A Round-Efficient Distributed Betweenness Centrality Algorithm

Loc Hoang, Matteo Pontecorvi, Roshan Dathathri, Gurbinder Gill, Bozhi You, Keshav Pingali, and Vijaya Ramachandran

The University of Texas at Austin
Computer Science

NOKIA Bell Labs

西安交通大學
XI'AN JIAOTONG UNIVERSITY

Betweenness Centrality

- Betweenness Centrality (BC) used to determine relative importance of node in graph
- Applications
  - Key actor detection in terrorist nets
  - Disease studies
  - Power grid analysis
  - River flow confluence
- Distributed implementations necessary
  - Large graphs with billions of nodes/edges
  - BC takes hours to complete even if approximating

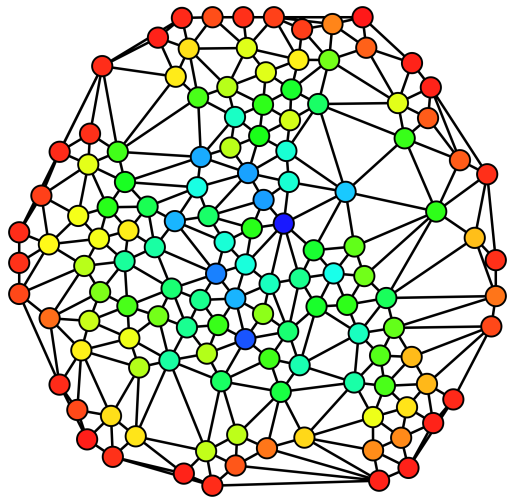
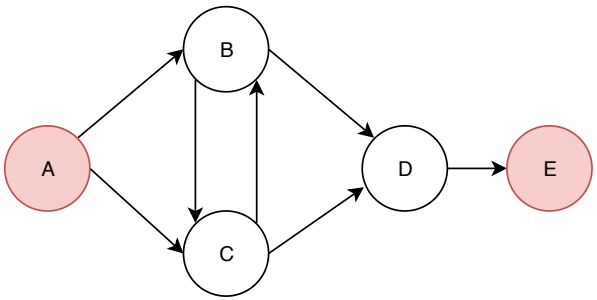


Figure Credit: Claudio Rocchini, Creative Commons Attribution 2.5 Generic

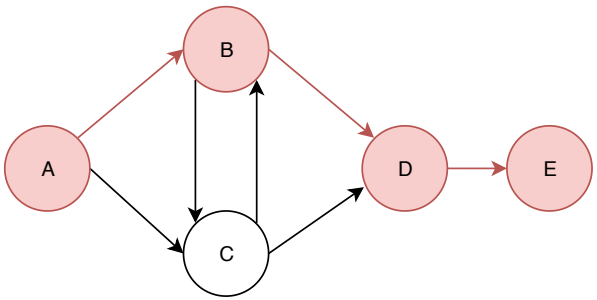
Betweenness Centrality Definition



- BC: fraction of shortest paths in which node appears

- Example: consider the 2 shortest paths from A to E:
  - B appears in 1: $\frac{1}{2}$; C appears in 1: $\frac{1}{2}$; D appears in 2: $\frac{2}{2} = 1$

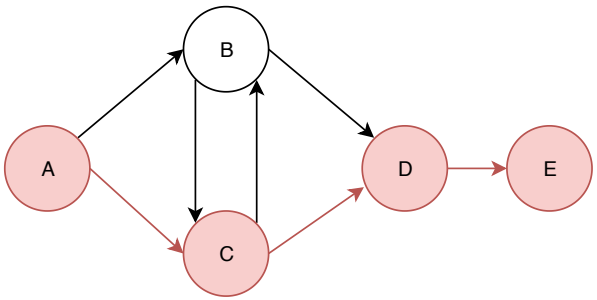
Betweenness Centrality Definition



- BC: fraction of shortest paths in which node appears

- Example: consider the 2 shortest paths from A to E:
  - B appears in 1: $\frac{1}{2}$; C appears in 1: $\frac{1}{2}$; D appears in 2: $\frac{2}{2} = 1$

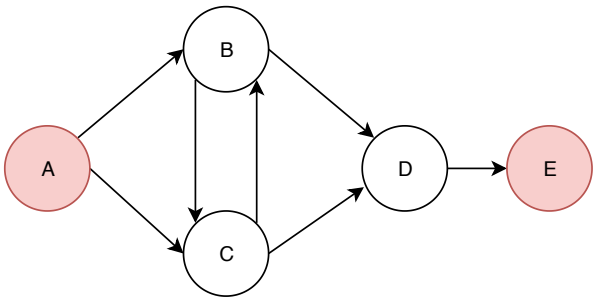
Betweenness Centrality Definition



- BC: fraction of shortest paths in which node appears

- Example: consider the 2 shortest paths from A to E:
  - B appears in 1: $\frac{1}{2}$; C appears in 1: $\frac{1}{2}$; D appears in 2: $\frac{2}{2} = 1$

Betweenness Centrality Definition



- BC: fraction of shortest paths in which node appears

- Example: consider the 2 shortest paths from A to E:
  - B appears in 1: $\frac{1}{2}$; C appears in 1: $\frac{1}{2}$; D appears in 2: $\frac{2}{2} = 1$

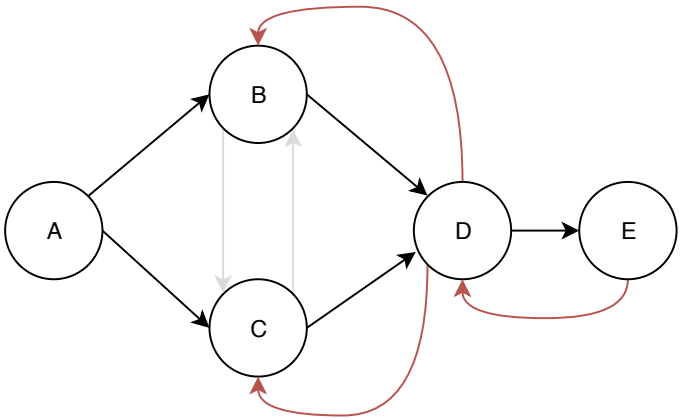
σ_{st}, number of shortest paths from s to t; $\sigma_{st}(v)$, number of shortest paths from s to t passing through v, $v \neq s \neq t$.

Betweenness Centrality (BC)

$$BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma st}$$

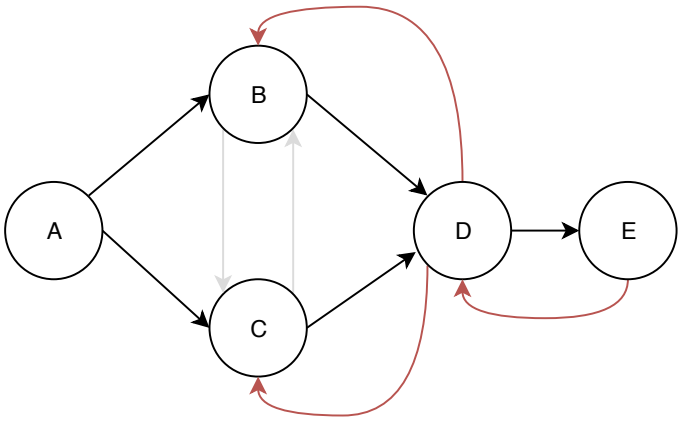
From definition: about n^3 operations (n is number of vertices)

Brandes Betweenness Centrality



- Shortest-path DAG with shortest path counts rooted at node s: propagate dependencies ($\delta_{s\bullet}$) along DAG predecessors

Brandes Betweenness Centrality



- Shortest-path DAG with shortest path counts rooted at node s: propagate dependencies ($\delta_{s\bullet}$) along DAG predecessors

BC from Dependencies of a Node

$$BC(v) = \sum_{s \neq v} \delta_{s\bullet}(v)$$

$$\text{where} \quad \delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))$$

$P_s(w)$ are predecessors of w in DAG

- Brandes BC [1]: sum dependencies from all DAGs: $O(nm)$ operations (m is number of edges)
- All-pairs shortest paths (APSP) or k-source shortest paths (k-SSP, shortest paths for subset of k nodes) to find DAGs

[1] U. Brandes. A Faster Algorithm for Betweenness Centrality. Journal of Mathematical Sociology 2001.

Related APSP and BC Work

- APSP
  - $O(n)$ round undirected, unweighted APSP algorithms [2,3,4]
    - Lenzen-Peleg: prior best unweighted APSP
- BC
  - Asynchronous Brandes BC (ABBC): asynchronous, shared-memory [5]
  - Maximal Frontier BC (MFBC): distributed, sparse-matrix Brandes BC [6]
  - Hua et al.: distributed BC for undirected, unweighted graphs [7]

[2] S. Holzer and R. Wattenhofer. Optimal Distributed All Pairs Shortest Paths and Applications. PODC 2012.
[3] D. Peleg, L. Roditty, and E. Tal. Distributed Algorithms for Network Diameter and Girth. ICALP 2012.
[4] C. Lenzen and D. Peleg. Efficient Distributed Source Detection with Limited Bandiwidth. PODC 2013
[5] D. Prountzos and K. Pingali. Betweenness centrality: algorithms and implementations. PPoPP'13.
[6] E. Solomonik, M. Besta, F. Vella, and T. Hoefler. Scaling Betweenness Centrality Using Communication-efficient Sparse Matrix Multiplication.
[7] Q. S. Hua, H. Fan, M. Ai, L. Qian, Y. Li, X. Shi, and X. Jin. Nearly Optimal Distributed Algorithm for Computing Betweenness Centrality. ICDCS 2016.

Motivation for Our Work

- Practical implementations of theoretical, distributed O(n)-round APSP/BC algorithms do not exist
- Existing distributed BC mainly use SSSP/k-SSP with Brandes BC
  - High amount of bulk-synchronous parallel (BSP) rounds with expensive communication barriers

Tradeoff exploration:
decreasing number of rounds at cost of increasing computation per round

Our Contributions: Theory

Min-Rounds APSP and Min-Rounds Betweenness Centrality (MRBC) for directed and undirected unweighted graphs

- CONGEST: (known) n nodes, m edges, diameter D: APSP in $min(n + O(D), 2n)$ rounds and $mn + O(m)$ messages

Our Contributions: Theory

Min-Rounds APSP and Min-Rounds Betweenness Centrality (MRBC) for directed and undirected unweighted graphs

- CONGEST: (known) n nodes, m edges, diameter D: APSP in $min(n + O(D), 2n)$ rounds and $mn + O(m)$ messages
- In systems that detect termination: k-SSP in at most $k + H$ rounds and $m \cdot k$ messages, H is largest finite shortest path distance for the k sources

Our Contributions: Theory

Min-Rounds APSP and Min-Rounds Betweenness Centrality (MRBC) for directed and undirected unweighted graphs

- CONGEST: (known) n nodes, m edges, diameter D: APSP in $min(n + O(D), 2n)$ rounds and $mn + O(m)$ messages
- In systems that detect termination: k-SSP in at most $k + H$ rounds and $m \cdot k$ messages, H is largest finite shortest path distance for the k sources
- BC: at most twice the rounds/messages as APSP/k-SSP

Our Contributions: Practice

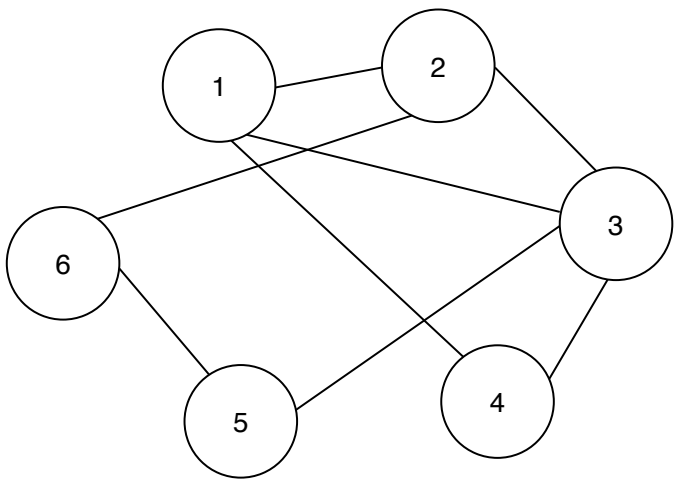
- MRBC implementation in D-Galois[8] with communication optimization exploiting MRBC properties
- MRBC evaluation
  - 3× faster than prior state-of-the-art MFBC
  - 2.8× speedup over Brandes BC on high diameter graphs

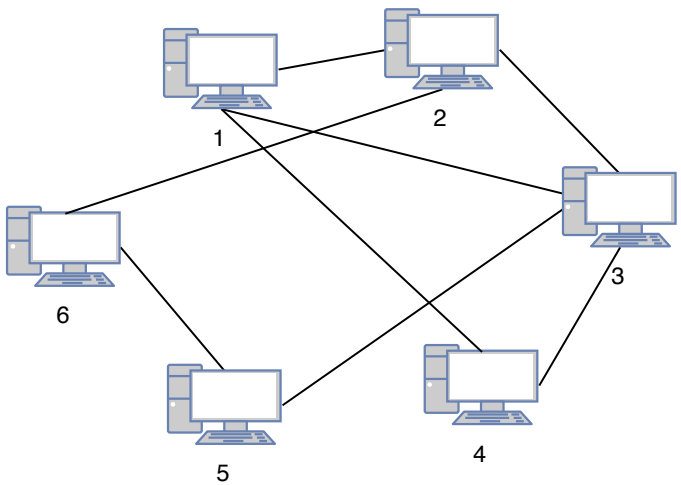
[8] R. Dathathri, G. Gill, L. Hoang, H.V. Dang, A. Brooks, N. Dryden, M. Snir, K. Pingali. Gluon: A Communication-Optimizing Substrate for Distributed Heterogeneous Graph Analytics. PLDI 2018.

