

Token Flow Control

Amit Kumar, Li-Shiuan Peh and Niraj K. Jha
Department of Electrical Engineering
Princeton University, Princeton, NJ 08544
Email: {amitk, peh, jha}@princeton.edu

Abstract

As companies move towards many-core chips, an efficient on-chip communication fabric to connect these cores assumes critical importance. To address limitations to wire delay scalability and increasing bandwidth demands, state-of-the-art on-chip networks use a modular packet-switched design with routers at every hop which allow sharing of network channels over multiple packet flows. This, however, leads to packets going through a complex router pipeline at every hop, resulting in the overall communication energy/delay being dominated by the router overhead, as opposed to just wire energy/delay.

In this work, we propose token flow control (TFC), a flow control mechanism in which nodes in the network send out tokens in their local neighborhood to communicate information about their available resources. These tokens are then used in both routing and flow control: to choose less congested paths in the network and to bypass the router pipeline along those paths. These bypass paths are formed dynamically, can be arbitrarily long and, are highly flexible with the ability to match to a packet's exact route. Hence, this allows packets to potentially skip all routers along their path from source to destination, approaching the communication energy-delay-throughput of dedicated wires. Our detailed implementation analysis shows TFC to be highly scalable and realizable at an aggressive target clock cycle delay of 21FO4 for large networks while requiring low hardware complexity.

Evaluations of TFC using both synthetic traffic and traces from the SPLASH-2 benchmark suite show reduction in packet latency by up to 77.1% with upto 39.6% reduction in average router energy consumption as compared to a state-of-the-art baseline packet-switched design. For the same saturation throughput as the baseline network, TFC is able to reduce the amount of buffering by 65% leading to a 48.8% reduction in leakage energy and a 55.4% lower total router energy.

1. Introduction

The current trend in utilizing the growing number of transistors provided by each technology generation is to use a modular design with several computation cores on the same chip. As the number of such on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important. As a result, packet-switched on-chip networks are fast replacing buses and crossbars to emerge as the pervasive communication fabric in both general-purpose chip multi-processor (CMP) [1]–[3] as well as application-specific system-on-a-chip (SoC) [4] domains.

Apart from providing scalable and high-bandwidth communication, on-chip networks are required to provide ultra-low latency with an extremely constrained power envelope and a low area budget. Most state-of-the-art packet-switched designs use a complex router at every node to orchestrate communication, and packets travel only a short distance on the link wires

before having to go through a complete router pipeline at every intermediate hop along their path. As a result, communication energy/delay in such networks is dominated by the router overhead, in contrast to an ideal network where packet latency and energy are solely due to the wires between the source and destination. For instance, routers consume around 61% of the average network power in the MIT Raw chip as opposed to 39% consumed by the links [5]. Similarly, the Intel 80-core teraflops chip has router power taking 83% of network power versus 17% consumed by the links [3]. The large energy-delay-throughput gap between the state-of-the-art packet-switched network and the ideal interconnect of dedicated point-to-point wires was pointed out in [6].

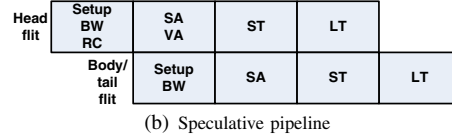
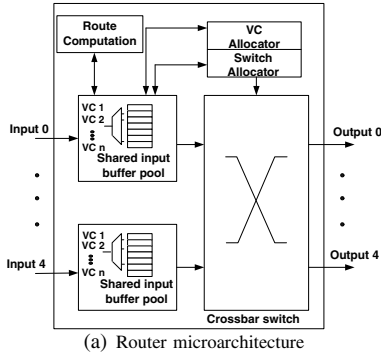
In this work, we propose TFC, a flow-control mechanism which aims to deliver the energy-delay-throughput of dedicated wires through the use of *tokens*. Tokens are indications of resource availability in the network. Each node in the network sends out tokens in its fixed local neighborhood of d_{max} hops to disseminate information about availability of resources, such as buffers and virtual channels (VCs) at its input ports. Individual packets then use these tokens during both routing – to find less congested routes in chunks of up to d_{max} hops, and flow control – to bypass the router pipeline at intermediate nodes along these d_{max} -hop routes. When one such d_{max} -hop token route ends, another token route can be chained to it seamlessly without any additional energy-delay overhead. Thus, packets can use an arbitrary number of tokens to bypass all intermediate routers between their source to destination, like that in an ideal network.

In the rest of this paper, Section 2 provides background for this work by looking at router energy/delay overhead in state-of-the-art packet-switched designs. This is followed by the working of TFC in Section 3 and its implementation details in Section 4. Evaluation results are presented in Section 5. Section 6 presents related work while Section 7 concludes the paper.

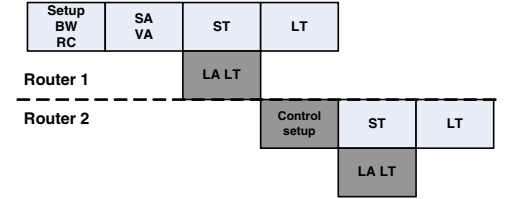
2. Background

2.1. Baseline state-of-the-art router

Fig. 1(a) shows the microarchitecture of a state-of-the-art baseline VC router used for comparison in all our experiments. We assume a two-dimensional mesh topology for simplicity. Flit-level buffering and on/off VC flow control [7] are used to minimize the amount of buffering per router and hence its area footprint. This design incorporates several features which are critical to on-chip networks – low pipeline delay using lookahead routing [8], speculation [11], [12], no-load bypassing



(b) Speculative pipeline



(c) No-load bypass pipeline – lookahead pipeline stages are shaded in dark (LA LT: Lookahead link traversal)

Figure 1. State-of-the-art packet-switched router

and lookaheads [9], [13], [21], high throughput using dynamic buffer management [10], high clock frequency using pipelining, simplified VC allocation [9] and separable switch allocation [7] and low router energy using no-load bypassing [9], [13], [21] and power-optimized straight-through buffers and cut-through crossbar [5]. Fig. 1(b) presents the router pipeline with the different stages being setup (in which flits send arbitration requests for their desired output port), buffer write (BW), route computation (RC), switch allocation (SA), VC allocation (VA), switch traversal (ST) and link traversal (LT). Fig. 1(c) presents the no-load bypass pipeline applicable under very low loads (when router ports are mostly empty) where lookaheads, which travel one cycle ahead of data flits on separate narrow channels¹, are used to do control setup to allow data flits to directly go through ST upon arrival. Control setup involves checking for input/output port conflicts (with local flits and other lookaheads) and free resources (buffer/VC) at the next hop. This no-load bypassing is *only* effective under very low loads since the router ports will be mostly busy as network load increases, leading to the lookaheads failing to do control setup most of the time. We refer the readers to [6], [9] for further details on the baseline.

2.2. Benefits of bypassing

Router energy/delay overhead: Out of all router pipestages described above, ST and LT are the only stages where a packet actually covers the physical distance towards its destination with all other stages contributing to the delay overhead which packets incur at every hop. In terms of energy consumption, apart from the energy spent in traversing the physical links, packets consume additional energy at each router while being written to the buffer, going through VC arbitration and switch arbitration, being read out of the buffer and finally while traversing the crossbar. In contrast to this, if packets are allowed to bypass the router pipeline at all levels of network load by skipping through buffer read/write and allocation operations and only going through ST and LT, it would lead to a significantly shorter pipeline delay per hop, savings in router energy as well as a higher throughput due to lower contention for resources. Moreover, since a flit does not use up any buffers when bypassing a router, this allows other flits to have more free

buffers available to them which helps push performance at high loads. When looked at in another way, bypassing helps to cut down the total buffering needed to achieve a target performance, resulting in significant buffer leakage and dynamic energy as well as area savings. This helps approach the energy/delay of an ideal network in which packets would incur only wire energy/delay and no router overhead along their path.

3. Token flow control (TFC)

In this section we describe the working of TFC. Tokens can have two flavors, *normal tokens* and *guaranteed tokens*, as explained next by answering the following questions.

3.1. Normal tokens

What are normal tokens? A normal token is a hint for buffer and VC availability at an input port of a router.

How are tokens turned on and off? A router turns *on* a normal token for a particular input port if there are greater than a fixed threshold of b_{thr} buffers and v_{thr} VCs available at that port. These thresholds act as a proxy for congestion at that router port. Similarly, a router turns *off* a normal token for an input port if either the number of free buffers becomes $\leq b_{thr}$ or the number of free VCs becomes $\leq v_{thr}$ at that port. It should be noted that the use of the word token here differs from the traditional usage – a token is not turned off if a packet grabs it. It is turned off explicitly by its source router based on the number of buffers and VCs at that input port.

