Application-Aware Deadlock-Free Oblivious Routing

Michel Kinsy, Myong Hyo Cho, Tina Wen,
Edward Suh (Cornell University), Marten van Dijk
and Srinivas Devadas

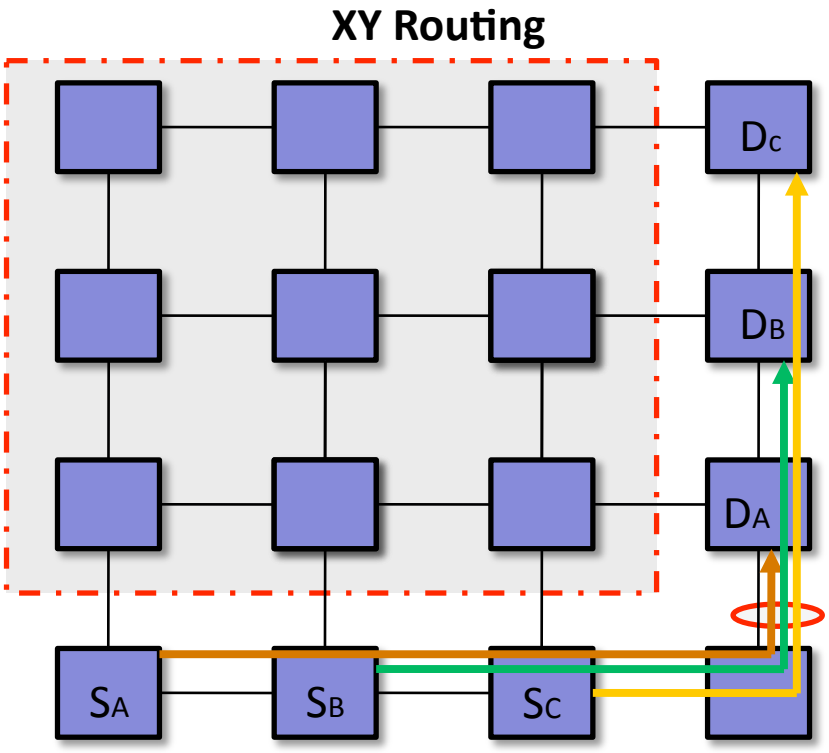
Massachusetts Institute of Technology

Outline

- Routing algorithms and Motivation
- Application-Aware Oblivious Routing
- Bandwidth-Sensitive Routing Approach
- Router Architecture and Performance Analysis
- Plans for the Future

Oblivious Routing

- Statically determined given the source and destination addresses

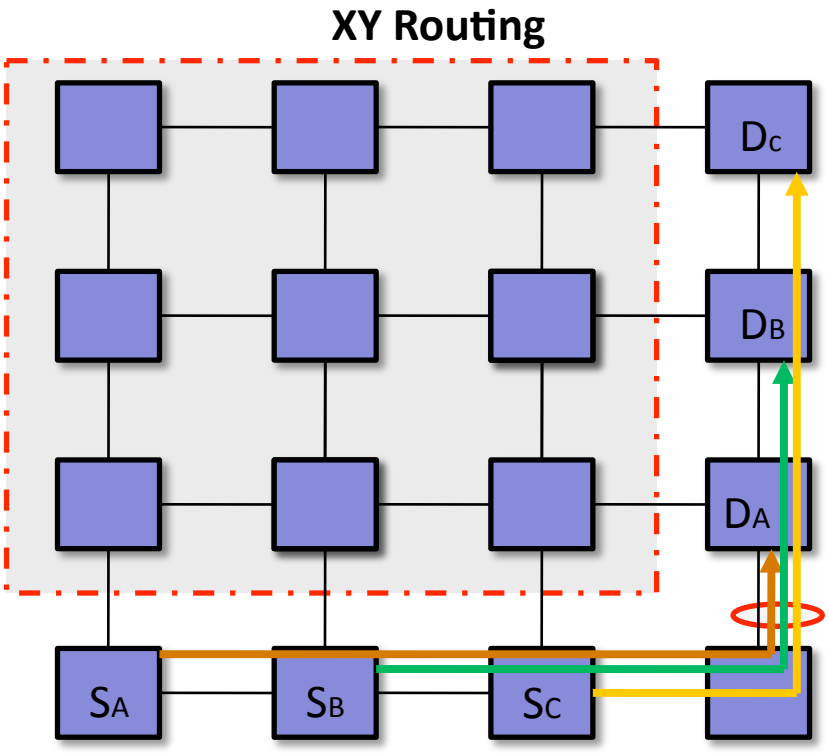


Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

Oblivious Routing

- Statically determined given the source and destination addresses

(+) Simple and fast router designs



Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

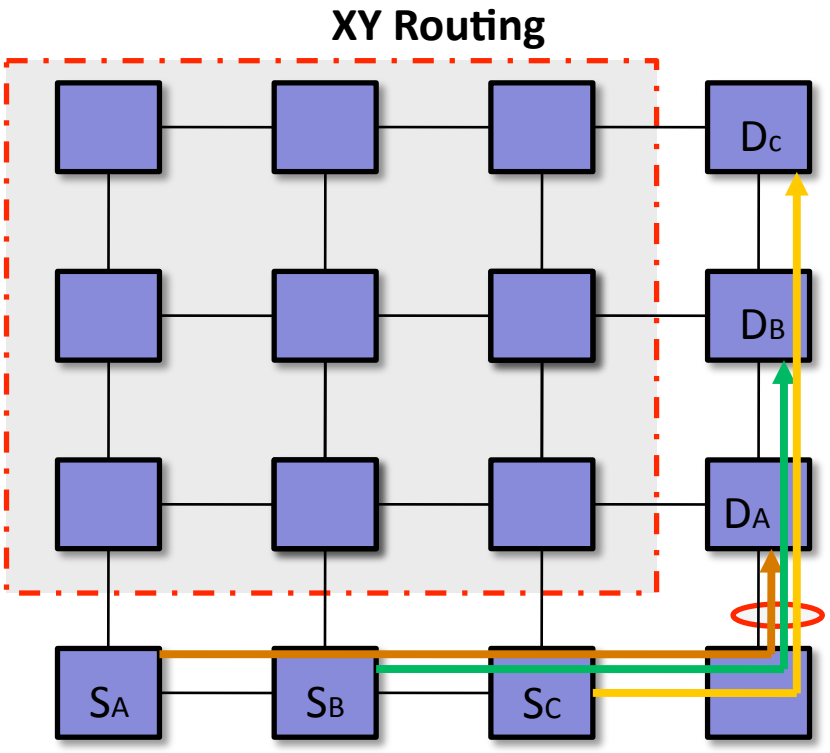
Oblivious Routing

- Statically determined given the source and destination addresses

(+) Simple and fast router designs

(-) Lead to network under-utilization

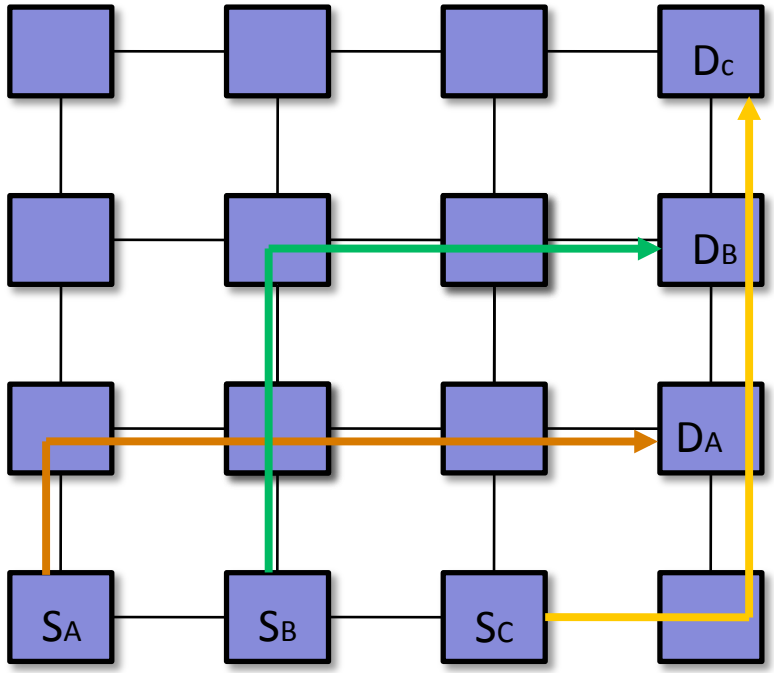
(-) Lack proper load balancing



Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

Adaptive Routing

- Routes dynamically adjusted based on network status



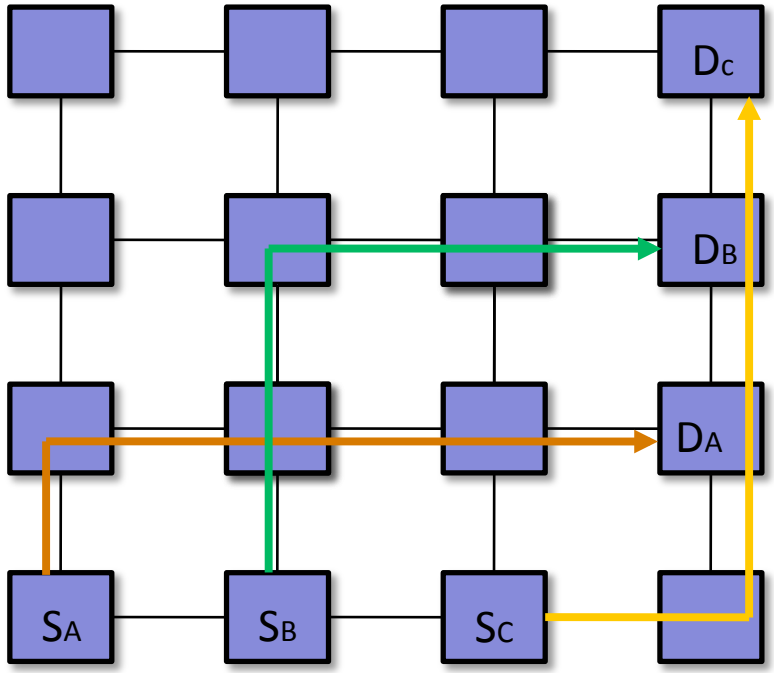
Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

Adaptive Routing

- Routes dynamically adjusted based on network status

(+) Better load balancing and path diversity

(+) Potentially better throughput and latency



Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

Adaptive Routing

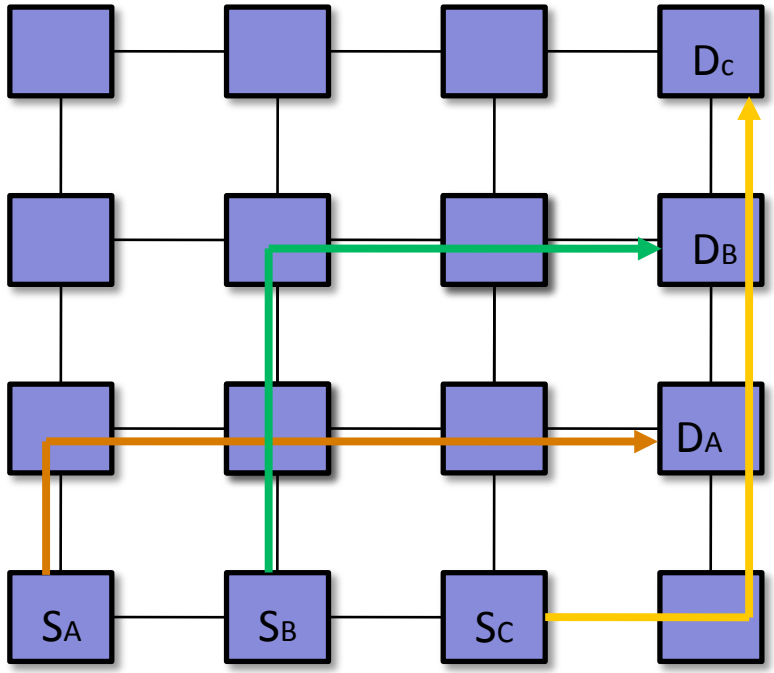
- Routes dynamically adjusted based on network status

(+) Better load balancing and path diversity

(+) Potentially better throughput and latency

(-) Need for global or local knowledge of network conditions

(-) Router complexity



Link Capacity 50 Mbytes/sec
Each flow has 25 Mbytes/sec bandwidth demand

Motivation

- Can we get the best of both worlds?