Outline

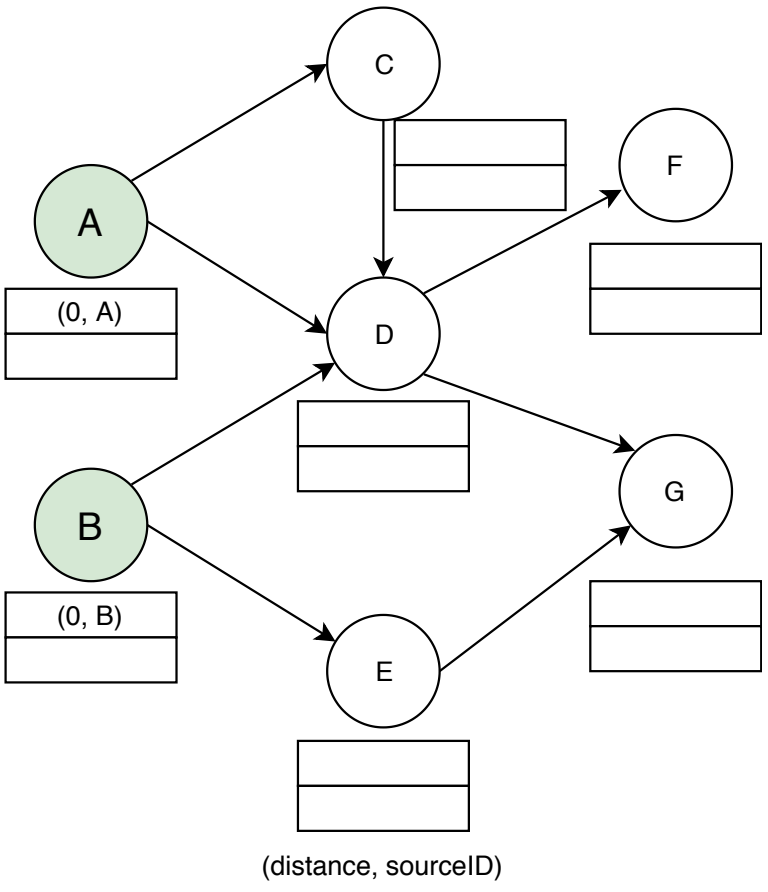
CONGEST Model for Distributed Algorithms

- Machines are nodes, edges are communication channels
- Send message (constant number of words) per round to do updates





k-SSP Example: Initial State



- Left: Initial State of k-SSP where $k = 2$ sources A and B
- Vertices store current distance from a source to self in lexicographically sorted vector
- Every round, vertex chooses 1 (distance, source) pair to send along outgoing edges

APSP: When To Send A Pair?

- Problem: sent distance may not be final distance associated with source

APSP: When To Send A Pair?

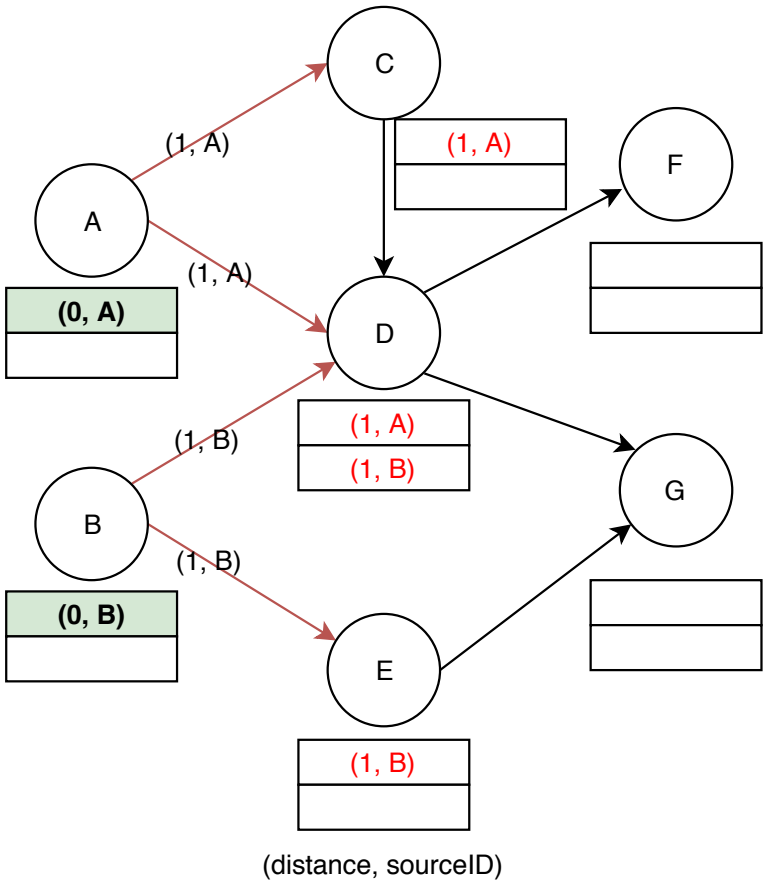
- Problem: sent distance may not be final distance associated with source

Min-Rounds APSP New Insight: Message Send Rule

Send unsent distance d with position p on sorted vector with corresponding source in round r if $p + d = r$

- Like Dijkstra: sends only final distance
- Resulting algorithm pipelines messages: orchestrates updates across edges and reduces amount of messages sent

k-SSP Example: Round 1

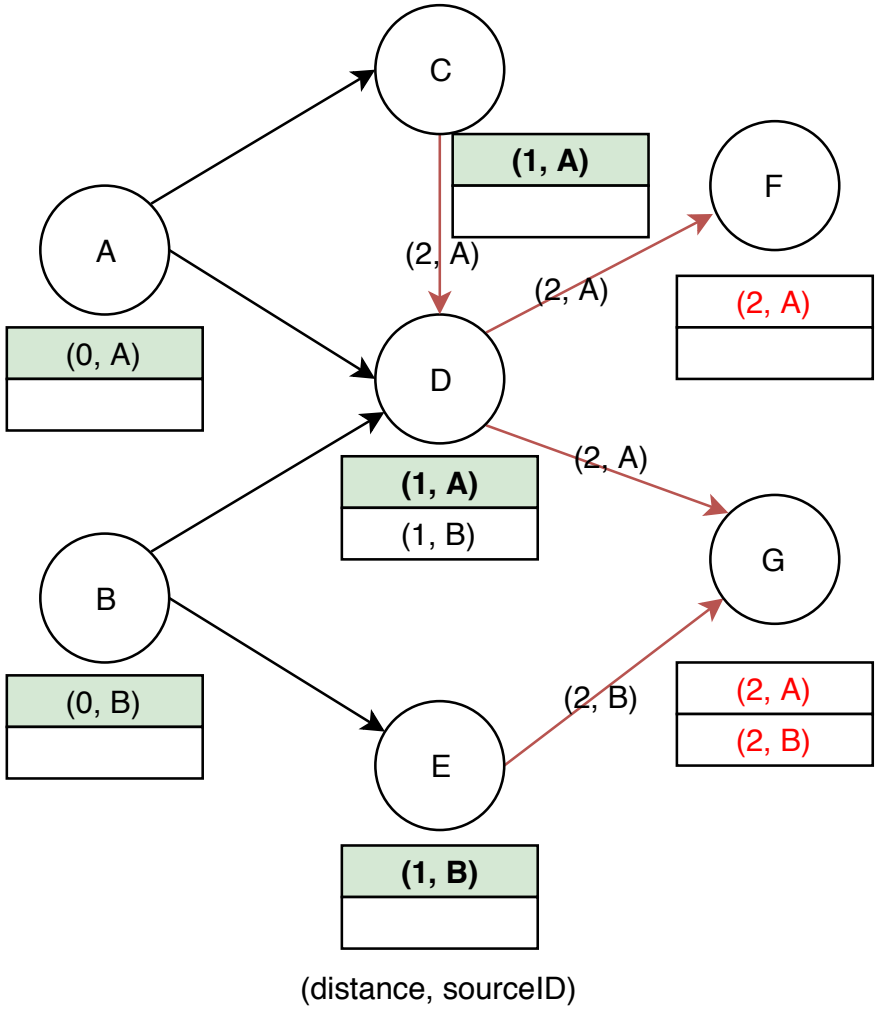


Message Send Rule

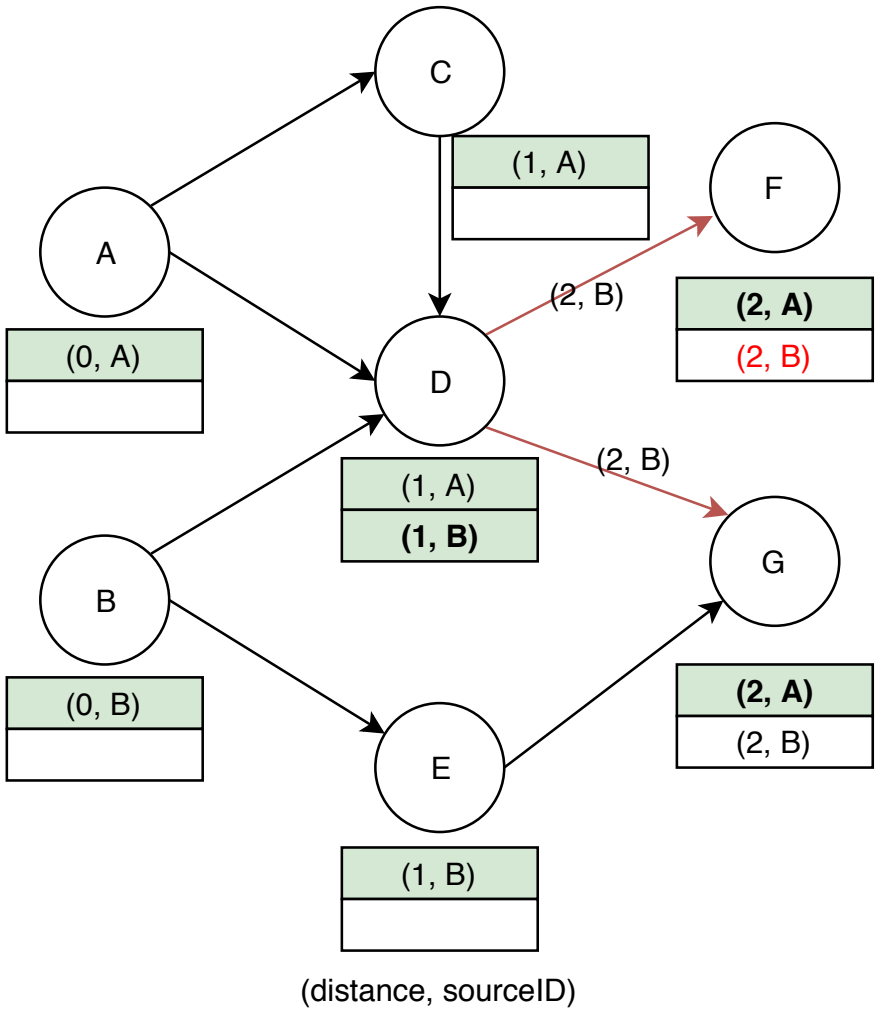
Send unsent distance d with position p on sorted vector with corresponding source in round r if $p + d = r$