How are tokens forwarded? While in state-of-the-art networks, availability of buffers and VCs is communicated only across adjacent nodes², tokens are used to communicate this information across all nodes in a d_{max} -hop neighborhood (where d_{max} is the maximum token route length). Fig. 2 demonstrates token forwarding for a particular input port (West input) at node 34. When there are greater than b_{thr} buffers and v_{thr} VCs available at the West input port of node 34, it turns on an E_{on} token for that input port to its West neighbor node 33, signaling to node 33 that its *East* output direction (hence the token E_{on}) is not congested. Node 33 then forwards this E_{on} token by *broadcasting* it to its neighbors (except in the East direction, assuming routes with U-turns are not allowed) by appending the received token with its own tokens (based on resource availability at its corresponding input ports). Hence, before forwarding a token along a particular port, the node

1. This lookahead, which uses a significantly narrower channel as compared to the data, also needs to traverse a correspondingly short distance of the crossbar switch (since its path can be laid out along the inner side of the wider normal data path) before traversing the link which takes place in parallel with reading of the data flit out of its buffer.

2. This is done using credit, on/off or ack/nack signaling in conventional networks [7].

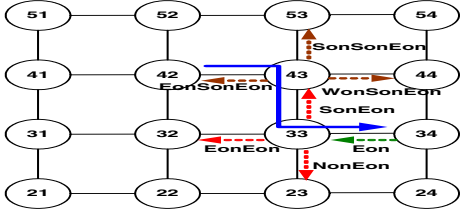


Figure 2. Token forwarding in a network with $d_{max} = 3$

checks to see if there are sufficient resources (greater than b_{thr} buffers and v_{thr} VCs) at that corresponding input port. For instance, node 33 is allowed to forward the E_{on} token to its North neighbor (node 43) using an appended $S_{on}E_{on}$ token, only if there are sufficient resources at its North input port (South output port of node 43). All nodes which receive tokens from node 33 do a similar forwarding to their neighbors by appending their own tokens, as shown in Fig. 2. This token forwarding takes place for up to d_{max} hops in the network, thus allowing each node to gather knowledge about resource availability at all other nodes within its d_{max} -hop vicinity.

Propagation of tokens between adjacent nodes uses separate point-to-point wires which are much narrower than the data channels. Hence, in a way, TFC makes use of tokens traveling on separate narrow channels between nodes to optimize communication energy/delay on the wider data channels.

How are tokens used? Tokens provide knowledge of congestion levels in a d_{max} -hop neighborhood. Packets at these nodes thus use these tokens during RC to form a *token route* – partial route of the packet computed for up to the next d_{max} hops at once along less congested sections of the network. Thus, for instance, the $E_{on}S_{on}E_{on}$ token at node 42 in Fig. 2 can be used by a packet at that node to travel to node 34 (assuming node 34 is closer to the packet’s destination) along a three-hop partial route (assuming $d_{max} \geq 3$) through the East output to reach node 43, followed by a South hop to node 33 and then an East hop to node 34 (route shown in bold), thus making use of the $E_{on}S_{on}E_{on}$ token to route through a path along which sufficient resources are available.

Once these token routes are known, lookaheads are sent (on separate channels) ahead of the data flits along these routes to do control setup at intermediate nodes one cycle prior to data arrival, thereby allowing the data flit to bypass the router pipeline and right away traverse the crossbar when it arrives at these intermediate nodes. In contrast to the baseline, which allows lookaheads to succeed only under very low loads when router ports are empty, lookaheads here hold tokens. When such a lookahead arrives at an intermediate router, it is prioritized to use its desired switch port over any locally-buffered flits at that router and hence succeeds in control setup even at high loads. Moreover, the likelihood of conflicts with other lookaheads is low along token routes as tokens are forwarded only along less congested sections in the network. Hence, in the example of Fig. 2, a packet using the $E_{on}S_{on}E_{on}$ token at node 42 can bypass intermediate nodes 43 and 33 on its way to node 34.

Are tokens used on a per-flit basis? Yes, tokens are used by individual flits of a packet. A header flit starting at a node is free to use any of the available tokens during route computation.

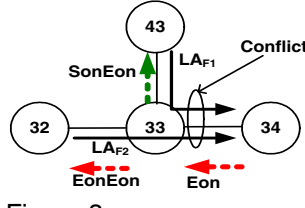


Figure 3. Example token conflict

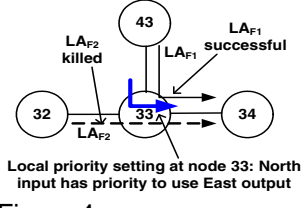


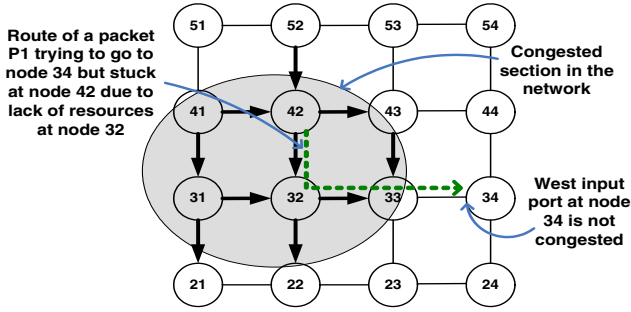
Figure 4. Token conflict resolution

Body/tail flits, on the other hand, do not go through route computation and need to follow this same route and use tokens to bypass intermediate routers along that route if such tokens are available or else they traverse that route without tokens. Similar to the baseline VC flow control design which uses a VC control table at each router to store information about each traversing packet [7], a packet’s route in the token scheme is stored in a control table at each intermediate router as its header flit traverses it, which allows the subsequent body/tail flits to follow the same route as the header.

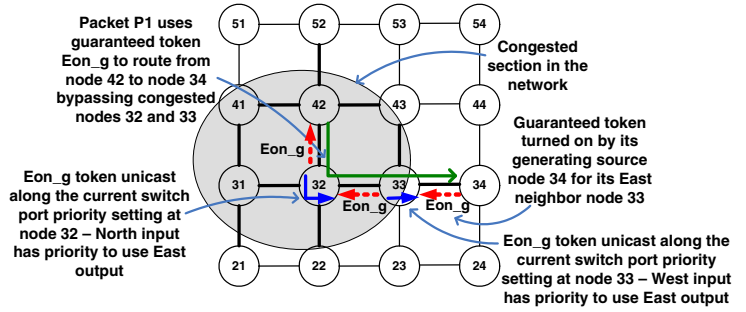
How are conflicts between multiple flits using tokens resolved? As discussed above, tokens are forwarded by broadcasting them in the network. However, since broadcasting involves forwarding the same token along multiple directions, it can lead to conflicts when multiple lookaheads (corresponding to different flits) arrive at a router from different input directions holding the token for the same output port to bypass that router. Fig. 3 shows an example of such a conflict where two lookaheads for flit F1 (using token $S_{on}E_{on}$) and flit F2 (using token $E_{on}E_{on}$) arrive simultaneously at router 33 at its North and West input ports, respectively, holding the token to go out at the East output.

To resolve such conflicts, each router maintains a local *switch port priority vector* which has one entry for each output port which denotes the particular input port which has priority to use that output port. If an incoming lookahead matches the priority setting for its desired output port at that router, it is deemed to have won the switch. For instance, as shown in Fig. 4, the current priority setting at node 33 denotes the North input having priority to use the East output. Hence, the lookahead for F1 (which matches this priority setting) is allowed to succeed while the lookahead for F2 is killed. This priority vector is varied dynamically to balance the request queues at each input based on the observed traffic, as explained in Section 4.4.