  (+) Simple and fast router designs

  (+) Better load balancing

  (+) Potentially better throughput and latency

Motivation

Given an application, with knowledge of data communication patterns, can we determine a set of static routes that performs better than conventional oblivious routing?

- ➡ Exploit knowledge of bandwidth demands (or latency requirements)
- ➡ Ensure deadlock freedom

Platforms and Suitable applications

- Suitable for applications with predictable communication patterns
  - video compression
  - processor simulation
  - rendering

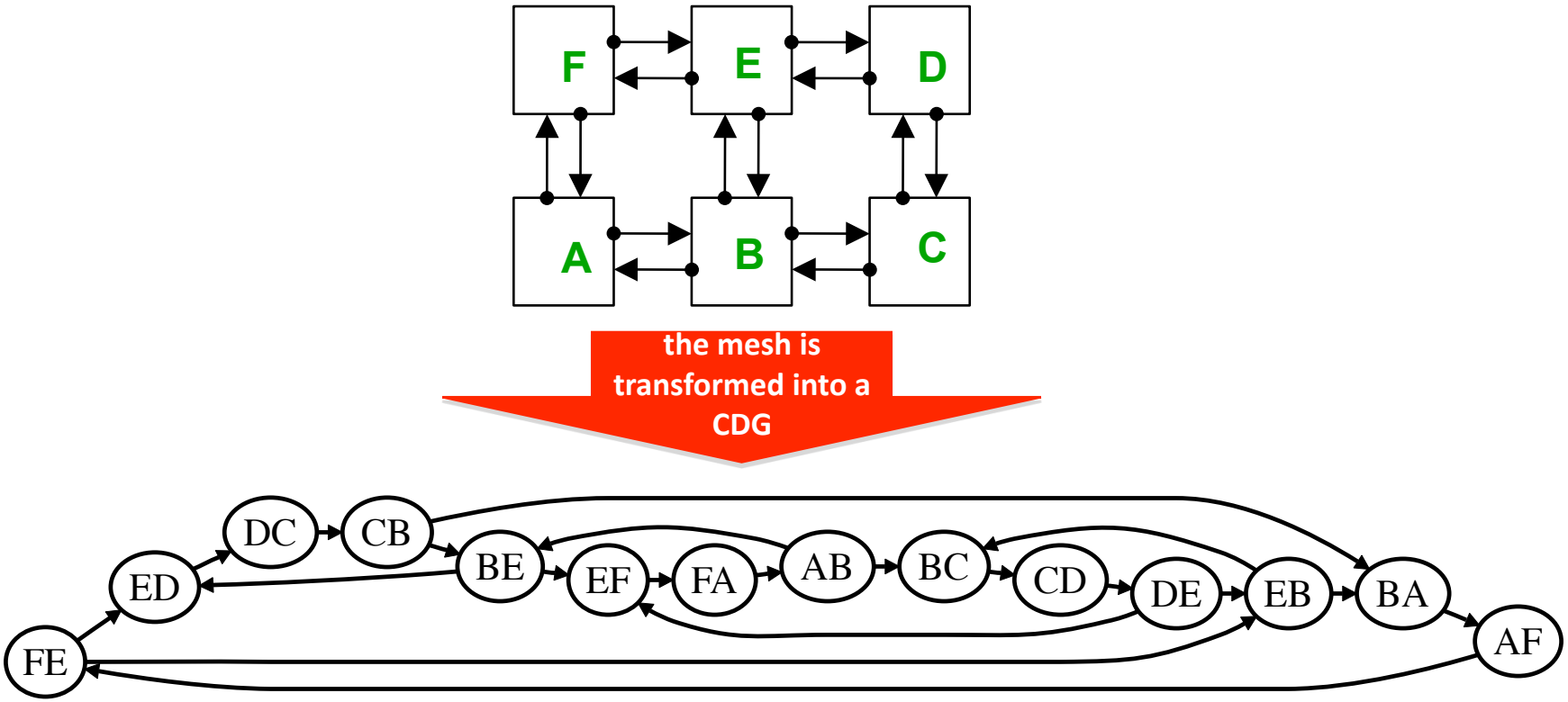
- Reconfigurable hardware: processing elements and their interconnection network can be configured

Outline

- Routing algorithms and Motivation
- Application-Aware Oblivious Routing
- Bandwidth-Sensitive Routing Approach
- Router Architecture and Performance Analysis
- Plans for the Future

Application-Aware Routing Framework

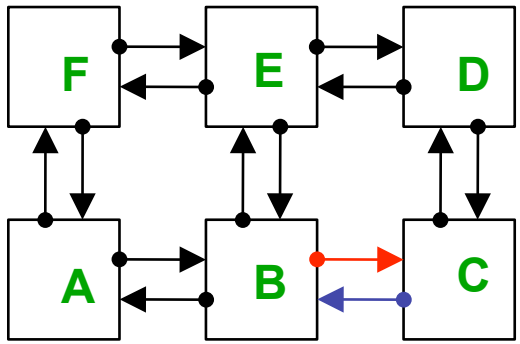
Step 1: *Use the targeted network topology and resources (e.g., buffer space) to create a conventional channel dependency graph (CDG) D of the network.*



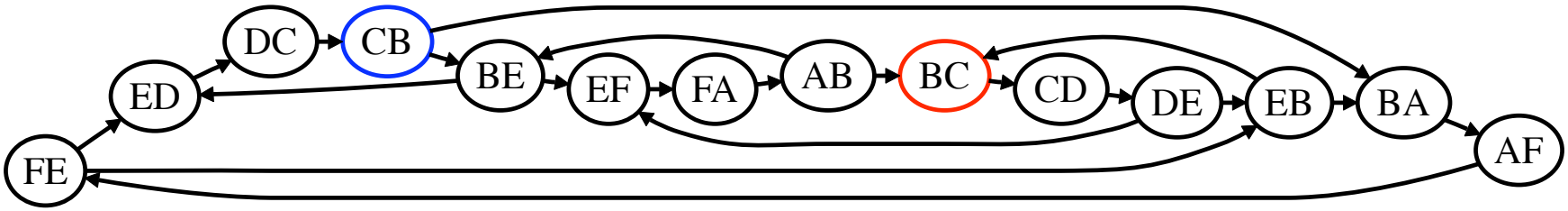
Application-Aware Routing Framework

Step 1: *Use the targeted network topology and resources (e.g., buffer space) to create a conventional channel dependency graph (CDG) D of the network.*

Vertices in the CDG represent network *links*



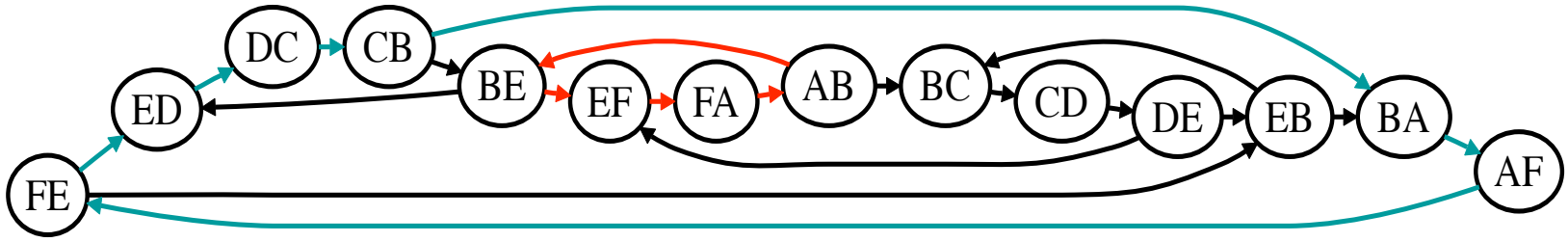
the mesh is transformed into a CDG



Application-Aware Routing Framework

Step 2: *Create (new) acyclic CDG D_A by deleting some edges from D*.