- Example: $(0, A)$ chosen because $0 + 1$ (1 is position on vector) equals round 1

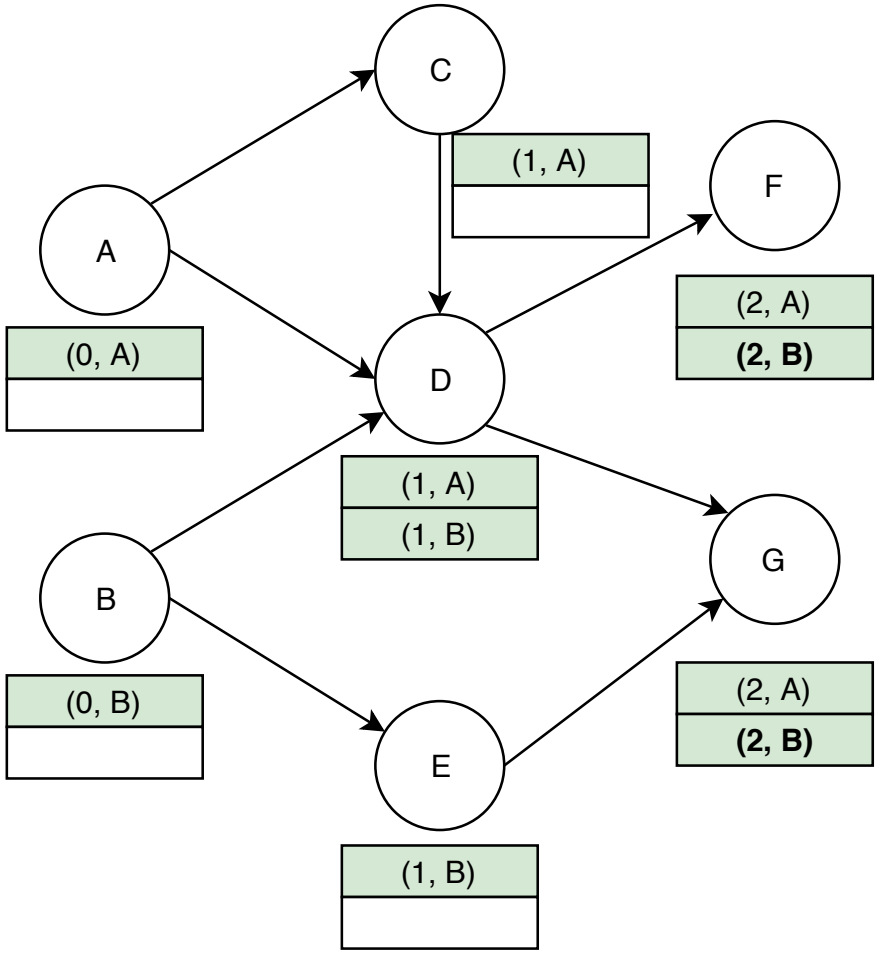
k-SSP Example: Round 2



k-SSP Example: Round 3



k-SSP Example: Round 4 (Final)



APSP for Brandes BC

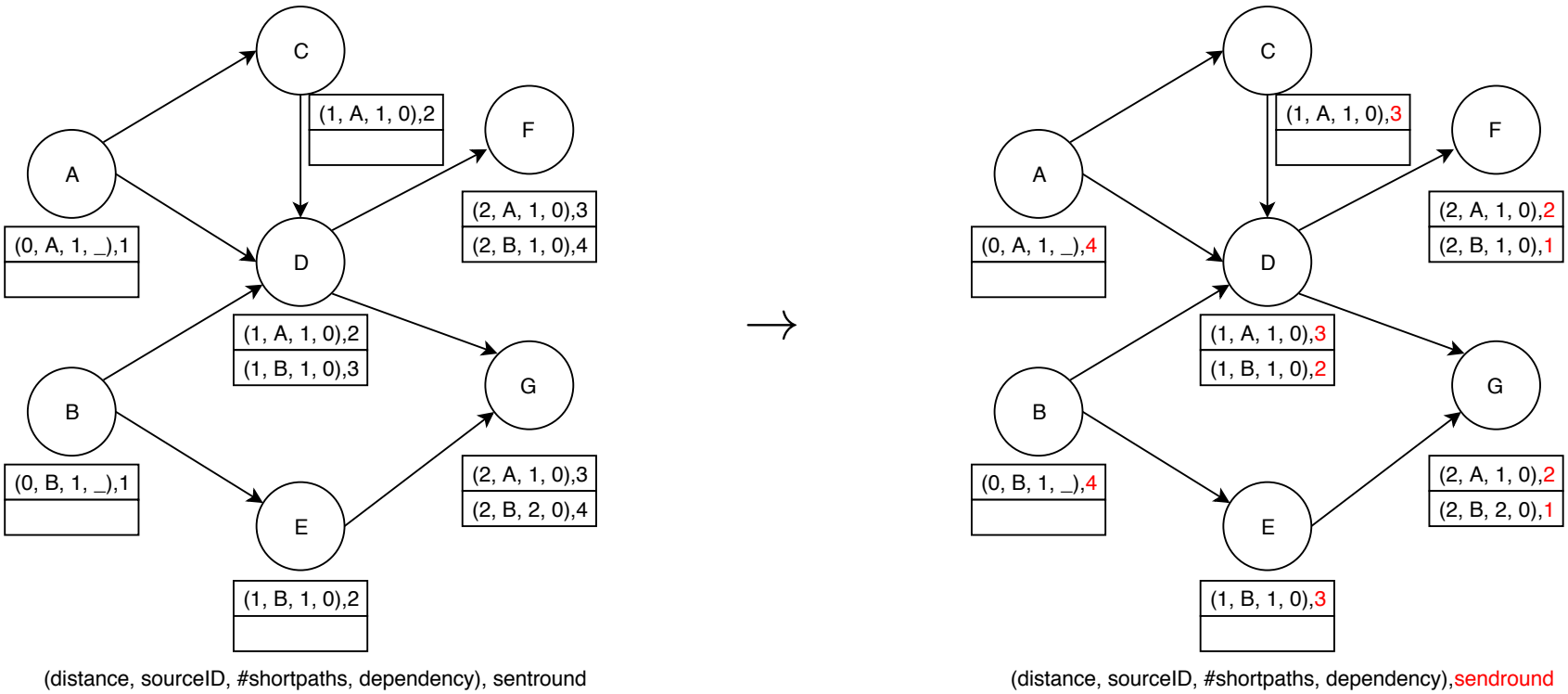
Min-Rounds APSP as subroutine for Brandes BC backward accumulation

Three Additions to APSP

- Send shortest path count with distance/source ID in APSP
- Timestamp round number in which message is sent
- Track predecessors of shortest path DAG for each source

Min-Rounds BC: Reversing Global Delays

Insight: leverage saved timestamps, send final values

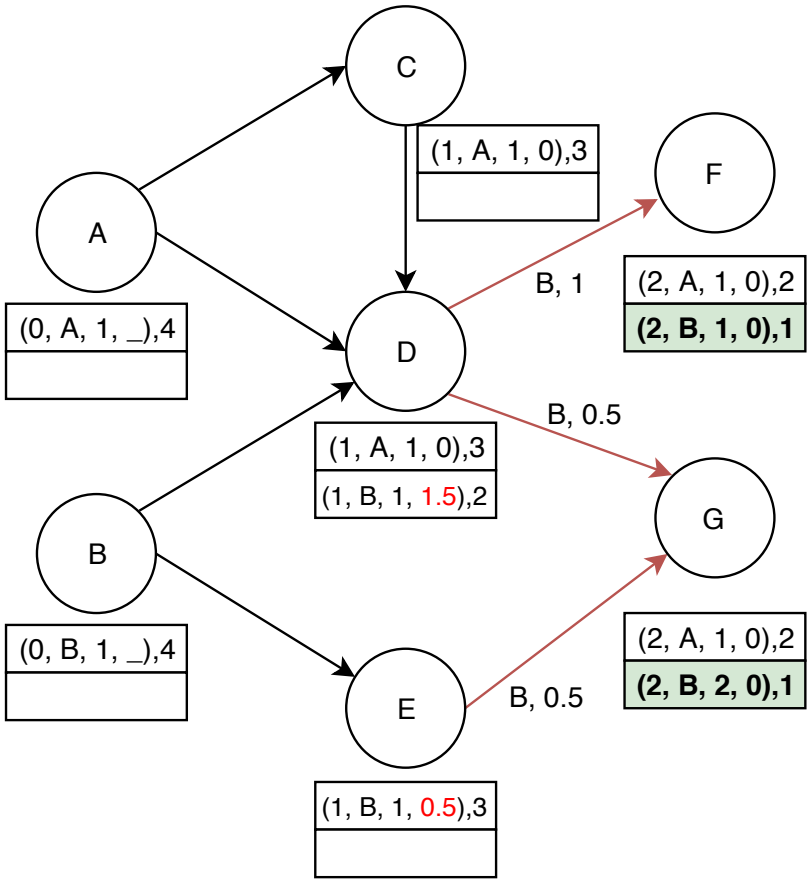


Timestamp Pipelining By Reversing Global Delay

Send source's dependency value to predecessors in source's DAG in *reverse round order*: total rounds + 1 - timestamp

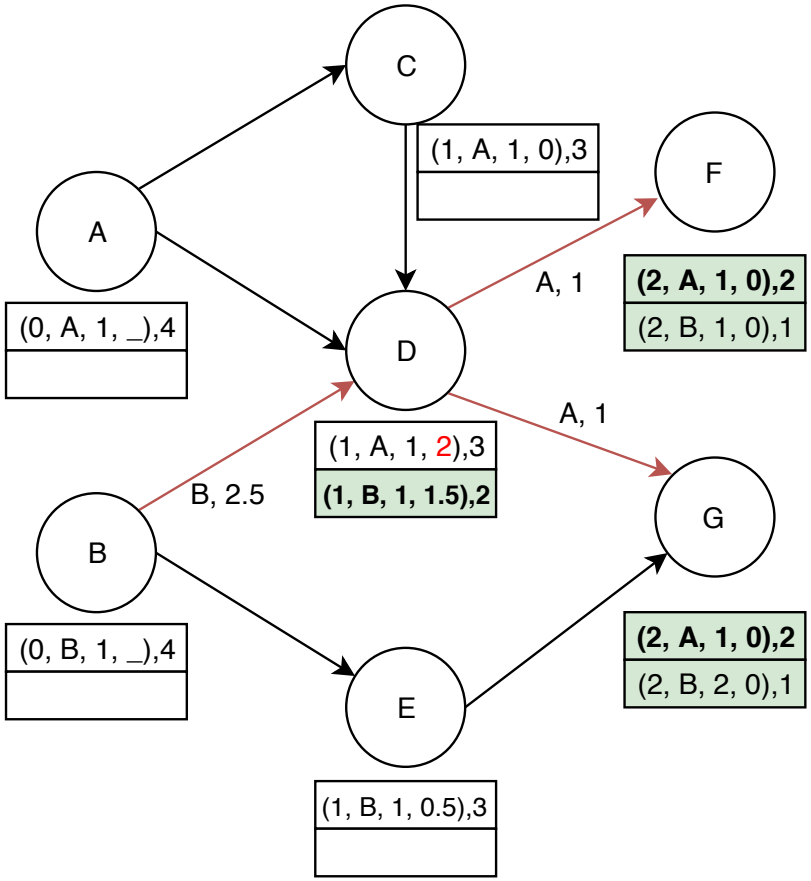
Backward Accumulation: Round 1

Brandes formulation to propagate finalized dependencies



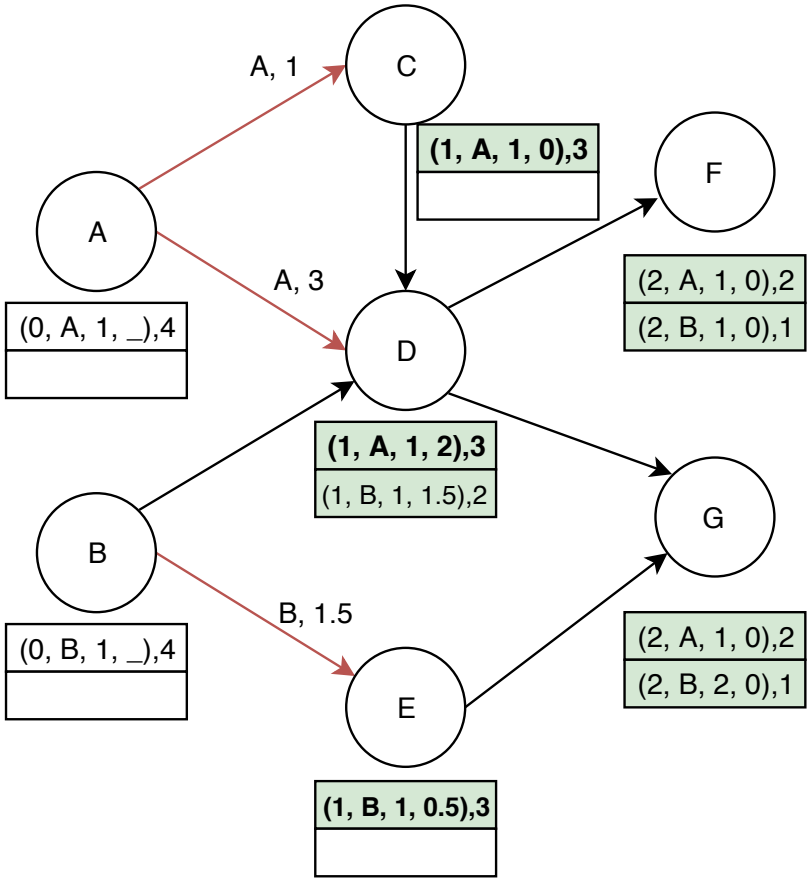
(distance, sourceID, #shortpaths, dependency),sendround