What happens when a lookahead carrying a token gets killed? Normal tokens only act as *hints* of resource (buffers/VCs) availability in the network. They do not need to guarantee resources at the endpoint node of a flit’s token route. Hence, a check for a free buffer/VC is done at each intermediate node along the token route and a lookahead carrying a token is killed at an intermediate node if there are no free buffers or VCs available at the next hop along the token route. A lookahead might also be killed due to conflicts with other lookaheads (as described above in the answer to the previous question). When a flit’s lookahead gets killed at an intermediate node, its corresponding data flit cannot bypass that node and needs to get buffered there while going through the normal pipeline. The key thing to note here is that since a check for a free



(a) An example of network congestion (bold links denote congestion at their endpoints)



(b) Using guaranteed token to bypass a congested network section (priority settings at intermediate nodes are shown using solid bold arrows)

Figure 5. Guaranteed tokens

buffer/VC is done on a hop-by-hop basis, it is completely safe to stop a flit mid-way if its lookahead gets killed. Continuing with the example shown in Fig. 2, consider the case when a packet P1 at node 42 is using the $E_{on}S_{on}E_{on}$ to travel to node 34 and buffers at the North input of node 33 become full by the time flits of P1 reach node 43. In this case, the in-flight lookaheads of these flits are killed at node 43 and the flits are stopped and buffered at that node. Thus, even though individual token routes can span up to d_{max} hops, buffer/VC management is still done on a hop-by-hop basis. This makes token routes highly scalable by allowing them to extend over long distances without the need to ensure a large amount of buffering to cover a correspondingly longer round-trip of all in-flight flits between the endpoints of that route. Hence, the benefits of longer token routes can be achieved with the buffering overhead of only single-hop routes. It should be noted that since normal tokens are forwarded only along paths where each port along that path has free resources, the probability of lookaheads getting killed due to lack of resources at intermediate nodes is small.

What happens at the end of a d_{max} -hop token route? A single token route with a lookahead carrying a token going ahead of the data flit can extend up to d_{max} hops, provided the lookahead is successful and does not get killed at every intermediate node along that route. To allow seamless transition from using one token route to another, we do RC in the lookahead pipeline. Specifically, one hop before the current token route is about to complete (i.e., at the $(d_{max} - 1)$ th hop from the starting node of the token route), a new token route is computed for up to the next d_{max} hops in the lookahead pipeline (explained further in Section 4.1). This allows chaining of multiple token routes: allowing the intermediate router between successive token routes to be bypassed as well. Thus, in cases where the lookahead does not get killed in between, a flit has the opportunity to bypass all routers along its path from the source to destination by chaining several token routes in succession.

3.2. Guaranteed tokens

What are guaranteed tokens? A guaranteed token is a guarantee for buffer and VC availability at an input port of a router. Similar to d_{max} for normal tokens, guaranteed tokens are forwarded in a g_{max} -hop neighborhood (where g_{max} is the maximum guaranteed path length). Unlike a normal token, which only acts as a hint of resource availability at its source node from where the token was generated, a guaranteed token guarantees the presence of free buffers and VCs at its generating source

node. Moreover, while lookaheads carrying normal tokens can get killed at intermediate nodes along the token route due to conflicts with other lookaheads, the mechanism using which guaranteed tokens are propagated in the network ensures that such conflicts never happen and lookaheads carrying guaranteed tokens never get killed at intermediate nodes. Thus, flits using these tokens are guaranteed to bypass intermediate nodes along the token route. Similar to normal tokens, these tokens should not be viewed as being turned off if a packet uses them. They are turned off explicitly by their source routers.

Why do we need guaranteed tokens? Consider a packet starting at a node which is in a congested section of the network. In this case, there will likely be no normal tokens for bypassing neighboring nodes in that congested neighborhood due to lack of buffers/VCs at neighbors. Thus, the packet will be forced to be buffered and wait for VCs/buffers by going through the router pipeline at each hop which will most likely further exacerbate contention for these resources in that congested section, hence leading to an increase in queuing delay.

The key intuition behind guaranteed tokens is the observation that if a packet's destination does not lie in a congested section of the network, then ideally it should not care about the availability of resources at intermediate nodes in that section and should be able to jump out of it by zooming through neighboring routers. Fig. 5(a) depicts such a scenario where a packet going from node 42 to destination node 34 is not able to bypass nodes in a congested network section due to lack of normal tokens from neighboring nodes even though its destination node lies outside that congested section and has enough free buffers/VCs. In such cases, having guaranteed tokens which allow flits to skip through the congested routers, from 42 directly to 34, will help improve energy-delay-throughput.

What needs to be ensured by a g_{max} -hop guaranteed token?

A g_{max} -hop guaranteed token provides a guarantee (to a flit using it) for resource availability at the endpoint node of that token route (i.e., at the source node from where that token was generated), but not at any of the intermediate nodes along that route. Hence, such a token needs to ensure three things:

- There is at least one free buffer so the flit is ensured of a buffer at the endpoint node of the guaranteed token route.
- There is at least one free VC so the flit is ensured of finding a free VC at that endpoint node.
- A lookahead carrying a guaranteed token is not killed at any intermediate node along that token route due to

conflicts with other lookaheads. Since a flit starting out at a node using a guaranteed token is not guaranteed resources at intermediate hops along its token route, it is not safe to kill its lookahead in between at a node which may have no free buffers/VCs. Hence, a guaranteed token route needs to be conflict-free.

How are these conditions ensured when turning on/off and forwarding a guaranteed token?

Turning on: In order to ensure the first two conditions of at least one free buffer and VC, a source router of a guaranteed token turns *on* a guaranteed token for an input port only if there are greater than b_{gthr} buffers and v_{gthr} VCs available at that port. These threshold values are higher than the corresponding ones for normal tokens (b_{thr} buffers and v_{thr} VCs) in order to cover a longer g_{max} -hop round-trip delay so that all already-in-flight flits are ensured of buffers/VCs during the time it takes to turn off the guaranteed token. An example is shown in Fig. 5(b), where node 34 sends out a guaranteed token E_{on_g} to its West neighbor when its West input port has greater than b_{gthr} free buffers and v_{gthr} free VCs.

Forwarding: The third condition of no lookahead conflicts is ensured by the mechanism used to forward guaranteed tokens. When a node receives a guaranteed token, it forwards the token by *unicasting* the token along the direction given by the switch port priority vector entry for that output port. This is unlike normal tokens which are forwarded by *broadcasting* to neighboring nodes. This unicasting ensures that a lookahead arriving at a node carrying guaranteed token will always have priority to use its desired output port and hence will never be killed due to conflicts with other lookaheads. Thus, as shown in Fig. 5(b), node 33 on receiving the E_{on_g} guaranteed token from the token's source node 34, unicasts it to its West neighbor, assuming in the current priority setting at node 33, the West input has priority to use the East output. This forwarding of the guaranteed token is done similarly at other nodes and continues for g_{max} hops in the network. Assuming a $g_{max} = 3$, the E_{on_g} token ends at node 42. It should be noted that, unlike normal tokens, a node does not need to check for buffers/VCs at its own input ports before forwarding guaranteed tokens.

Turning off: A guaranteed token for an input port is turned *off* by its source router if either the number of free buffers at that port becomes $\leq b_{gthr}$ or the number of free VCs at that port becomes $\leq v_{gthr}$. On the other hand, at routers which are not the source of a guaranteed token but are involved in forwarding of that token, whenever there is a change in their local switch port priority vector setting, any guaranteed tokens forwarded along the previous priority direction are turned off and instead now forwarded along the new priority direction. Handling of already in-transit flits using the old guaranteed tokens is explained later in Section 4.4.

How are guaranteed tokens used? If a packet is not able to get a normal token for the next hop along its route (due to lack of buffers/VCs), it looks for any available guaranteed tokens which match its already-computed route. If such a guaranteed token is available, the packet is allowed to make progress by sending

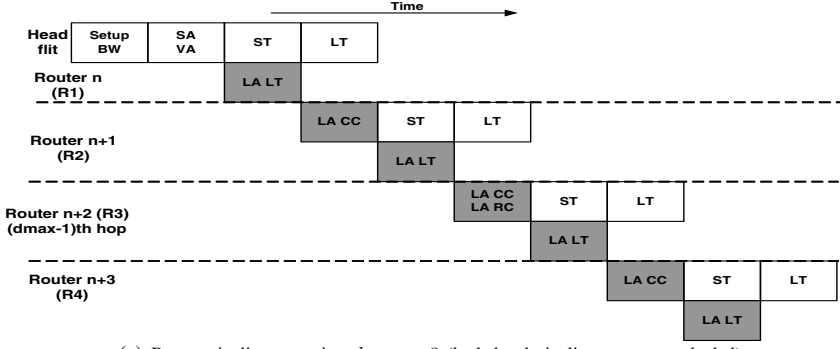
out a lookahead (similar to lookaheads for normal tokens) even if there are no VCs/buffers available at the next hop. As described above, since guaranteed tokens are only unicast along switch allocator priority setting paths at each router, a lookahead arriving at an intermediate router holding a guaranteed token is ensured of having priority to use its desired switch port at that router. Moreover, now there is no need to check for VCs/buffers at the intermediate hops because of the guarantee to get a VC/buffer at the endpoint node of the guaranteed token route. Hence, such a lookahead is allowed to directly make progress to the next hop with its corresponding data flit directly traversing the switch upon arrival in the next cycle. Continuing with the example in Fig. 5(b), even though there may be no buffers/VCs available at the North input port of node 32, a flit starting at node 42 and going to node 34 (which may be its destination node or a node closer to its destination) can use the E_{on_g} guaranteed token to bypass nodes 32 and 33 along the way without checking for buffers and VCs at those nodes. The flit only needs to allocate a free VC for the endpoint node of its guaranteed token route (node 34) which is done by the lookahead at the previous node 33. Moreover, several guaranteed tokens can be chained together in the lookahead pipeline, just like chaining of normal tokens, thus enabling arbitrarily long guaranteed paths.