Because the channel dependency graph D derived from the network topology may contain many cycles

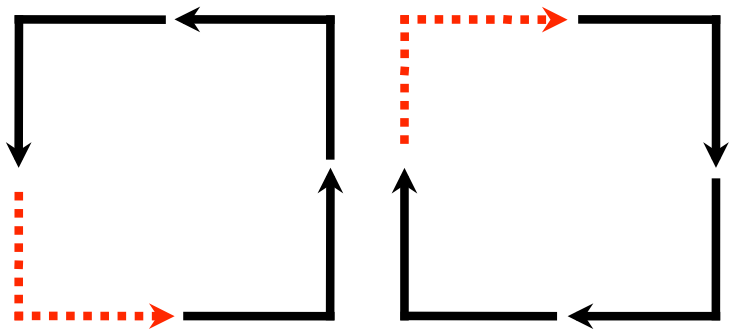


Well known result: Having cycle-free dependency graph ensures deadlock freedom

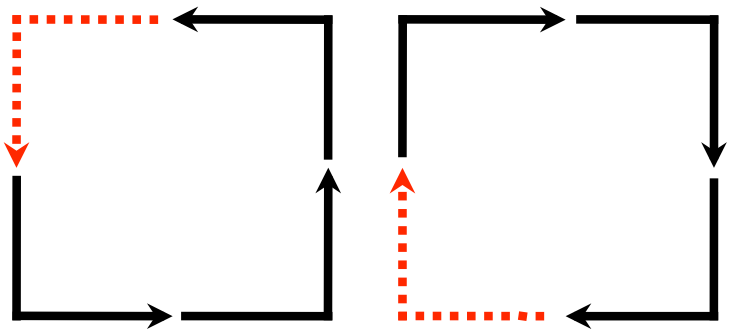
Turn Model (Glass and Ni, 1994)

- A systematic way of generating deadlock-free routes with small number of prohibited turns

- Deadlock-free if routes conform to at least ONE of the turn models (acyclic channel dependence graph)



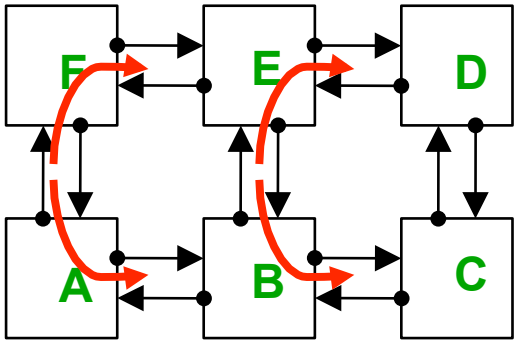
West-First Turn Model

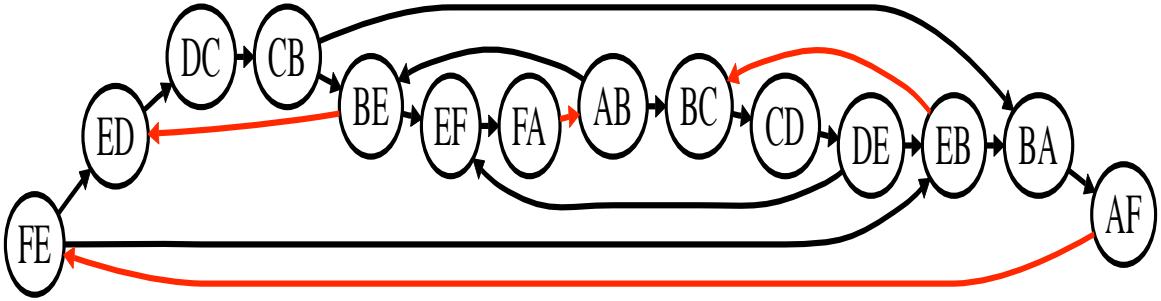


North-Last Turn Model

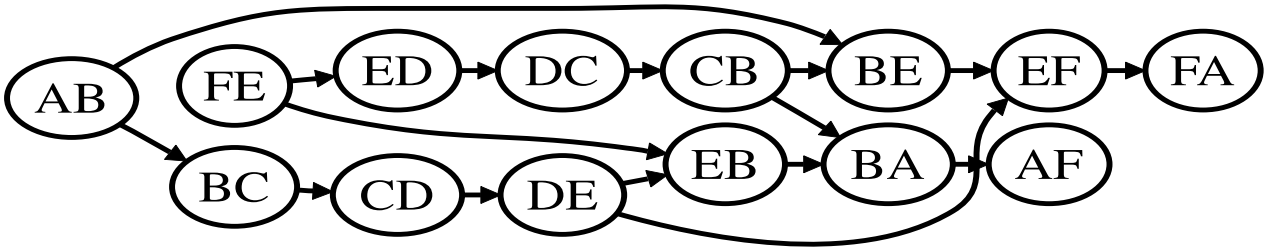
Acyclic CDG ⇒ Deadlock-free routes

Per the North-Last prohibited turns, all the edges in red are deleted



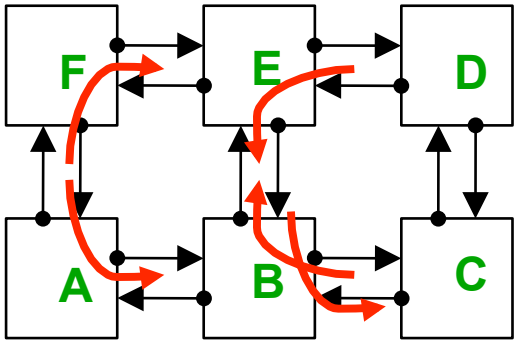


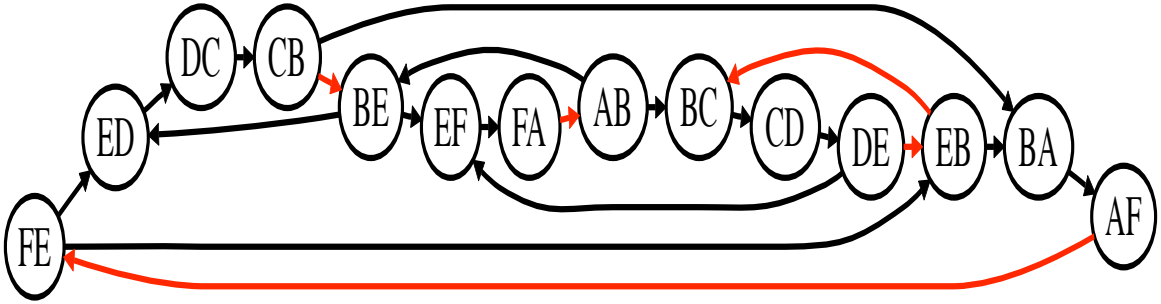
North-Last Acyclic CDG



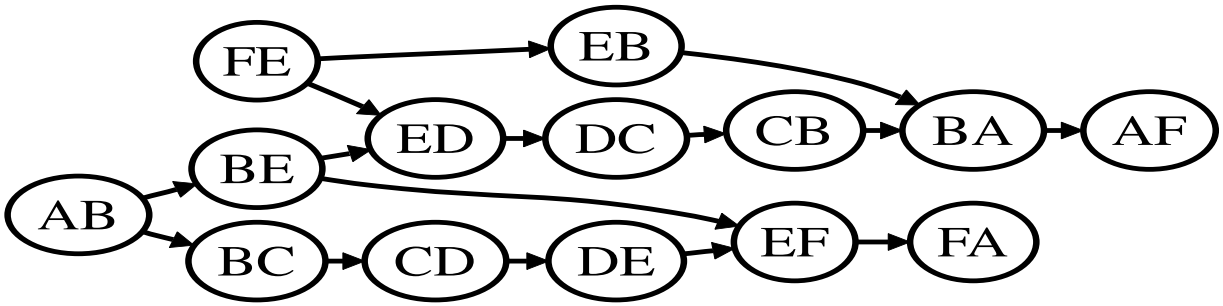
Acyclic CDG ⇒ Deadlock-free routes

Turns could be prohibited at ad-hoc, all the edges in red are deleted



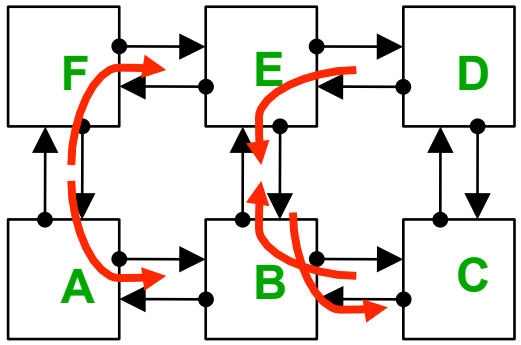


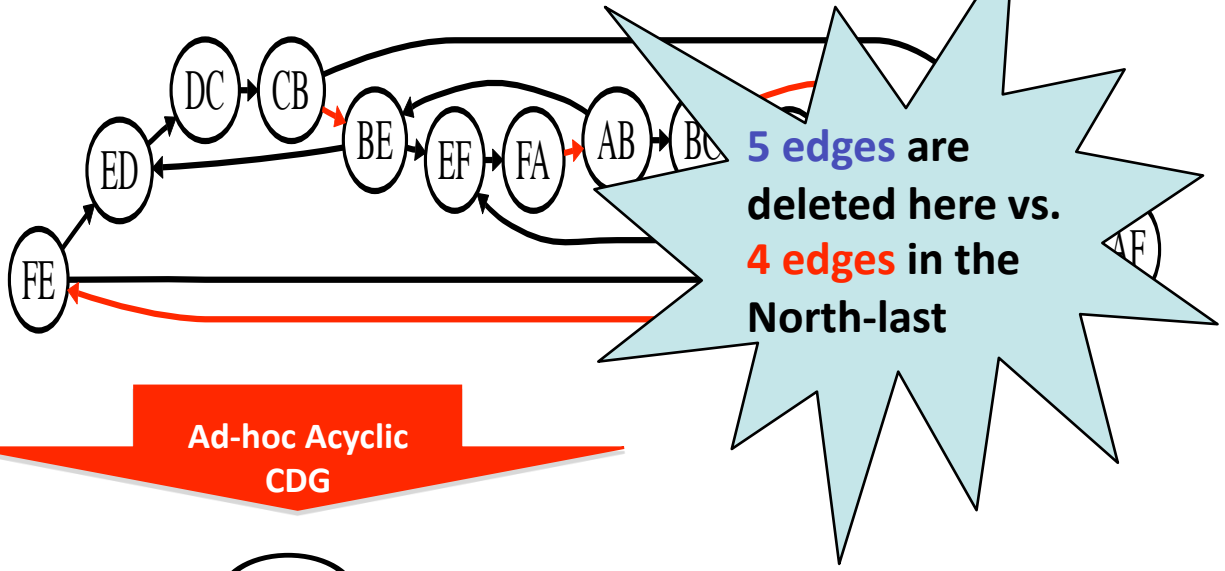
Ad-hoc Acyclic CDG



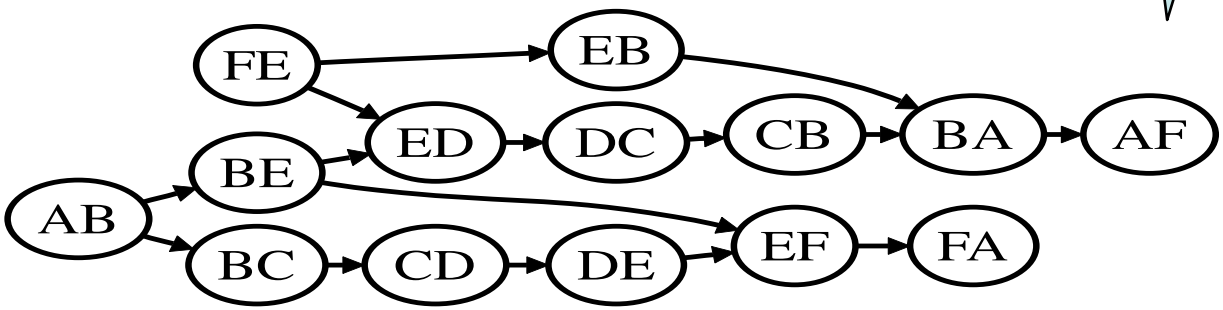
Acyclic CDG ⇒ Deadlock-free routes

Turns could be prohibited at ad-hoc, all the edges in red are deleted.