Backward Accumulation: Round 2



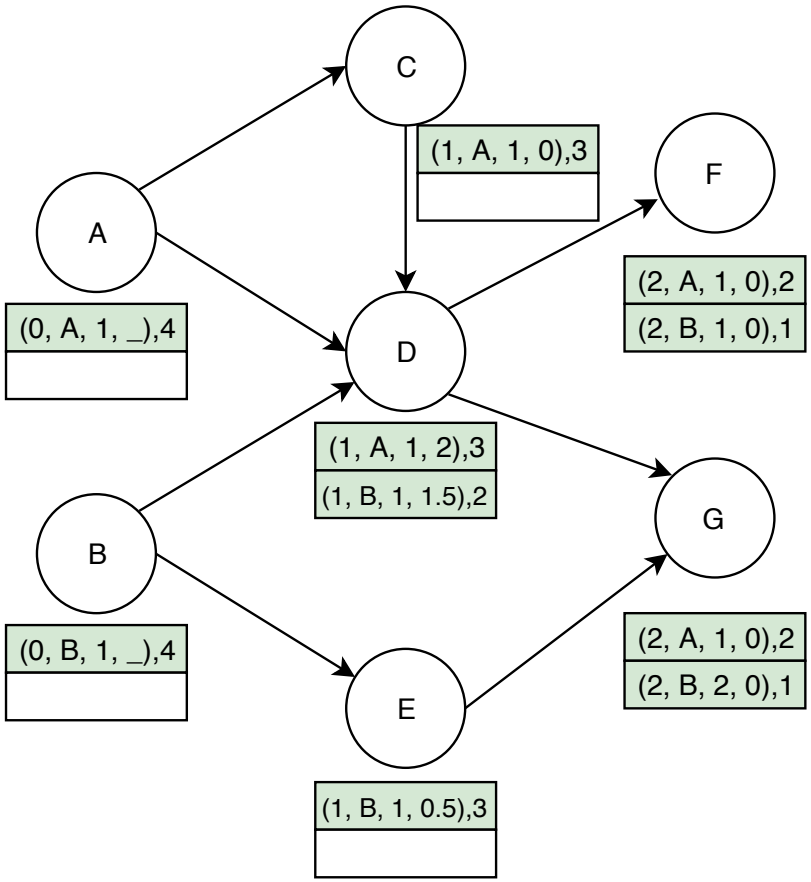
(distance, sourceID, #shortpaths, dependency),sendround

Backward Accumulation: Round 3



(distance, sourceID, #shortpaths, dependency),sendround

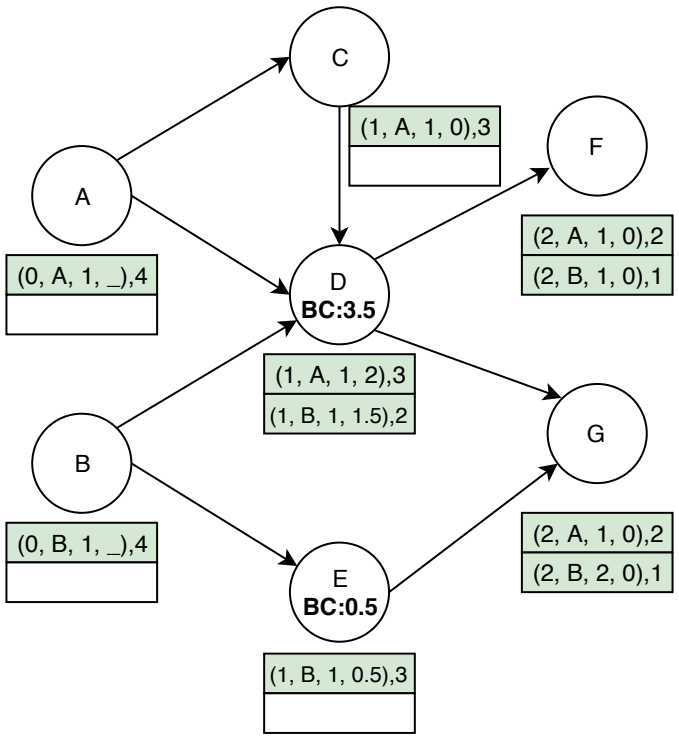
Backward Accumulation: Round 4



(distance, sourceID, #shortpaths, dependency),sendround

Final Result

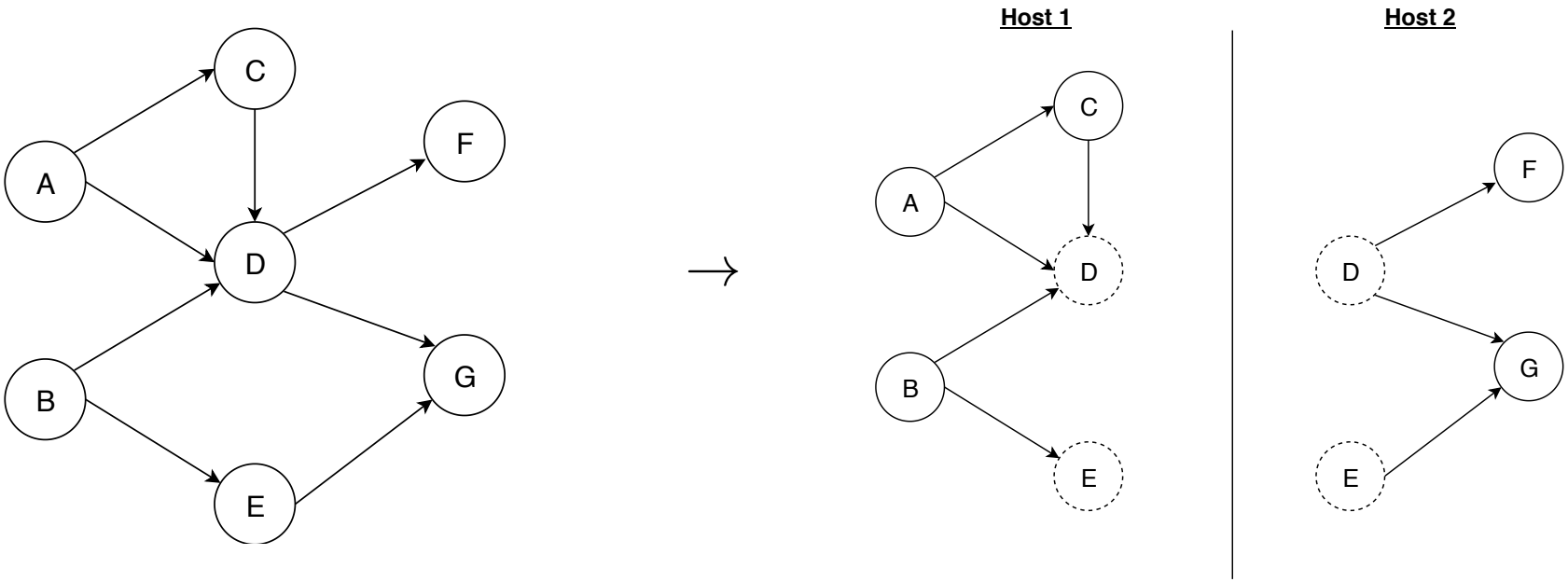
Add source dependencies to get BC contribution



To get BC, use APSP rather than k-SSP

D-Galois and the Execution Model

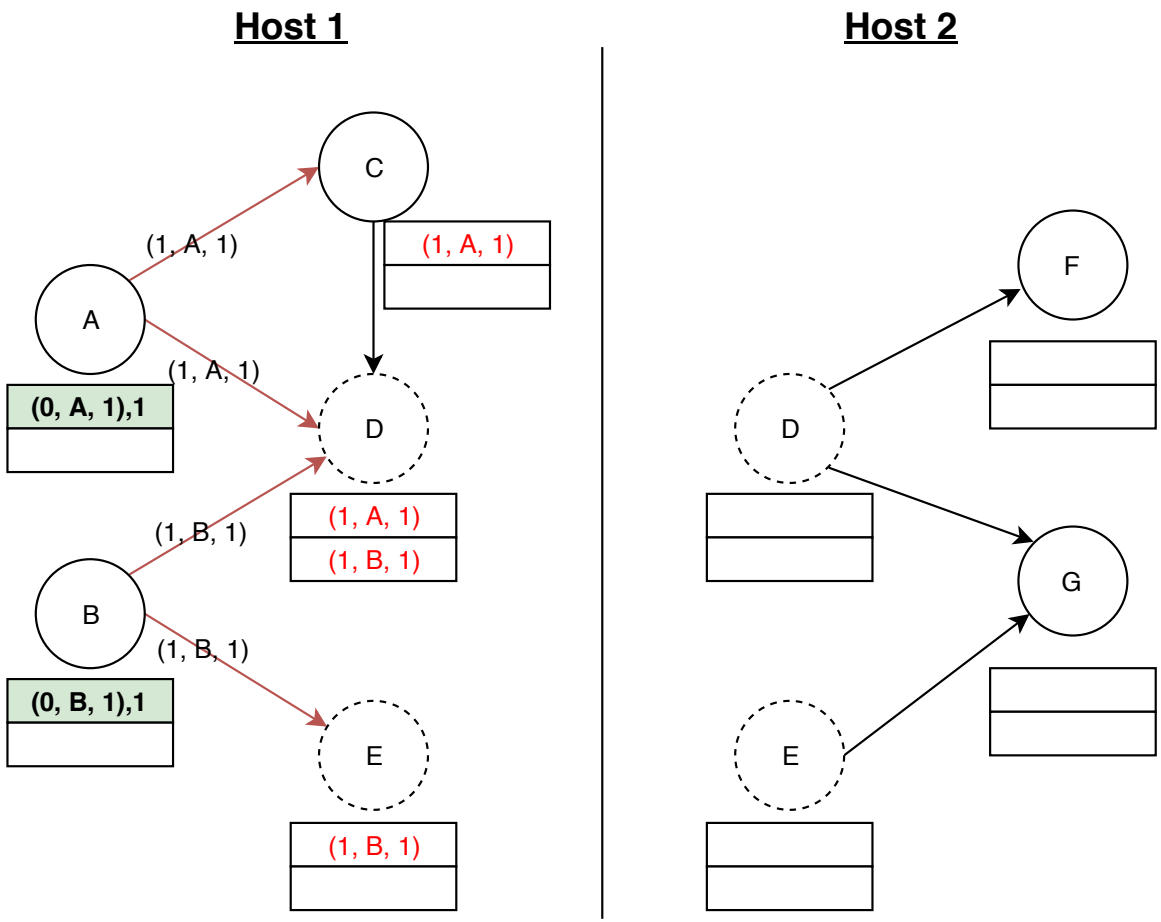
- D-Galois: distributed graph analytics system using shared-memory Galois and Gluon communication substrate



- Distribute edges from graph; cached-copies (proxies) of endpoints created
- Execution in bulk-synchronous parallel (BSP) rounds: computation then communication to sync proxies

