Simple, Fast and Deterministic Gossip and Rumor Spreading

Main paper by: B. Haeupler, MIT
Talk by: Alessandro Dovis, ETH

Presentation Outline

- What is gossip?
- Applications
- Basic Algorithms
- Advanced Algorithms
- Other Results & Current Research
- Q & A

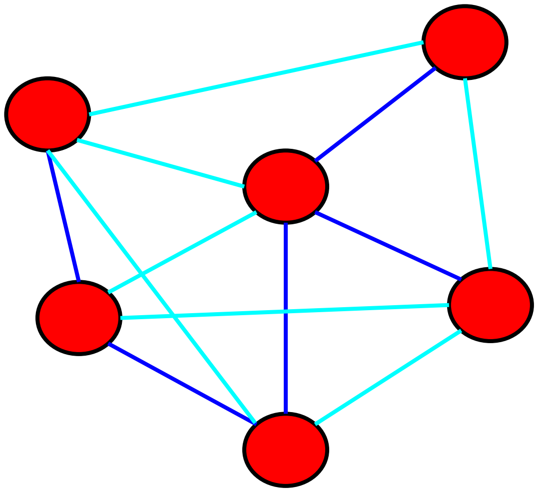
What is gossip?

What is gossip?

Broadcast strategies

What is gossip?

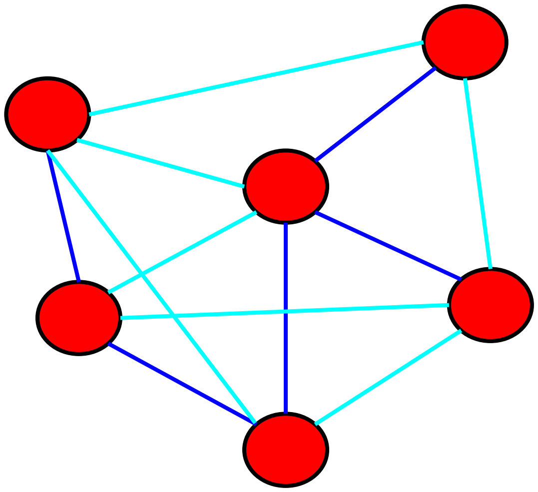
Broadcast strategies



STRUCTURED

What is gossip?

Broadcast strategies

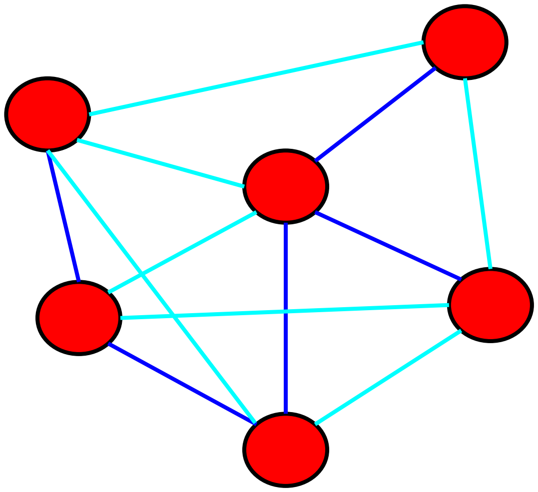


STRUCTURED

ISSUES

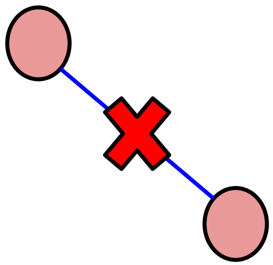
What is gossip?