Ad-hoc Acyclic CDG



Application-Aware Routing Framework

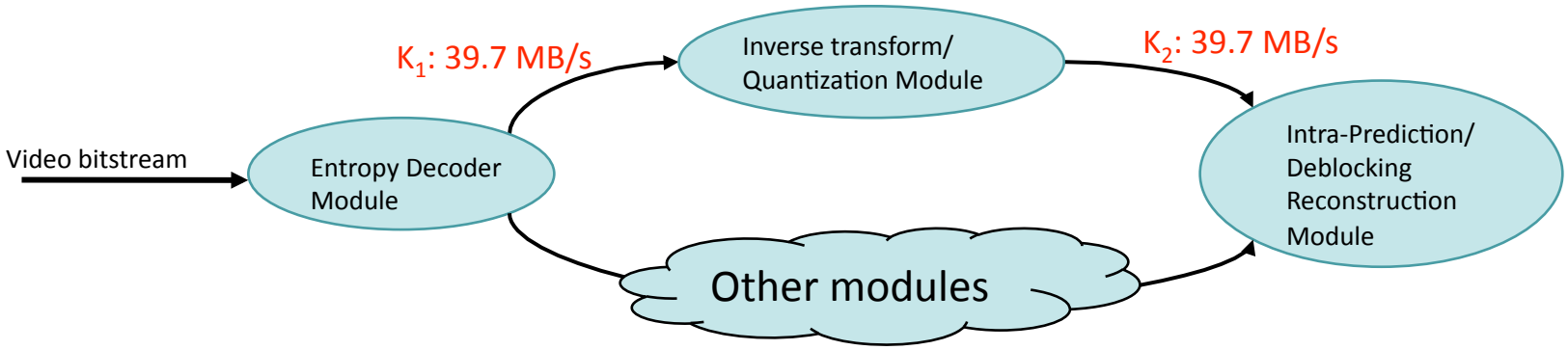
Step 3: *Transform D_A into a flow network G_A, given a set of k flows denoted K.*

Flows $K = \{K_1, K_2, \ldots, K_k\}$. $K_i = (s_i, t_i, d_i)$, where s_i and t_i are the source and sink, for connection i, and d_i is the demand

Application-Aware Routing Framework

Step 3: *Transform D_A into a flow network G_A, given a set of k flows denoted K.*

Part of the modular decomposition of the H.264 decoder, with the following estimated bandwidths and placement:

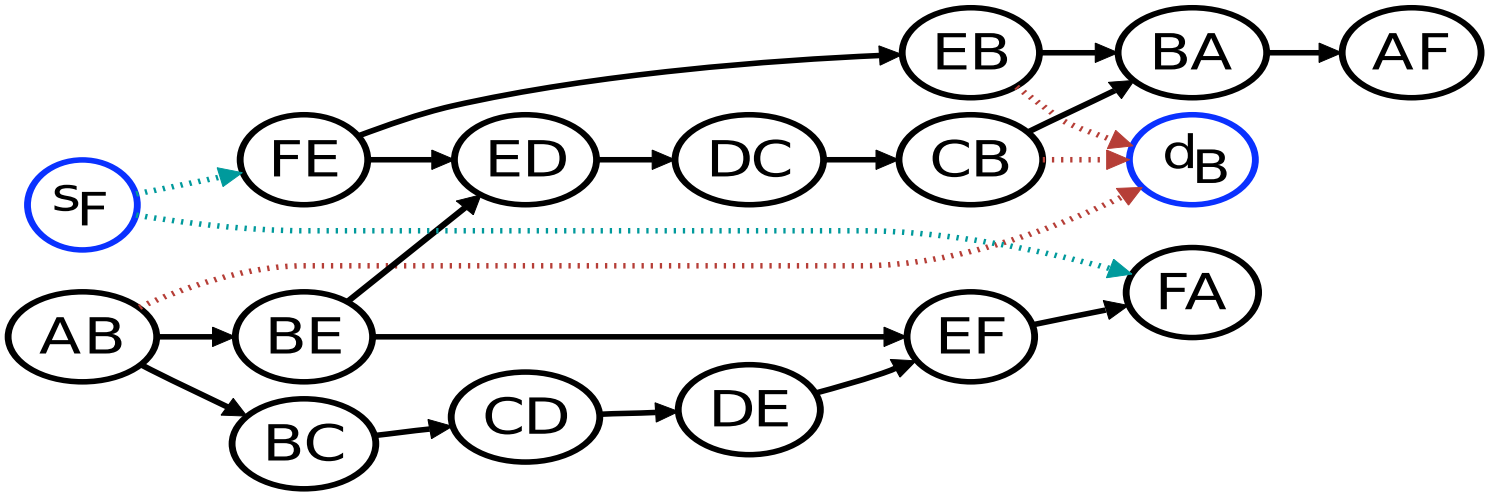


Flow ID	Source	Destination	Demands
K_1	F	B	39.7 MB/s
K_2	B	D	39.7 MB/s

Application-Aware Routing Framework

Step 3: *Transform D_A into a flow network G_A, given a set of k flows denoted K.*

- Flows are routed on CDG not on network
- To routing K_1 = (F, B, 39.7 MB/s) on the ad-hoc acyclic CDG

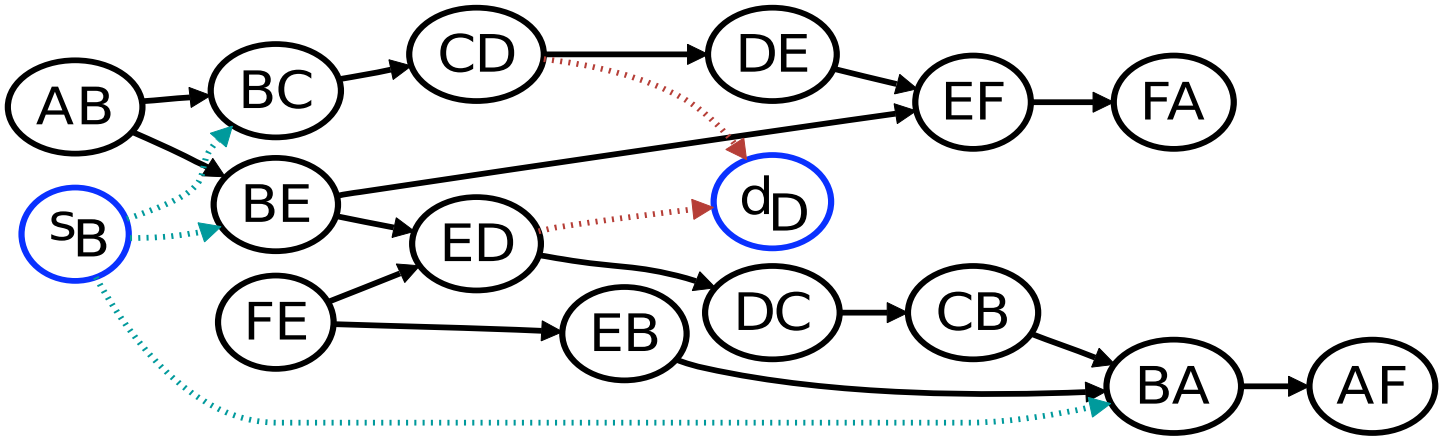


Dummy nodes s_F and d_B are created to drive flow K_1 from its source F and to sink it into B

Application-Aware Routing Framework

Step 3: *Transform D_A into a flow network G_A, given a set of k flows denoted K.*

- Flows are routed on CDG not on network
- To routing K_2 = (B, D, 39.7 MB/s) on the ad-hoc acyclic CDG

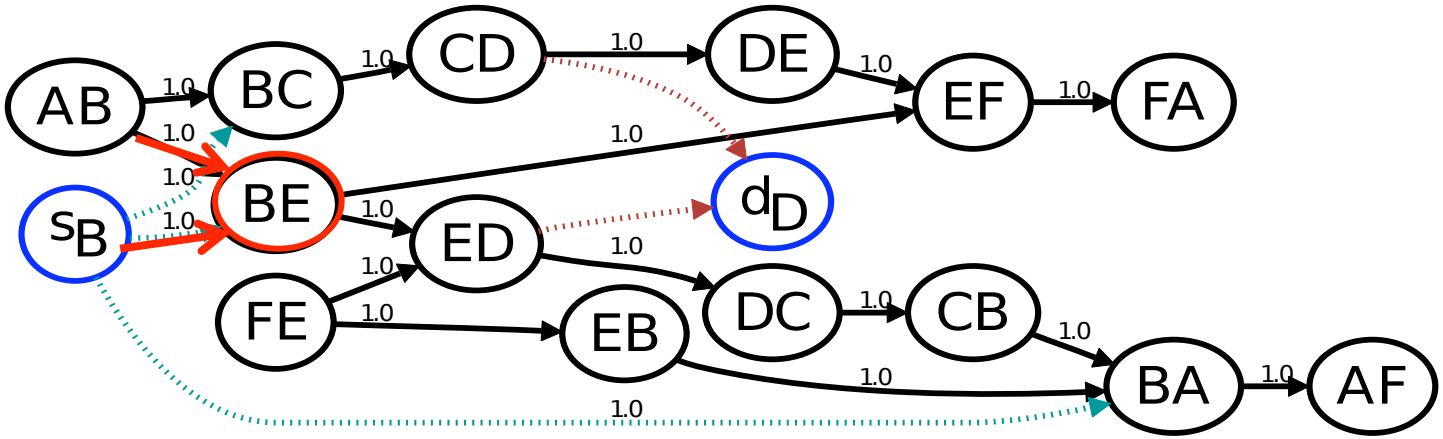


Dummy nodes s_B and d_D are created to drive flow K_2 from its source B and to sink it into D

Application-Aware Routing Framework

Step 3: *Transform D_A into a flow network G_A, given a set of k flows denoted K.*

- Flows are routed on CDG not on network
- To routing K_2 = (B, D, 39.7 MB/s) on the ad-hoc acyclic CDG



- Edges into BE are assigned the capacity of link BE in the mesh
- No capacity or weight is assigned to the edges incident on sink nodes

Application-Aware Routing Framework

Step 4: *Perform application-aware routing of the flows in* G_A .