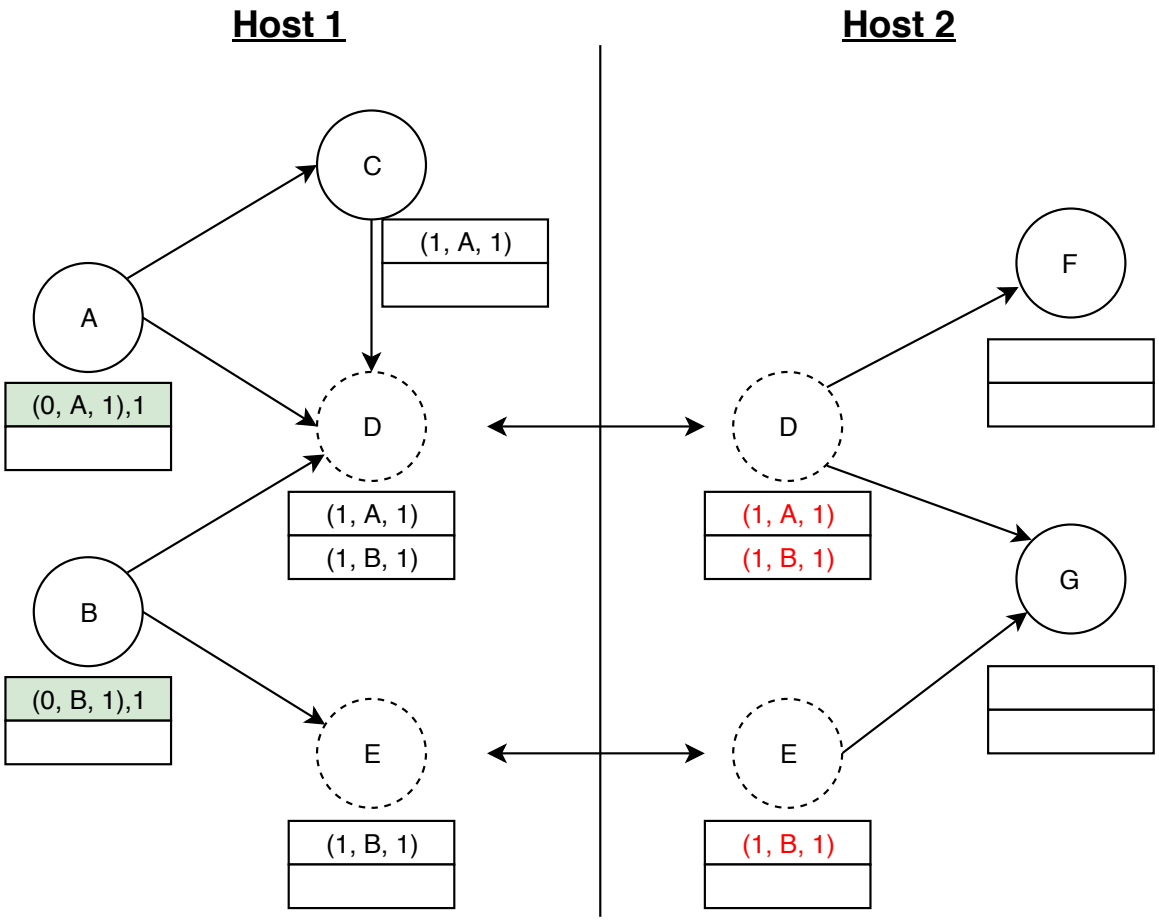
Example Execution in D-Galois: Round 1 Compute

Computation: CONGEST "message sends" along edges



Example Execution in D-Galois: Round 2 Sync

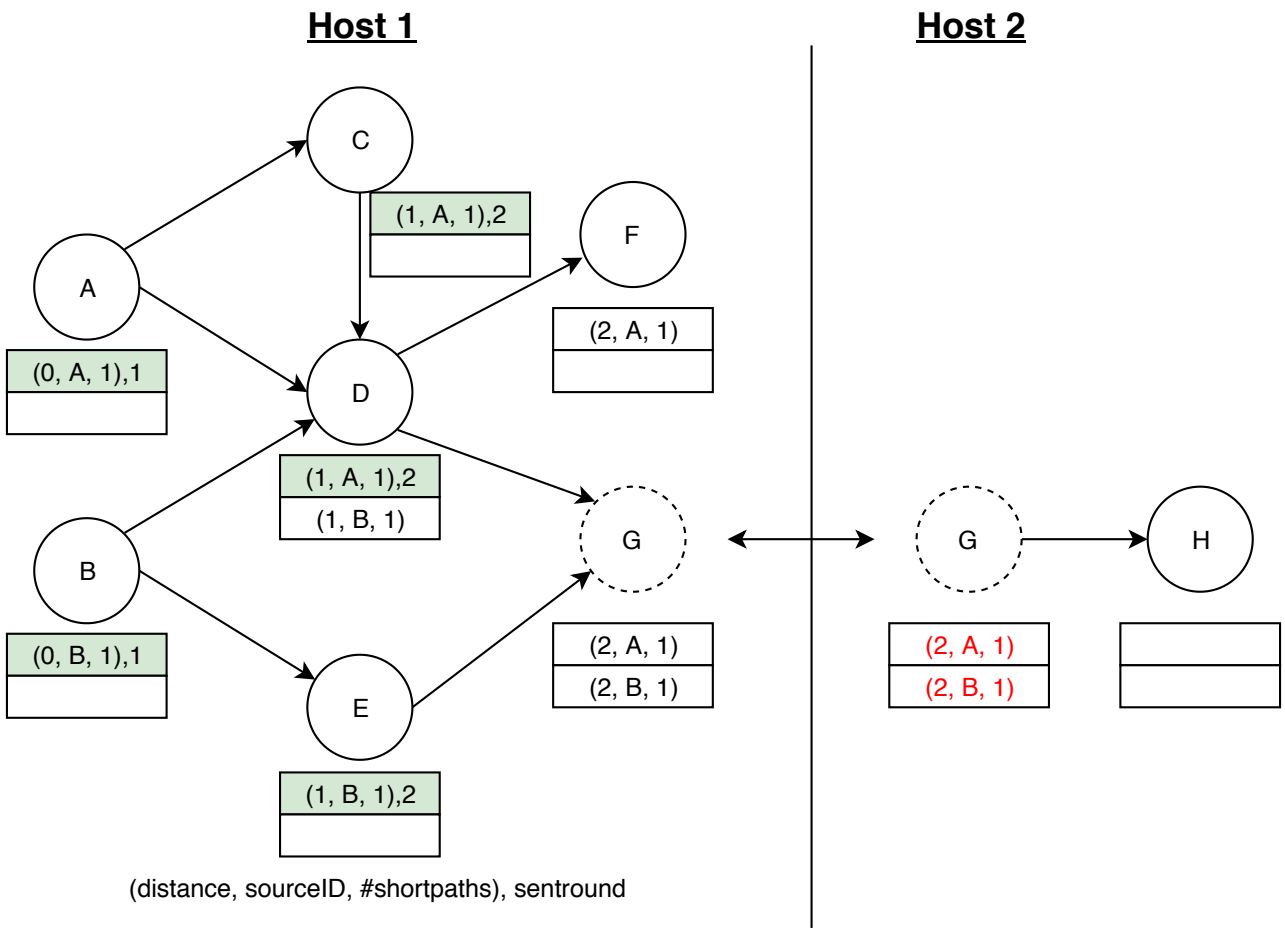
Synchronization of proxies D and E after computation



(distance, sourceID, #shortpaths), sentround

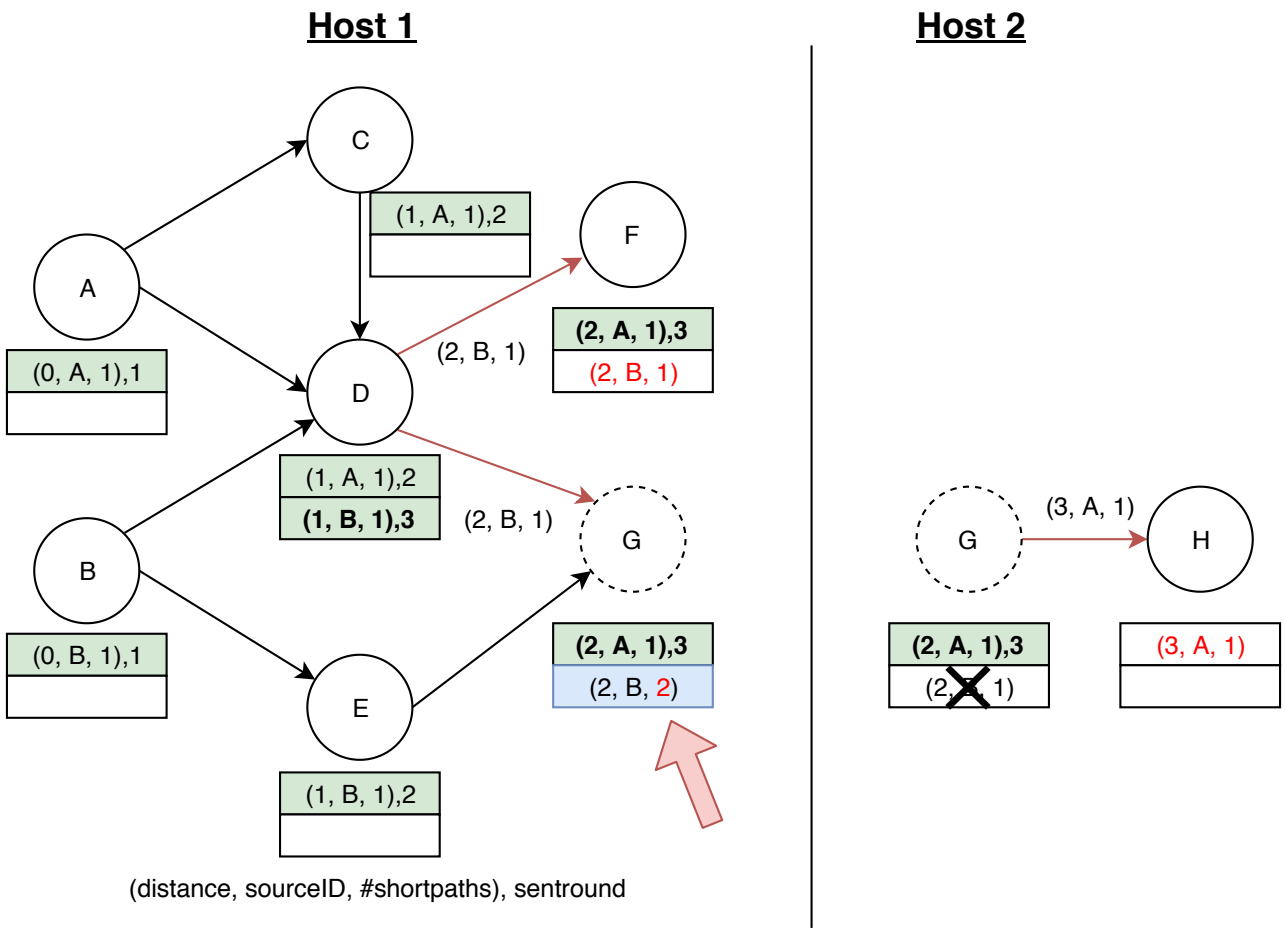
Redundant Synchronization (I)

Beginning of Round 3: Synchronize all data on proxy *G*



Redundant Synchronization (II)

After compute, stale value on host 2: needs synchronization again!

