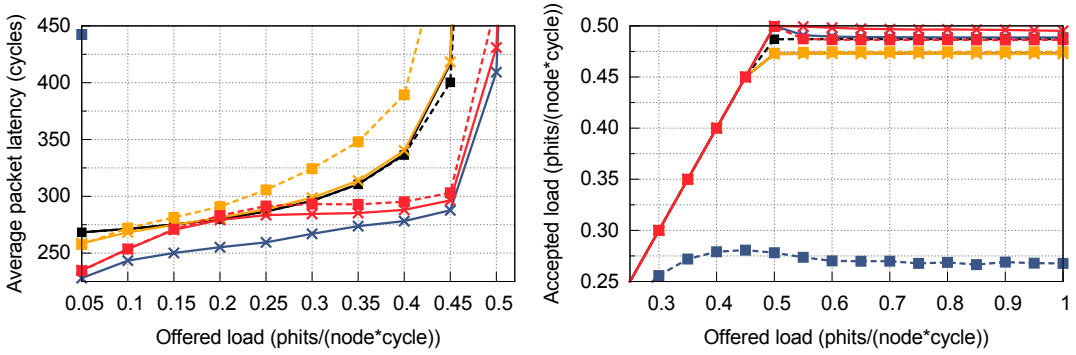
Figure 5.13 illustrates the severity of the problem. Some of the points in the curve for FlexVC with *per-port* sensing drop due to averaging at least one simulation with pathological HoLB; however, throughput for FlexVC is bad in general, with *FlexVC-minCred-per-port* achieving 15% less accepted load than PB without FlexVC. In contrast, all baseline and FlexVC implementations that recompute the Valiant router in Figure 5.13b achieve competitive throughput, with the best performers reaching the theoretical limit described in Section 1.4.1.2.



(a) UN traffic.



(b) BURSTY-UN traffic.



(c) ADV+1 traffic.

Figure 5.14: Latency and throughput under request-reply uniform (UN, BURSTY-UN) and adversarial (ADV+1) traffic patterns with in-transit adaptive routing. MIN and VAL are the oblivious routing reference for uniform and adversarial traffic, respectively.

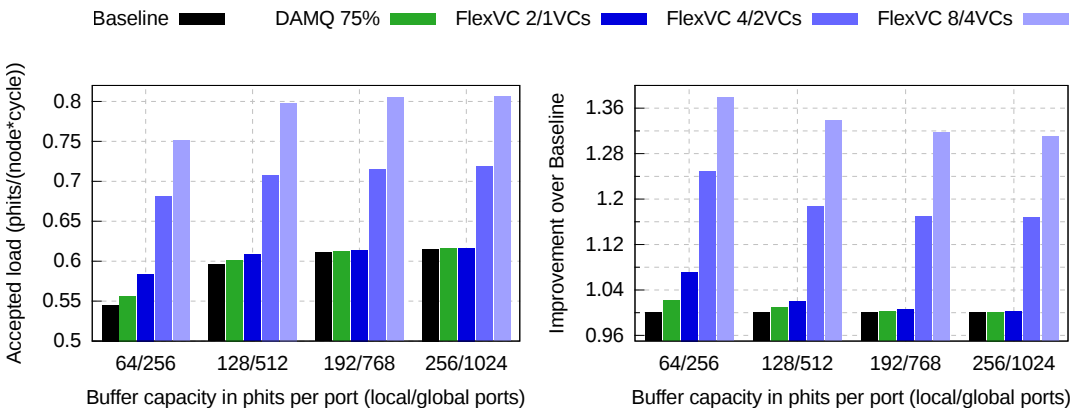
5.3.4 Results with in-transit adaptive routing

Figure 5.14 shows the performance results with in-transit adaptive routing under request-reply traffic. Results with oblivious and source-adaptive routing with base VC management and with FlexVC are also shown as a reference. In-transit adaptive routing follows the OLM implementation described in Section 1.4.2.2.1. In the case of adaptive routing, the best overall performing variants are chosen for each configuration: baseline with *perVC* sensing and *FlexVC-minCred-perPort*.

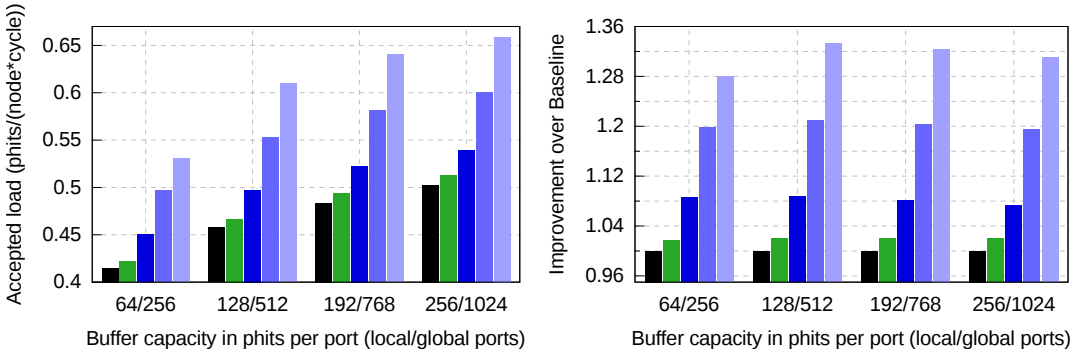
Performance with in-transit routing and FlexVC is competitive under uniform traffic patterns, but saturates at a very low load under ADV+1 traffic. Since FlexVC allows to reuse VCs in each hop, congestion sensing presents a similar limitation to detect adversarial patterns as observed with source-adaptive routing in Section 5.3.3. In contrast to source-adaptive routing, in-transit routing needs to trigger misrouting in-transit considering the availability of free slots in the next buffer, including traffic routed nonminimally. However, in-transit routing can be paired to the contention-based misrouting trigger described in Section 4.2.1.1 to decouple the misrouting decision from the buffer occupancy. Results with in-transit routing and contention-based misrouting decision, denoted as *In-transit Cta* in the figure, achieve competitive throughput and latency with FlexVC, even outperforming source-adaptive routing under BURSTY-UN and ADV+1 patterns. Furthermore, the use of FlexVC allows these results to be achieved with half the number of VCs needed for the base in-transit adaptive implementation, and fewer buffers per port than any of the source-adaptive implementations.

5.3.5 Simulation results without internal speedup

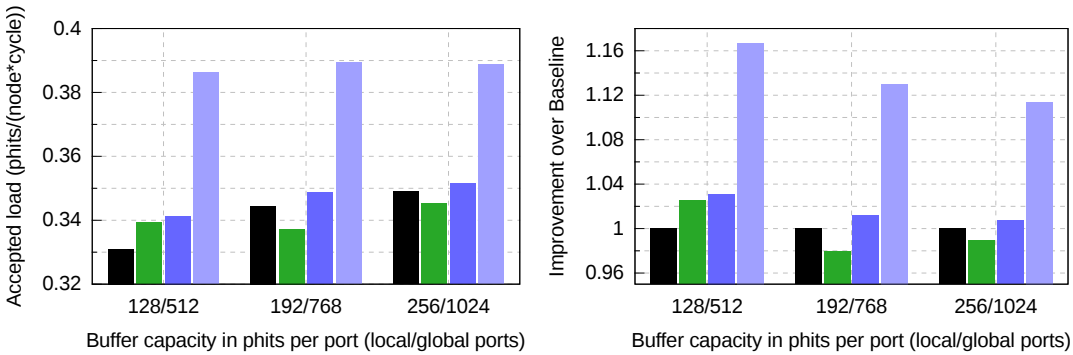
Results in Sections 5.3.2- 5.3.4 have been achieved simulating routers with internal speedup, as detailed in Table 1.2. Figure 5.15 reproduces the absolute and relative maximum throughput results from Figure 5.8 but removing the internal speedup, with network routers and their crossbars working at the same frequency as the network links. Base throughput is significantly lower due to HoLB, but the impact of FlexVC is higher, and it performs consistently better than DAMQ. FlexVC performs consistently better than DAMQ, which shows little benefit from the base case. Under ADV+1 traffic the impact of buffer organization is lower than under uniform patterns, and DAMQ is detrimental when larger buffer sizes are considered. Increasing the number of VCs magnifies the throughput improvement with FlexVC, although it is also beneficial from the baseline in all the evaluated cases. Relative improvement diminishes with deeper buffers, since the absolute throughput is larger and other factors aside HoLB limit the network performance. FlexVC allows a relative throughput improvement of up to 37% from the base case, with 8/4 VCs under UN traffic. Results with request-reply traffic show less congestion and better performance at maximum offered load under UN traffic, and are omitted for the sake of brevity. FlexVC still outperforms baseline and DAMQ-based versions with the minimal set of VCs, but the relative gains with a larger set of VCs are significantly higher.