Step 5: *If desired, go to Step 2 and repeat using a different acyclic CDG.*

Step 6: *Select the best set of routes found , per the routing function used in Step 4.*

- In Step 4, bandwidth-sensitive routing can be used as a type of application-aware routing scheme

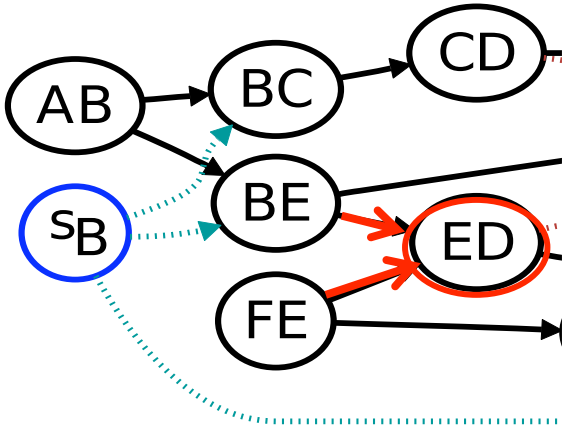
Outline

- Routing algorithms and Motivation
- Application-Aware Oblivious Routing
- Bandwidth-Sensitive Routing Approach
- Router Architecture and Performance Analysis
- Plans for the Future

Bandwidth-Sensitive Oblivious Routing (BSOR)

Goal: Route flows while minimizing the maximum channel load (MCL) U in the network:

minimize $$U = \max_{v \in E} \sum_{i=1}^{k} \sum_{(u,v) \in E} f_i(u,v)$$



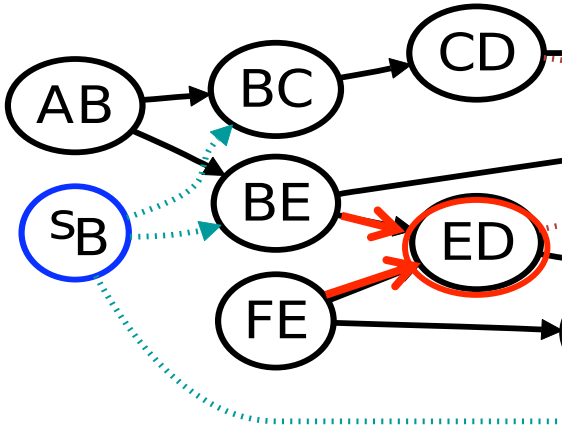
v is a link/vertex in the CDG (e.g., ED)

$f_i(u, v)$ is the edge's bandwidth used by flow i (e.g., $f_1(BE, ED) = 0$ where $f_2(BE, ED) = 39.7$)

Bandwidth-Sensitive Oblivious Routing (BSOR)

Goal: Route flows while minimizing the maximum channel load (MCL) U in the network:

minimize $$U = \max_{v \in E} \sum_{i=1}^{k} \sum_{(u,v) \in E} f_i(u,v)$$



v is a link/vertex in the CDG (e.g., ED)

$f_i(u, v)$ is the edge's bandwidth used by flow i (e.g., $f_1(BE, ED) = 0$ where $f_2(BE, ED) = 39.7$)

➡ U denotes the channel with the highest load which is the bottleneck channel in the entire network and determines the saturation throughput of the system

BSOR Algorithms

- Unsplittable flow problem is NP-hard

- Mixed Integer-Linear Programming (MILP) can provide an optimal solution in worst-case exponential time
  - Works for small problems with ~100 flows

- Dijkstra's weighted shortest path algorithm provides a polynomial-time heuristic that produces good results

Mixed Integer-Linear Programming

Capacity:

$$\forall v \neq s_i, t_i \quad h(v) = \sum_{i=1}^{k} \sum_{(u,v) \in E} f_i(u,v) \leq c(u,v)$$

Flow conservation:

$$\forall i, \forall u \neq s_i, t_i \quad \sum_{(w,u) \in E} f_i(w,u) = \sum_{(u,w) \in E} f_i(u,w)$$

$$\forall i, \quad \sum_{(s_i,w) \in E} f_i(s_i,w) = \sum_{(w,t_i) \in E} f_i(w,t_i) = g_i$$

Unsplittable flow:

$$\forall i, \forall (u,v) \in E, f_i(u,v) \leq b_i(u,v) \bullet d_i$$

$$\forall i, \forall u \quad \sum_{(u,v) \in E} b_i(u,v) \leq 1$$

Hop count:

$$\forall i \quad \sum_{(u,v) \in E} b_i(u,v) \leq hop_i$$

helps control path lengths

Mixed Integer-Linear Programming

Capacity:

$$\forall v \neq s_i, t_i \quad h(v) = \sum_{i=1}^{k} \sum_{(u,v) \in E} f_i(u,v) \leq c(u,v)$$

Flow conservation:

$$\forall i, \forall u \neq s_i, t_i \quad \sum_{(w,u) \in E} f_i(w,u) = \sum_{(u,w) \in E} f_i(u,w)$$

$$\forall i, \quad \sum_{(s_i,w) \in E} f_i(s_i,w) = \sum_{(w,t_i) \in E} f_i(w,t_i) = g_i$$

Unsplittable flow:

$$\forall i, \forall (u,v) \in E, f_i(u,v) \leq b_i(u,v) \bullet d_i$$

$$\forall i, \forall u \quad \sum_{(u,v) \in E} b_i(u,v) \leq 1$$

Hop count:

$$\forall i \quad \sum_{(u,v) \in E} b_i(u,v) \leq hop_i$$

helps control path lengths

Dijkstra's weighted shortest path BSOR

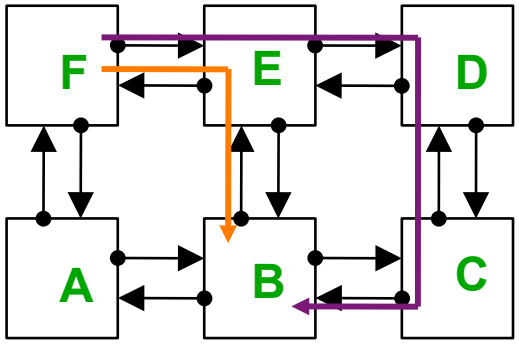
- Polynomial-time (suitable for large size problems)
- Greedily route one flow at the time
- The weighting function:

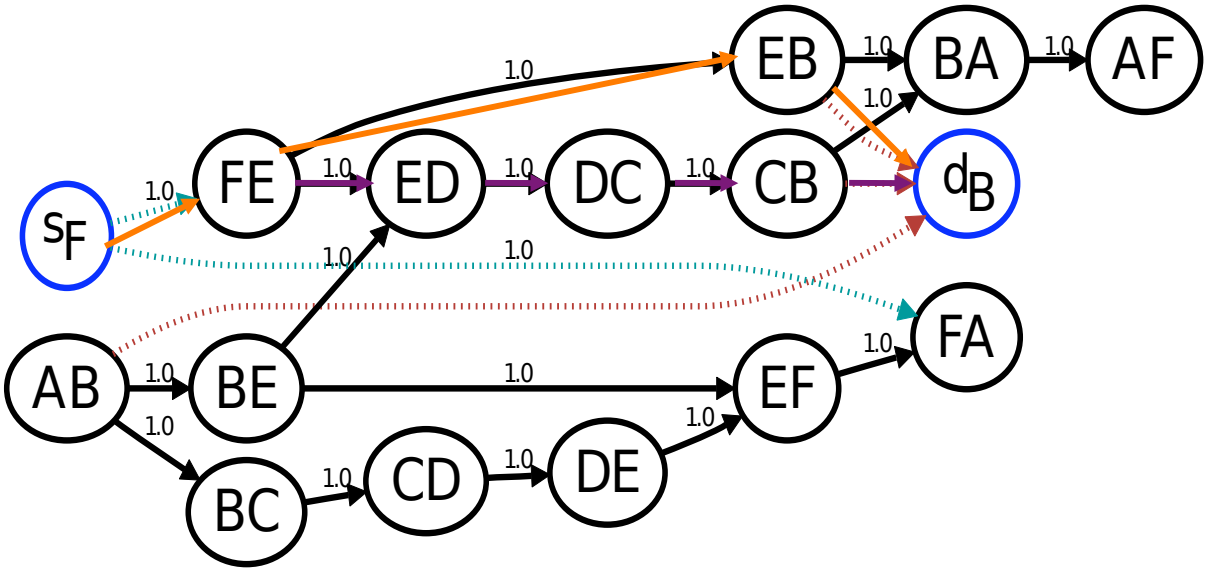
$$w(u,v) = \begin{cases} \dfrac{1}{c'(u,v) - d_i}, & if\ c'(u,v) > d_i \\ \infty, & if\ c'(u,v) \le d_i \end{cases}$$

Residual capacity: $c'(u,v) = c(u,v) - \sum_{1 \le i \le k} d_i(u,v)$

Dijkstra-based Flows routing Illustration

➡ For flow K_1 (F, B, 39.7 MB/s)

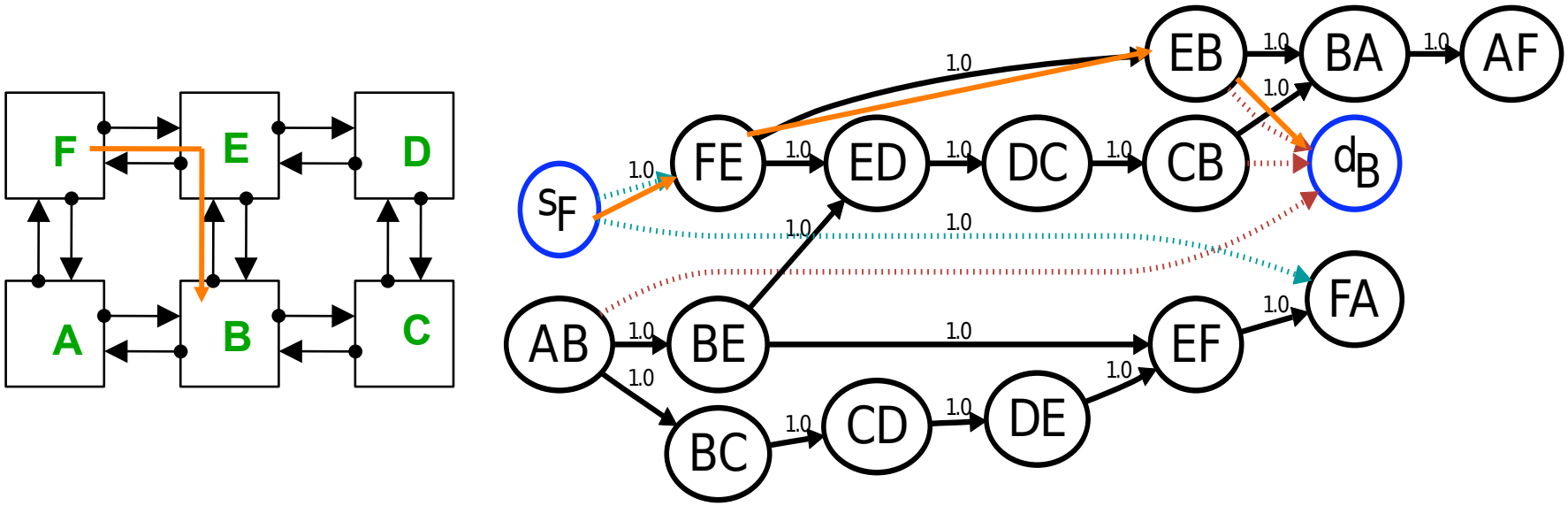




- We consider all possible routes for flow K_1 conforming to the acyclic graph
- Here the ad-hoc acyclic CDG is used