Broadcast strategies



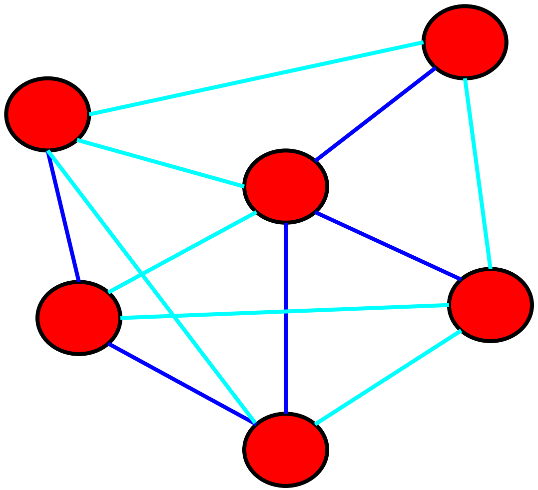
STRUCTURED

ISSUES



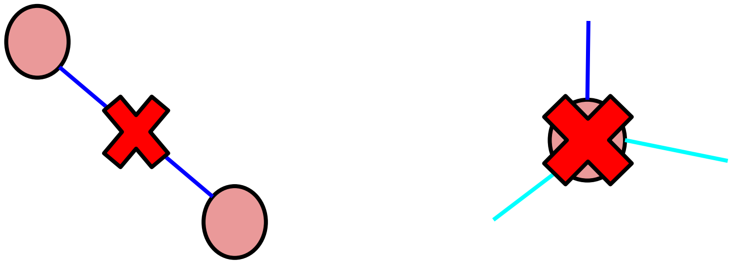
What is gossip?

Broadcast strategies



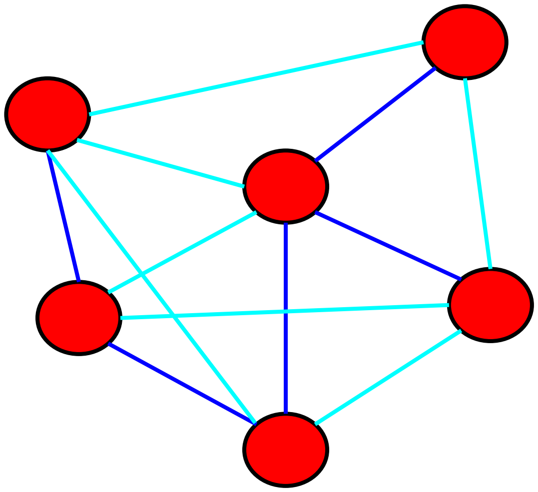
STRUCTURED

ISSUES



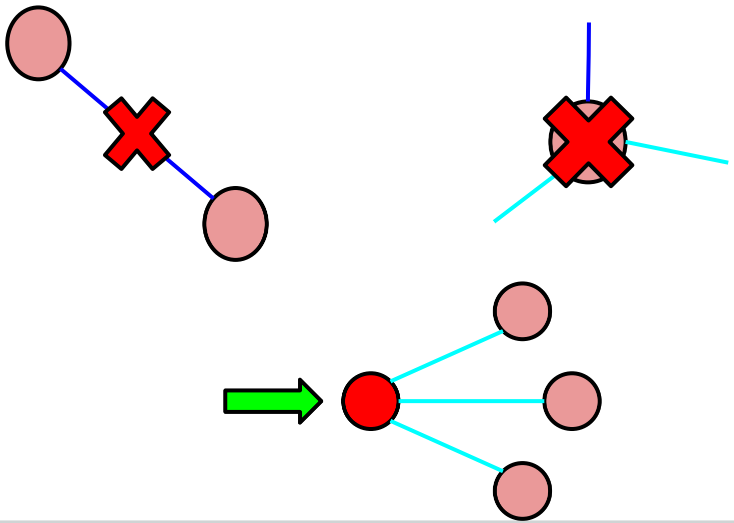
What is gossip?