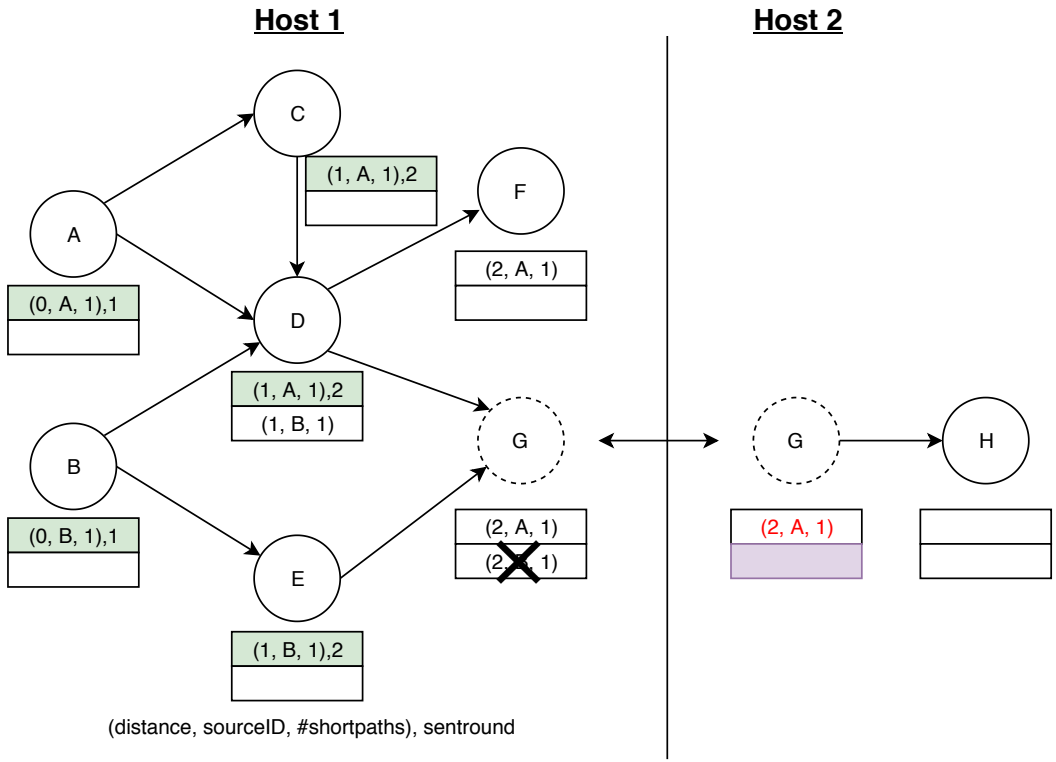


Optimization: Delayed Synchronization in D-Galois

Delayed Synchronization

Synchronize updated data associated with a source on a proxy only if that data meets the message send rule's conditions

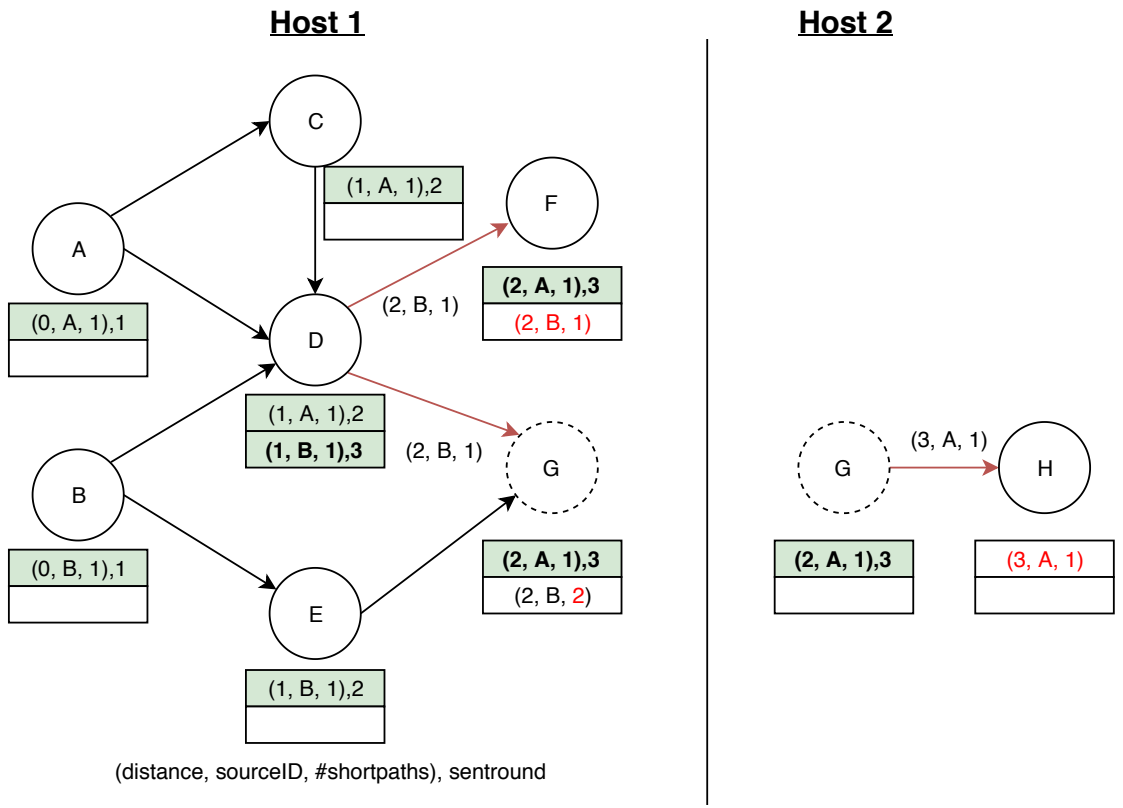
Beginning of Round 3, sync only source A data on G
(distance 2 + position 1 = round 3)



- Intuition: data not read until round it is sent
- Availability of proxies allows delaying synchronization

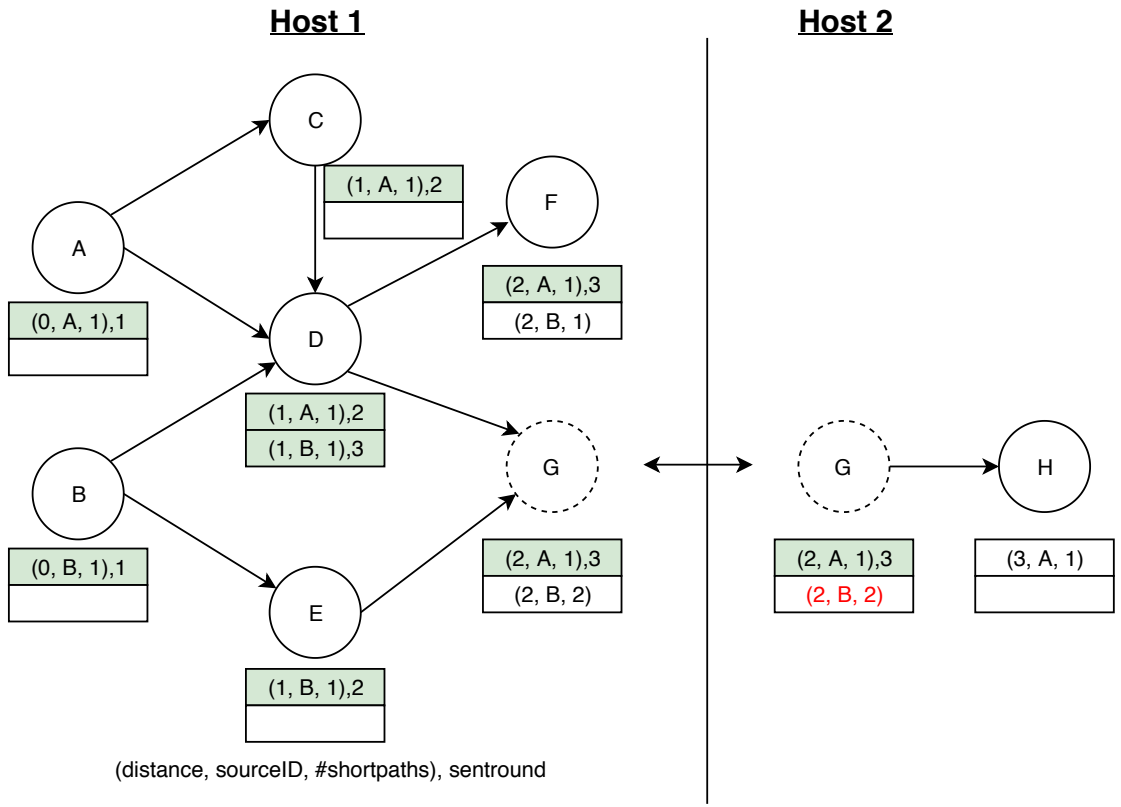
Delayed Synchronization Example Continued (I)

Round 3 compute



Delayed Synchronization Example Continued (II)

Beginning of Round 4: synchronize source B data on proxy G (distance 2 + position 2 = round 4)



Delayed sync reduces network congestion and communication volume

Outline

Experimental Setup: Evaluated Algorithms

1. Asynchronous Brandes BC (ABBC)
2. Maximal Frontier BC (MFBC), sparse-matrix-based
3. Synchronous Brandes BC (SBBC), Brandes in D-Galois
4. Min-Rounds BC (MRBC)

	System	(A)synchronous?	Distributed?	Batching?
ABBC	Galois	Async	N	N
MFBC	CTF	Sync	Y	Y
SBBC	D-Galois	Sync	Y	N
MRBC	D-Galois	Sync	Y	Y

We focus on SBBC and MRBC

- ABBC excellent for high diameter graphs if fits in memory
- MFBC performs moderately well, slows as graphs grow

Experimental Setup: Platform

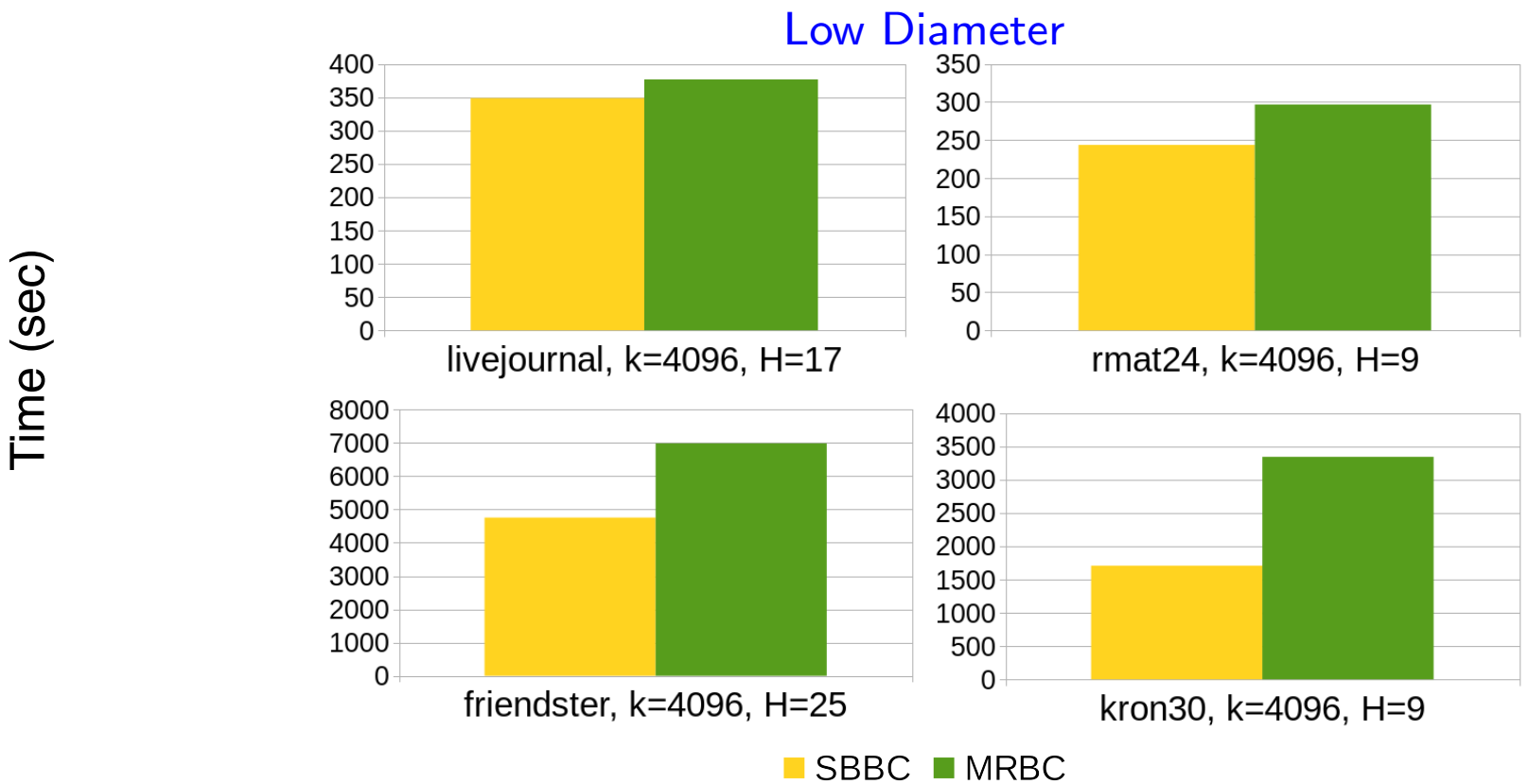
	Low Diameter				High Diameter			
	livejournal	rmat24	friendster	kron30	indochina04	road-europe	gsh15	clueweb12
$\|V\|$	4.8M	17M	66M	1,073M	7.4M	174M	988M	978M
$\|E\|$	69M	268M	3,612M	17,091M	194M	348M	33,877M	42,574M
H (Estimated Diameter)	17	9	25	9	45	22541	103	501

