Decentralized Slicing in Mobile Low-Power Wireless Networks

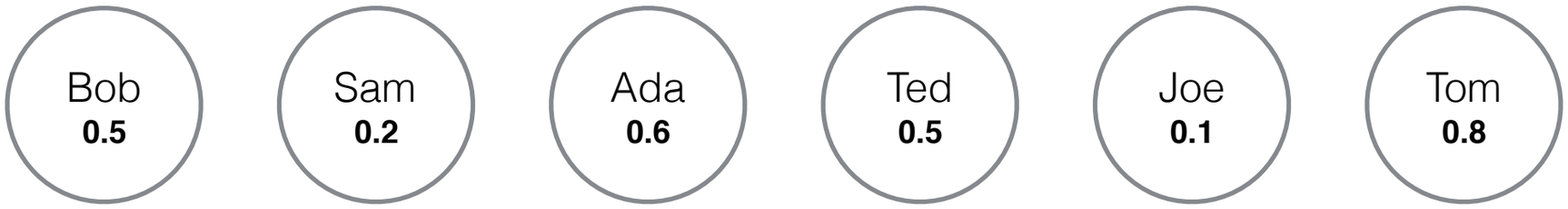
Piotr Jaszkowski, Pawel Sienkowski, Konrad Iwanicki
University of Warsaw

pj306249@students.mimuw.edu.pl
ps319383@students.mimuw.edu.pl
iwanicki@mimuw.edu.pl

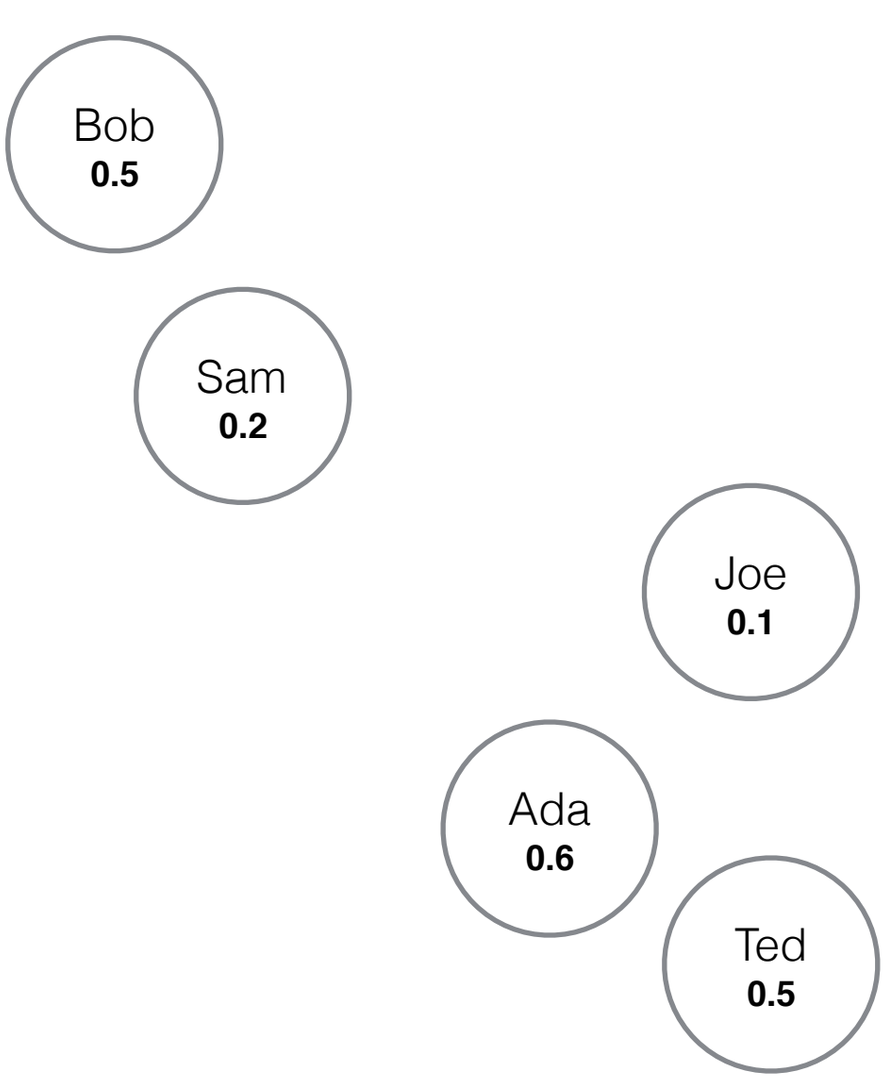
DCOSS 2016 Washington, DC May 27th, 2016