(a) UN traffic with MIN routing.



(b) BURSTY-UN traffic with MIN routing.



(c) ADV+1 traffic with VAL routing.

Figure 5.15: Absolute and relative maximum throughput under uniform and adversarial traffic with oblivious routing without router speedup.

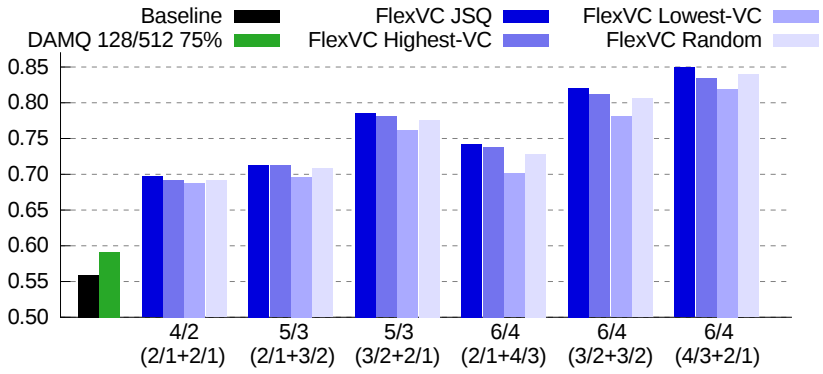


Figure 5.16: Throughput under UN request-reply traffic at 100% load, with multiple VC selection functions and amount of VCs. MIN routing.

5.3.6 Evaluation of the VC allocation policy

FlexVC requires a VC selection policy to establish which VC to employ from the range of allowed VCs for each hop, as described in Section 5.2.5. Results from previous sections employ a JSQ (Join the Shortest Queue) policy which chooses the queue with the largest number of available slots. Figure 5.16 presents the throughput under request-reply UN traffic with different VC allocation policies and different distributions of the set of VCs. Although the absolute values change from one case to another, the trend between the performance of each policy does not alter with the number and distribution of VCs.

For each configuration, JSQ provides the best performance on average, since it balances the utilization of all VCs. Interestingly, a Random policy is also competitive in all the configurations. Lowest-VC tends to saturate lower-index VCs that are more used in the first hops of requests, eventually restricting the injection; it consistently provides the lowest performance. A side effect of this restriction which cannot be observed in this figure is that Lowest-VC also presents lower peak throughput. In any case, the difference between policies under maximum load varies less than 3.4% in average.

5.4 Conclusions

This chapter has identified some of the main limitations of distance-based deadlock avoidance in low-diameter networks: HoLB, inefficient utilization of router buffers, and scaling restrictions with the longer paths associated to nonminimal adaptive routing and protocol-deadlock avoidance. Dynamically-allocated buffers such as DAMQs increase design complexity and introduce access delays, and only partially mitigate the previous limitations. Furthermore, they require statical partitioning for a significant amount of the memory to avoid congestion in small diameter networks as the Dragonfly. By contrast, the FlexVC mechanism relies on a simple design with stati-

cally partitioned buffers and on opportunistic routing, relaxing VC usage restrictions. FlexVC partially decouples the amount of VCs from the deadlock avoidance mechanism. It allows longer network paths with a low amount of resources, reducing memory up to a 50% (considering in-transit adaptive routing and protocol-deadlock avoidance).

A thorough evaluation in a Dragonfly network shows that FlexVC improves throughput up to 12% compared to a base oblivious case with the same buffering and number of VCs. It also reduces latency under traffic bursts at medium traffic loads below the saturation point. Furthermore, FlexVC leverages additional VCs, reaching 23% improvements in throughput with less buffering per VC but the same total amount of memory per port. FlexVC is even more efficient when the internal speedup of the router is removed, with an improvement of up to 37.8%, because the baseline performance degrades due to HoLB. Doubling the number of buffers (but keeping total buffering unaltered) allows FlexVC to match or outperform the behavior of the baseline with the internal router speedup. Increasing the amount of VCs also mitigates the congestion that appears with long oblivious paths used to avoid protocol deadlock. It can be observed that additional VCs are more useful when assigned to both subpaths for request and reply packets rather than reserving them for the replies, even when the number of VCs used solely for replies is lower than in the baseline mechanism.

However, reusing VCs for packets in different hops of their paths complicates the identification of adversarial traffic patterns for adaptive routing decisions. *FlexVC-minCred* handles credits separately for packets traveling minimally/nonminimally, properly identifying the communication pattern for adaptive routing. Combined with a *per-port* sensing strategy, it outperforms alternative designs and provides a 20.4% increase in throughput and noticeable latency reductions over the base PB implementation, while requiring 25% less buffers and total memory.

In-transit adaptive routing presents a similar problem to properly identify adversarial traffic when combined with FlexVC. Unfortunately, using separate credits for minimally routed packets does not offer acceptable performance in this case. However, the drop in performance is clearly associated to an inability to detect adversarial traffic from the congestion information; resorting to the contention-based misrouting decision proposed in the previous chapter allows to distinguish uniform from adversarial traffic workloads. In-transit adaptive routing with FlexVC and the *Base* contention counters implementation is competitive against the baseline and FlexVC variants of oblivious and source-adaptive routing, outperforming them in BURSTY-UN traffic and providing lower latency in all cases, while halving the number of buffers and the total memory per port required for in-transit routing.

Overall, FlexVC is a simple design that maximizes buffer utilization and outperforms state-of-the-art and more complex alternatives.

Chapter 6

Related Work

This chapter explores the existence of related work to the analysis and proposals of this thesis. It is organized in six blocks, following the order of the sections to which they are related. In Section 6.1 there is an analysis of BigData applications and simulation tools to evaluate the performance and behavior of the system interconnect. Section 6.2 reviews the different network topologies proposed for and implemented in different systems, as well as the routing mechanisms to forward the information between the source and destination nodes. Section 6.3 cites different mechanisms exploited to guarantee fairness in the network. In Section 6.4 discusses different congestion detection mechanisms and the use of contention metrics in previous works. Section 6.5 explores different deadlock avoidance mechanisms, and Section 6.6 describes different buffer implementations and the impact of the buffer size in other works.

6.1 Graph500 and simulation tools for network architects

Network simulators constitute a useful tool for network architects in the design and evaluation of new systems. Simulators are commonly based on full-system simulation, trace-driven execution or synthetic traffic patterns. Full system simulators such as Gem5 [25] present high computational and memory requirements; an alternative is to replace the application by a skeleton restricted to the most relevant sections for the network evaluation, as done in the SST simulator [126]. Such option is not feasible for the Graph500 benchmark because the core interest of the network simulation requires the graph traversal, and presents similar memory restrictions to the simulation of the whole application. Trace-driven simulators sometimes fail to accurately represent the dependencies in the execution.

Synthetic traffic models have smaller computational and memory requirements than both alternatives while retaining the core characteristics of the workload they represent. However, synthetic traffic models have traditionally consisted of permutations to determine the destination or set of destinations for the messages from a given node,

which isn't a fitting scheme for the behavior of BigData applications. SynFull [17] generates synthetic network traffic which preserves the characteristics (temporality, destinations, volume, etc) from executions of a real application. It relies on Markov Chains to model the behaviour of multiple phases of an application. However, it is focused on Networks-on-Chip only, modeling memory accesses and their associated coherence traffic, which is not appropriate for system-level interconnects.

BigData applications like MapReduce [47] or Spark [154] have become ubiquitous and gather the interest of system architects and designers. The Graph500 benchmark [2, 109] appeared in 2010 with the aim of influencing the design of new systems to match the needs of BigData applications. It is based on the execution of a Breadth-First Search (BFS), a strategy to explore a graph and organize its data into a tree. The benchmark consists of three kernels (generation of a small-world graph, BFS over the graph, and validation of the resulting tree from the BFS) of which only the BFS is accounted for the performance measurements. Graph500 ranks the execution of a machine through a metric called Traversed Edges Per Second (TEPS), calculated as the division of the number of edges traversed in the graph by the execution time of the BFS. An in-depth characterization of the benchmark and its communications can be found in the works of Suzumura *et al.* [136] and Anghel *et al.* [14, 15]. Beamer *et al.* characterize the memory requirements and locality of the benchmark in [19], but they do not study the impact of the network and its utilization.

The benchmark includes three implementations of the BFS algorithm, plus a template to define alternative versions of the algorithm. Several authors have proposed efficient and scalable shared memory implementations of BFS on commodity multicore processors. Agarwal *et al.* present in [9] a multi-core implementation with several optimizations. Beamer, Asanovic and Patterson develop in [18] a hybrid approach combining a conventional top-down algorithm along with a bottom-up one. The bottom-up algorithm reduces the number of edges visited and accelerates the search as a whole. This approach is advantageous for low-diameter graphs.