4. Implementation details

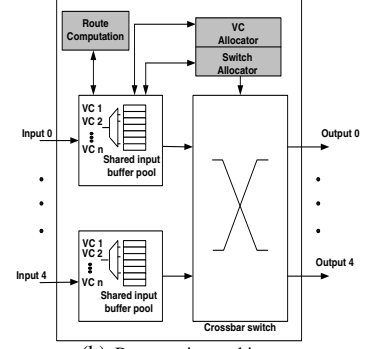
4.1. Router pipeline and microarchitecture

Fig. 6(a) shows the typical pipeline of the head flit of a packet traveling over multiple hops using a token. Normal tokens are used during RC to compute a packet's token route for up to the next d_{max} hops in one go. This is done one hop in advance, just as in the baseline router. At node R1, from where the packet's token route starts, the head flit first goes through the complete router pipeline starting with the setup stage where it sends an output port request to the switch allocator based on its immediately next output port given by its route. In parallel, it is written to the payload buffer. In the next cycle, SA and VA are done using using a scheme similar to the one used in the baseline router (Section 2). After successfully winning the switch port, the flit traverses the crossbar during the ST stage. A lookahead pipeline operates in parallel with the data flit pipeline, where a lookahead, which encodes the flit's token route and its remaining hops to the destination, is sent out on a separate dedicated narrow channel during the lookahead link traversal (LA LT) stage to travel to the next node R2. During the time that the data flit goes through LT in the next cycle, a lookahead conflict check (LA CC) takes place at the next node (R2) for the subsequent output port encoded in the token route carried by the lookahead.

During LA CC, the lookahead succeeds in winning its desired switch port if there are no conflicts, i.e., if it is the only lookahead asking for that output port in that cycle. However, if more than one lookaheads arrive simultaneously at R2 asking for the same output port, the local switch port priority vector maintained by R2 is used to break conflicts (as explained earlier in Fig 4), with a lookahead succeeding if it arrived at the input



(a) Router pipeline assuming $d_{max} = 3$ (lookahead pipeline stages are shaded)



(b) Router microarchitecture

Figure 6. Token flow control router pipeline and microarchitecture

port which has priority to use its desired output port. Thus, the complexity of LA CC is minimal as compared to a full-blown switch arbitration scheme and it only involves checking if a lookahead matches the priority vector setting for its desired port. There is never a need to do full-blown arbitration. For instance, if two lookaheads conflict and none of them matches the priority vector setting then both are killed. Hence, LA CC only uses a one-hot detect logic to check if there is a conflict (similar to the baseline), along with a simple priority match to break conflicts. In parallel with LA CC, a free VC for the next hop is picked from a pool of free VCs along with checking for a free buffer, just as in the baseline. However, in this case, if there are no buffers/VCs available at the next hop, the lookahead is still allowed to succeed if a guaranteed token which matches either the entire or a part of the packet's remaining token route is available³. Thus, while normal tokens are used to form a packet's token route for up to d_{max} hops during RC, a guaranteed token can be selected for this already-computed route at any node along the route during LA CC if there are no resources available at the next hop.

As mentioned in Section 3.1, lookaheads, in this case, are prioritized over any locally-buffered flits at that router waiting to exit the same output port. Thus, if the lookahead succeeds in LA CC, the local SA result at R2 for that output port is nullified. The data flit corresponding to that successful lookahead is able to bypass the router pipeline at R2 and directly traverses the crossbar during ST when it arrives in the next cycle, with the lookahead continuing to travel to the next hop along the token route. However, if the lookahead fails, it is killed and its corresponding data flit goes through the complete pipeline.

Another token route can be chained one hop before the end of the current token route (at router R3, assuming the current token route extends up to $d_{max} = 3$ hops, i.e., up to router R4) by doing RC in the lookahead pipeline (LA RC), thus avoiding any delay overhead to the data pipeline due to this chaining from one token route to another. There is a similar data and lookahead pipeline for each body/tail flit of the packet except that they do not go through RC and VA stages, instead inheriting the route and VC used by the header flit.

Fig. 6(b) shows the microarchitecture of a router in a network using TFC. The modifications to the different units, as compared

to the baseline microarchitecture of Fig. 1(a), are minimal and are highlighted using shaded regions. The RC unit is modified to incorporate available normal tokens while calculating packet routes. The switch allocator now maintains a switch port priority vector for resolving clashes among lookaheads during LA CC. Moreover, SA for locally-buffered flits is nullified if a lookahead is using a particular output port. The VC allocator now also checks for guaranteed tokens which match the packet's route if there are no VCs available for the next hop. Note that these modifications are mainly to the control logic and arbiters which are only a small portion of the router complexity. For instance, these units consume only 7% of the router power in the Intel teraflops router [3].

4.2. Buffer and VC thresholds for sending tokens

As mentioned in Section 2, the baseline design uses dynamic buffer management with on/off flow control to signal buffer availability across adjacent nodes. As the number of free buffers at an input port of a node becomes equal to or falls below a threshold b_{thr} , it sends an *off* signal to its adjacent node, signaling to it to stop sending further flits. The threshold value b_{thr} needs to be enough to ensure that all flits which are already in-flight are ensured of free buffers during the time the *off* signal is transmitted and processed. Hence, b_{thr} is calculated based on the round-trip delay between adjacent nodes. Assuming the off signal takes one cycle each for transmission to the adjacent node, one cycle for being processed and there can be at most two flits already in-flight which need to be ensured of free buffers, gives a value of $b_{thr} = 4$. Similarly, an *on* signal is sent out if the number of free buffers exceeds b_{thr} .

In TFC, even though a token route using a normal token can extend up to d_{max} hops, flits traveling along normal token routes need to check for buffers at every intermediate node along their route and, hence, normal tokens require buffers to be managed on a hop-by-hop basis similar to the baseline. Therefore, a normal token for an input port is turned off if the number of free buffers at that port becomes less than or equal to b_{thr} and *vice versa*. On the other hand, the corresponding buffer threshold value, b_{gthr} , for turning on/off guaranteed tokens, needs to cover a longer up to g_{max} -hop round-trip to ensure buffer availability across the g_{max} hops. b_{gthr} is given by:

$$b_{gthr} = g_{max} + 1 + 2g_{max} = 3g_{max} + 1 \quad (1)$$

where the off signaling is assumed to take g_{max} cycles to travel back g_{max} hops to the farthest away node which may be using

3. This check for guaranteed tokens is done in parallel with LA CC to allow chaining from one token route to another without overhead.

the guaranteed token, it takes one cycle for processing the off signal and there can be at most $2g_{max}$ flits already in-flight at the intermediate nodes which may be using this guaranteed token and which need to be ensured of free buffers. The value of $2g_{max}$ in-flight flits stems from the bypassing data pipeline in TFC which has two stages (ST and LT) per hop (Fig 6(a)).

In case of VCs, the threshold for the number of free VCs, v_{thr} , to turn on/off a normal token, is a parameter which is used as a proxy to communicate the level of congestion. Hence, a normal token is turned off for a port having less than or equal to v_{thr} VCs and *vice versa*. On the other hand, the threshold value to turn on/off a guaranteed token, v_{gthr} , needs to be such that a packet using a guaranteed token is ensured to find a free VC when it reaches the endpoint of that guaranteed token route (which may be up to g_{max} hops away from where the packet starts using that token). In our design, v_{gthr} is set to be equal to b_{gthr} to cover a g_{max} -hop round-trip.