- Platform: Stampede2's Skylake cluster
  - Intel Xeon Platinum 8160, 48 cores on 2 sockets per machine
  - 2.1GHz clock rate, 192GB DDR4 RAM
- Graphs run on up to 256 machines
- Low diameter graphs ≤ 25, high diameter greater than 25
  - Web crawls (such as clueweb12) also high-diameter

Execution Times, Low Diameter Graphs

SBBC: $O(k \cdot H)$ BSP rounds **MRBC**: $O(k + H)$ BSP rounds

H = largest shortest path distance for k sources

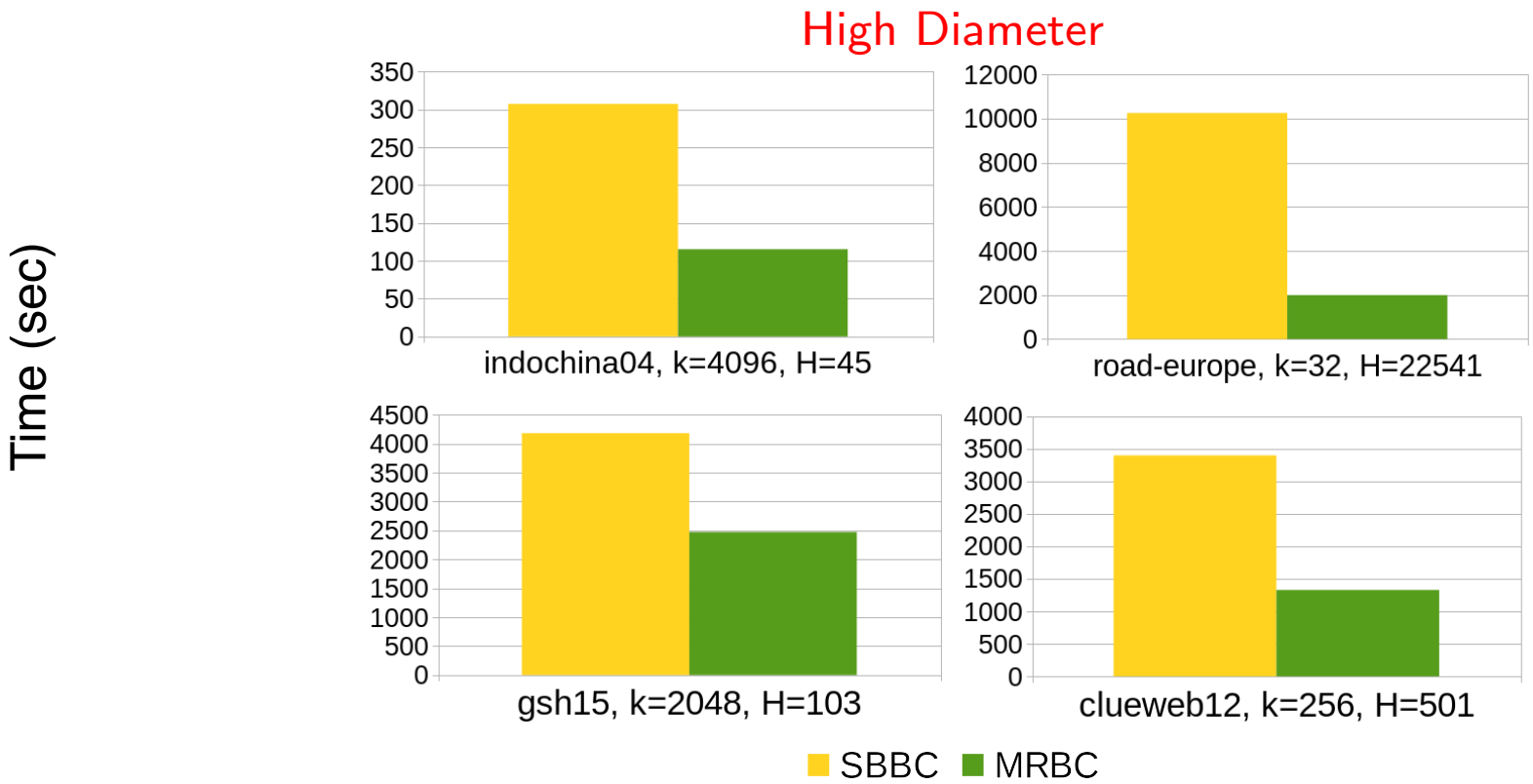


- MRBC round reduction (which leads to communication improvements) does not outweigh compute overhead

Execution Times, High Diameter Graphs

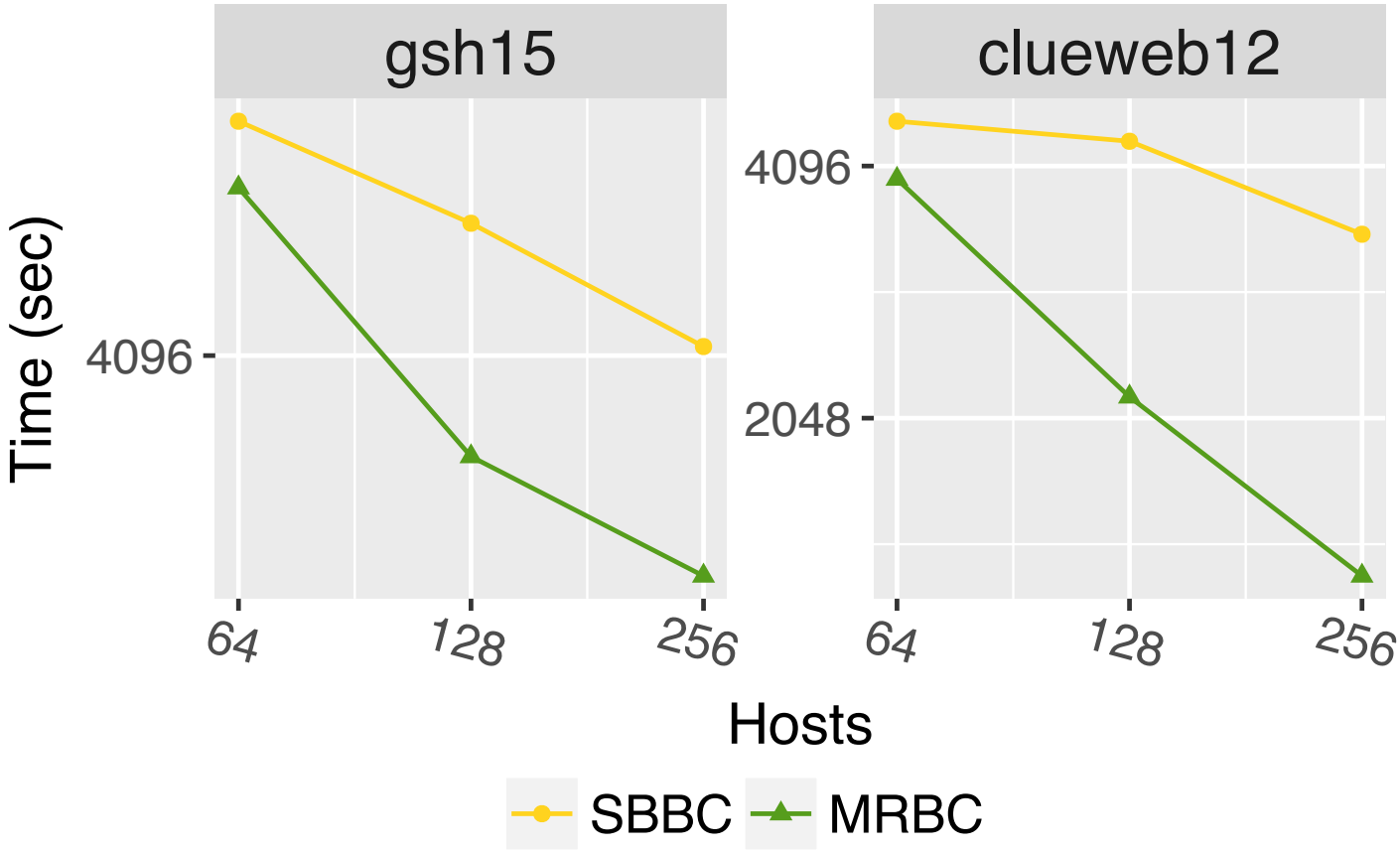
SBBC: $O(k \cdot H)$ BSP rounds **MRBC**: $O(k + H)$ BSP rounds

H = largest shortest path distance for k sources



- MRBC outperforms SBBC (2.8× faster): round reduction and communication improvement more significant

Execution Time of SBBC/MRBC from 64 to 256 Hosts



- Both SBBC and MRBC scale as number of hosts increase
- Communication time of SBBC does not scale as well compared to MRBC

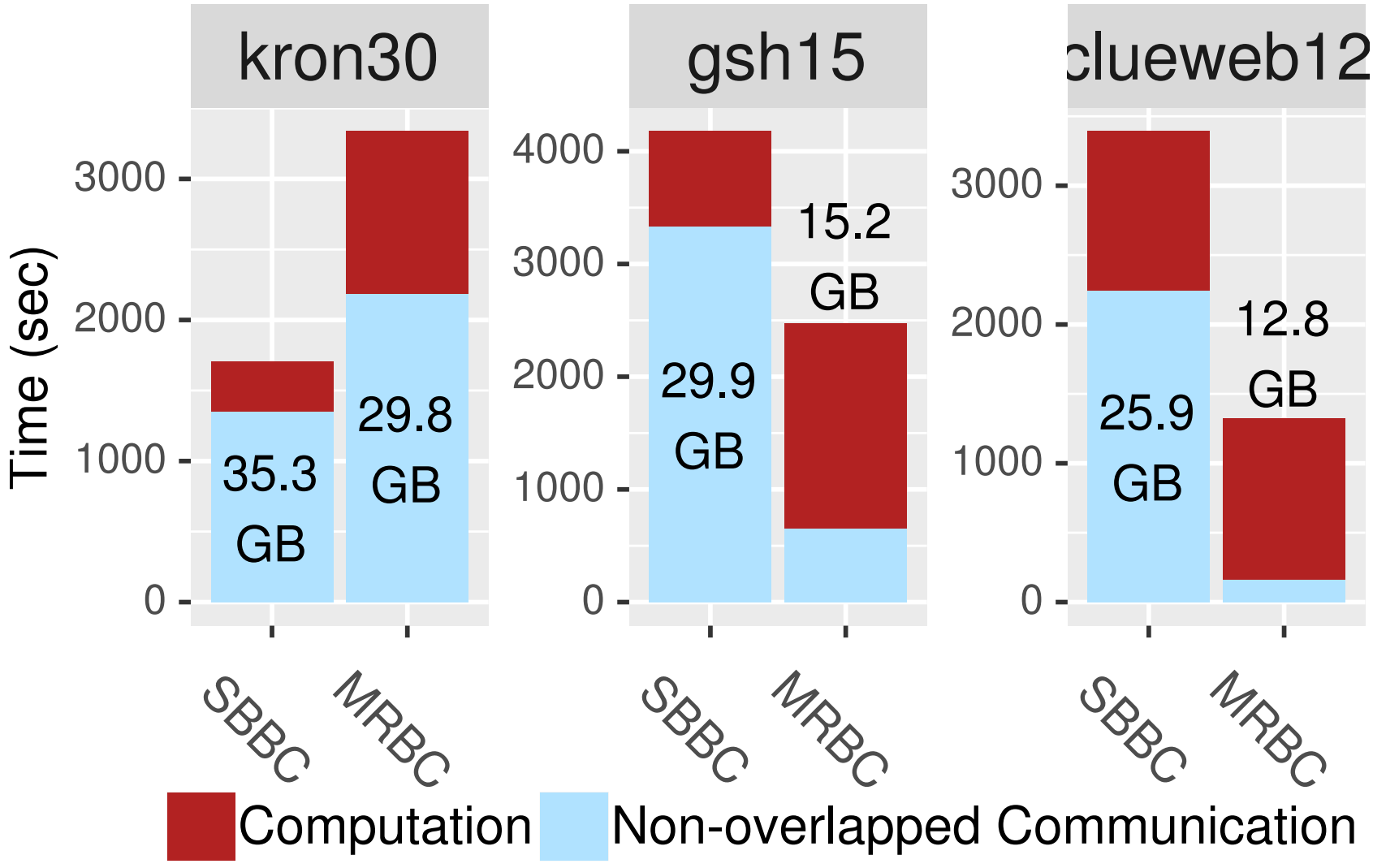
Conclusion

- Presented round-efficient distributed APSP and BC algorithm (MRBC) that improves communication by pipelining message sends
- MRBC in D-Galois over Brandes BC: 14× reduction in rounds, 2.8× speedup for high-diameter graphs