Alternative implementations have also been introduced to exploit architectures with hardware accelerators and custom hardware. Hong *et al.* [73] present a hybrid method which dynamically decides the best execution method for each BFS-level iteration, shifting between sequential execution, multi-core CPU-only execution, and GPUs. Tao, Yutong and Guang [139] develop two different approaches to improve the performance of BFS algorithm on an Intel Xeon Phi coprocessor. Checconi *et al.* [38] describe a family of highly-efficient Breadth-First Search (BFS) algorithms optimized for their execution on IBM Blue Gene/P and Blue Gene/Q supercomputers. Alternatively, Buluç and Madduri [31] conducted a performance evaluation of a distributed BFS using 1D and 2D partitioning on Cray XE6 and XT4 systems.

6.2 Network topologies and routing mechanisms

The design of large-radix routers has been studied in multiple works, such as [92, 11, 117]. Large-radix routers allow interconnection networks to scale to a large number of

nodes retaining full bisection bandwidth. They are assumed to optimally exploit the available pin bandwidth of current chips. Many highly-scalable low-diameter network topologies have been proposed based on high-radix routers. Some well-known examples are 2D or 3D Flattened-Butterflies (FB, [90, 10]), diameter-3 Dragonflies [91], and diameter-2 Slim Flies (SF, [24]), Orthogonal Fat Trees (OFT, [142]) and Projective Networks (PN, [32]). Low-diameter networks present low average distance between nodes and permit to reduce latency, cost and energy consumption, while achieving high scalability.

Minimal routing in these networks proves insufficient when adversarial traffic loads are considered, leading to high congestion and poor performance. Adversarial patterns of many of these networks are studied in [91, 84]. Valiant routing [143] avoids network hotspots by sending all packets minimally to a random intermediate router, and then minimally to destination. The impact of using an intermediate group in the Dragonfly, instead of an intermediate router, was evaluated in [122].

Adaptive routing has been used in multiple interconnection networks to exploit their path diversity [28, 27, 89, 130]. Nonminimal adaptive routing allows for the selection of minimal or nonminimal paths, in terms of their cost. Different variants of nonminimal adaptive routing have been proposed for multiple network topologies, such as Flattened Butterflies [90, 10] or Dragonflies [91, 82, 63]. This work only considers a single class of traffic that can be routed minimally or nonminimally depending only on the status of the network links. In a real system this would typically not happen because not all traffic can be sent adaptively: in Cascade [56] minimal routing is used for packets that need to preserve in-order delivery, and in PERCS [16] it is the programmer who decides whether MIN or VAL is employed.

Universal Globally-Adaptive Load-balanced routing (UGAL, [134]) selects at injection between a minimal and a Valiant path, based on their respective buffer occupancy. Buffer occupancy within the router is insufficient when congestion occurs in links not directly connected to the source router; this happens for example in Dragonflies, in which global inter-group links are more prone to congestion, but the global link to be used by a packet is often connected to a neighbor router in the source group. PiggyBack (PB, [91]) employs a variant of Explicit Congestion Notifications (ECN) to propagate the occupancy status of the queues within every group. In-transit adaptive routing mechanisms re-evaluate the routing decision in some hops of the path; Progressive Adaptive Routing (PAR, [82]) may switch from MIN to VAL after a minimal hop. Opportunistic Local Misrouting (OLM, [64]) can resort to VAL paths at injection, after a minimal local hop in the source group, and in the intermediate group.

6.3 Throughput unfairness

A previous analysis of the performance impact due to throughput unfairness and its mitigation through the global misrouting policy has been presented in [60].

End-to-end congestion control mechanisms such as TCP [13] typically also deal with fairness, in particular with an Additive-Increase, Multiplicative Decrease (AIMD)

policy. However, fairness is provided only between flows which compete for some given link in their paths. Adaptive nonminimal routing mechanisms typically employed in dragonflies are not suited for such congestion control policies, since packets from each flow follow different paths.

Explicit fairness mechanisms include age-based arbitration [5] and SAT [77]. Age-based arbitration employs a modified allocator which considers the age of the packets for arbitration. Tracking packet age is quite costly, and multiple implementations in the Network-on-Chip environment try to mimic its performance with lower cost [97, 98, 106]. SAT restricts injection when some nodes are starving and cannot inject at their desired rate. To do so, SAT relies on a circulating signal. When some node starves, it holds the SAT signal, what eventually slows down other nodes which are waiting for the periodic message. As far as the authors know, SAT has not been applied before in Dragonfly networks.

The ADVc traffic pattern described and used for the evaluation in Chapter 3 can correspond to a real case of execution where applications with uniform traffic loads are assigned a set of consecutive groups in a Dragonfly network. Some approaches to prevent this assignation have been discussed in Section 3.1.3.1, modifying the job scheduler [3] or increasing network wiring between every pair of groups (called *trunking*). Another option is to alter the topology itself to avoid ADVc traffic; arranging the global links randomly has been considered before in [33]. Such mechanism would however randomize output nodes for a given subset of the network, but would not guarantee the absence of ADVc or similar traffic patterns.

6.4 Congestion detection and contention counters

Nonminimal adaptive routing mechanisms usually employ a misrouting trigger (the mechanism used to select between minimal or nonminimal routing) that relies on a congestion detection scheme based on buffer occupancy [89, 90, 10, 91, 82, 63]. This presents a problem of oscillations that has been known for a long time [86, 148].

Congestion detection mechanisms in WAN and lossy networks have been typically indirect, based on collisions, packet drops or jitter [78, 30, 113]. Random Early Detection (RED, [58]) mechanisms analyze the buffer occupancy to determine the congestion status. When routers detect congestion, the sources can be notified indirectly (i.e., by dropping packets) or explicitly (ECN: Explicit Congestion Notification). ECN is used in many technologies, such as the FECN and BECN messages in Frame Relay, the EFCI bit in ATM cells, the ECN bits in IP [125], the Quantized Congestion Notification in Datacenter Ethernet (802.1Qau) [74] or the congestion control in Infiniband [66]. All of these mechanisms rely, ultimately, in the buffer occupancy to determine the congestion status.

Most congestion-control implementations react by throttling injection [96, 78]. Focusing on HPC and Datacenter networks, the Datacenter TCP protocol [12] uses the IP ECN bits to restrict the transmission window of the sources, relying on an estimation of the amount of congestion. There exist alternative mechanisms that use adaptive

routing to circumvent congested network areas. Such routing was proposed for lossless Datacenter Ethernet networks in [108], while PiggyBack and Credit Round-Trip Time (PB and CRT, [82]) behave as ECN mechanisms to support adaptive source routing in Dragonfly networks. Alternative mechanisms to cope with congestion such as RECN [54] alleviate the impact of congestion by using separate buffers for congested traffic, but require additional hardware in the router logic.

In-network congestion in Dragonflies is typically employed to drive adaptive routing [91, 82, 64]. Nonminimal routing is employed in such case to avoid congested areas, and throughput is typically reduced in half due to the use of Valiant routing. In [82], Jiang *et al.* propose two source-based adaptive routing mechanisms for Dragonflies in addition to the PB routing whose limitations have been analyzed in Chapter 3. These two proposals, Credit Round Trip (CRT) and Reservation (RES) routing, are outperformed by PB on steady-state latency evaluations and have not been considered in this work. End-point congestion, described at the beginning of Section 1.4.2.2, requires different handling. In [80] and [81] the authors propose several reservation mechanism for dragonflies, which avoid congestion by pre-reserving bandwidth for each flow. Alternative proposals include the use of dynamically allocated side-buffers in the network switches [54]. In [151], Won *et al.* propose the use of a *history window* to avoid the uncertainty in credit-based congestion detection with saturation in remote network routers.

Contention indicators have been employed to drive routing in alternative contexts. Elwhishi *et al.* introduce their use in the context of shared-medium mobile wireless networks [55]. In the context of mesh-based networks-on-chip, Regional Congestion Awareness (RCA) [67] explores the use of contention information for minimal adaptive routing. It shows that contention information can be effectively employed to select between different minimal paths. RCA relies on the evolution of crossbar demand (i.e. allocator requests) for the output ports, whereas the contention counters proposed in Chapter 4 track the minimal output port of each packet, regardless of its actual followed path. Although both mechanisms could be similar under uniform traffic, their behaviour could differ with adversarial traffic: crossbar demand could oscillate between alternative paths, whereas contention counters not. In the same context, Chang *et al.* [36] consider the rate of change in the buffer levels to predict congestion, what avoids uncertainty issues with small buffers. In the context of interconnection networks, Dynamic Routing Control [118] employs a set of counters to track the number of packets targeting a certain destination to detect hotspots in Omega networks. Based on these counters, it prioritizes traffic flows aiming non-hotspot destinations. However, it does not employ adaptive routing (Omega networks do not have path diversity) and the size of the counters is related to the depth of the buffers since all packets are counted. Finally, Contention-Based Congestion Management (CBCM, [87]) applies the use of contention information and remote notifications to detect and overcome endpoint congestion. CBCM is based on the concept of the Explicit Contention Notification (ECtN) mechanism introduced in this work in Section 4.2.1.4, and extends it to the detection of endpoint congestion.