Dijkstra-based Flows routing Illustration

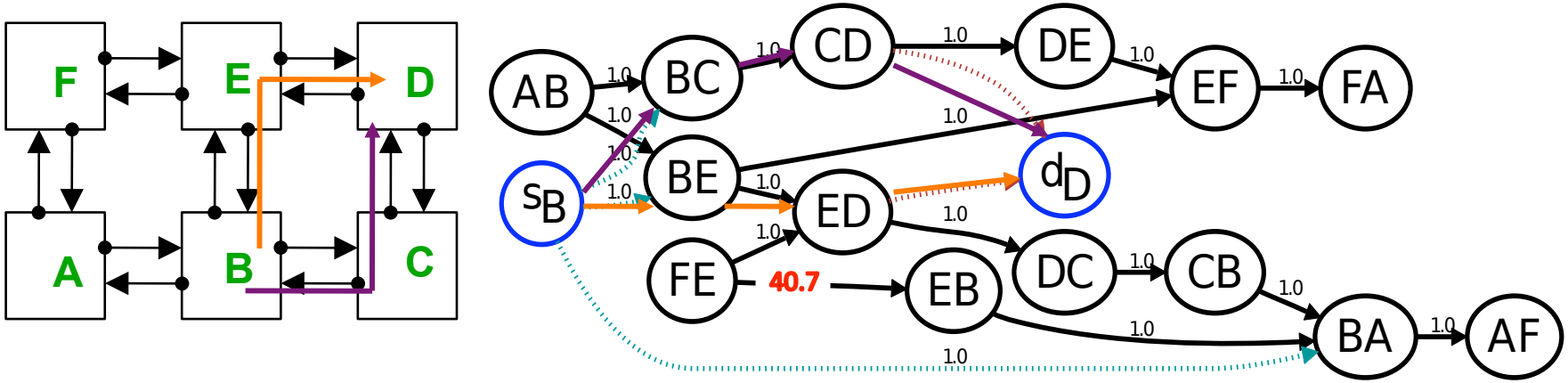
➡ For flow K_1 (F, B, 39.7 MB/s)



- We consider all possible routes for flow K_1 conforming to the acyclic graph
- Final route corresponds to the "best" path in CDG determined by the weighting function

Dijkstra-based Flows routing Illustration

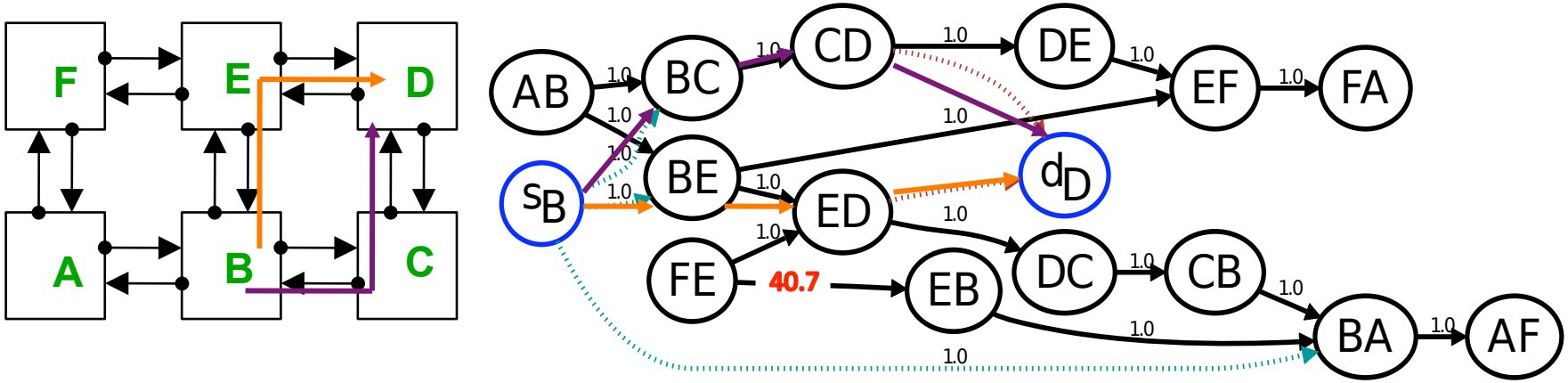
➡ For flow K_2 (B, D) 39.7 MB/s



- After the edge weights are adjusted from the routing of K_1, we consider all possible routes for K_2 conforming to the acyclic graph

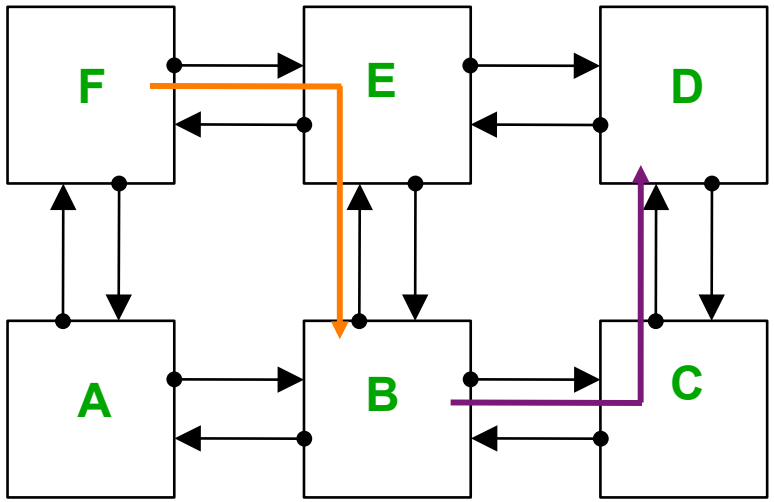
Dijkstra-based Flows routing Illustration

➡ For flow K_2 (B, D) 39.7 MB/s



- Final route corresponds to the "best" path in CDG determined by the weighting function
- Here both routes permitted under the acyclic CDG have the same weight

Dijkstra-based Flows routing Illustration



- Final routes for the two flows are as shown
- Routing order of flows in Dijkstra-based algorithms does affect route selection
- MILP produces the minimal MCL through exhaustive search

Comparison of Maximum Channel Load

Transpose: $d_i = s_{i+b/2 \bmod b} \quad where\ b = \log_2 n$

Bit-Complement: $d_i = \neg s_i$

Shuffle: $d_i = s_{i-1 \bmod b}$

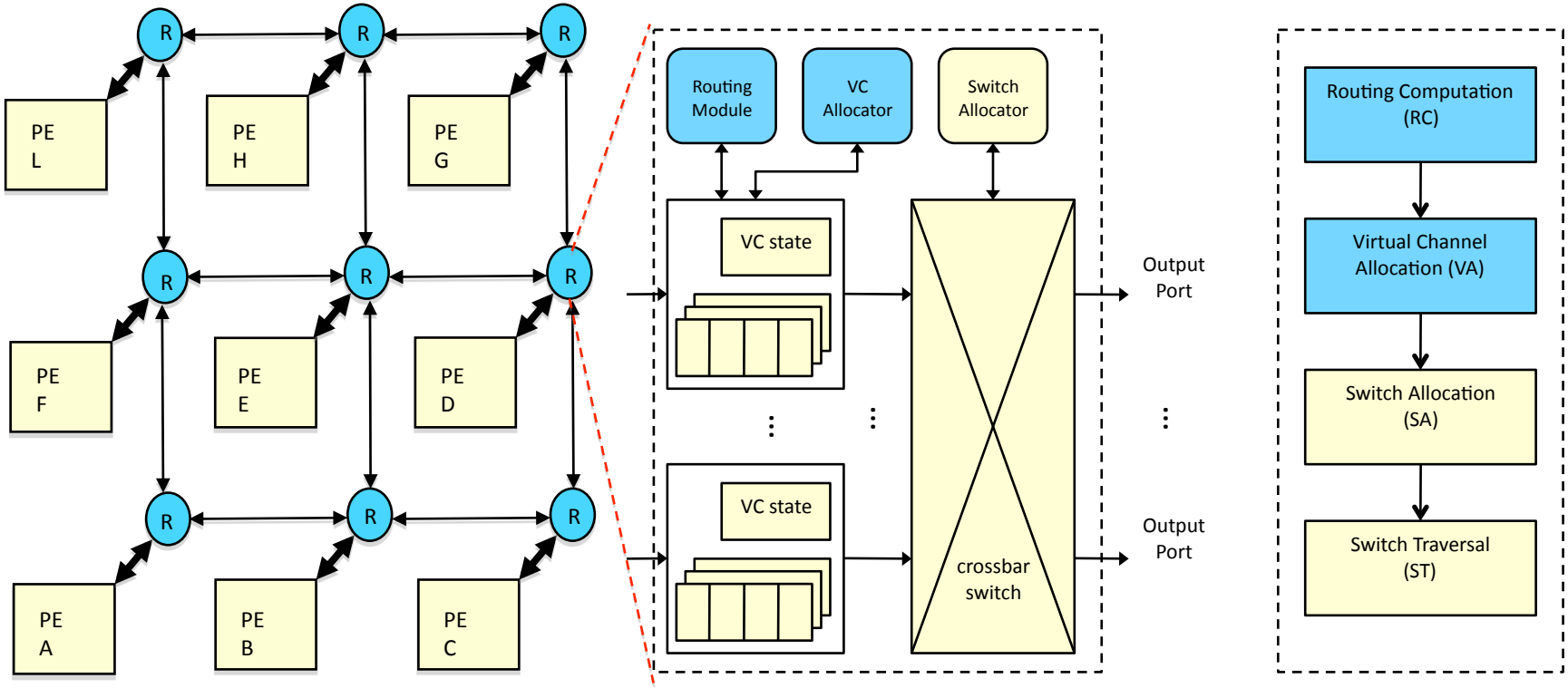
H.264 decoder: bandwidths and flows derived through profiling

Traffic	XY	YX	ROMM	Valiant	Dijkstra	MILP
Transpose	175	175	200	175	75	75
Bit-comp	100	100	400	200	100	100
Shuffle	100	100	150	200	75	75
H.264	214	365	336	352	124	120

Outline

- Routing algorithms and Motivation
- Application-Aware Oblivious Routing
- Bandwidth-Sensitive Routing Approach
- Router Architecture and Performance Analysis
- Plans for the Future