Broadcast strategies



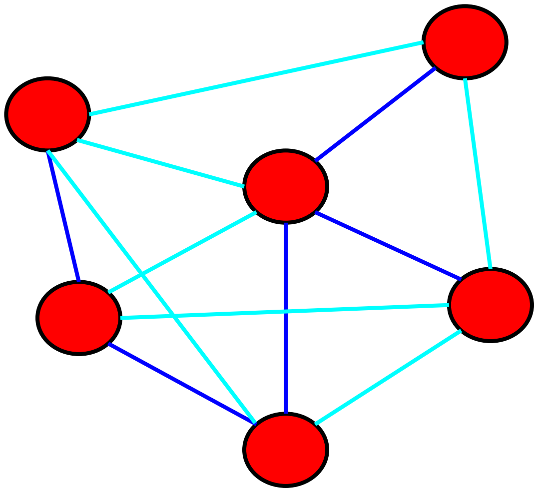
STRUCTURED

ISSUES



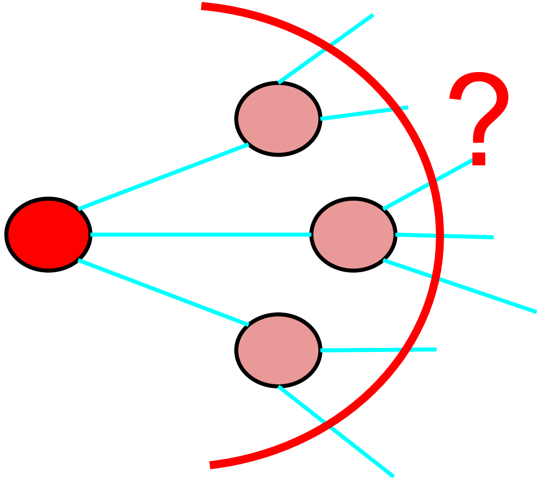
What is gossip?

Broadcast strategies



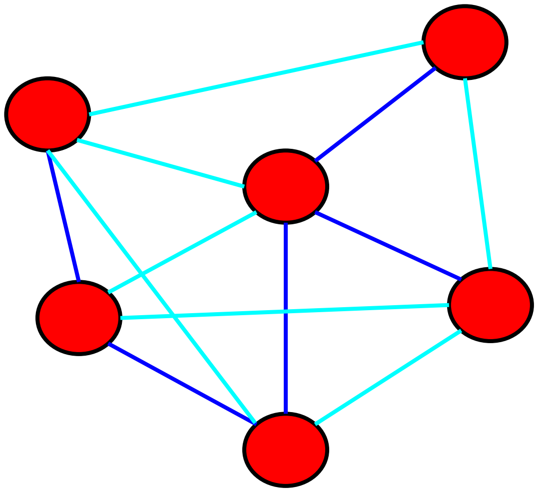
STRUCTURED

ISSUES

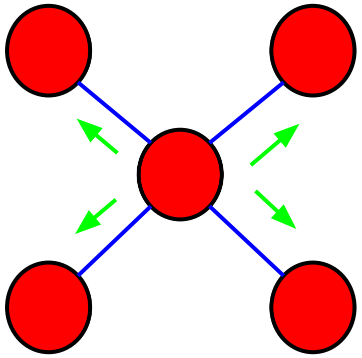


What is gossip?

Broadcast strategies



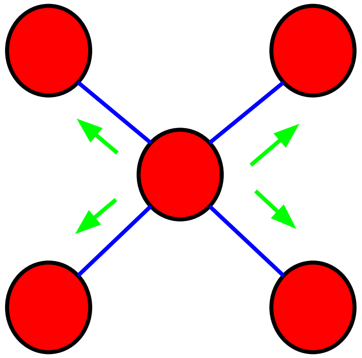
STRUCTURED



FLOODING

What is gossip?

Broadcast strategies

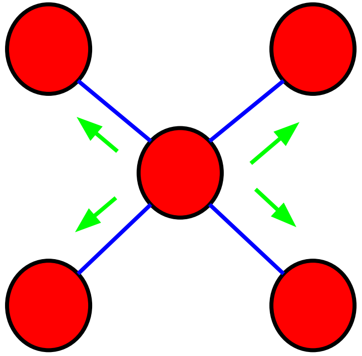


FLOODING

What is gossip?

Broadcast strategies

ISSUES

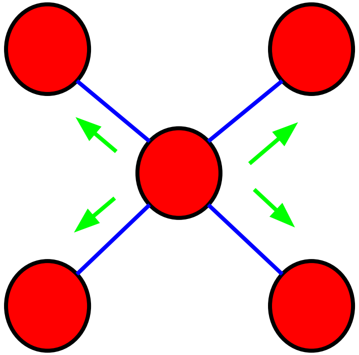


FLOODING

What is gossip?