6.5 Deadlock avoidance

In [52], Duato defines the existence of an acyclic channel dependency graph as a sufficient condition to guarantee deadlock-free networks employing adaptive routing and wormhole switching; furthermore, deadlock-freedom can be achieved as long as there exists a routing subfunction with a subset of the channels that is connected and complies the first condition. This theorem can be identically applied [53] to Virtual Cut-Through switching with lower restrictions, since the extended dependencies are simpler and restricted to buffers.

The most widespread mechanism proposed for deadlock avoidance in low-diameter networks relies on a fixed order in the use of virtual channels (VCs). Seminal works on distance-based deadlock avoidance in store-and-forward networks were introduced by Günther [69] and Gopal [65]. Several current systems employ such mechanisms (or a variation of them), such as IBM PERCS [16] or Cray Cascade [56], and have been extended to commodity InfiniBand [129]. In these proposals, the amount of required VCs increases with the maximum path length. Therefore, supporting nonminimal paths, in-transit adaptive routing, multiple QoS traffic classes and avoiding protocol deadlock significantly increases buffer requirements (e.g., the Dragonfly network in Cray Cascade requires 8 VCs to support nonminimal routing and avoid protocol deadlock [56]).

Opportunistic Local Misrouting (OLM, [64]) violates the base order of increasing VC index only for certain local hops in Dragonflies. However, compared to the FlexVC proposal in Chapter 5 it does not fully exploit the available VCs to reduce HoLB, does not consider protocol-deadlock and is not exploited to simplify buffer management.

Distance-based deadlock avoidance allows to deal separately with deadlock avoidance and routing. However, this mechanism is not supported in all network technologies. For example, Infiniband switches select the output VC (denoted Virtual Lane, VL) based on the input and output ports and the packet service level (which does not change during the path). For this reason, most routing mechanisms in Infiniband (such as LASH [135], SSSP [72] and DF-SSSP [50]) assign a single VC to a complete path from source to destination. These routing protocols typically calculate sets of paths with a reduced amount of cyclic dependencies, so that the VL assignment phase result fits in a low amount of VLs. NUE routing [49] provides better results by combining path computing and VL assignment in a single calculation, but still assigns VLs to complete paths. Schneider *et al* extend distance-based deadlock avoidance to Infiniband in [129], but still determine a single fixed output VC per packet.

Protocol deadlock can occur when the arrival of certain packets triggers the generation of another packet in response; a cyclic dependency between both traffic classes results in deadlock if they share the same network resources. The typical mechanism employed to avoid protocol deadlock in lossless networks relies on two virtual networks, one for requests and other for replies (e.g., as implemented in Alpha 21364), what doubles the buffering requirements. In [147], Wang *et al.* introduce a bubble-based deadlock avoidance protocol for on-chip networks which does not employ separate networks. Instead, their mechanism shares router buffers but employs a separate

bubble for each type of message. The FlexVC mechanism in Chapter 5 similarly avoids a strict separation in different virtual networks, but ensures that replies have exclusive buffers to avoid deadlock.

6.6 Buffer sizing and organization

Current ASICs are constrained to use limited buffers [130], making them one of the most critical areas in the router design. Shared buffer structures as the DAMQs [138] described in Chapter 1 try to improve the utilization of the available space and overcome some of the limitations of statically partitioned buffers, hence its widespread use (e.g. in the SCOC design [41] and the Tianhe-2 network switches [100]).

Statically partitioned FIFO buffers are typically implemented through circular buffers using SRAM [140]. DAMQs also rely on SRAM to store data, but need a control structure to share a single buffer across multiple queues. This can be done using linked lists [138], devoting some of the buffer space to store the pointers for the lists. The associated buffer overhead is small but not negligible: for a 4KB DAMQ with 8-byte phits (512 phits per DAMQ), pointers need to be 9 bits long and the overhead is roughly 576 bytes, a 14% increase. Choi *et al.*[40] minimize this overhead by considering per-packet pointers; for this case of 8-phit packets, the overhead shrinks to 1.6%, but the flexibility of the buffer for variable packet sizes is reduced.

Shared buffer organizations present also a penalty in access latency, due to the indirections required in the linked lists. The implementation in [59] adds three cycles to read or write access latency. Choi *et al.* measure in [40] slowdowns in packet access time ranging 59-77% for different DAMQ implementations. This work has not evaluated the impact of this penalty on the performance of the DAMQ configurations, constituting an optimistic scenario for shared buffer structures.

Alternative control structures have been employed for the implementation of shared buffers with multiple VCs per port, such as self-compacting buffers [115] or Fully-connected circular buffers [110]. Alternative designs allow for a variable number of VCs, what helps reduce HoLB particularly under adaptive routing [40, 111]. These dynamically allocated buffers present two main drawbacks: a more complex design that leads to increases in area, power and delay; and a pathological case of congestion when a single VC occupies the complete buffer space, as studied in Section 5.3.1. The second effect can be partially avoided in an implementation with reserved space per VC, such as those presented in [101, 156]; such approach has been employed in the Tianhe-2 network switch [100]. Alternative approaches suggest to extend flow control to detect such congestion and regulate buffer usage [21, 155], but increase buffer complexity even further.

This work has considered a small buffer size sufficient to cover the link round-trip latency and permit lossless flow control. However, larger buffers are typically implemented to properly deal with congestion bursts of traffic. Yébenes *et al.* [153] reduce HoLB in Dragonflies by implementing multiple buffers per VC. This multiplies the complexity and buffer requirements, whereas the FlexVC mechanism proposed in

Chapter 5 produces a similar outcome by only exploiting the buffers already available for longer paths.

Chapter 7

Conclusions and future work

This work has focused in VCT system networks not only considering the traditional HPC workloads, which are mainly constrained by latency, but also BigData applications. To evaluate the performance with the latter, a synthetic traffic model of the communications in the Graph500 benchmark has been developed. This traffic model has allowed to observe that BigData workloads are more constrained by the network throughput than the average latency. Consequently, the maximum throughput in a Dragonfly network has been analyzed, considering the throughput fairness of the system and identifying pathological starvation effects. Implicit and explicit fairness mechanisms have been exploited to mitigate these starvation effects.

Regarding network design, two mechanisms have been proposed, the use of contention counters and FlexVC. In these kind of networks with nonminimal adaptive routing, they allow to diminish the size of the buffers, decoupling them from the routing decision and using them more efficiently. Relaxing the buffer restrictions results in lower implementation costs and provides better performance.

7.1 Conclusions

This work makes an extensive analysis of high radix-based, low-diameter networks aimed for Exascale systems. It starts by providing a synthetic traffic model of the communications of the Graph500 benchmark. This benchmark reproduces the behavior of BigData applications, which is more constrained by the memory and the network than traditional HPC workloads. The model introduced in this work permits to evaluate the impact of different network designs under a network-intensive workload without resorting to more time-consuming and non-scalable alternatives such as whole system or trace-driven simulations. As far as this author knows, it is the first synthetic traffic model of these characteristics to be proposed. To develop the model, an in-depth analysis of the benchmark communications has been performed. The analysis observes the asynchronous nature and spatial uniformity of the messages, and evaluates the impact of message aggregation on the overall performance. Communications have a staged structure with almost temporal message uniformity within each stage. The model pre-

dicts the number of messages for each stage and network node, receiving only the size of the graph and an estimation of the computing capabilities of the network nodes.

Results from the execution of the traffic model in a Dragonfly network evince a range of situations in which the interconnection is the performance bottleneck in the benchmark execution. The impact of the network is tightly linked to the routing mechanism and the number and distribution of processes assigned to the model execution in the simulator. Although as a whole the workload resembles a random uniform pattern, the execution over a subset of network nodes steers the behavior towards adversarial traffic scenarios.

One of the evaluated scenarios presents performance degradation due to throughput unfairness. Throughput unfairness occurs when the network nodes are assigned different ratios of resources, even reaching node starvation. To analyze the fairness of the network, a novel adversarial-consecutive (ADVc) pattern is used. ADVc induces throughput unfairness in a Dragonfly network when combined with an adaptive routing mechanism. The analysis evaluates the impact of in-transit traffic over injection priority, the global misrouting policy for adaptive routing, and the arbitration policy.