4.3. Route computation

Routing algorithm: As explained in Section 3.1, packets use normal tokens to form up to d_{max} -hop long token routes along less congested sections of the network. This assumes the routing algorithm to be adaptive. In this work, we use minimal west-first routing [14] for both the baseline and TFC. In this routing scheme, packets are restricted to cover all their hops in the West direction (if any) before turning to a different dimension, while routes can be adaptive if a packet has hops remaining in both East and North directions or both East and South directions (in which case it chooses the direction which is more profitable, i.e., has more buffers [15], [16]). Minimal west-first routing is easy to implement and provides inherent deadlock freedom by prohibiting turns which lead to cyclic dependencies while using only minimal paths by not allowing misroutes which can adversely affect packet energy/delay.

It should be noted that even though we choose minimal west-first routing, TFC can be used along with any adaptive routing algorithm. Moreover, even if the routing algorithm is non-adaptive, packets can still bypass nodes using tokens along a deterministic route although fewer token route choices will be available. We evaluate this aspect in Section 5.7.

RC using longest match: RC is done by finding the longest feasible token route (which can be up to d_{max} hops long) based on the routes permitted by the adaptive routing scheme and the available tokens. Since packet routes are computed for up to d_{max} hops in one go, the hardware complexity of RC (which is done in the lookahead pipeline, as shown in Fig. 6(a)) is proportional to d_{max} . The lookahead corresponding to the head flit of a packet carries the magnitude of the remaining hops to the packet's destination node in each dimension (x and y), with two sign bits indicating the corresponding direction to be taken in each dimension (sx and sy). Since minimal west-first routing allows adaptivity only if there is distance to be covered in both East ($x > 0$ and $sx = 0$) and North ($y > 0$ and $sy = 0$) directions or in both East ($x > 0$ and $sx = 0$) and South ($y > 0$ and $sy = 1$) directions, finding less-congested routes using tokens can be done only in these cases. Consider the case

when the packet has hops left in the East (E) and North (N) directions. In this case, the possible values for x either lie in the range 1 to $d_{max} - 1$ or $x \geq d_{max}$, and similarly for y , thus leading to a total of $d_{max} \times d_{max}$ possible (x, y) tuples. Based on the available tokens at a given instant of time, each router maintains a corresponding $d_{max} \times d_{max}$ vector of the longest match (LM) token route for each corresponding (x, y) tuple. For instance, in a network with $d_{max} = 3$, if a packet has hops left in the E and N directions with $x = 2$ and $y = 1$, it can take up to two E hops and up to one N hop in any order leading to the following eight possible route combinations: EEN, ENE, NEE, EE, EN, NE, E and N. The LM vector entry corresponding to the $(x = 2, y = 1)$ tuple then contains the longest possible consecutive sequence of available tokens out of the above eight possibilities. Hence, if there is an E_{on} token available at a node from its immediate East neighbor, an E_{on} token available from the node which lies two hops to the East, and an N_{on} token available from the node two hops East and then one hop North, the LM vector entry for $(x = 2, y = 1)$ would contain EEN. Among two or more token sequences of the same length, the one in which there are more profitable hops (hops along directions with more buffers than the other alternative direction) is chosen as the LM entry⁴. All LM vector entries can be precalculated based on the available tokens before a lookahead arrives. Hence, LM vector calculation does not fall on the RC critical path.

Route padding: The LM vector entries, for which the longest sequence of available normal tokens is smaller than the maximum hops possible for that (x, y) tuple (which is $\leq d_{max}$), are padded assuming X-Y routing to fill in the remaining hops even though there may be no tokens available for the padded portion of the route. Hence, considering the above example when the longest token route for the tuple $(x = 2, y = 1)$ is EE, the corresponding LM vector entry contains the padded route EEN (assuming $d_{max} = 3$). Route padding helps packets to bypass routers when there are no normal tokens available or when there is no adaptivity possible along their route due to constraints imposed by the routing algorithm⁵. In such cases, adaptive token routes cannot be created and padding is used instead to compute the packet's route for up to d_{max} hops. Lookaheads, which do not carry tokens, are then sent out ahead of the data flits along this padded route. These lookaheads go through a similar pipeline as the lookaheads carrying tokens (Fig. 6(a)) and are prioritized over local flits at intermediate nodes, thereby allowing packets to bypass these intermediate nodes. However, the chances of successfully bypassing a node is lower for lookaheads which do not carry a token than when bypassing using a token since routes formed using tokens are ensured to be along less congested paths in the network and, hence, have relatively less chances of conflicts.

As mentioned in Section 4.1, packets are allowed to use guaranteed tokens to jump out of congested pockets *only along their already-computed route*. Hence, using padding to compute

4. If the number of profitable hops is the same, the one, in which the packet covers its remaining hops in the x -dimension first, is chosen.

5. For minimal west-first routing, there is no adaptivity possible if the packet needs to travel west or has hops left only in either the east or the Y-dimension.

Table 1. Route computation critical path delay

d_{max}	Delay (FO4)
1	3.59
2	5.09
3	5.96

longer routes (up to d_{max} hops) when there are no normal tokens available also enables packets to use any available guaranteed tokens to bypass nodes when there are no buffers/VCs available at the next node.

RC logic complexity: We analyzed the critical path delay of the RC logic circuit in terms of fanout-of-four (FO4) delays using the theory of logical effort with the methodology used in [12]. This delay T in FO4 units is given by:

$$T = \log_4(3 \cdot d_{max}^3) + 2.8 \quad (2)$$

The values of T for different values of d_{max} are presented in Table 1. As can be seen, this delay grows with larger values of d_{max} . However, since RC is done in the lookahead pipeline one hop before it is required (as shown in Fig. 6(a)), it does not have to be serialized before doing LA CC and can be done in parallel. Using circuit-level analysis data from [9] (which targets a clock period of 18FO4), the only stage where we extend the critical path in comparison is LA CC (due to the extra RC logic which dominates the LA CC logic in delay). We estimate this RC logic of up to $d_{max} = 3$ to require an additional delay of 3 FO4 on top of using the already available wire delay slack of 3 FO4 in the lookahead pipeline, as shown in [9] (which is due to a relatively short switch path which these narrow bit-width lookaheads need to traverse as compared to the data flit). This gives a total clock delay of 21 FO4 for our design.

4.4. Switch port priority vector

As mentioned in Section 3, each router maintains a switch port priority vector with an entry for each output port denoting which input port has priority to use that output. This vector is used for both – breaking conflicts between multiple lookaheads carrying normal tokens and figuring out the unicast direction along which the router forwards the guaranteed tokens it receives from its neighbors. We vary this priority vector entry at each router dynamically to favor different input-output combinations based on the observed traffic with the intuition being to *balance the output port request queues at each input*. Specifically, each router keeps a count of the number of waiting requests for each output port at each input queue. After every e clock cycles (where e is a parameter defined as the epoch period), these counters are examined, and the input port i with the highest number of requests for a particular output port o is assigned priority to use that output. This assumes that the past traffic seen in the previous epoch is a predictor of the future traffic to be seen in the next epoch.

The forwarding of normal tokens is unaffected by local changes in priority settings at each router. On the other hand, any guaranteed tokens, which were forwarded earlier by unicasting along the direction given by the old priority setting, need to be turned off while now being forwarded along the direction given by the new priority setting. Hence, for instance, if the West input had priority over the East output at a router in the old setting and the North input has that priority in the new

setting, any guaranteed tokens received at the East input by that router and forwarded along the West neighboring node earlier are turned off and now forwarded along the North neighboring node. However, there may already be in-transit flits using the old guaranteed tokens corresponding to the old priority setting. In order to ensure such flits do not get stalled due to lookahead conflicts, even though the old guaranteed tokens are turned off they are not forwarded along their new direction and the new priority setting is not applied locally at that router immediately at the end of the epoch e . Rather, each router maintains a transition window equal to $2 \cdot g_{max}$ to cover all potential head flits which may have started up to g_{max} hops away using the old guaranteed token while tracking any outstanding body/tail flits (which are yet to arrive) corresponding to these header flits. When this window is complete and there are no outstanding flits, it is ensured that there will be no more flits arriving at that router using the old guaranteed tokens. It is then safe to apply the new priority settings which are applied and guaranteed tokens are forwarded along the new priority direction.

4.5. Wiring overhead

TFC uses additional wiring to communicate tokens and lookaheads between nodes. Assuming minimal west-first routing using number of free buffers to indicate the more profitable direction and lookahead routing, the wiring required for a token design with $d_{max} = 1$ is the same as that of the baseline design. This overhead, however, increases with d_{max} .