Baseline architecture



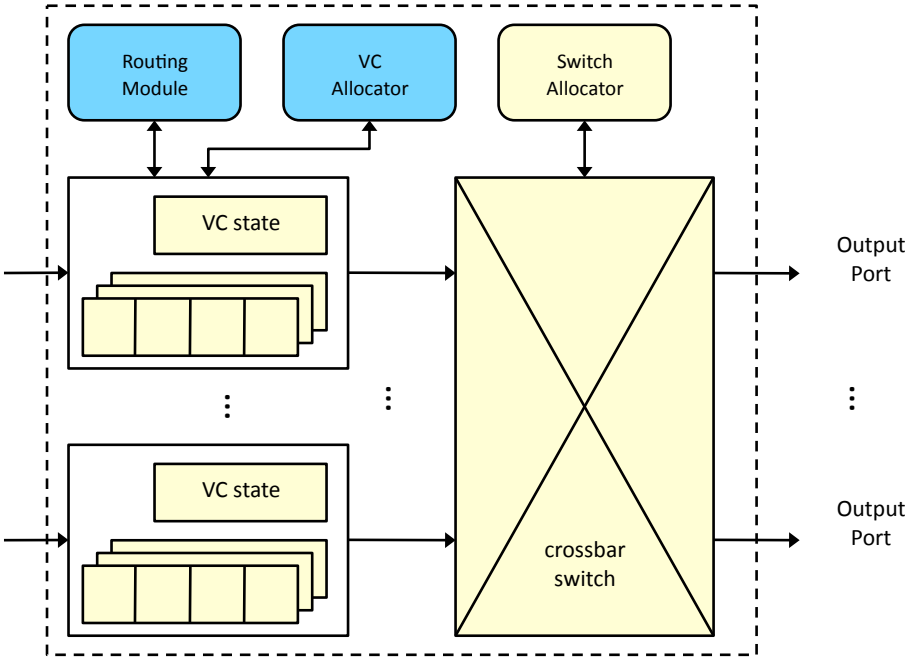
3x3 Mesh network

Router architecture

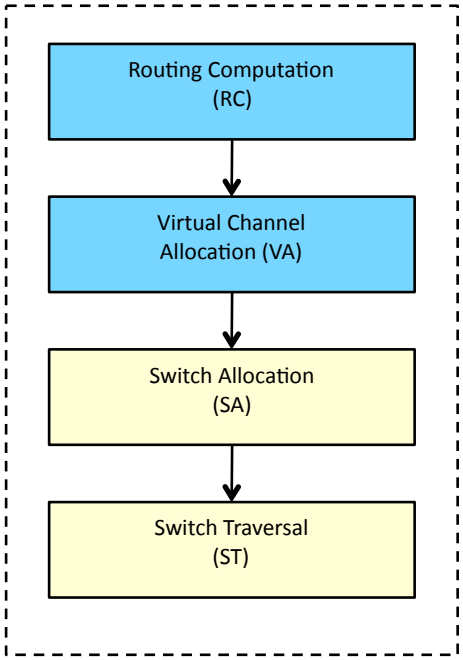
Router: routing phases

Typical virtual-channel router

Router for Application-Aware Routing



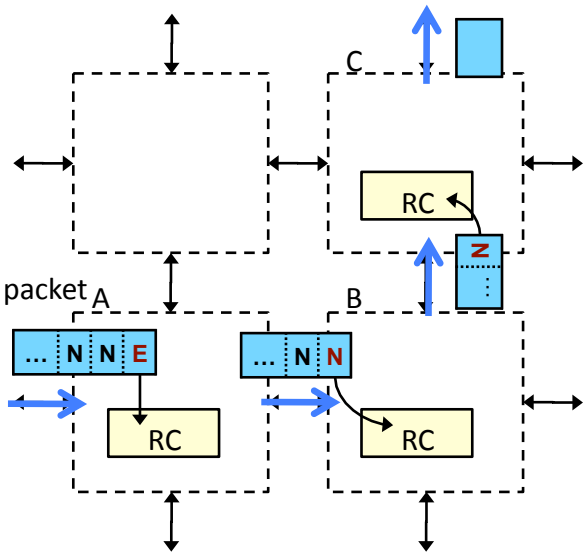
Router architecture



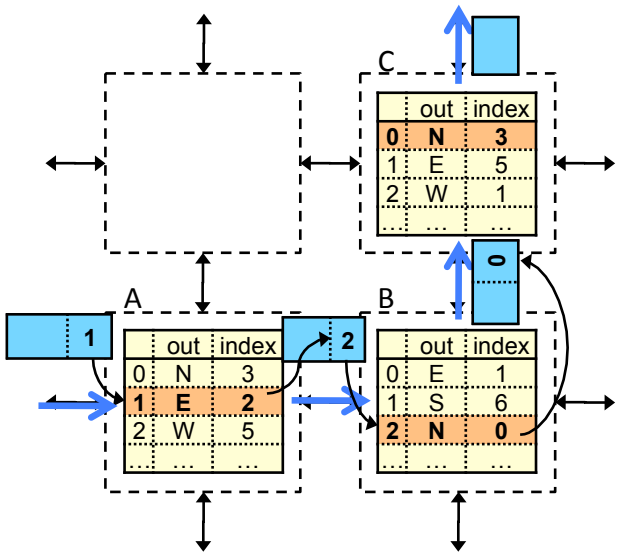
Router: routing phases

- We need modifications to the standard router architecture for application-aware routing
  - The *main* change required is in the routing module
  - The routing module needs *table-based routing*

Two ways of Table-Based Routing



Source routing



Node-table routing

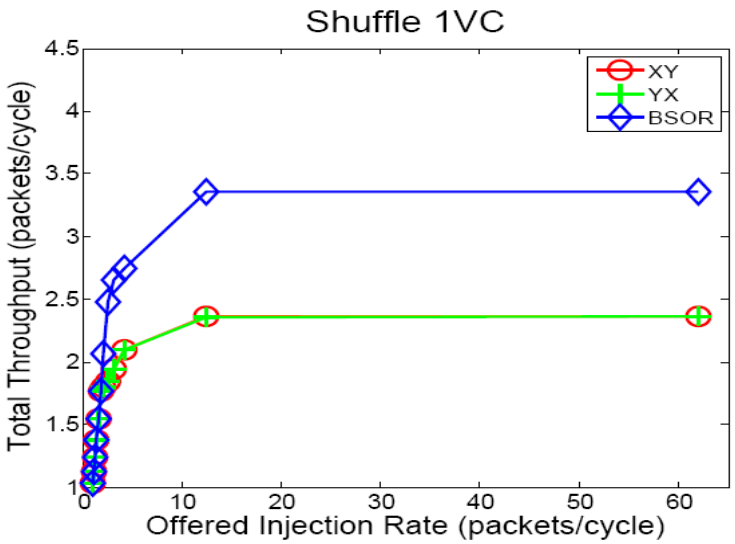
- Source routing
  - eliminates the routing step, but results in longer packets
- Node-table routing
  - Each module contains a routing table, which is looked up at every hop but this does not change the per-hop latency

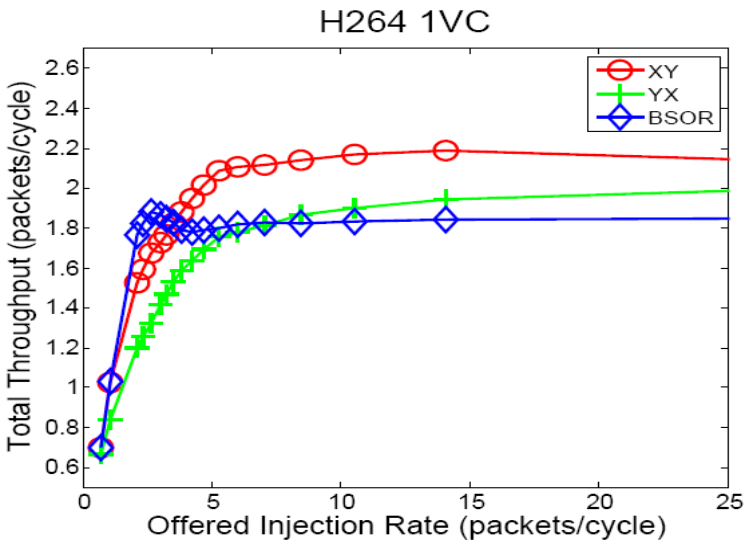
Performance Analysis

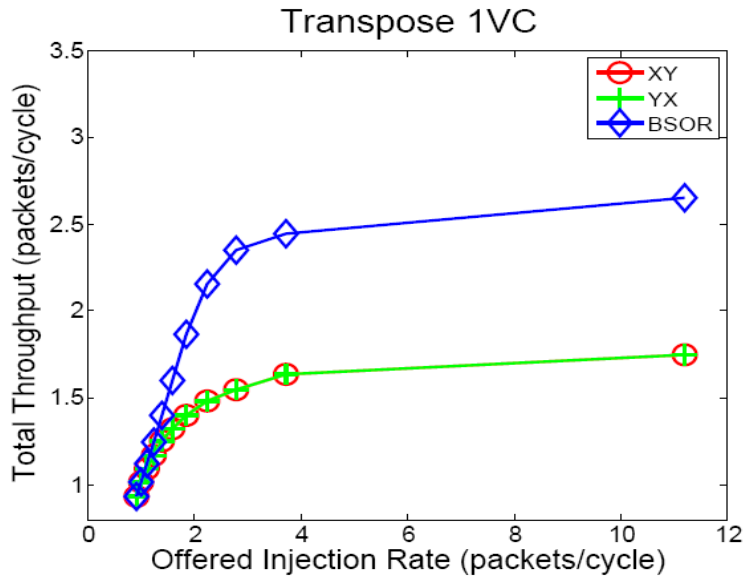
- Benchmarks :
  - Synthetic: Transpose, Bit-Complement, and Shuffle
  - Application: H.264 Decoder

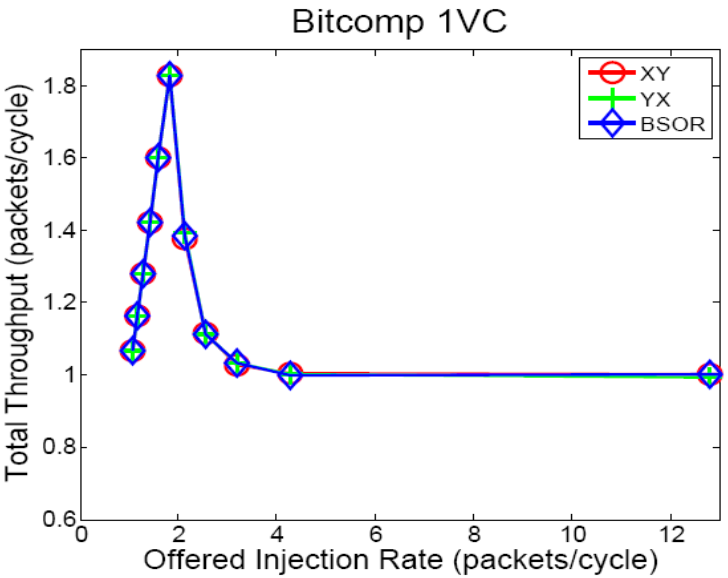
- Simulator :
  - a cycle-accurate network simulator
  - 8 X 8 2-D mesh network with 1, 2, 4 or 8 VCs per port
  - Fixed packet length : 8 flits
  - Per-hop latency : 1 cycle
  - Flit buffer size per VC : 16 flits
  - Simulation for 100,000 cycles after 20,000 cycles of warm-up