In-transit adaptive routing with the MM policy achieves the best overall performance. However, under ADVc traffic the in-transit traffic priority prevents the injection from nodes attached to the bottleneck router which connects to the destination groups, leading to a pathological case of starvation. The PB implementation used for source-adaptive routing presents an inherent incapability to detect the ADVc pattern as adversarial at injection. Removing the priority dismisses the implicit throttle injection and reduces performance even further. Furthermore, priority removal is insufficient to fully avoid unfairness under ADVc traffic. An explicit fairness mechanism such as the use of age-based arbitration achieves good performance and complete fairness with in-transit adaptive routing, at the expense of a more complex arbitration policy that leads to higher implementation costs. The rest of the work adheres to the simpler round-robin policy for the crossbar allocation without prioritizing in-transit traffic over new injections.

One of the sources of the unfairness in said scenarios is a tight dependence of the misrouting decision with the buffer size and its utilization. The misrouting decision, which selects between a minimal and a nonminimal path in adaptive routing mechanisms, typically relies in a congestion metric such as the number of available slots in the neighbor routers. Congestion-based misrouting decisions suffer from a series of shortcomings, such as a strong dependency on the buffer size, oscillations in routing, and a slow adaption to changes in the traffic pattern. The dependency on the buffer size translates into deeper buffers needed to achieve fine granularity in the congestion detection. However, greater buffer sizes slow the adaption to traffic changes, incurring in a latency penalty to fill up or empty the buffer past the threshold to update the misrouting decision. Additionally, using nonminimal paths allows the buffers in the minimal route to drain, introducing a loop in the misrouting decision which provokes misrouting oscillations.

Using a contention-based misrouting decision avoids these limitations and detects

the cause of the congestion (contention for the same resources) rather than its outcome (full buffers). The use of a contention metric is proposed, with a counter per output port tracking the demand from head-of-buffer packets at the input ports. These counters only consider the output in the minimal path of a packet, in order to identify the communication pattern. Four different implementations with in-transit adaptive routing are considered; all have a fast adaption to traffic changes, avoid routing oscillations and are competitive in performance. Overall, the *hybrid* and *ECtN* variants are the most enticing because they outperform the credit-based in-transit adaptive routing mechanism while averting its drawbacks, although they entail a higher implementation cost: *hybrid* combines the contention statistics with credit-based congestion information, and *ECtN* broadcasts contention statistics for the global links within each group.

In low-diameter networks deadlock is typically prevented through distance-based deadlock avoidance, relying on the use of virtual channels (VCs) with a separate buffer for each VC at every input port, and following an ever-increasing VC index for every hop of a packet path. Distance-based deadlock avoidance mechanisms prevent an efficient utilization of the buffers and incur in Head-of-Line Blocking (HoLB). Moreover, they couple the number of required buffers per port with the length of the longest non-minimal path, hindering the scalability of the network. The share of memory across the VCs within the port (e.g., through the use of a Dynamically Allocated Multi-Queue, DAMQ) theoretically overcomes those limitations, in exchange for higher implementation costs and access delays. However, they require a significant amount of memory to be assigned statically in small diameter networks, partially negating their benefits.

This work introduces a novel VC management called FlexVC, which relaxes VC usage restrictions and relies only on statically partitioned buffers and opportunistic routing. The core idea behind FlexVC is that for a path to be deadlock free it is sufficient that an increasing VC-index sequence towards the destination exists, *although it does not need to be followed*. Instead, lower VC indices can be used for every hop, and additional VCs can be used to further mitigate HoLB and increase performance. FlexVC is particularly useful when deadlock avoidance is accounted for under traffic with request-reply dependencies, since it can reuse the VCs from the petitions for the requests. FlexVC achieves significant improvements in throughput with oblivious routing, without suffering latency penalties, particularly with request-reply traffic. Results evidence that it is more useful to perform opportunistic misrouting and assign additional VCs to both request and reply subpaths rather than reserving them to ensure longer safe paths. VC reuse complicates the implicit identification of adversarial traffic that the baseline source-adaptive routing performs. To regain this capability, an additional set of counters can be used to account separately those packets that are traveling minimally. When combined with a *per-port* sensing strategy, it allows FlexVC to outperform the base source-adaptive mechanism while demanding 25% less buffer resources. *Per-port* sensing is insufficient to achieve good performance with FlexVC combined with in-transit adaptive routing. Resorting to a contention-based misrouting decision like the use of contention counters permits to FlexVC to be competitive with in-transit adaptive routing while halving the number of buffers needed at each port.

7.2 Future work

Future lines of research contemplate the use of FlexVC and contention counters in other topologies, where the benefits of a more flexible use of the VCs can be equally useful. Furthermore, in networks without link restrictions the potential savings in the number of resources required can be higher, because the VCs are not grouped into disjoint sets.

Other topics that will be pursued are the application of contention-based misrouting decisions to employ speculative flow control [107] with adaptive routing. The use of speculative flow control allows to diminish the size of the buffers, relying on the retransmission of discarded packets; however, less shallow buffers are detrimental for a credit-based misrouting decision. The metric based on contention counters that has been proposed in this thesis overcomes that limitation and could exploit speculative flow control, reducing the router area devoted to buffers.

The base implementation of contention counters can also be combined with injection throttling to prevent endpoint congestion. In a case of endpoint congestion, exploiting path diversity through adaptive routing only spreads the congestion to the network and can reduce the performance. The use of contention counters has opened a line of researched pursued by Kim *et al.* in [87]; our mechanism could simplify their proposal while still being able to prevent endpoint congestion from saturating the network links.

It is also intended to evaluate the performance of other network topologies proposed for Exascale system interconnects, such as the SlimFly [24] or Projective Networks [32], through the synthetic traffic model of the Graph500 benchmark. In the long term, the impact of other fairness mechanisms in Dragonfly networks may also be explored.

7.3 Publications

During the development of this thesis a significant collaboration has been established with the IBM Zurich Resarch Laboratory in Rüschlikon, Switzerland. This collaboration has allowed the author to perform two internships, from September 6th 2012 until July 31st 2013 (11 months), and from September 1st 2015 until November 30th 2015 (3 months).

The research included in this thesis has lead to a number of publications. A preliminary version of the Graph500 synthetic traffic model has been published in [3] and is the first model for Graph500 traffic as far as we know. An extended version has been submitted to a special issue of the Journal of Concurrency and Computation: Practice and Experience [1]. The analysis of the benchmark communications conducted prior to the model development was first presented in [7], where the impact of message aggregation was first brought to light.

An earlier analysis of the unfairness issues in Dragonflies discussed in Chapter 3 has been presented in [5], without evaluating any effective solution to the causes of the

unfairness problem. A more compact version of the evaluation in this work has been published in [4].

The core idea behind the use of contention counters and the variants described in Chapter 4 (minus the filtered implementation) have been introduced in [6]; an early evaluation was presented in [8]. A paper describing the FlexVC mechanism and evaluating its performance with oblivious and PB routing has been accepted for publication in [2].

[1] P. Fuentes, M. Benito, E. Vallejo, J. L. Bosque, R. Beivide, A. Anghel, G. Rodríguez, M. Gusat, C. Minkenberg, and M. Valero, "A scalable synthetic traffic model of Graph500 for computer networks analysis," Submitted for publication in *Concurrency and Computation: Practice and Experience (CCPE)*, 2017.

[2] P. Fuentes, E. Vallejo, R. Beivide, C. Minkenberg, and M. Valero, "FlexVC: Flexible virtual channel management in low-diameter networks," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017.

[3] P. Fuentes, E. Vallejo, J. L. Bosque, R. Beivide, A. Anghel, G. Rodríguez, M. Gusat, and C. Minkenberg, "Synthetic traffic model of the Graph500 communications," in *Proceedings of the 16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2016.

[4] P. Fuentes, E. Vallejo, C. Camarero, R. Beivide, and M. Valero, "Network unfairness in Dragonfly topologies," *The Journal of Supercomputing*, pp. 1–29, 2016.

[5] ——, "Throughput unfairness in Dragonfly networks under realistic traffic patterns," in *1st IEEE International Workshop on High-Performance Interconnection Networks Towards the Exascale and Big-Data Era (HiPINEB)*, Sept 2015, pp. 801–808.

[6] P. Fuentes, E. Vallejo, M. García, R. Beivide, G. Rodríguez, C. Minkenberg, and M. Valero, "Contention-based nonminimal adaptive routing in high-radix networks," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 103–112.

[7] P. Fuentes, J. L. Bosque, R. Beivide, M. Valero, and C. Minkenberg, "Characterizing the communication demands of the Graph500 benchmark on a commodity cluster," in *Proceedings of the 2014 IEEE/ACM Int. Symposium on Big Data Computing*, pp. 83–89.

[8] P. Fuentes, E. Vallejo, M. Garcia, and R. Beivide, "On the use of contention information for adaptive routing," *Tenth International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES 2014)*, vol. 0, pp. 243–246, 2014.