Decentralized Slicing Problem

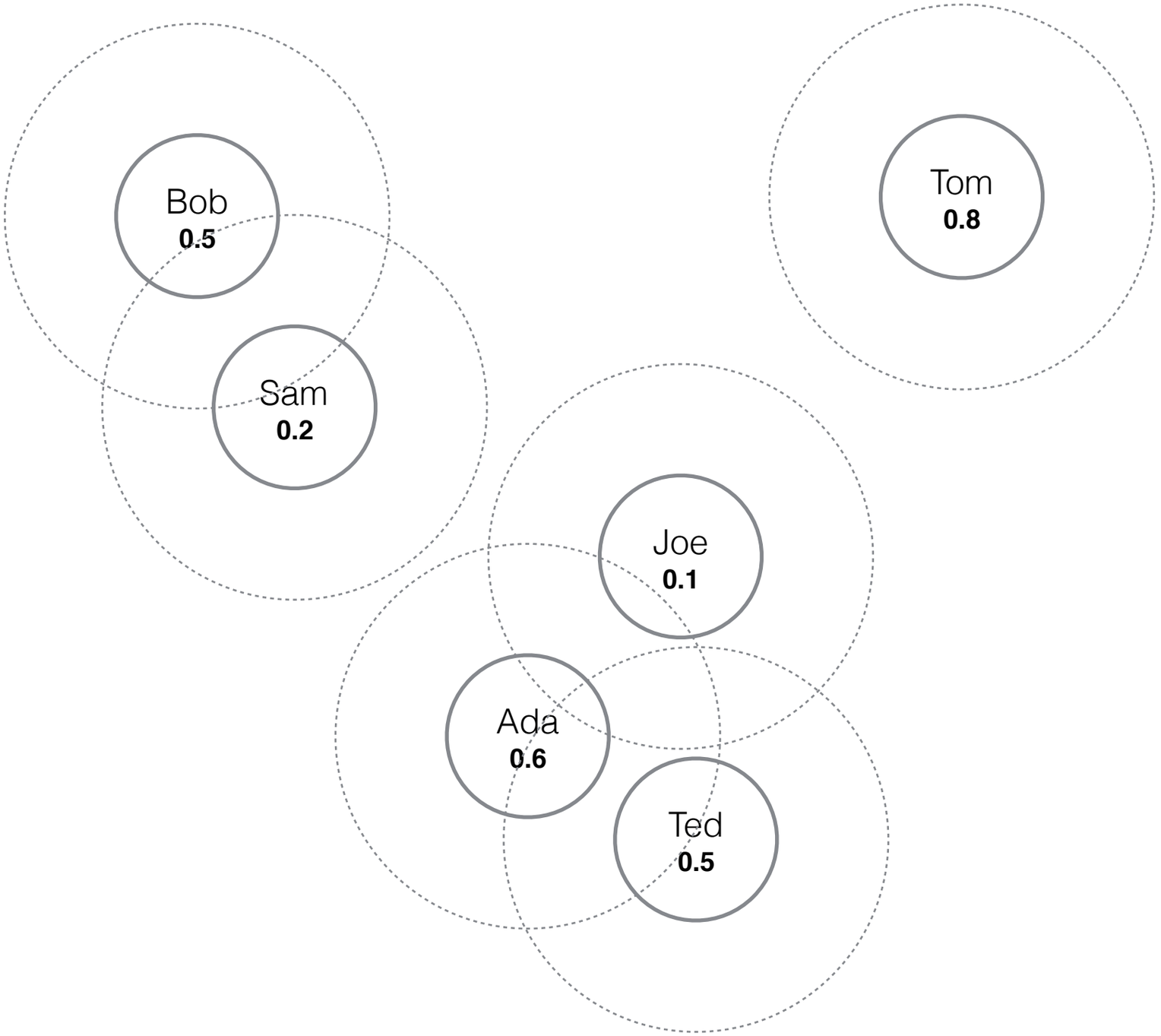
Decentralized Slicing Problem



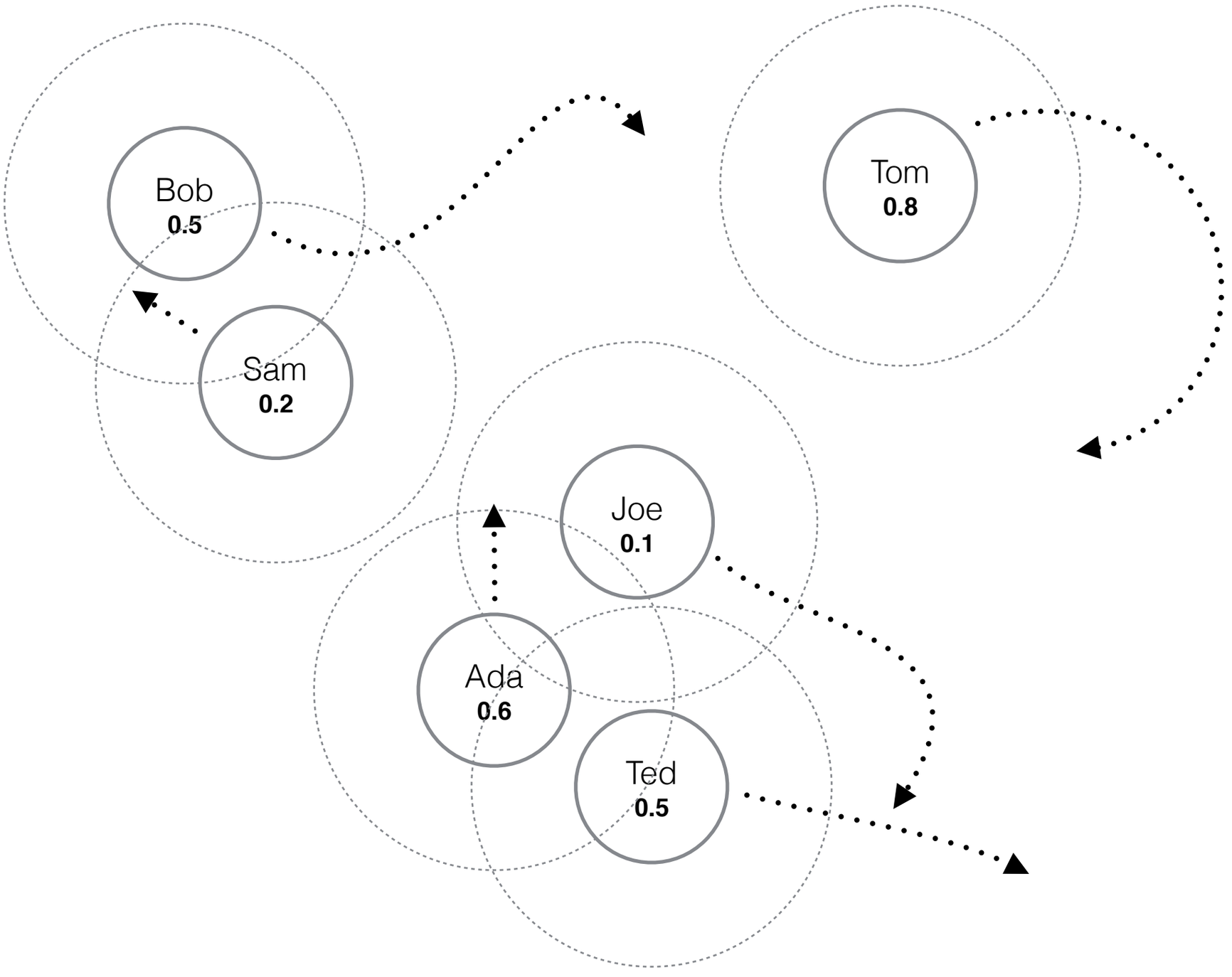
Decentralized Slicing Problem



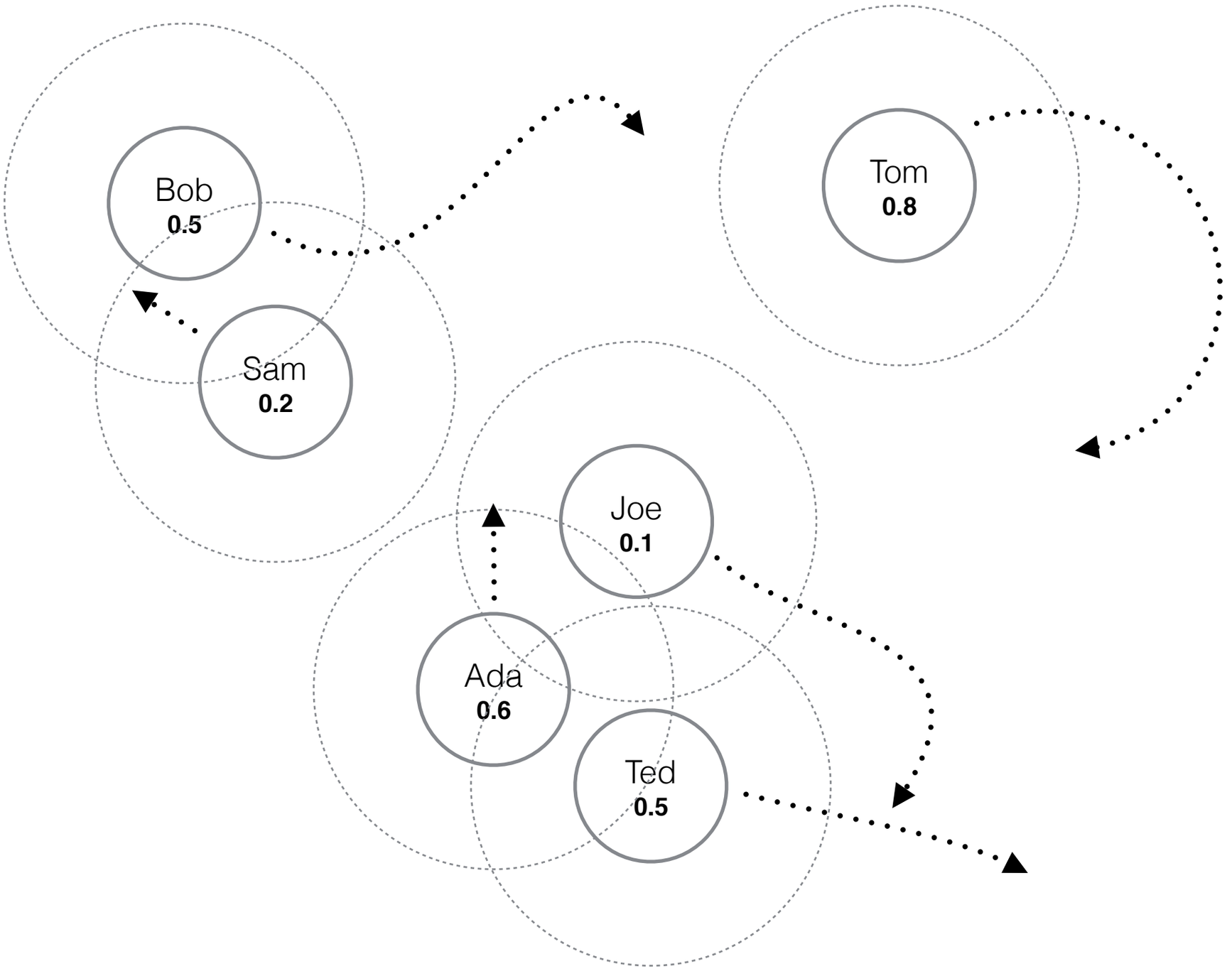
Decentralized Slicing Problem



Decentralized Slicing Problem

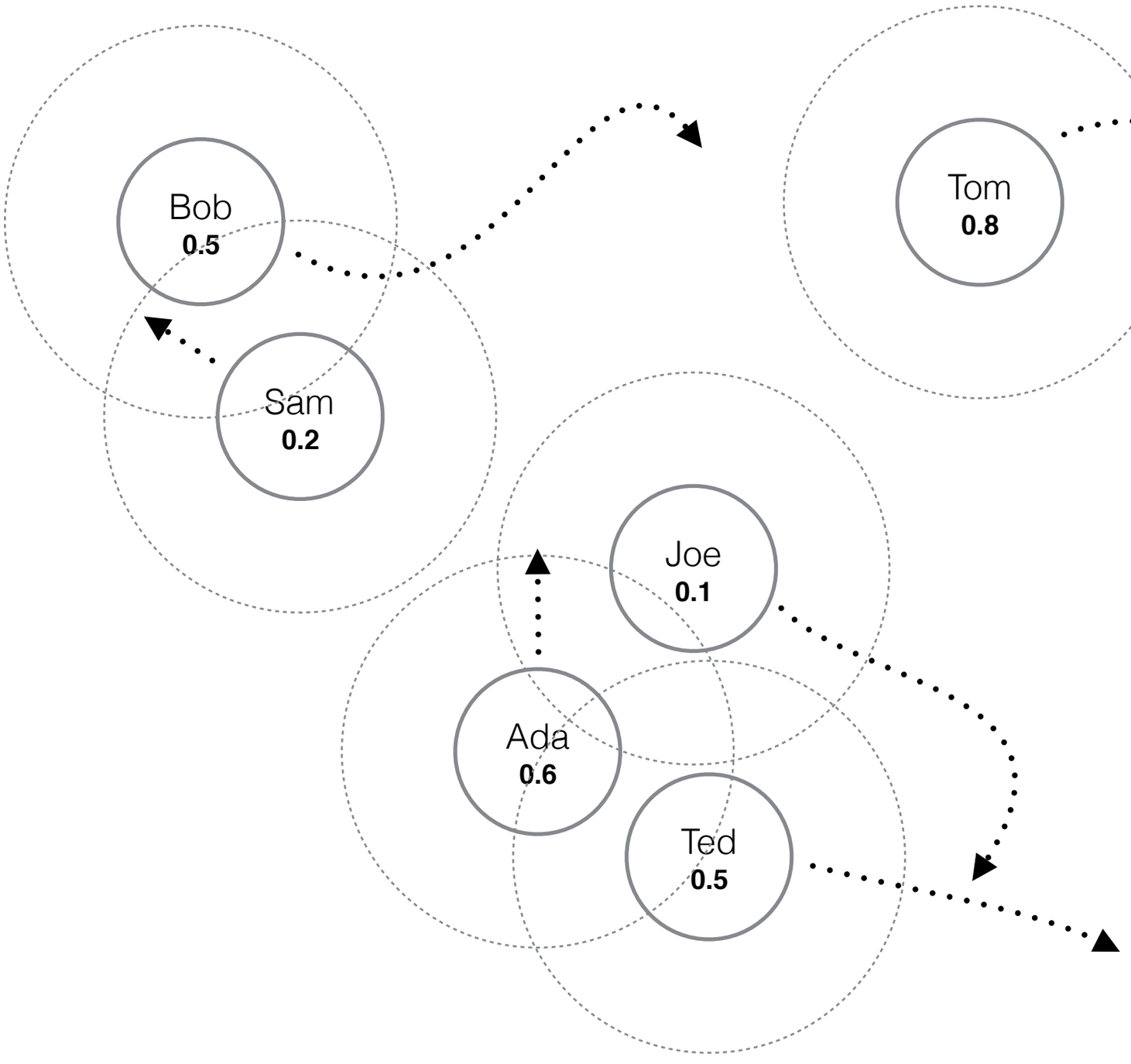


Decentralized Slicing Problem



Tom	0.8
Ada	0.6
Ted	0.5
Bob	0.5
Sam	0.2
Joe	0.1

Decentralized Slicing Problem



Slice 1: 1/6	Tom	0.8
Slice 2: 3/6	Ada	0.6
	Ted	0.5
	Bob	0.5
Slice 3: 2/6	Sam	0.2
	Joe	0.1

Slice Disorder Measure

Definition

$$SDM = \sum_{node} |actual\_slice(node) - slice\_estimate(node)|$$

Slice Disorder Measure

Definition

$$SDM = \sum_{node} |actual\_slice(node) - slice\_estimate(node)|$$

Example

	id	value	estimate
Slice 1	Tom	0.8	**Slice 2**
Slice 2	Ada	0.6	Slice 1
	Ted	0.5	Slice 2
	Bob	0.5	Slice 3
Slice 3	Sam	0.2	Slice 3
	Joe	0.1	Slice 3