Normal tokens: Indicating greater than a threshold value of free buffers/VCs using normal tokens requires a single-bit wire per port. With minimal west-first routing, a node needs to receive normal tokens only from router ports which lie in either its NE or SE quadrant (along which adaptivity is possible) extending up to d_{max} -hops. One extra wire for every reachable node in the NE quadrant is needed to indicate which one of the N and E ports is more profitable (has more buffers) and similarly for nodes in the SE quadrant. In addition, extra wires proportional to d_{max} are needed to make tokens visible to neighboring nodes in order to enable lookahead routing. Since neighboring nodes in only NE and SE quadrants need to be considered, the total additional wiring overhead of normal tokens is not symmetric across all directions and the per-port overhead at a router, W_{normal} , is given by:

$$W_{normal} = \lceil (3 \cdot (d_{max}^2) + 8 \cdot d_{max} - 11) / 4 \rceil \quad (3)$$

Guaranteed tokens: In contrast to normal tokens, a node receives guaranteed tokens from all nodes in its g_{max} -hop neighborhood with the associated wiring being proportional to the total number of ports reachable in this neighborhood. This overhead when divided across each port at the router, $W_{guaranteed}$, is given by:

$$W_{guaranteed} = g_{max}^2 \quad (4)$$

Lookahead signals: Lookaheads for each flit in the baseline design encode the remaining hops in each dimension and the immediate next hop output port of that flit. In addition to this information, lookaheads in TFC need to encode the flit’s route for an additional $d_{max} - 1$ hops. Hence, using three bits to

Table 2. Additional wiring overhead over the baseline design

$d_{max} (= g_{max})$	W_{normal}	$W_{guarantee}$	$W_{lookahead}$	Total
2	5	4	3	12
3	10	9	6	25

encode each output port in this route leads to an additional per-port wiring overhead, $W_{lookahead}$, given by:

$$W_{lookahead} = 3 \cdot (d_{max} - 1) \quad (5)$$

Table 2 presents the total wiring overhead in addition to the baseline design for different values of $d_{max} > 1$ (assuming $g_{max} = d_{max}$), with the wiring for $d_{max} = 1$ being the same as that of the baseline. As can be seen, this additional overhead is small as compared to the data channel width (which we assume to be 128 bits in this work) for up to reasonable values of d_{max} , and grows as d_{max} increases. However, it should be noted that TFC enables arbitrarily long bypass paths even with a small d_{max} , since multiple small token routes can be chained together without any delay overhead.

4.6. Starvation avoidance

In TFC, since lookaheads carrying tokens are given priority in using their desired switch port over any locally-buffered flits at a router, this can lead to starvation of locally-buffered flits. To avoid such scenarios, each node keeps track of the number of consecutive flits bypassed using tokens for each output port. When this number for a particular output port o at a router r exceeds a pre-defined threshold, str_{off} , and if there are locally-buffered flits at that router which are waiting to use output o , a starvation avoidance mechanism is triggered. Further lookaheads carrying normal tokens, which arrive at r asking for port o , are killed and the corresponding flit is forced to get buffered and go through the normal pipeline at r . However, similar lookaheads carrying guaranteed tokens cannot be killed in between. To prevent starvation due to flits using guaranteed tokens, any guaranteed tokens forwarded by r , along input i which has priority to use output o , are turned off, thereby preventing any new lookaheads carrying guaranteed tokens along the starved route. This is done for str_{cycles} after which forwarding of the turned off guaranteed tokens is resumed. str_{off} and str_{cycles} are pre-defined system parameters.

5. Evaluation

5.1. Comparison with express virtual channels (EVCs)

In addition to comparing TFC against the state-of-the-art baseline design (presented in Section 2), we also compare it against EVCs [6], a recently proposed technique which allow packets to virtually bypass nodes along their path similar to bypassing of nodes in TFC. EVCs allow bypassing using pre-defined virtual lanes in the network which connect distant nodes along a straight line. However, EVC paths have certain limitations. They have fixed endpoints and, in the general case, can only run straight without turning. The maximum length, l_{max} , of an EVC is limited by the amount of buffering available at router ports. In addition, resources such as VCs in an EVC design are partitioned among normal VCs and EVCs of varying lengths which prohibits efficiently sharing them across different paths based on the actual traffic.

While both TFC and EVCs share a common goal of bypassing routers, they are inherently different in their working. The use

Table 3. Network and technology parameters

Technology	65 nm
V_{dd}	1.1 V
$V_{threshold}$	0.17 V
Link length	1 mm
Wire pitch	0.45 μ m
Flit size/channel width	128 bits
Frequency	3 GHz
Topology	7-ary 2-mesh
Routing algorithm	Minimal West-First
Number of message classes	3
VCs per message class	8
Buffers per port	48 (dynamically shared)
Token d_{max}	3
Token g_{max}	3
str_{off}	20
str_{cycles}	3
Epoch e	40
v_{thr}	3

of per-hop tokens in TFC helps improve on EVCs by removing all the above limitations. Bypass paths can be dynamically formed using tokens from one node of the network to any other node allowing them to adapt to a packet’s exact route. Packets can chain a succession of tokens without additional overhead, making token routes highly scalable with arbitrary lengths and including an arbitrary number of turns without being limited by the amount of buffering in the network. Moreover, resources, such as VCs, in TFC are not statically partitioned and can be shared across different token routes on a hop-by-hop basis.

5.2. Simulation setup

We use a detailed cycle-accurate microarchitecture simulator to model network performance. All major components of the router microarchitecture, including RC, buffer management, VA, SA, token generation and forwarding, on/off flow control, etc., are modeled in detail while tracking the different operations a flit and its corresponding lookahead go through every cycle as they traverse the data and lookahead pipeline stages, respectively. Contention for resources, such as buffers, VCs, crossbar input and output ports, etc., are accurately modeled with the delay of each pipestage derived from detailed circuit-level schematics, similar to the approach in [9]. To evaluate network energy, we use Orion [17], an architecture-level network energy model, which is integrated in our performance model to keep track of both dynamic and leakage energy of all major microarchitectural components, including buffers, crossbar, allocation logic and network links. All designs incorporate power-optimized straight-through buffer and cut-through crossbar designs [5].

Table 3 lists the default network and technology parameters used in this study. These parameters were chosen for the best baseline as well as EVC performance and unless otherwise mentioned, all experiments use these parameters. The comparison with EVCs uses a dynamic EVC design (in which every node is a source/sink of variable-length EVCs) using the non-aggressive express pipeline since the design with an aggressive express pipeline incurs additional area overhead in the form of extra straight-through links around the crossbar in order to allow flits to skip ST as well when bypassing a router [6]. It should be noted, however, that using an aggressive design with extra straight-through link bandwidth will benefit TFC as well. The maximum EVC length l_{max} was chosen to be three and the design incorporates the routing flexibility feature proposed

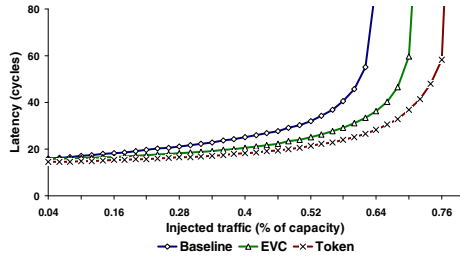


Figure 7. Uniform random traffic

in [6] which allows packets to switch between EVCs of varying lengths to reduce VC contention. All runs using synthetic traffic are 1 million cycles long with a 100k cycles warm-up period and use a mix of single-flit short packets and five-flit long packets spread across three protocol message classes. In all results, we define network saturation to be the point where flit latency becomes three times the no-load latency. Network capacity is calculated based on the channel load of the most heavily utilized links in the network based on the traffic pattern.

5.3. Uniform random traffic

Fig. 7 presents network performance using uniform random traffic by plotting flit latencies as a function of injected load using the default parameters presented in Table 3. When the network load is low, the baseline latency is close to that of EVCs and TFC because of the use of lookaheads for no-load bypassing when router ports are mostly empty. However, at higher values of network load, TFC significantly outperforms both the baseline and EVC designs in terms of latency while simultaneously improving throughput. As compared to the baseline, TFC leads to 51.7% reduction in latency near the baseline saturation point. When compared to EVCs, TFC reduces latency by 38.2% near the EVC saturation point. This is mainly attributable to the flexible nature of TFC which allows packets to choose tokens that exactly match their route and bypass routers along them. Moreover, unlike EVCs, packets are able to bypass routers even while turning from one dimension to another. At very high load, when there are few normal tokens available due to lack of sufficient resources, TFC continues to show lower latency and higher throughput because of route padding which is used to still compute up to d_{max} -hop long routes using X-Y routing (Section 4.3) and, lookaheads, which may not be carrying tokens, are sent along these routes to bypass routers.