Out of the scope of this work, but performed in the same interval, is the collaboration in a proposal for an optical switch for data-intensive computing [10], a comparison of interconnection networks for HPC [11] and two evaluations of different mechanisms for better teaching activities, one submitted for publication [9] and another published in [12].

[9] E. Vallejo, P. Fuentes, and M. Benito, "Aprendizaje autónomo del estudiante apoyado en recursos audiovisuales en el contexto de un Grado de Ingeniería Informática: experiencias con metodologías de enseñanza activas," Submitted for publication in *Congreso Nacional de Innovación Educativa y de Docencia en Red (IN-RED)*, 2017.

[10] L. Schares, B. G. Lee, F. Checconi, R. Budd, A. Rylyakov, N. Dupuis, F. Petrini, C. L. Schow, P. Fuentes, O. Mattes, and C. Minkenberg, "A throughput-optimized optical network for data-intensive computing," *IEEE Micro*, vol. 34, no. 5, pp. 52–63, Sept 2014.

[11] P. Fuentes, E. Vallejo, C. Martinez, M. Garcia, and R. Beivide, "Comparison study of scalable and cost-effective interconnection networks for HPC," *2012 41st International Conference on Parallel Processing Workshops*, vol. 0, pp. 594–595, 2012.

[12] P. Fuentes, C. Martınez, E. Vallejo, E. Stafford, and J. L. Bosque, "Plataforma web para retroalimentación automática en la docencia de ensamblador," in *XXIII Jornadas de Paralelismo (JP2012)*, 2012.

Bibliography

[1] "Sun datacenter switch 3456 architecture white paper," Nobember 2007.

[2] "Graph500 benchmark," May 2016. [Online]. Available: http://www.graph500.org/

[3] A. H. Abdel-Gawad, M. Thottethodi, and A. Bhatele, "RAHTM: Routing algorithm aware hierarchical task mapping," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 325–335. [Online]. Available: https://doi.org/10.1109/SC.2014.32

[4] F. Abel, C. Minkenberg, R. P. Luijten, M. Gusat, and I. Iliadis, "A four-terabit packet switch supporting long round-trip times," *IEEE Micro*, vol. 23, no. 1, pp. 10–24, Jan 2003.

[5] D. Abts and D. Weisser, "Age-based packet arbitration in large-radix k-ary n-cubes," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, Nov 2007, pp. 1–11.

[6] D. Abts, "Cray XT4 and Seastar 3-D torus interconnect," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 470–477.

[7] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue Gene/L torus interconnection network," *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 265 –276, march 2005.

[8] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 33–42.

[9] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader, "Scalable graph exploration on multicore processors," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.46

[10] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "HyperX: Topology, routing, and packaging of efficient large-scale networks," in *SC '09: Conf. on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 41:1–41:11.

[11] J. H. Ahn, Y. H. Son, and J. Kim, "Scalable high-radix router microarchitecture using a network switch organization," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 3, pp. 17:1–17:25, Sep. 2008.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM Conference*, 2010, pp. 63–74.

[13] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, RFC 5681, Internet Engineering Task Force Std., September 2009.

[14] A. Anghel, G. Rodríguez, and B. Prisacari, "The importance and characteristics of communication in high performance data analytics," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 80–81.

[15] A. Anghel, G. Rodríguez, B. Prisacari, C. Minkenberg, and G. Dittmann, "Quantifying communication in graph analytics," in *High Performance Computing*. Springer, 2015, pp. 472–487.

[16] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li *et al.*, "The PERCS high-performance interconnect," in *18th Symposium on High Performance Interconnects*. IEEE, 2010, pp. 75–82.

[17] M. Badr and N. E. Jerger, "SynFull: Synthetic traffic models capturing cache coherent behaviour," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, June 2014, pp. 109–120.

[18] S. Beamer, K. Asanovic, and D. Patterson, "Direction-optimizing breadth-first search," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, ser. SC '12, IEEE. IEEE Computer Society Press, 2012, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389013

[19] ——, "Locality exists in graph processing: Workload characterization on an Ivy Bridge server," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*, Oct 2015, pp. 56–65.

[20] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–12.

[21] D. U. Becker, "Adaptive backpressure: Efficient buffer management for on-chip networks," in *Intl Conf. on Computer Design*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 419–426.

[22] M. Benito, E. Vallejo, and R. Beivide, "On the use of commodity Ethernet technology in exascale HPC systems," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, Dec 2015, pp. 254–263.

[23] M. Benito, E. Vallejo, R. Beivide, and C. Izu, "Extending commodity OpenFlow switches for large-scale HPC deployments," in *3rd IEEE International Workshop on High-Performance Interconnection Networks Towards the Exascale and Big-Data Era (HiPINEB)*, 2017.

[24] M. Besta and T. Hoefler, "Slim Fly: A cost effective low-diameter network topology," in *Int. Conf. High Performance Computing, Networking, Storage and Analysis*. Piscataway, NJ, USA: IEEE Press, 2014, pp. 348–359.

[25] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[26] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel®Omni-Path architecture: Enabling scalable, high performance fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, Aug 2015, pp. 1–9.

[27] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, and P. Vranas, "Design and analysis of the BlueGene/L torus interconnection network," *IBM Research Report RC23025 (W0312-022)*, vol. 3, 2003.

[28] R. V. Boppana and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," in *ISCA '93: 20th International Symposium on Computer Architecture*, 1993, pp. 351–360.

[29] S. Bradner and J. McQuaid, *Request for Comments 2544: Benchmarking Methodology for Network Interconnect Devices*, Internet Engineering Task Force, Network Working Group Std., 1999.

[30] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *SIGCOMM '94: Conf. on Communications architectures, protocols and applications*. ACM, 1994, pp. 24–35.

[31] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceedings of 2011 International Conference for High*

Performance Computing, Networking, Storage and Analysis. ACM, 2011, p. 65. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063471

[32] C. Camarero, C. Martínez, E. Vallejo, and R. Beivide, "Projective networks: Topologies for large parallel computer systems," *IEEE Transactions on Parallel and Distributed Systems (To appear)*, 2016.

[33] C. Camarero, E. Vallejo, and R. Beivide, "Topological characterization of Hamming and Dragonfly networks and its implications on routing," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 39:1–39:25, 2014.

[34] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 442–446.

[35] J. Chambers and T. Hastie, *Statistical Models in S*. Wadsworth & Brooks/Cole, 1992.

[36] E.-J. Chang, H.-K. Hsin, S.-Y. Lin, and A.-Y. Wu, "Path-congestion-aware adaptive routing with a contention prediction scheme for network-on-chip systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 1, pp. 113–126, 2014.

[37] K.-Y. K. Chang, S.-T. Chuang, N. McKeown, and M. Horowitz, "A 50 gb/s 32/spl times/32 CMOS crossbar chip using asymmetric serial links," in *1999 Symposium on VLSI Circuits. Digest of Papers (IEEE Cat. No.99CH36326)*, June 1999, pp. 19–22.

[38] F. Checconi, F. Petrini, J. Willcock, A. Lumsdaine, A. R. Choudhury, and Y. Sabharwal, "Breaking the speed and scalability barriers for graph exploration on distributed-memory machines," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE Computer Society Press, 2012, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389014

[39] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The IBM Blue Gene/Q interconnection network and message unit," in *SC: Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–10.

[40] Y. Choi and T. M. Pinkston, "Evaluation of queue designs for true fully adaptive routers," *J. Parallel Distrib. Comput.*, vol. 64, no. 5, pp. 606–616, May 2004.

[41] N. Chrysos, C. Minkenberg, M. Rudquist, C. Basso, and B. Vanderpool, "SCOC: High-radix switches made of bufferless Clos networks," in *Intl Symp. on High Performance Computer Architecture*, 2015, pp. 402–414.

[42] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953. [Online]. Available: http://dx.doi.org/10.1002/j.1538-7305.1953.tb01433.x

[43] W. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, Jun 1990.

[44] ——, "Virtual-channel flow control," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, no. 2, pp. 194 –205, mar 1992.

[45] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. 100, no. 5, pp. 547–553, 1987.

[46] W. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[47] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[48] S. Derradji, T. Palfer-Sollier, J. P. Panziera, A. Poudes, and F. W. Atos, "The BXI interconnect architecture," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, Aug 2015, pp. 18–25.

[49] J. Domke, T. Hoefler, and S. Matsuoka, "Routing on the dependency graph: A new approach to deadlock-free high-performance routing," in *Symposium on High-Performance Parallel and Distributed Computing (HPDC'16)*, Jun. 2016.

[50] J. Domke, T. Hoefler, and W. E. Nagel, "Deadlock-free oblivious routing for arbitrary topologies," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, May 2011, pp. 616–627.

[51] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. [Online]. Available: http://dx.doi.org/10.1002/cpe.728

[52] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 4, no. 12, pp. 1320 –1331, dec 1993.