$$SDM = \underset{\text{Tom}}{|1 - 2|} + \underset{\text{Ada}}{|2 - 1|} + \underset{\text{Ted}}{|2 - 2|} + \underset{\text{Bob}}{|2 - 3|} + \underset{\text{Sam}}{|3 - 3|} + \underset{\text{Joe}}{|3 - 3|} = 1 + 1 + 1 + 0 = 3$$

Applications

- gamification mechanisms
- self-division of a robobee swarm
- finding potential cluster heads in an area hierarchy over sensors

Related work

- To this date, a few algorithms have been proposed:

JK, mod-JK, Dynamic ranking by sampling, Sliver, Q-digest

- All of the solutions proposed so far either:
  - relay on a global connectivity of a network (point-to-point communication)
  - assume that nodes are static (so an overlay network can be created)
  - were not designed for resource-constrained devices (memory or bandwidth)

Our algorithms: *BSort*

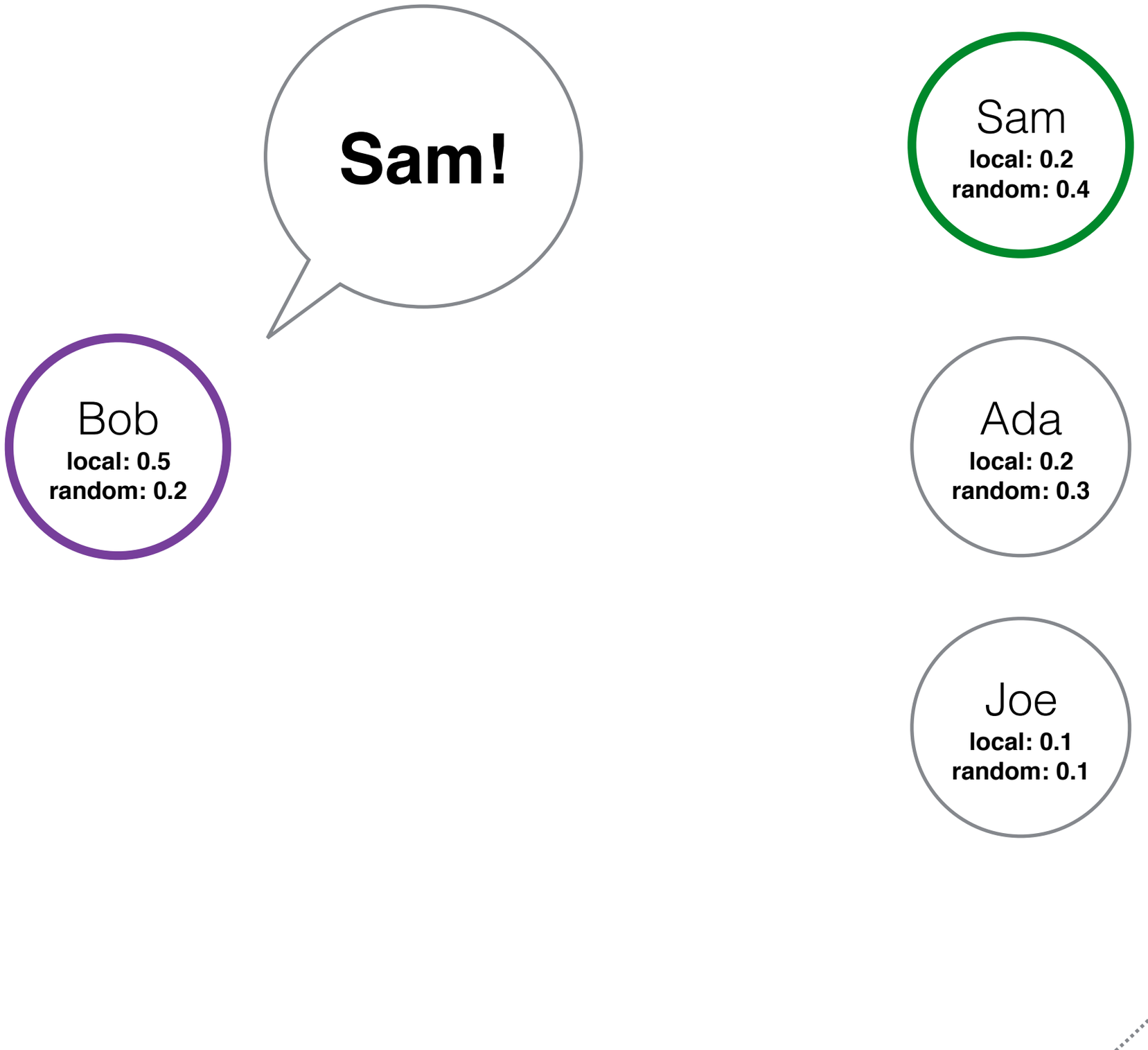
Sam
local: 0.2
random: 0.4

Bob
local: 0.5
random: 0.2

Ada
local: 0.2
random: 0.3

Joe
local: 0.1
random: 0.1

Our algorithms: *BSort*



Our algorithms: *BSort*

Hey Sam!
my local > your local,
but
my random < your random.