Broadcast strategies

ISSUES

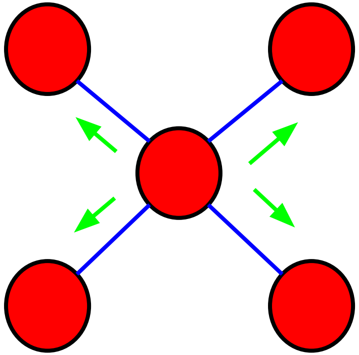


FLOODING

What is gossip?

Broadcast strategies

ISSUES

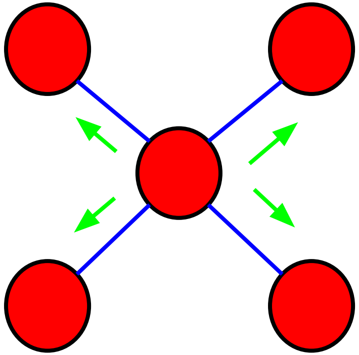


FLOODING

What is gossip?

Broadcast strategies

ISSUES



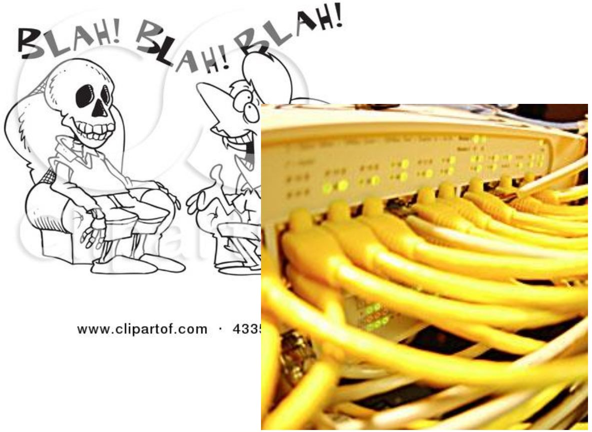
FLOODING

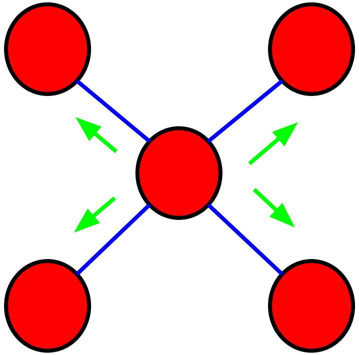
COMPLEXITY

What is gossip?