[53] ——, "A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 841–854, Aug 1996.

[54] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *HPCA-11: Intl. Symp. on High-Performance Computer Architecture.*, 2005, pp. 108–119.

[55] A. Elwhishi, P.-H. Ho, K. Naik, and B. Shihada, "Self-adaptive contention aware routing protocol for intermittently connected mobile networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 7, pp. 1422–1435, Jul. 2013.

[56] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: a scalable HPC system based on a Dragonfly network," in *SC: Intl Conf on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 103:1–103:9.

[57] J. Flich and D. Bertozzi, *Designing network on-chip architectures in the nanoscale era*. Chapman and Hall/CRC, 2010.

[58] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[59] G. L. Frazier and Y. Tamir, "The design and implementation of a multiqueue buffer for VLSI communication switches," in *Intl Conference on Computer Design*, Oct 1989, pp. 466–471.

[60] M. García, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, J. Labarta, and G. Rodríguez, "Global misrouting policies in two-level hierarchical networks," in *INA-OCMC: Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*, 2013, pp. 13–16.

[61] M. García, P. Fuentes, M. Odriozola, E. Vallejo, and R. Beivide. (2014) FOGSim interconnection network simulator. University of Cantabria. [Online]. Available: http://fuentesp.github.io/fogsim/

[62] M. García, E. Vallejo, R. Beivide, C. Camarero, M. Valero, G. Rodríguez, and C. Minkenberg, "On-the-fly adaptive routing for Dragonfly interconnection networks," *The Journal of Supercomputing*, vol. 71, no. 3, pp. 1116–1142, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11227-014-1357-9

[63] M. García, E. Vallejo, R. Beivide, M. Odriozola, C. Camarero, M. Valero, G. Rodríguez, J. Labarta, and C. Minkenberg, "On-the-fly adaptive routing in high-radix hierarchical networks," in *The 41st International Conference on Parallel Processing (ICPP)*, 09 2012.

[64] M. García, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient routing mechanisms for Dragonfly networks," in *The 42nd International Conference on Parallel Processing (ICPP)*, 10 2013.

[65] I. Gopal, "Interconnection networks for high-performance parallel computers," I. D. Scherson and A. S. Youssef, Eds. IEEE Computer Society Press, 1994, ch. Prevention of Store-and-forward Deadlock in Computer Networks, pp. 338–344.

[66] E. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *IEEE Intl. Symp. on Parallel Distributed Processing*, 2010.

[67] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *HPCA'08: IEEE 14th Intl. Symp. on High Performance Computer Architecture.*, 2008, pp. 203–214.

[68] C. Groër, B. D. Sullivan, and S. Poole, "A mathematical analysis of the R-MAT random graph generator," *Networks*, vol. 58, no. 3, pp. 159–170, 2011.

[69] K. Günther, "Prevention of deadlocks in packet-switched data transport systems," *Communications, IEEE Transactions on*, vol. 29, no. 4, pp. 512 – 524, apr 1981.

[70] M. G. Hluchyj and M. J. Karol, "Queueing in space-division packet switching," in *IEEE Infocom*, vol. 88, 1988, pp. 334–343.

[71] Y. Ho Song and T. M. Pinkston, "A progressive approach to handling message-dependent deadlock in parallel computer systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 3, pp. 259–275, Mar. 2003.

[72] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized routing for large-scale InfiniBand networks," in *2009 17th IEEE Symposium on High Performance Interconnects*, Aug 2009, pp. 103–111.

[73] S. Hong, T. Oguntebi, and K. Olukotun, "Efficient parallel graph exploration on multi-core CPU and GPU," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*. IEEE Computer Society, 2011, pp. 78–88, doi:10.1109/PACT.2011.14. [Online]. Available: http://dx.doi.org/10.1109/PACT.2011.14

[74] *"IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification," 802.1Qau*, IEEE Std., April 2010.

[75] I. Iliadis and W. E. Denzel, "Performance of packet switches with input and output queueing," in *IEEE International Conference on Communications, Including Supercomm Technical Sessions*, Apr 1990, pp. 747–753 vol.2.

[76] *Infiniband architecture specification*, InfiniBand Trade Association Std., November 2007.

[77] C. Izu and E. Vallejo, "Throughput fairness in indirect interconnection networks," in *13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, ser. PDCAT '12. IEEE Computer Society, 2012, pp. 233–238.

[78] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM '88: Communications architectures and protocols*. ACM, 1988, pp. 314–329.

[79] N. Jiang, J. Balfour, D. Becker, B. Towles, W. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 86–96.

[80] N. Jiang, D. Becker, G. Michelogiannakis, and W. Dally, "Network congestion avoidance through speculative reservation," in *High Performance Computer Architecture (HPCA), IEEE 18th International Symposium on*, Feb 2012, pp. 1–12.

[81] N. Jiang, L. Dennison, and W. Dally, "Network endpoint congestion control for fine-grained communication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. ACM, 2015, pp. 35:1–35:12. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807600

[82] N. Jiang, J. Kim, and W. Dally, "Indirect adaptive routing on large scale interconnection networks," in *Intl. Symp. on Computer Architecture (ISCA)*, 2009, pp. 220–231.

[83] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (CICQ) switches," in *2004 IEEE International Conference on Communications (IEEE Cat. No.04CH37577)*, vol. 2, June 2004, pp. 1090–1096 Vol.2.

[84] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler, "Cost-effective diameter-two topologies: Analysis and evaluation," in *Intl Conf High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, pp. 36:1–36:11.

[85] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, 1979.

[86] A. Khanna and J. Zinky, "The revised ARPANET routing metric," in *Symp. on Communications Architectures & Protocols*, ser. SIGCOMM '89, 1989, pp. 45–56.

[87] G. Kim, C. Kim, J. Jeong, M. Parker, and J. Kim, "Contention-based congestion management in large-scale networks," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[88] J. Kim, W. Dally, S. Scott, and D. Abts, "Cost-efficient Dragonfly topology for large-scale systems," *Micro, IEEE*, vol. 29, no. 1, pp. 33–40, 2009.

[89] J. Kim, W. Dally, and D. Abts, "Adaptive routing in high-radix Clos network," in *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006.

[90] ——, "Flattened Butterfly: a cost-efficient topology for high-radix networks," in *ISCA: Intl. Symposium on Computer architecture*, 2007, pp. 126–137.

[91] J. Kim, W. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable Dragonfly topology," in *ISCA'08: 35th International Symposium on Computer Architecture*. IEEE Computer Society, 2008, pp. 77–88.

[92] J. Kim, W. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *International Symposium on Computer Architecture*, ser. ISCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 420–431.

[93] M. Kim and J. Leskovec, "Multiplicative attribute graph model of real-world networks," in *Algorithms and Models for the Web-Graph*. Springer, 2010, pp. 62–73.

[94] P. Koka, M. O. McCracken, H. Schwetman, X. Zheng, R. Ho, and A. V. Krishnamoorthy, "Silicon-photonic network architectures for scalable, power-efficient multi-chip systems," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 117–128. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815977

[95] J. Labarta, S. Girona, and T. Cortes, "Analyzing scheduling policies using Dimemas," *Parallel Computing*, vol. 23, no. 1, pp. 23 – 34, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819196000944

[96] S. Lam and M. Reiser, "Congestion control of store-and-forward networks by input buffer limits–an analysis," *IEEE Trans. on Communications*, vol. 27, no. 1, pp. 127–134, 1979.

[97] J. W. Lee, M. C. Ng, and K. Asanovic, "Globally-synchronized frames for guaranteed quality-of-service in on-chip networks," in *35th International Symposium on Computer Architecture*. IEEE, 2008, pp. 89–100.

[98] M. Lee, J. Kim, D. Abts, M. Marty, and J. Lee, "Probabilistic distance-based arbitration: Providing equality of service for many-core CMPs," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, dec. 2010, pp. 509 –519.

[99] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985.

[100] X.-K. Liao, Z.-B. Pang, K.-F. Wang, Y.-T. Lu, M. Xie, J. Xia, D.-Z. Dong, and G. Suo, "High performance interconnect network for Tianhe system," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 259–272, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11390-015-1520-7

[101] J. Liu and J. G. Delgado-Frias, "A shared self-compacting buffer for network-on-chip systems," in *2006 49th IEEE International Midwest Symposium on Circuits and Systems*, vol. 2, Aug 2006, pp. 26–30.

[102] S. Ma, Z. Wang, Z. Liu, and N. E. Jerger, "Leaving one slot empty: Flit bubble flow control for torus cache-coherent NoCs," *IEEE Transactions on Computers*, vol. 64, no. 3, pp. 763–777, March 2015.

[103] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb 2002.

[104] R. Mandeville and J. Perser, "Benchmarking methodology for LAN switching devices," Tech. Rep., 2000.