Let's swap!

Sam
local: 0.2
random: 0.4

Bob
local: 0.5
random: 0.2

Ada
local: 0.2
random: 0.3

Joe
local: 0.1
random: 0.1

Our algorithms: *BSort*

Cool Sam! Now
my local > your local,
and
my random > your random.
That's what I like!

Sam
local: 0.2
random: 0.2

Bob
local: 0.5
random: 0.4

Ada
local: 0.2
random: 0.3

Joe
local: 0.1
random: 0.1

We have **10 slices**,
random values are from **[0.0, 1.0)**
and my random value is **0.4**.
So, I think I'm in the **4th** slice!

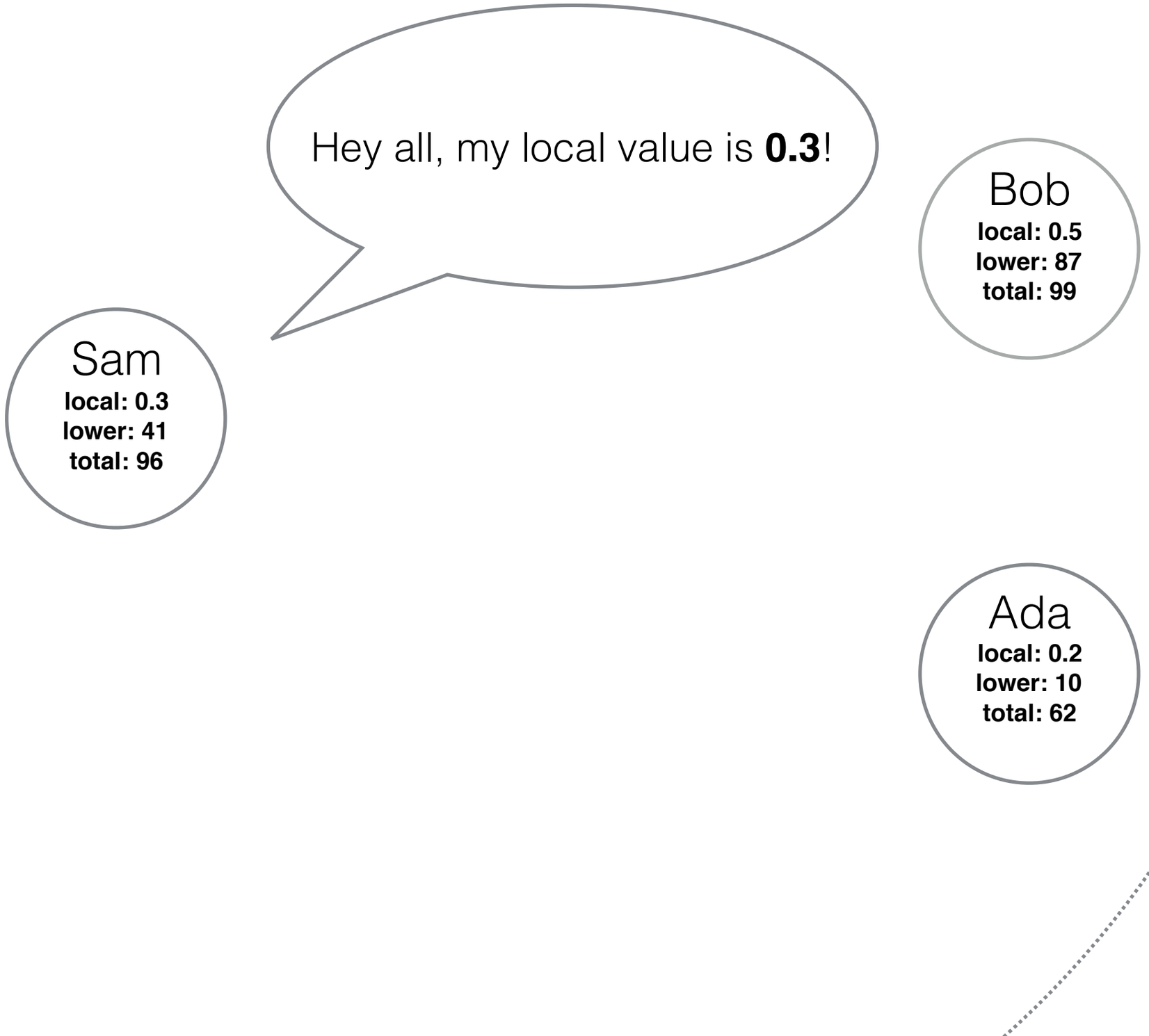
Our algorithms: *ICount*

Bob
local: 0.5
lower: 87
total: 99

Sam
local: 0.3
lower: 41
total: 96

Ada
local: 0.2
lower: 10
total: 62

Our algorithms: *ICount*



Our algorithms: *ICount*

0.3, huh? Less than my value…

Bob
local: 0.5
lower: 87 + 1
total: 99 + 1

Sam
local: 0.3
lower: 41
total: 96

0.3 is more than my **0.2** :(

Ada
local: 0.2
lower: 10
total: 62 + 1

Our algorithms: *ICount*

Sam
local: 0.3
lower: 41
total: 96

We have **10** slices,
84% of nodes I met
had greater values then mine,
so I think I am in the **9th** slice.

Bob
local: 0.5
lower: 88
total: 100

Ada
local: 0.2
lower: 10
total: 63

Digression: SharedState

- A scheme for data distribution in a wireless network
- Idea:
  - Each node maintains a set of values
  - Initially nodes store only own values in their sets
  - Each node periodically broadcasts a subset of its set
  - Recipients merge local and received sets
  - Random entries are discarded to meet size limits

Our algorithms: *LCount*

Sam

local: 0.3
lower: 12
total: 54

{**(Sam, 0.3), (Ada, 0.6),** (Joe, 0.2)}

Bob

local: 0.5
lower: 30
total: 64

{(Bob, 0.3), (Ted, 0.6), (Joe, 0.2)}

Our algorithms: *LCount*

Hey all,
here is a sample of our population:
{(Sam, 0.3), (Ada, 0.6)}

Sam

local: 0.3
lower: 12
total: 54

{**(Sam, 0.3), (Ada, 0.6),** (Joe, 0.2)}

Bob

local: 0.5
lower: 30
total: 64

{(Bob, 0.3), (Ted, 0.6), (Joe, 0.2)}

Our algorithms: *LCount*

Sam

local: 0.3
lower: 12
total: 54

{**(Sam, 0.3), (Ada, 0.6)**, (Joe, 0.2)}

Sam's value is lower,
Ada's is greater.

Bob

local: 0.5
lower: 30 + 1
total: 64 + 2

{(Bob, 0.3), (Ted, 0.6), (Joe, 0.2),
(Sam, 0.3), (Ada, 0.6)}

Our algorithms: *LCount*

I need to discard some entries from my local set to satisfy the limit.

Sam

local: 0.3
lower: 12
total: 54

{(Sam, 0.3), (Ada, 0.6), (Joe, 0.2)}

Bob

local: 0.5
lower: 31
total: 66

{(Bob, 0.3), ~~**(Ted, 0.6)**~~, (Joe, 0.2), ~~**(Sam, 0.3)**~~, (Ada, 0.6)}

Our algorithms: *LCount*

Sam

local: 0.3
lower: 12
total: 54

{(Sam, 0.3), (Ada, 0.6), (Joe, 0.2)}

We have **10** slices, **53%** of nodes I met had greater values then mine, so I think I am in the **6th** slice.