Source Code:
`https://github.com/IntelligentSoftwareSystems/Galois/`
Artifact:
`https://zenodo.org/record/2399798`

Backup Slides

Execution Time of SBBC/MRBC at 256 Hosts, Breakdown



More Topics Covered in the Paper

- Termination detection routine for Min-Rounds APSP which reduces the round complexity and termination detection in D-Galois
- Proofs of correctness for the algorithm and its optimizations
- More detailed analysis of experiments

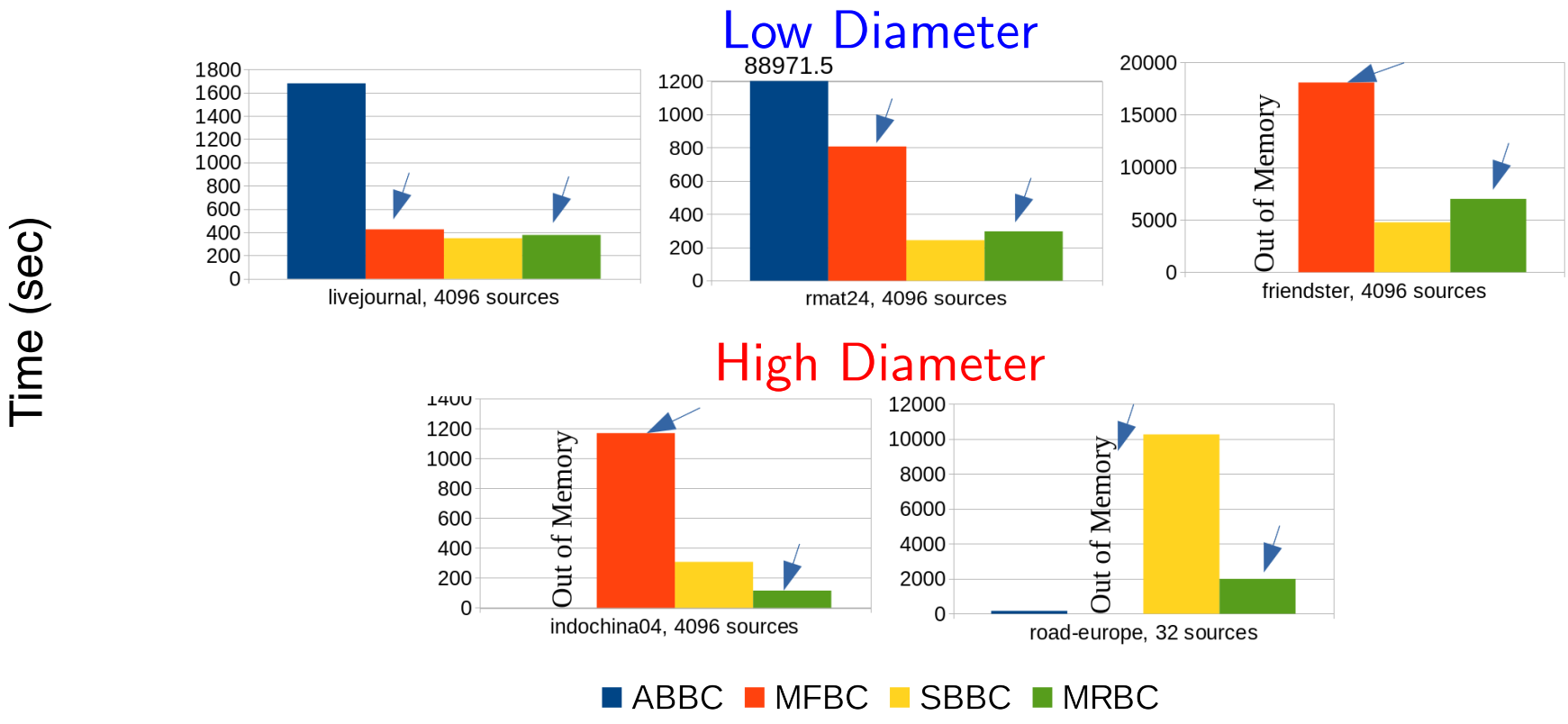
Effect of D-Galois Optimization

Note difference in CONGEST model and D-Galois model

- Number of Rounds
  - MRBC reduces rounds over SBBC in both models (same bounds apply)
- Messages Sent
  - CONGEST: messages sent along edges in SBBC/MRBC are same (only final value is sent)
  - D-Galois
    - SBBC: proxy distance from source updated/sent only once (updated value is final value)
    - MRBC: proxy distance updated multiple times before finalization, i.e. communicate every update, not just when value is finalized

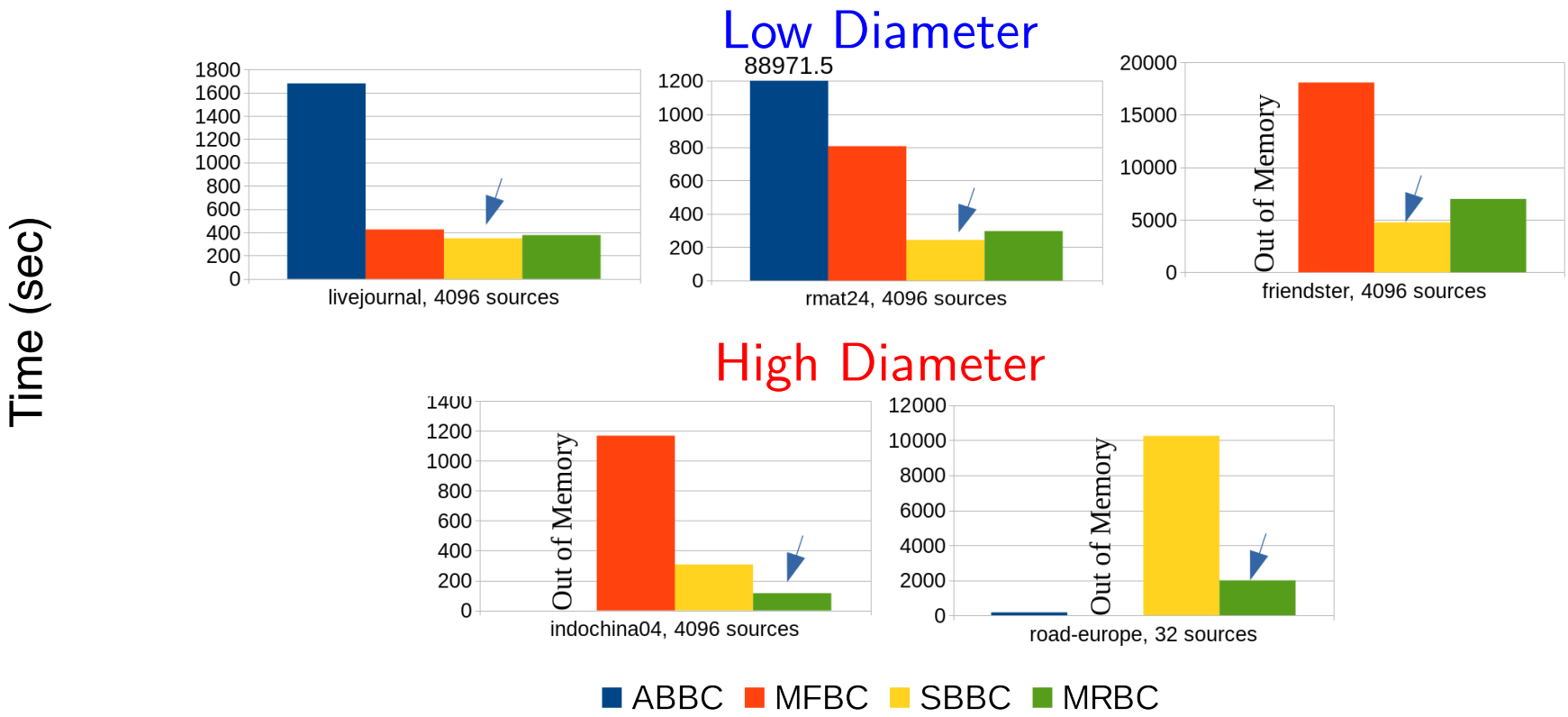
Without optimization, MRBC may send more messages; expected to perform worse

Best Execution Times (1, 32 Hosts) on Small Graphs (I)



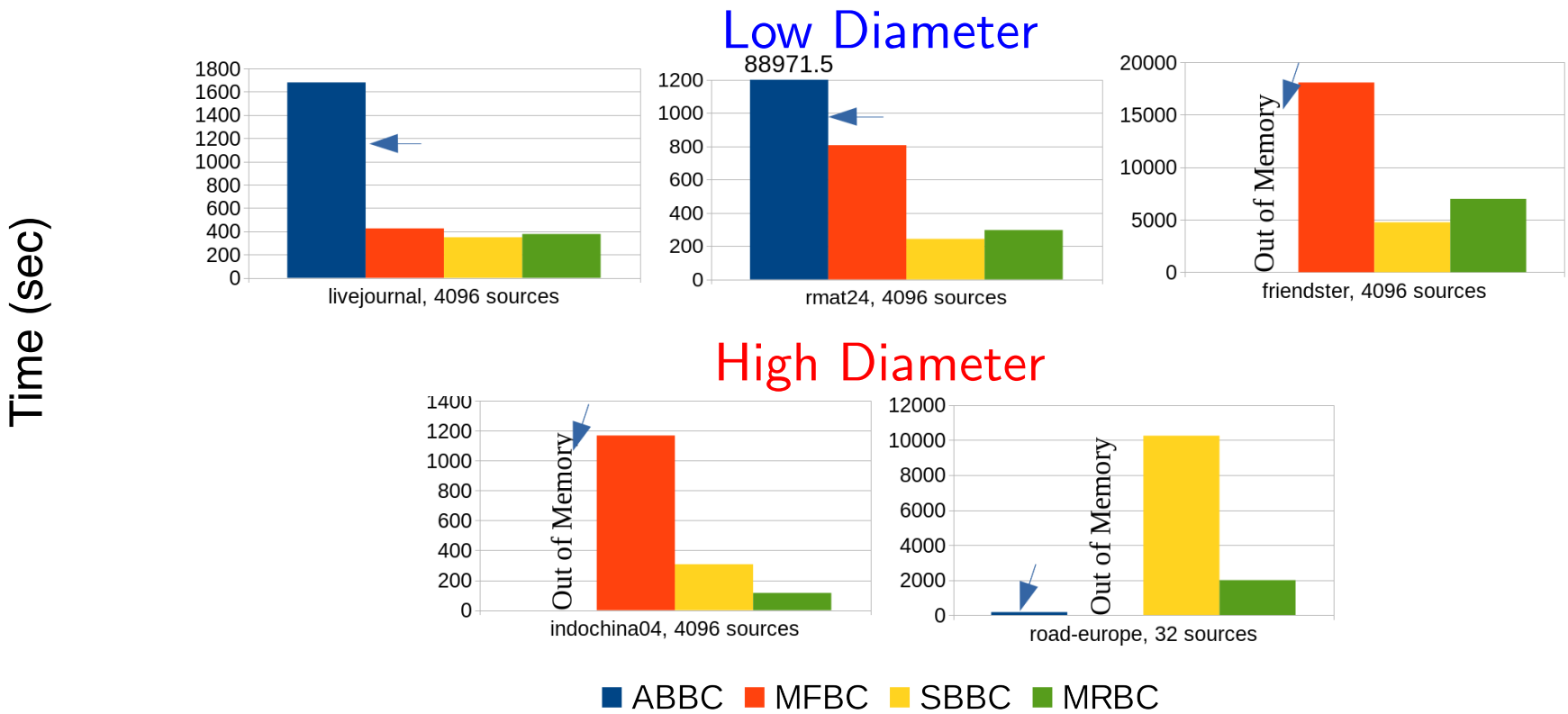
- MRBC is 3× faster than MFBC on average
- SBBC also outperforms MFBC

Best Execution Times (1, 32 Hosts) on Small Graphs (II)



- SBBC best for graphs with low-diameter
- MRBC better for high-diameter

Best Execution Times (1, 32 Hosts) on Small Graphs (III)



- ABBC fast on high diameter graphs
- ABBC extremely slow otherwise