[105] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny Tera: a packet switch core," *IEEE Micro*, vol. 17, no. 1, pp. 26–33, Jan 1997.

[106] S.-J. Miao and Y. Hsu, "Group allocation: A novel fairness mechanism for on-chip network," in *Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on*, Dec 2011, pp. 1–7.

[107] C. Minkenberg and M. Gusat, "Design and performance of speculative flow control for high-radix datacenter interconnect switches," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 680 – 695, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731508001482

[108] C. Minkenberg, M. Gusat, and G. Rodríguez, "Adaptive routing in data center bridges," in *17th IEEE Symposium on High Performance Interconnects (HOTI'09)*. IEEE Computer Society, 2009, pp. 33–41.

[109] R. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the Graph 500," *Cray User's Group (CUG)*, May 5, 2010.

[110] N. Ni, M. Pirvu, and L. Bhuyan, "Circular buffered switch design with wormhole routing and virtual channels," in *Intl Conf on Computer Design*, Oct 1998, pp. 466–473.

[111] C. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. Yousif, and C. Das, "ViChaR: A dynamic virtual channel regulator for network-on-chip routers," in *IntlSymp. on Microarchitecture*, 2006, pp. 333–346.

[112] Y. Oie, M. Murata, K. Kubota, and H. Miyahara, "Effect of speedup in nonblocking packet switch," in *IEEE International Conference on Communications, World Prosperity Through Communications,*, Jun 1989, pp. 410–414 vol.1.

[113] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 133–145, Apr. 2000.

[114] A. Papoulis, "Bernoulli trials," in *Random Variables, and Stochastic Processes*. McGraw-Hill, New York, 1990, pp. 57–63.

[115] J. Park, B. O'Krafka, S. Vassiliadis, and J. Delgado-Frias, "Design and evaluation of a DAMQ multiprocessor network with self-compacting buffers," in *Supercomputing '94., Proceedings*, Nov 1994, pp. 713–722.

[116] M. Parker, S. Scott, A. Cheng, and J. Kim, "Progressive adaptive routing in a Dragonfly processor interconnect network," Patent, Sep. 15, 2015, uS Patent 9,137,143. [Online]. Available: https://www.google.com/patents/US9137143

[117] G. Passas, "VLSI micro-architectures for high-radix crossbars," Ph.D. dissertation, FORTH-ICS, April 2012.

[118] J.-K. Peir and Y.-H. Lee, "Improving multistage network performance under uniform and hot-spot traffics," in *2nd IEEE Symposium on Parallel and Distributed Processing*, dec 1990, pp. 548 –551.

[119] M. Petracca, B. G. Lee, K. Bergman, and L. P. Carloni, "Design exploration of optical interconnection networks for chip multiprocessors," in *2008 16th IEEE Symposium on High Performance Interconnects*, Aug 2008, pp. 31–40.

[120] T. Pinkston, "Deadlock characterization and resolution in interconnection networks," *Deadlock Resolution in Computer-Integrated Systems*, pp. 445–492, 2004.

[121] B. Prabhakar and N. McKeown, "On the speedup required for combined input- and output-queued switching," *Automatica*, vol. 35, no. 12, pp. 1909 – 1920, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109899001296

[122] B. Prisacari, G. Rodríguez, M. García, E. Vallejo, R. Beivide, and C. Minkenberg, "Performance implications of remote-only load balancing under adversarial traffic in Dragonflies," in *INA-OCMC: Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*. ACM, 2014, pp. 5:1–5:4.

[123] V. Puente, C. Izu, R. Beivide, J. Gregorio, F. Vallejo, and J. Prellezo, "The adaptive bubble router," *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1180–1208, 2001.

[124] N. Rajovic, A. Rico, F. Mantovani, D. Ruiz, J. Vilarrubi, C. Gomez, D. Nieto, H. Servat, X. Martorell, J. Labarta *et al.*, "The Mont-Blanc prototype: An alternative approach for HPC systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[125] K. Ramakrishnan, S. Floyd, and D. Black, *RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP*, Std., 2001.

[126] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, Mar. 2011.

[127] R. Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: combined input-crosspoint-output buffered packet switch," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, vol. 4, 2001, pp. 2654–2660 vol.4.

[128] L. Schares, B. G. Lee, F. Checconi, R. Budd, A. Rylyakov, N. Dupuis, F. Petrini, C. L. Schow, P. Fuentes, O. Mattes, and C. Minkenberg, "A throughput-optimized optical network for data-intensive computing," *IEEE Micro*, vol. 34, no. 5, pp. 52–63, Sept 2014.

[129] T. Schneider, O. Bibartiu, and T. Hoefler, "Ensuring deadlock-freedom in low-diameter InfiniBand networks," in *IEEE Hot Interconnects*, 2016.

[130] S. Scott, D. Abts, J. Kim, and W. J. Dally, "The BlackWidow high-radix Clos network," in *Intl Symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 16–28.

[131] R. Sedgewick, "Algorithms in C, part 5: Graph algorithms," 2002.

[132] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ser. ISCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 432–443. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2005.37

[133] C. Seshadhri, A. Pinar, and T. G. Kolda, "An in-depth study of stochastic Kronecker graphs," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 2011, pp. 587–596.

[134] A. Singh, "Load-balanced routing in interconnection networks," Ph.D. dissertation, 2005.

[135] T. Skeie, O. Lysne, and I. Theiss, "Layered shortest path (LASH) routing in irregular system area networks," in *Intl Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2002, pp. 194–.

[136] T. Suzumura, K. Ueno, H. Sato, K. Fujisawa, and S. Matsuoka, "Performance characteristics of Graph500 on large-scale distributed environment," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 149–158.

[137] E. E. Swartzlander, "Parallel counters," *IEEE Trans. Comput.*, vol. 22, no. 11, pp. 1021–1024, Nov. 1973.

[138] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Trans. Comput.*, vol. 41, no. 6, pp. 725–737, Jun. 1992.

[139] G. Tao, L. Yutong, and S. Guang, "Using MIC to accelerate a typical data-intensive application: The breadth-first search," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1117–1125.

[140] Texas Instruments, *FIFO Architecture, Functions, and Applications*, 1999.

[141] K. Ueno and T. Suzumura, "Highly scalable graph search for the Graph500 benchmark," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '12, ACM. ACM, 2012, pp. 149–160, doi:10.1145/2287076.2287104.

[142] M. Valerio, L. E. Moser, and P. M. Melliar-Smith, "Using fat-trees to maximize the number of processors in a massively parallel computer," in *Intl Conf on Parallel and Distributed Systems*, 1993, pp. 128–134.

[143] L. G. Valiant, "A scheme for fast parallel communication," *SIAM journal on computing*, vol. 11, p. 350, 1982.

[144] ——, "Bulk-synchronous parallel computers," *Parallel Processing and Artificial Intelligence*, pp. 15–22, 1989.

[145] ——, "A bridging model for parallel computation," *Communications of the ACM*, vol. 22, no. 8, pp. 103–111, aug 1990.

[146] D. W. Walker and J. J. Dongarra, "MPI: a standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.

[147] R. Wang, L. Chen, and T. M. Pinkston, "Bubble coloring: Avoiding routing- and protocol-induced deadlocks with minimal virtual channel requirement," in *International Conference on Supercomputing*, 2013, pp. 193–202.

[148] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 63–71, Apr. 1992.

[149] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[150] P. Wijetunga, "High-performance crossbar design for system-on-chip," in *The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. Proceedings.*, June 2003, pp. 138–143.

[151] J. Won, G. Kim, J. Kim, T. Jiang, M. Parker, and S. Scott, "Overcoming far-end congestion in large-scale networks," in *Intl. Symp. on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 415–427.

[152] X.-J. Yang, X.-K. Liao, K. Lu, Q.-F. Hu, J.-Q. Song, and J.-S. Su, "The TianHe-1A supercomputer: Its hardware and software," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 344–351, 2011. [Online]. Available: http://dx.doi.org/10.1007/s02011-011-1137-8

[153] P. Yébenes, J. Escudero-Sahuquillo, P. J. García, and F. J. Quiles, "Straightforward solutions to reduce HoL blocking in different Dragonfly fully-connected interconnection patterns," *The Journal of Supercomputing*, pp. 1–23, 2016.

[154] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[155] H. Zhang, K. Wang, Z. Pang, L. Xiao, Q. Dou, and Y. Yuan, "An area-efficient DAMQ buffer with congestion control support," *Journal of Circuits, Systems and Computers*, vol. 25, no. 10, p. 1650125, 2016.

[156] H. Zhang, K. Wang, J. Zhang, N. Wu, and Y. Dai, "A fast and fair shared buffer for high-radix router," *Journal of Circuits, Systems and Computers*, vol. 23, no. 01, p. 1450012, 2014.