Bob

local: 0.5
lower: 31
total: 66

{(Bob, 0.3), (Joe, 0.2), (Ada, 0.6)}

Digression: Counting Sketch

- Probabilistic data structure
- Aims cardinality estimation problem
- Uses sublinear-space
- Operations:
  - *add(element)* - idempotent
  - *merge(sketch)* - idempotent, associative and commutative
  - *count()* - retrieves cardinality approximation

Our algorithms: *SCount*

Hey all, here is some info:
{(Sam, 0.3, lower, greater)}

Sam

local: 0.3
sketches: lower, greater

{(Sam, 0.3, lower, greater),
(Ada, 0.6, lower, greater)}

Bob

local: 0.5
sketches: lower, greater

{(Bob, 0.5, lower, greater),
(Joe, 0.2, lower, greater)}

Our algorithms: *SCount*

$local_{Sam} < local_{Bob}$, so $lower_{Bob} := merge(lower_{Bob}, lower_{Sam})$
and
$local_{Sam} > local_{Joe}$, so $greater_{Joe} := merge(greater_{Joe}, greater_{Sam})$

Sam

local: 0.3
sketches: lower, greater

{(Sam, 0.3, lower, greater),
(Ada, 0.6, lower, greater)}

Updating sketches…

Bob

local: 0.5
sketches: lower, greater

{(Bob, 0.5, **lower**, greater),
(Joe, 0.2, lower, **greater**)}

Our algorithms: *SCount*

Sam

local: 0.3
sketches: lower, greater

{(Sam, 0.3, lower, greater),
(Ada, 0.6, lower, greater)}

Adding received entries…

Bob

local: 0.5
sketches: lower, greater

{(Bob, 0.5, lower, greater),
(Joe, 0.2, lower, greater),
(Sam, 0.3, lower, greater)}

Our algorithms: *SCount*

Sam

local: 0.3
sketches: lower, greater

{(Sam, 0.3, lower, greater),
(Ada, 0.6, lower, greater)}

Discarding entries to meet the limit.

Bob

local: 0.5
sketches: lower, greater

{(Bob, 0.5, lower, greater),
~~**(Joe, 0.2, lower, greater)**~~,
(Sam, 0.3, lower, greater)}

Our algorithms: *SCount*

count(greater$_{Bob}$) / (count(lower$_{Bob}$) + count(greater$_{Bob}$)) = 12 / (12 + 68) = **15%**

Sam

local: 0.3
sketches: lower, greater

{(Sam, 0.3, lower, greater),
(Ada, 0.6, lower, greater)}

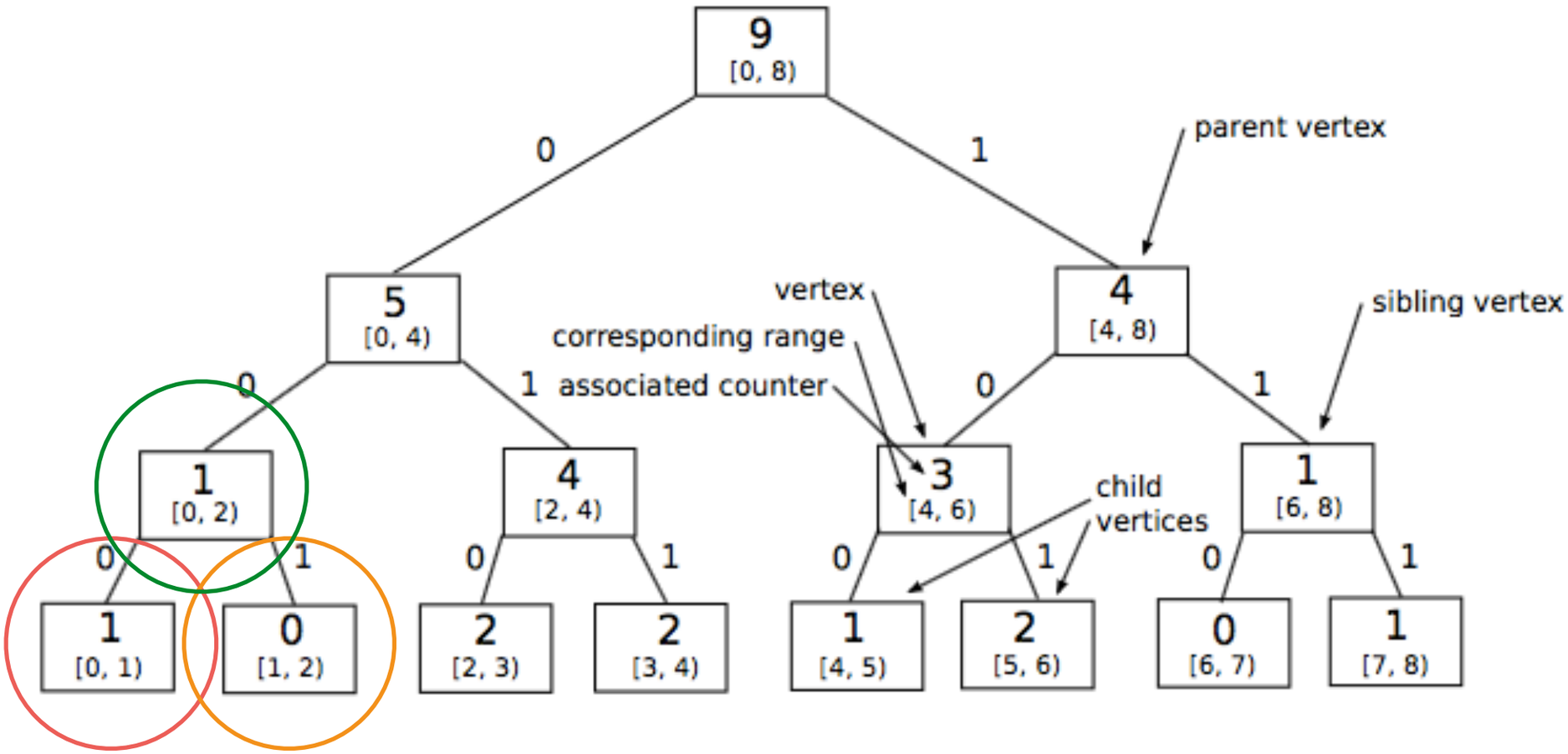
We have **10** slices,
15% of nodes I met had greater values.
I think I'm in the **2nd** slice.