Broadcast strategies

ISSUES





FLOODING

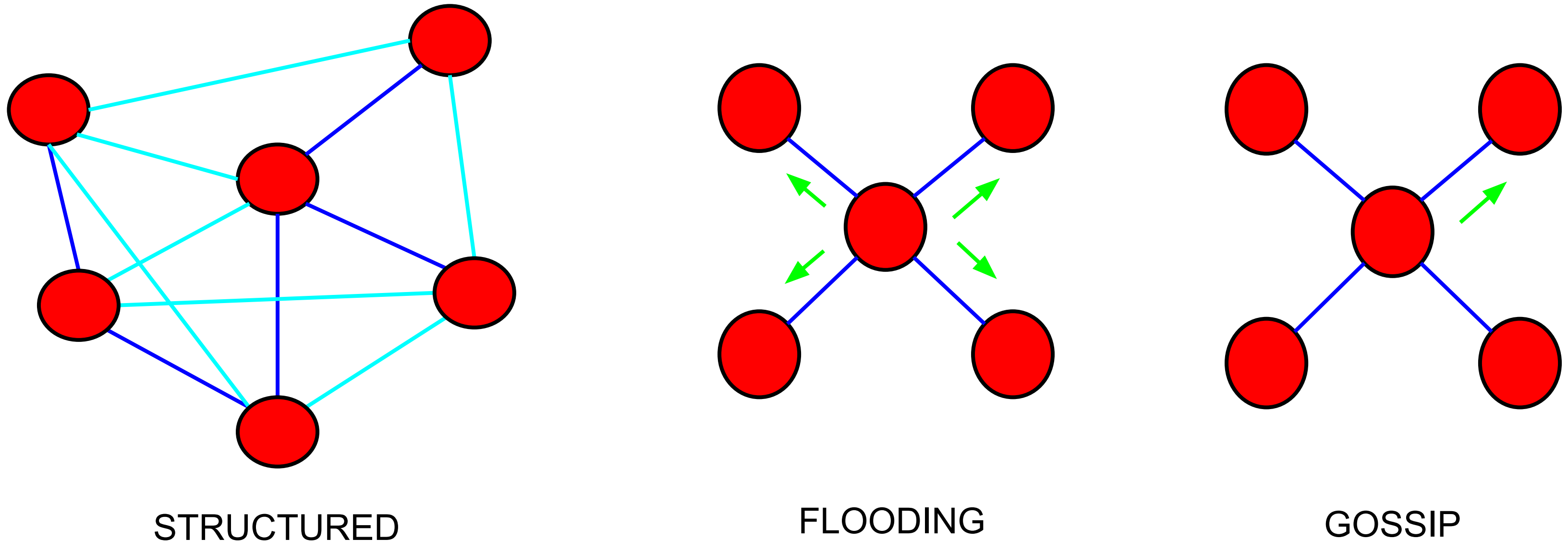
COMPLEXITY

TIME: **O(D)**

MESSAGE: **O(m)**
(or **O(D*m)**)

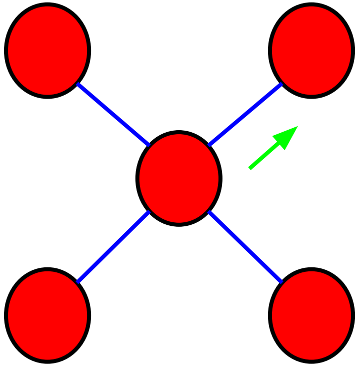
What is gossip?

Broadcast strategies



What is gossip?

Broadcast strategies

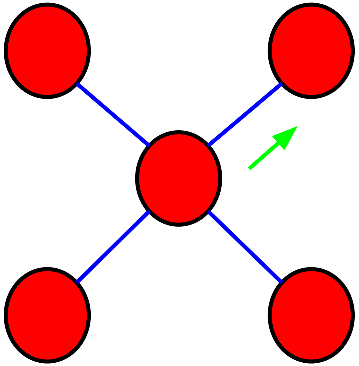


GOSSIP

What is gossip?

Broadcast strategies

- choice of active edge
- randomized vs. deterministic
- time complexity



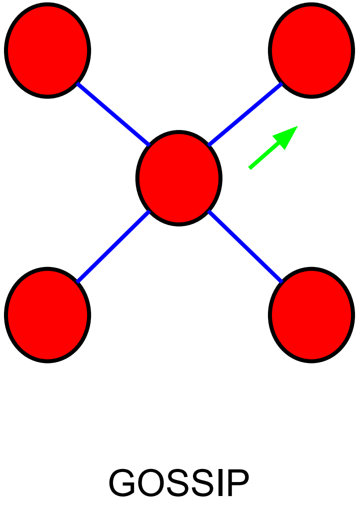
GOSSIP

What is gossip?

Broadcast strategies

- choice of active edge
- randomized vs. deterministic
- time complexity

?