In terms of energy, while EVCs lead to a 26.5% reduction in router energy over the baseline near the baseline saturation point, this reduction is 37.3% using tokens. This again is mainly due to packets bypassing more nodes without having to get buffered. Specifically, while EVCs are able to reduce buffer energy by 45.1% as compared to the baseline, this reduction is a significant 68% using tokens.

Both EVCs and TFC allow packets to bypass nodes non-speculatively even under medium/high loads and, hence, lead to latency and throughput improvements. But while EVCs allow bypassing only along restricted straight line paths, tokens allow packets to choose less congested routes in the network and then bypass nodes along them including along turns. When compared to the ideal packet-switched design, where packets can bypass all intermediate nodes along their route, tokens allow packets to

bypass 87.1% of all intermediate nodes near baseline saturation.

5.4. Effect of fewer resources

The significantly higher throughput achieved by TFC, as compared to the baseline, allows for an energy-performance trade-off where the same throughput as that of the baseline can be attained with significantly smaller amount of buffering in the network. Here, we study this effect by starting with a leaner router design with 4 VCs per message class and 40 buffers per port⁶, and gradually reducing the buffers per port in both the token and EVC designs, until their throughput became equal to that of the baseline. We found that while the EVC design achieved the same baseline throughput with a significantly lower 25 buffers per port (leading to a 53.1% reduction in dynamic buffer energy and a 28.4% reduction in leakage energy as compared to the baseline), the token design achieved the same baseline throughput with an even lower 14 buffers per port (leading to a dynamic buffer energy reduction of 79.4% and a leakage energy reduction of 48.8% as compared to the baseline). This translated to a total router energy reduction (as compared to the baseline) of 35.6% using EVCs while the corresponding reduction using tokens was 55.4%. Naturally, this reduces the area footprint as well.

5.5. Non-uniform traffic

Fig. 8 shows network performance for bit-complement traffic. While EVCs lead to a 18.4% latency reduction (24.9% lower router energy) over the baseline near saturation, tokens lead to a higher latency reduction by 56.9% (39.6% lower router energy) as well as push throughput. For this traffic, this is mainly because EVCs only allow straight bypass paths in the network, thus leading to a higher queuing imbalance at certain nodes where the fraction of turning traffic is high. In contrast, token bypass paths can turn and exhibit better queuing behavior by using dynamic switch port priority settings at each router which are changed based on the observed traffic, thereby balancing the queue lengths at different input ports in each router. For transpose traffic (Fig. 9), the ability of tokens to allow packets to take less congested routes in the network in addition to bypassing routers along those routes leads to a higher latency reduction of 20.6% over the baseline near saturation with the corresponding reduction in router energy being 29.3% (as compared to 19.9% lower energy using EVCs). Tornado traffic which involves packets going half-way around the mesh in a straight line represents the best-case for EVCs and as shown in Fig. 10, they lead to significant latency (48.7%) and router energy reduction (25.4%) over the baseline near saturation. While tokens also lead to a significant corresponding latency (48%) and router energy reduction (33.4%), their saturation throughput is slightly lower than EVCs. Analyzing this further, we found that while seamless chaining of successive token routes allows packets to bypass a higher fraction of the total nodes along their route (75.3%) as compared to 43% nodes bypassed in EVCs (also leading to a correspondingly lower dynamic router energy), bypassing more nodes does not always

6. This number was chosen to minimize buffer contention in the baseline since we saw the baseline performance degrading significantly below 40 buffers.

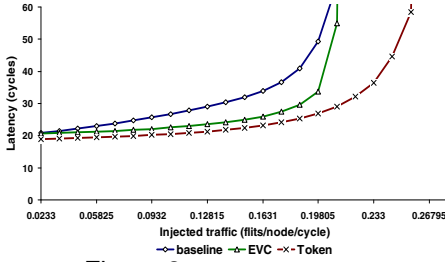


Figure 8. Bit-complement traffic

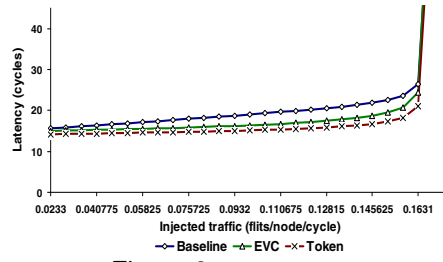


Figure 9. Transpose traffic

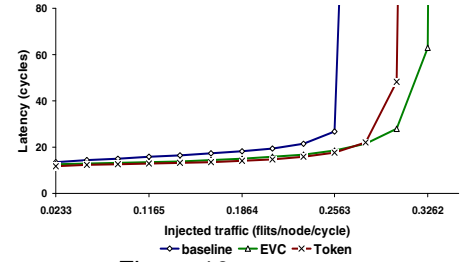


Figure 10. Tornado traffic

improve latency for this traffic at high loads. This is because nodes at the center of each row in the network see a lot of bypassing traffic causing their local flits to wait longer for using the switch leading to more occurrences of starvation. This effect can be alleviated through a more dynamic starvation-avoidance policy based on the observed queuing at nodes instead of using fixed thresholds to signal starvation and is left for future work.

5.6. Effect of varying d_{max}

As mentioned in Section 3.2, packets use guaranteed tokens only along their already computed d_{max} -hop route (with g_{max} being equal to d_{max} in our design). Hence, a larger d_{max} has two effects: it not only allows more adaptive routes by providing packets with normal tokens from a larger neighborhood of nodes but also provides packets the opportunity to use longer-hop guaranteed tokens to skip over more number of nodes in trying to get out of congested network sections at high loads. It should be noted that a smaller d_{max} does not necessarily mean bypassing of a smaller number of nodes because even in a token design with $d_{max} = 1$, packets can bypass more than one node in one go by chaining successive tokens at every hop without any delay overhead. Fig. 11 highlights this effect by presenting network performance for increasing values of d_{max} using tornado traffic. The performance at low and medium loads is similar and not affected due to smaller d_{max} values. This is because packets are still able to bypass a high fraction of the total nodes along their path by chaining several tokens (72.6% of the total nodes bypassed for $d_{max} = 1$, 74.2% for $d_{max} = 2$ and 75.3% for $d_{max} = 3$). However, as network load increases, a higher d_{max} ($= g_{max}$) leads to a higher saturation throughput. Since the tornado traffic pattern has packets traveling only along straight line, there is no adaptivity possible in packet routes when using minimal routing and hence the performance improvement at high loads is not due to increased adaptivity using normal tokens but solely due to the ability to use longer-hop guaranteed tokens to bypass several congested nodes in a guaranteed fashion. Hence, this depicts a scenario where guaranteed tokens benefit performance in a design with higher d_{max} ($= g_{max}$). In terms of energy consumption, since the number of nodes bypassed remains almost the same, there is almost no impact on dynamic router energy due to smaller d_{max} .

5.7. Flow control vs. routing

The benefits of TFC consist of both its adaptive routing aspect of choosing better routes using tokens and its flow control aspect of bypassing routers along the token routes. To study the impact of the flow control aspect, we use a token design, *token_no_bypassing*, which allows packets to compute

up to d_{max} -hop routes adaptively using tokens similar to the default TFC. But unlike the default TFC *token_no_bypassing* disables bypassing of routers along the token routes. Fig. 12 presents network performance for uniform traffic comparing *token_no_bypassing* with the default TFC (with $d_{max} = 3$). The figure shows the default TFC significantly outperforming *token_no_bypassing* (44.7% latency reduction near saturation), thereby implying that bypassing routers along token routes is a major contributor towards overall performance.

To study the adaptive routing aspect, we compared a design using dimension-ordered routing (DOR) when computing d_{max} -hop token routes versus the default scheme which uses adaptive minimal west-first routing. Fig. 13 plots this effect for both the baseline and TFC (with $d_{max} = 3$) using the transpose traffic pattern which benefits from adaptive routing. As seen, the performance using adaptive routing is better than DOR for both the baseline (18.9% lower latency near saturation), as well as for TFC (20.4% lower latency near saturation).