Bob

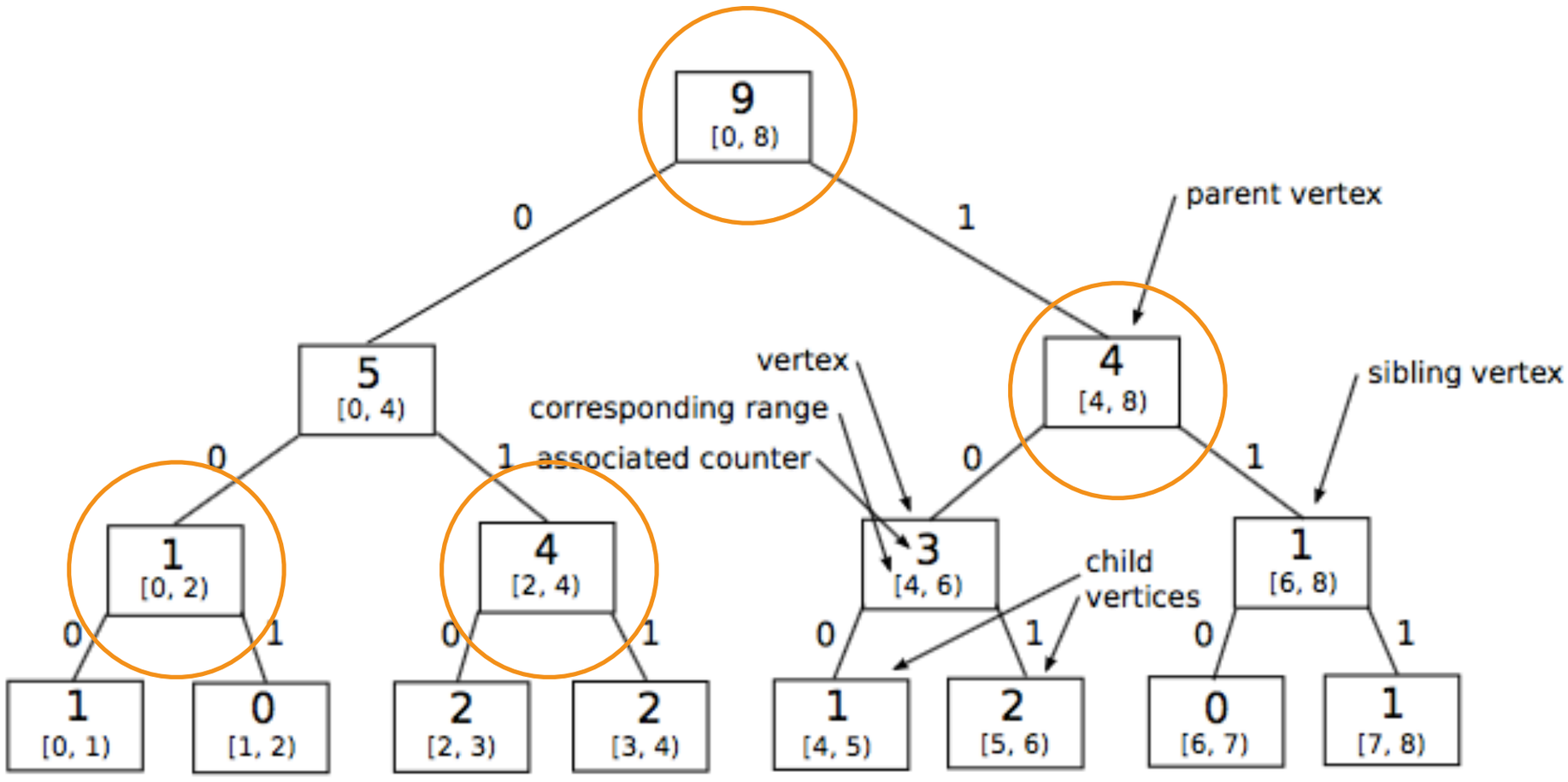
local: 0.5
sketches: lower, greater

{(Bob, 0.5, lower, greater),
(Sam, 0.3, lower, greater)}

Our algorithms: *DTree*



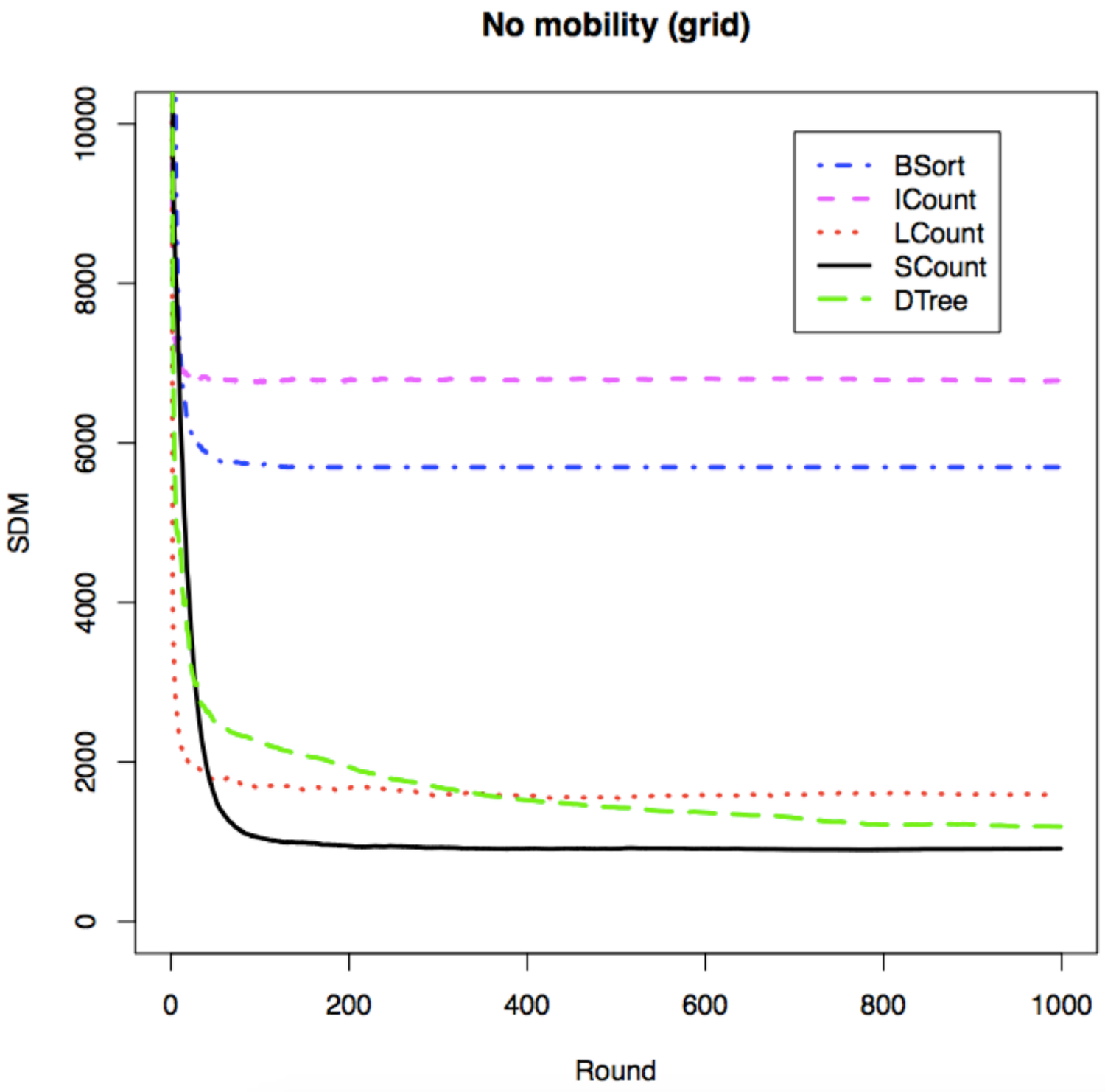
Our algorithms: *DTree*



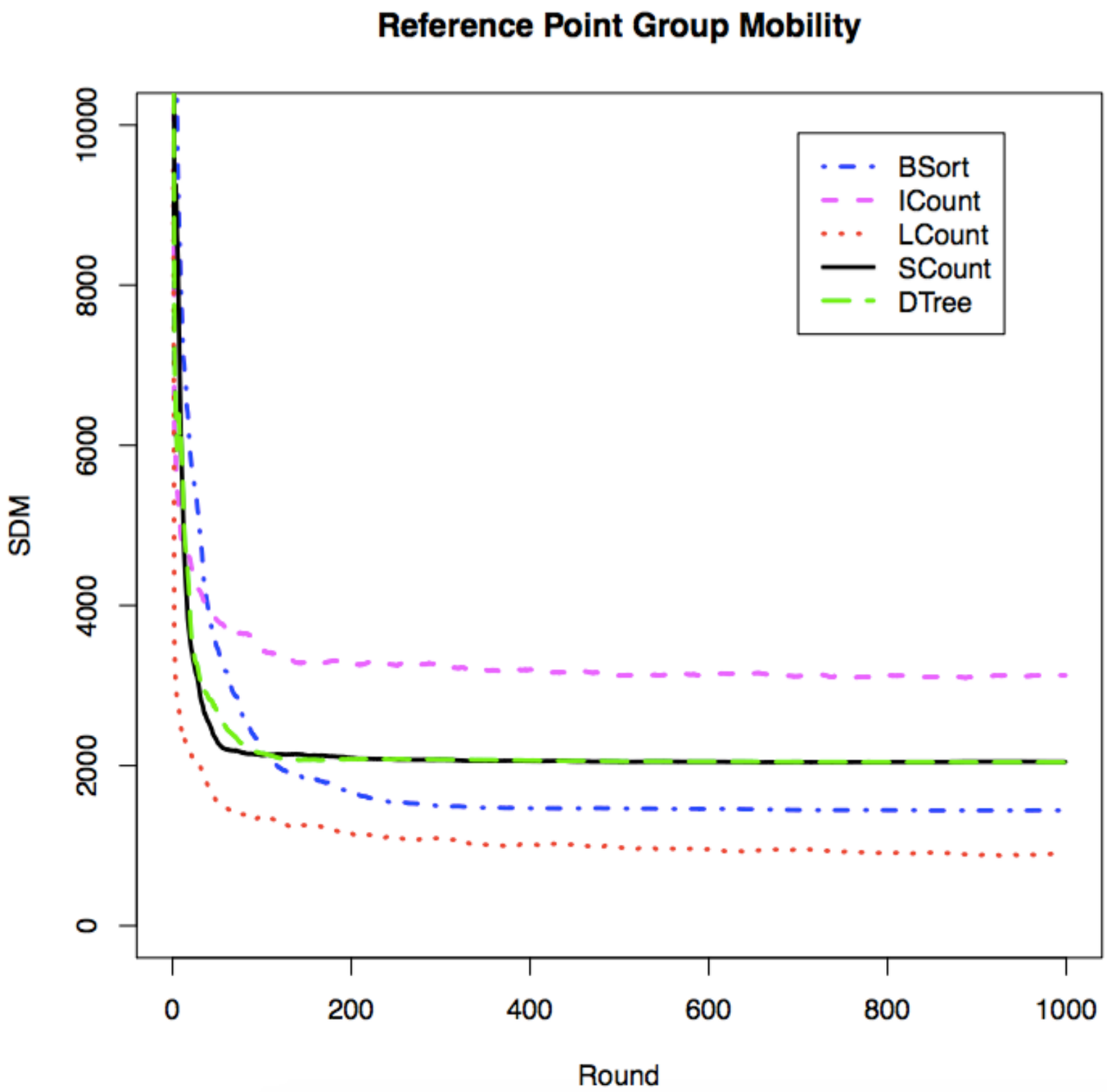
Simulations

- 1024 nodes
- effective radio range: 50 meters, 100 bytes per message
- square-shaped area, side length: 1024 meters
- 100 slices
- 3 mobility patterns:
  - static grid
  - Reference Point Group Mobility
  - Random Waypoint Mobility