Simulation Results

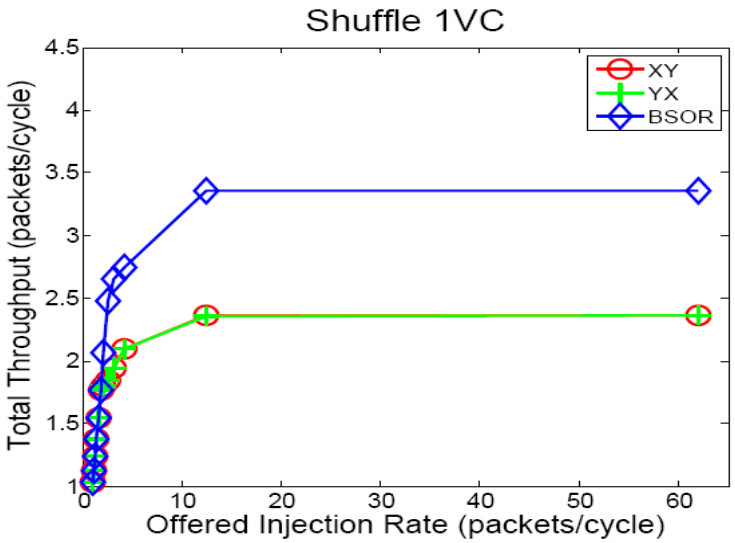


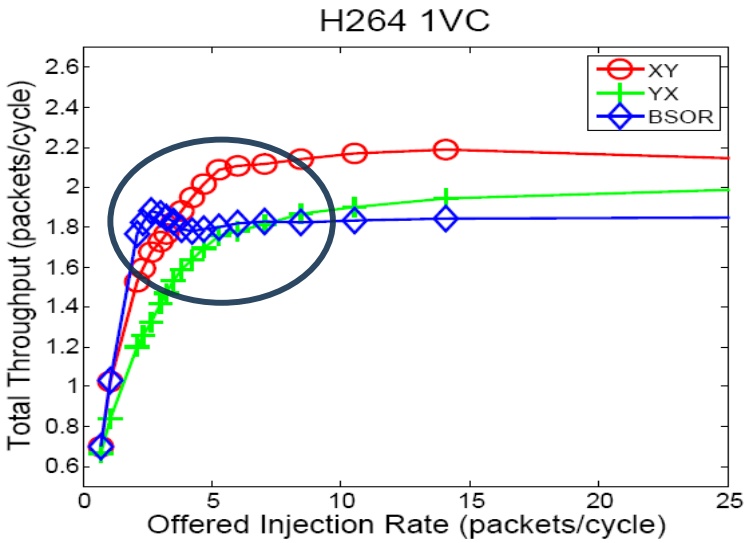


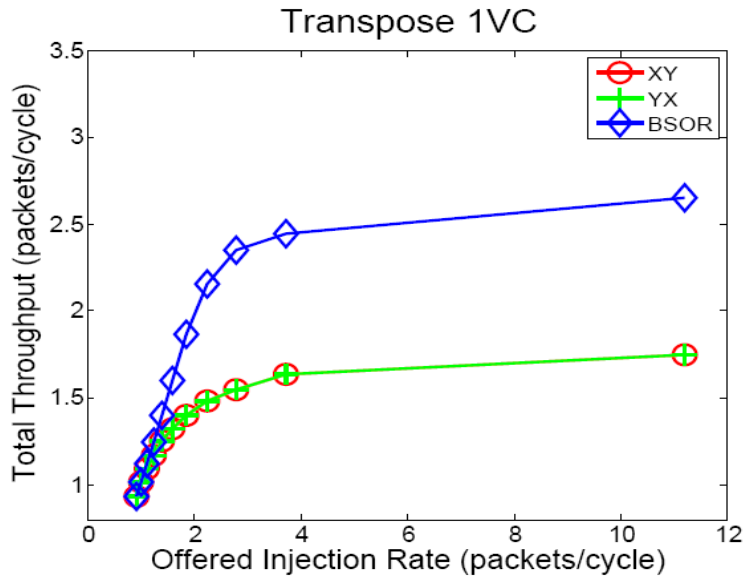


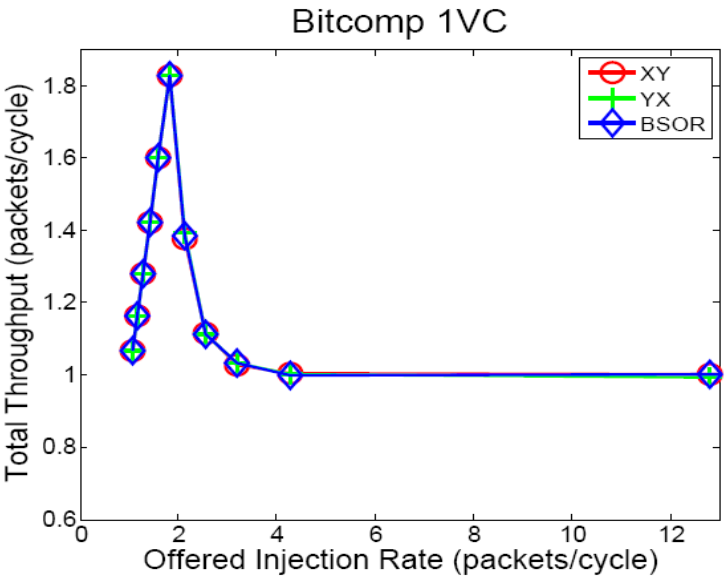


Simulation Results

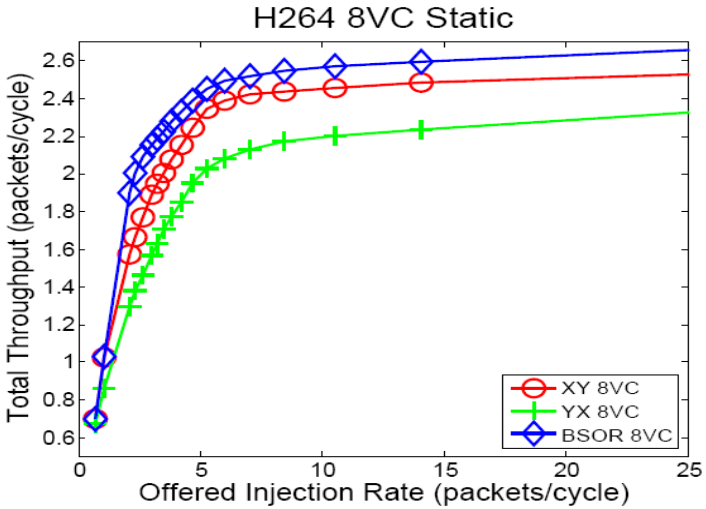








Simulation Results



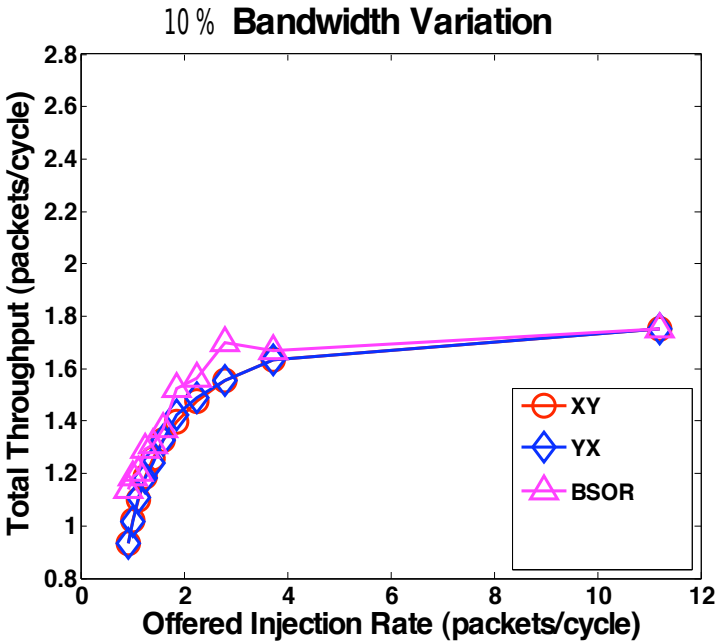
In H.264 head-of-blocking is the limiting factor for BSOR

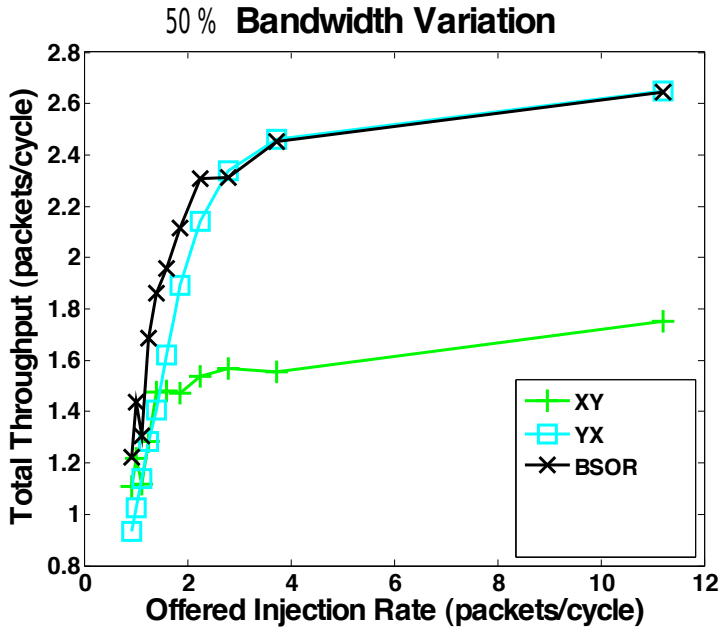
More VCs to mitigate the effects

➡ We also propose a different heuristic BSORM (Bandwidth-Sensitive Oblivious Routing with Minimal Routes) which requires two virtual channels [NOCS'09]

Stress-Test Results

Transpose





Bandwidth of each Individual flow is changed by 10% and 50% in a random fashion

➡ ROMM and Valiant do not do as well as DOR algorithms

Complete summary of our experimental results in the paper

Conclusion

- Our application-aware framework has same router speed and complexity as required by other oblivious algorithms
- It does better load-balancing, (therefore shows better performance) than other oblivious algorithms
- It can handle even substantial runtime bandwidth variation with no significant performance degradation

Conclusion

- Its limitation: need some knowledge of the application
- To handle bursty flows, we have proposed bandwidth-adaptive networks that contain adaptive bidirectional links [PACT'09]
- Ongoing work:
  - How does bandwidth-sensitive routing do on the bandwidth-adaptive network?

Q & A Section

Thank You !

More Information at
http://csg.csail.mit.edu

Also thanks to Keun Sup Shim and Mieszko Lis for their contribution, comments and valuable feedback