5.8. SPLASH benchmark results

TFC leads to significant improvements in energy/delay as compared to the baseline across a range of SPLASH-2 benchmarks (gathered using the methodology described in [6]). The reduction in latency is largest for the *water-spatial* benchmark (77.1%) mainly due to high traffic load at which the baseline exhibits large queuing delays while TFC continues to show lower latency by allowing packets to bypass nodes along token routes which match their path. Similarly, *water-nsquared* also exhibits relatively high traffic with the latency reduction using TFC being 69.6%. On the other extreme is *raytrace* which has very low traffic load at which no-load bypassing in the baseline is effective and hence the latency reduction due to TFC is only 2%. The other benchmarks – *lu*, *ocean*, *radix*, *barnes* and *fft* represent medium-load scenarios with TFC reducing latencies by 25.5%, 26.2%, 39.2%, 24% and 22%, respectively. TFC is also able to significantly reduce average router energy with the reductions being 24.5%, 21.6%, 22.9%, 25.1%, 20.6%, 23.5% and 15.4% for *lu*, *ocean*, *radix*, *barnes*, *water-nsquared*, *water-spatial* and *fft*, respectively. This is mainly due to lower buffer energy by eliminating the need for flits to get buffered when they are bypassing routers along token routes. For *raytrace*, the energy reduction is only 2.3%, again due to low traffic in this benchmark at which baseline’s no-load bypassing is effective.

6. Related work

Flow control: Circuit switching [7] uses pre-reservation of physical channels from the source to destination before the actual data transfer begins. This can approach link energy/delay once data transfer starts but incurs a large delay overhead in

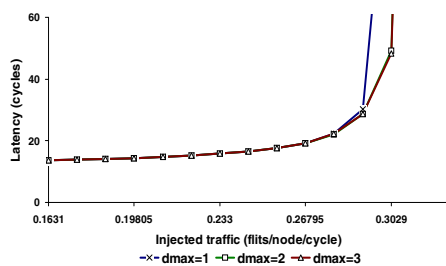


Figure 11. Effect of varying d_{max}

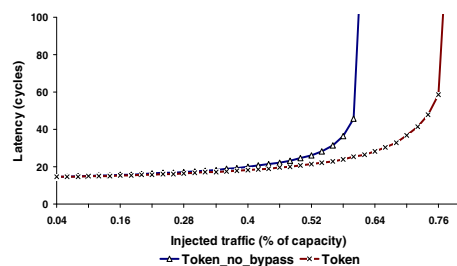


Figure 12. Effect of flow control

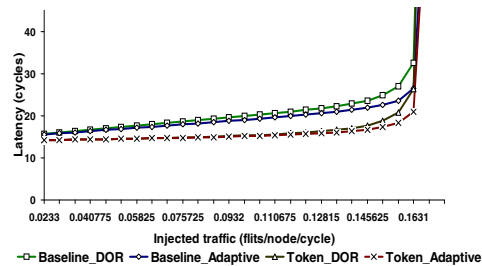


Figure 13. Effect of routing

setting up and tearing down circuits while inefficiently sharing available bandwidth by dedicating channels to particular packet flows. Variants of this such as pipelined circuit switching [18] and hybrid circuit switching [19] have been proposed which allow bandwidth sharing across multiple circuits. Unlike circuit switching and its variants, TFC allows packets to bypass routers along their path without any setup/tear-down delay while allowing complete sharing of link bandwidth across different flows.

Topology: TFC is completely orthogonal to network topology. Even though a mesh topology was chosen in this work because of its simplicity and ubiquity [1]–[3], tokens can significantly reduce per-hop router overhead (which dominates link overhead in on-chip networks) by allowing packets to bypass a bulk of the intermediate routers along their path irrespective of topology and can complement topologies which target lower hop counts such as high-radix [20] and butterfly [7] designs.

Techniques targeting router delay: Techniques such as speculation [11], [12], bypassing [21], lookahead pipelines [9], [13], simplified VA [9] and lookahead routing [8] which improve router delay are already incorporated in both the baseline and token designs. While these techniques drive towards ultra-low router latency, they only succeed in bypassing the router pipeline at low loads, performing poorly when network contention is high. TFC, however, allows router bypassing at all traffic loads.

Adaptive routing: A recent work by Gratz et al. [16] proposes adaptive routing using global status information aggregated across multiple hops. While this work has similarities with TFC in its propagation of status information and use in adaptive routing, the key distinction is that [16] is solely a routing algorithm; it does not allow packets to bypass the router pipeline. In contrast, like other flow control techniques, e.g., credit-based, TFC manages allocation of resources such as buffers, switch and link bandwidth. Specifically, router bypassing using tokens allows packets to skip buffering and get priority in winning the switch/link. TFC, in general, can work with any routing scheme.

7. Conclusion

Communication overhead in state-of-the-art packet-switched on-chip networks is dominated by the energy/delay incurred in traversing complex router pipelines at every hop. In this work, we proposed TFC which allows packets to use tokens, which are indications of resource availability at neighboring routers, to find routes along which intermediate nodes can be bypassed. The high flexibility and seamless chaining of token routes enables packets to bypass all nodes between their source and destination in the best case. We presented a detailed implementation of TFC along with a detailed evaluation showing tokens providing significant energy-delay-throughput gains over existing designs.

Acknowledgments

The authors would like to thank Partha Kundu of Intel Corp. for useful feedback and assistance with the simulation infrastructure. This work was supported in part by NSF (grant CCF-0702110), MARCO Gigascale Systems Research Center, MARCO Interconnect Focus Center, SRC (contract 2008HJ1793) and an Intel PhD Fellowship.

References

- [1] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, S. W. Keckler, D. Burger, and C. R. Moore, “Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture,” in *ISCA*, June 2003.
- [2] M. B. Taylor, W. Lee, J. Miller, D. Wentzloff, I. Bratt, B. Greenwald, H. Hoffman, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpfen, M. Frank, S. Amarasinghe, and A. Agarwal, “Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams,” in *ISCA*, June 2004.
- [3] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, “A 5-GHz mesh interconnect for a teraflops processor,” *IEEE Micro*, vol. 27, no. 5, Sept. 2007.
- [4] L. Benini and G. De Micheli, “Networks on chips: A new SoC paradigm,” *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
- [5] H.-S. Wang, L.-S. Peh, and S. Malik, “Power-driven design of router microarchitectures in on-chip networks,” in *MICRO*, Nov. 2003.
- [6] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, “Express virtual channels: Towards the ideal interconnection fabric,” in *Proc. ISCA (and IEEE Micro Top Picks 2008)*, June 2007.
- [7] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004.
- [8] M. Galles, “Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip,” in *HotI*, Aug. 1996.
- [9] A. Kumar, P. Kundu, A. Singh, L.-S. Peh, and N. K. Jha, “A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS,” in *ICCD*, Oct. 2007.
- [10] Y. Tamir and G. L. Frazier, “Dynamically-allocated multi-queue buffers for VLSI communication switches,” *IEEE Trans. Computers*, vol. 41, no. 6, June 1992.
- [11] R. Mullins, A. West, and S. Moore, “Low-latency virtual-channel routers for on-chip networks,” in *ISCA*, June 2004.
- [12] L.-S. Peh and W. J. Dally, “A delay model and speculative architecture for pipelined routers,” in *HPCA*, Jan. 2001.
- [13] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, “Implementation and evaluation of on-chip network architectures,” in *ICCD*, Oct. 2006.
- [14] C. J. Glass and L. M. Ni, “The turn model for adaptive routing,” in *ISCA*, May 1992.
- [15] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, “A low latency router supporting adaptivity for on-chip interconnects,” in *DAC*, June 2005.
- [16] P. Gratz, B. Grot, and S. W. Keckler, “Regional congestion awareness for load balance in networks-on-chip,” in *HPCA*, Feb. 2008.
- [17] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, “Orion: A power-performance simulator for interconnection networks,” in *MICRO*, Nov. 2002.
- [18] P. T. Gaughan and S. Yalamanchili, “A family of fault-tolerant routing protocols for direct multiprocessor networks,” *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 5, May 1995.
- [19] N. E. Jerger, M. Lipasti, and L.-S. Peh, “Circuit-switched coherence,” in *Proc. Int. Symp. Networks-on-Chip*, Apr. 2008.
- [20] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, “Microarchitecture of a high-radix router,” in *ISCA*, June 2006.
- [21] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, “The Alpha 21364 network architecture,” *IEEE Micro*, vol. 22, no. 1, pp. 26–35, Jan./Feb. 2002.