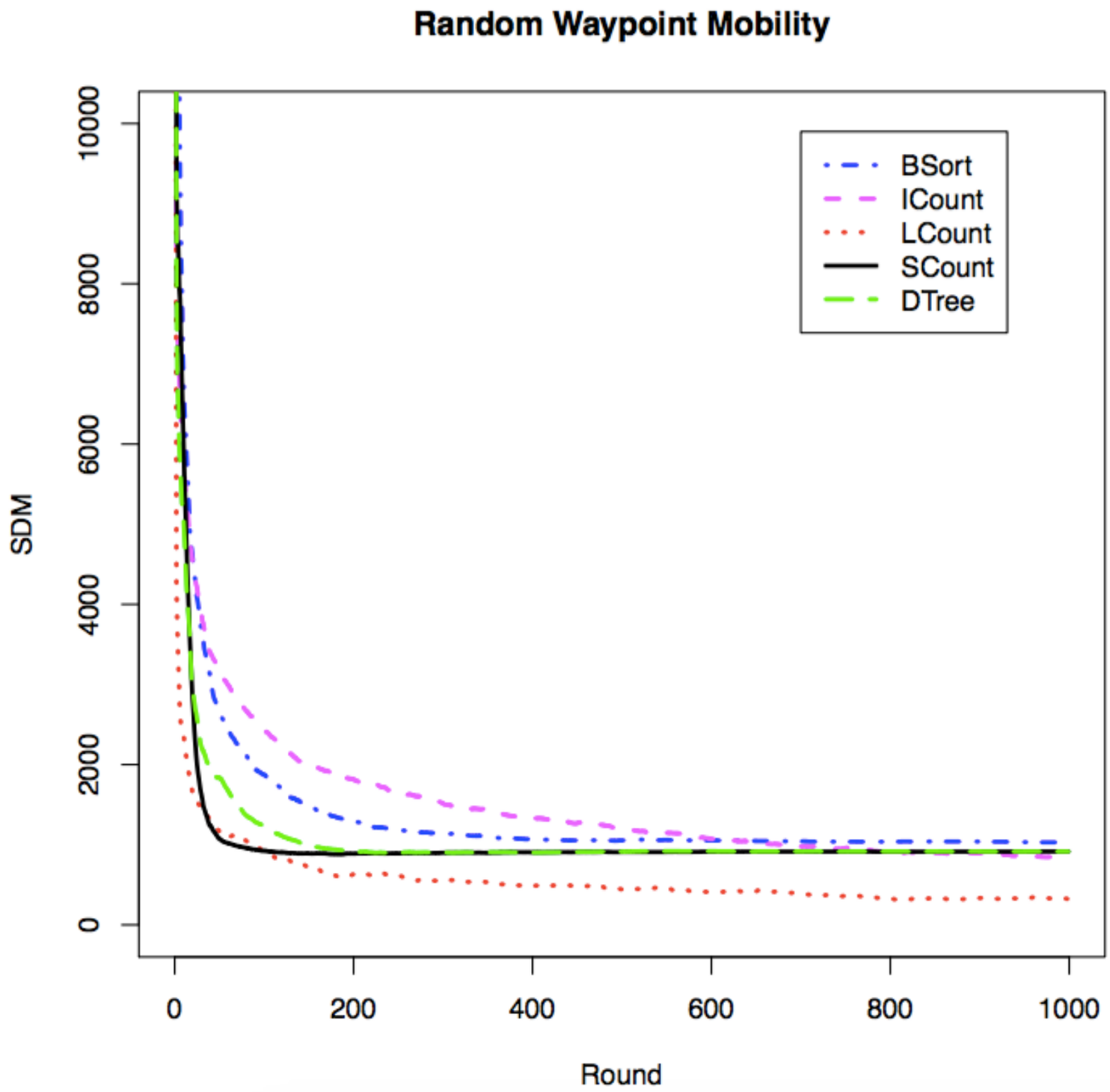
Performance comparison



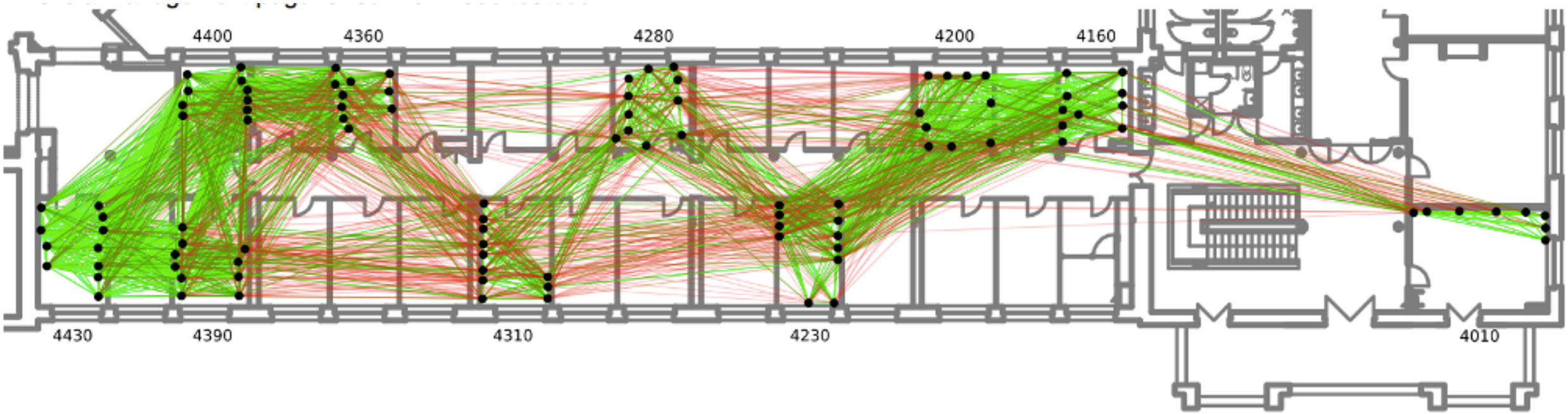
Performance comparison

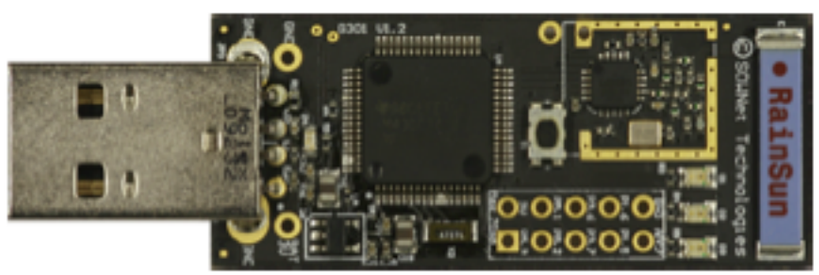


Performance comparison



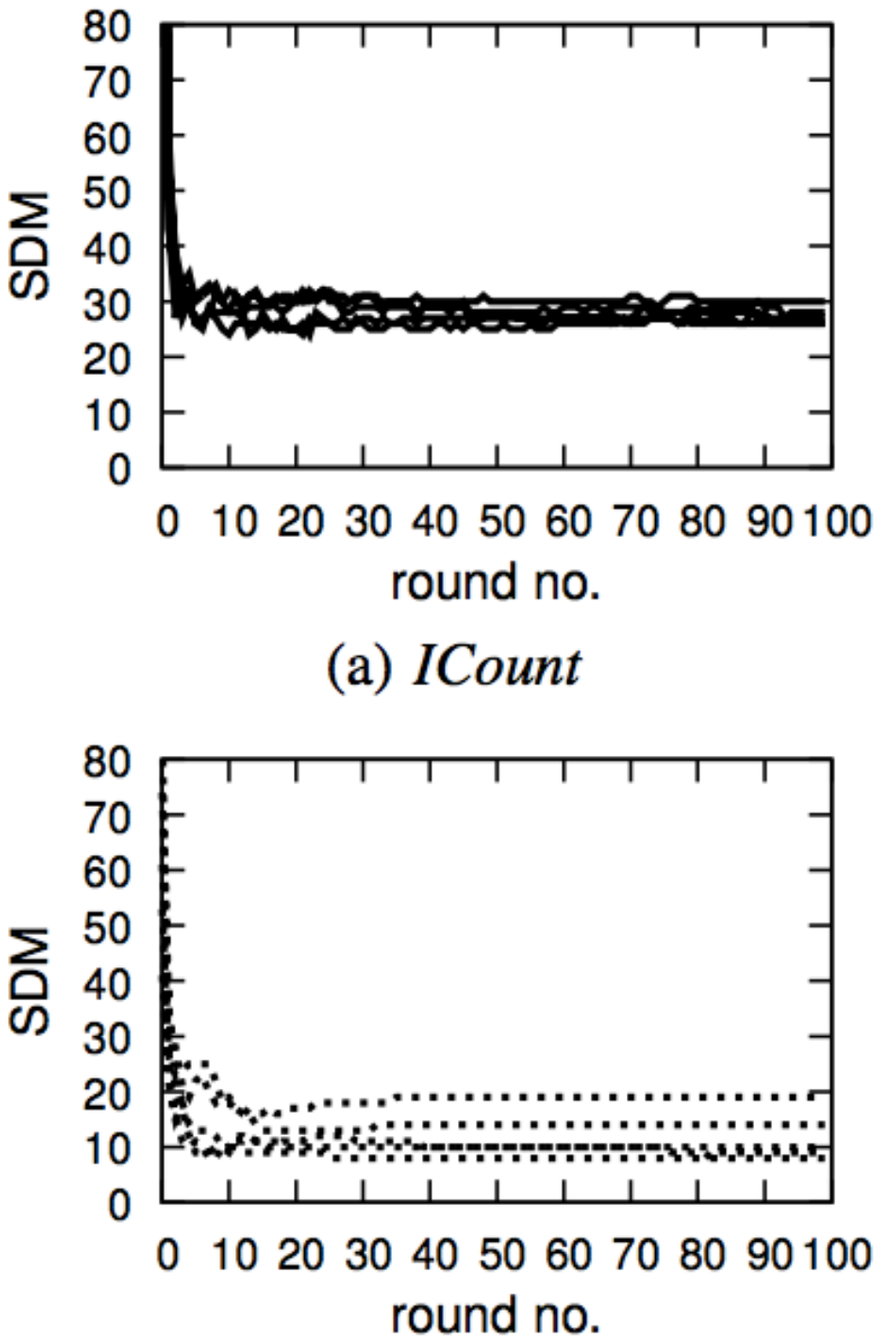
Testbed experiments



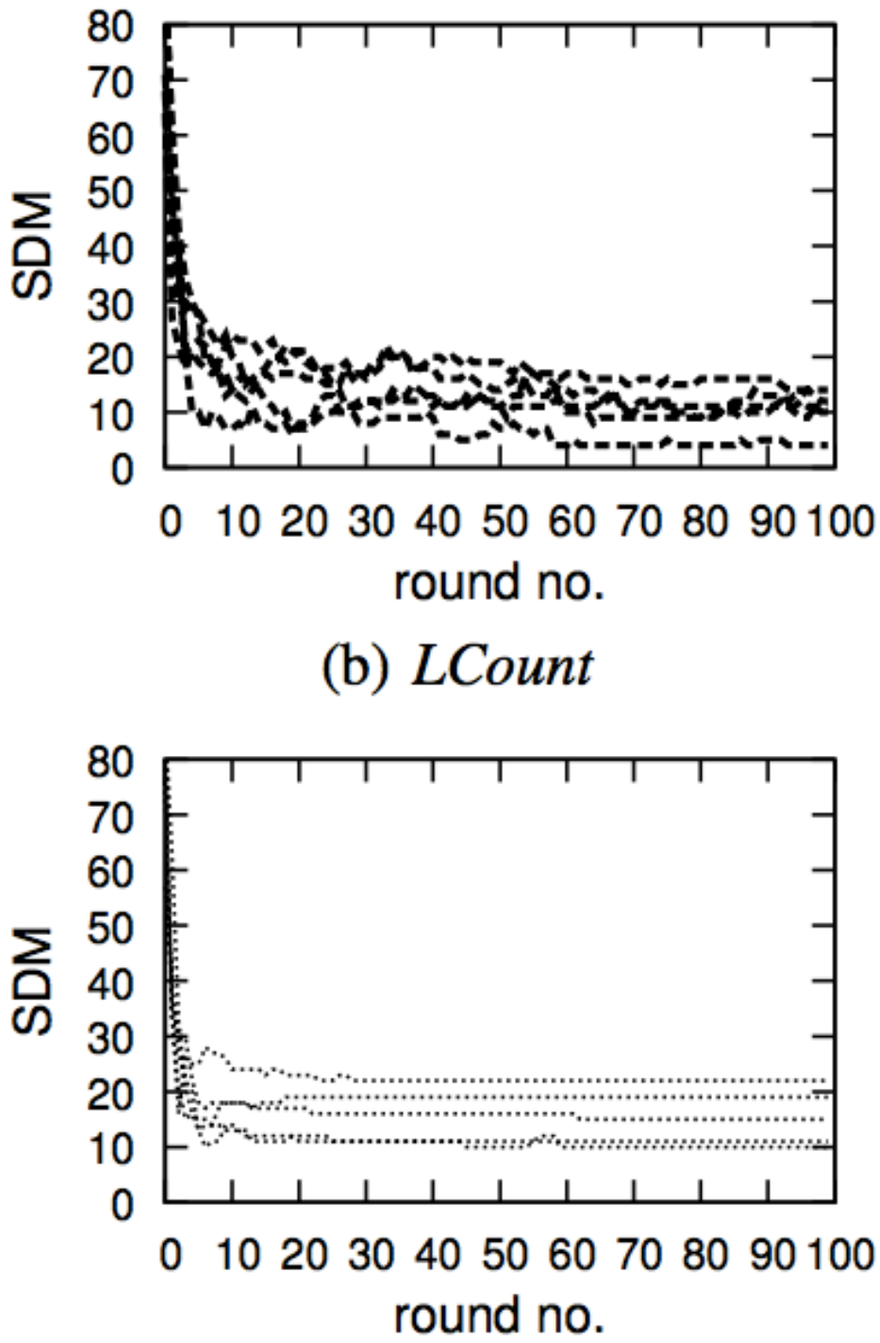


	G-Node
CPU	8 MHz
RAM	8 kb
ROM	116 kb
Radio	500 kb/s

Testbed experiments



(a) *ICount*



(b) *LCount*

(c) *SCount*

(d) *DTree*

Conclusions

- There were some solutions to the Decentralised Slicing Problem
- None of them worked in highly dynamic, low-powered networks
- 2 algorithms have been adapted, 3 new has been designed
- Our new algorithms yield promising results
- Unfortunately, there is no best algorithm - performance depends on configuration, network size and mobility patterns

Thank You

Questions?

Supported by the (Polish) National Science Centre (NCN) within the SONATA programme under grant no. DEC-2012/05/D/ST6/03582.
K. Iwanicki was additionally supported by a scholarship from the (Polish) Ministry of Science and Higher Education for outstanding young scientists.

NATIONAL SCIENCE CENTRE

Ministry of Science and Higher Education
Republic of Poland

Conclusions

Algorithm	Advantages	Drawbacks
BSort	Uses a fixed amount of memory and bandwidth. Guarantees that slice estimates finally converge.	Does not perform well in static networks. Converges slowly. May be difficult to implement.
ICount	Uses a fixed amount of memory and bandwidth. Is easy to implement.	Does not perform well in static and mobile clustered networks. Converges slowly. Has its slice estimates fluctuate even if *SDM* is low.
LCount	Converges fast initially. Has a relatively low final *SDM*. Is easy to implement. Benefits from an increased network throughput.	Converges slowly in later stages. Has its slice estimates fluctuate even if *SDM* is low.
SCount	Converges fast. Has its final *SDM* independent of the device density. Benefits from an increased network throughput. Is relatively easy to implement. Guarantees that slice estimates finally converge.	Has its final *SDM* limited by the accuracy of counting sketches.
DTree	Converges relatively fast. Has its final *SDM* independent of the device density. Benefits from an increased network throughput. Guarantees that slice estimates finally converge.	Has its final *SDM* limited by the accuracy of counting sketches. Is difficult to implement.