Applications

Applications

DATABASE REPLICATION

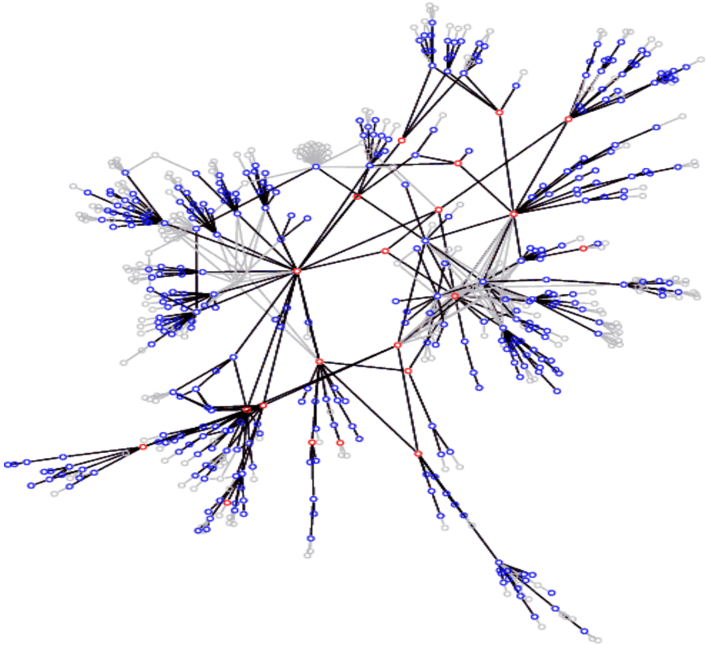
Applications

DATABASE REPLICATION

1. Direct mail
2. Anti-entropy
3. Rumor mongering

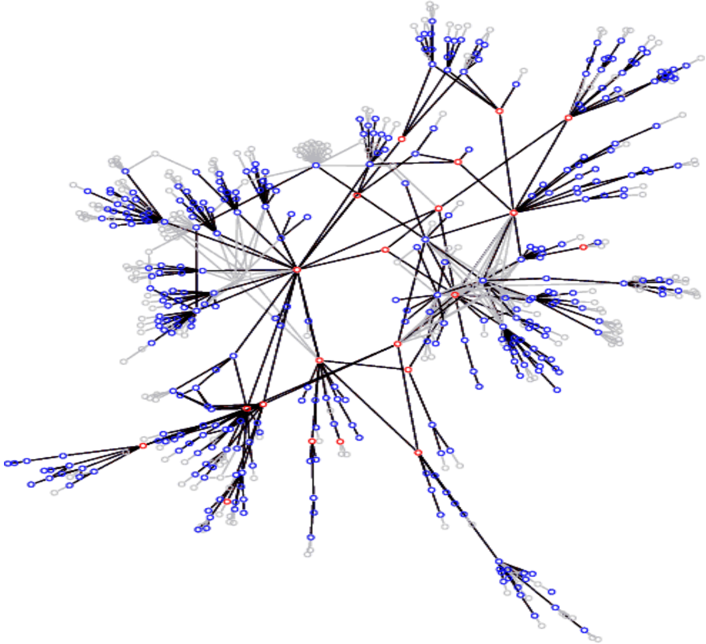
- PUSH
- PULL
- PUSH-PULL

Applications



RESOURCE DISCOVERY

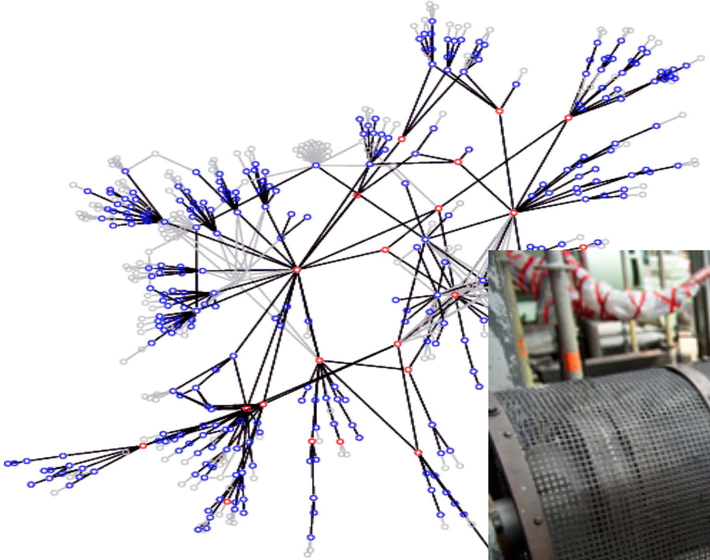
Applications



RESOURCE DISCOVERY

DISTRIBUTED COMPUTATION

Applications



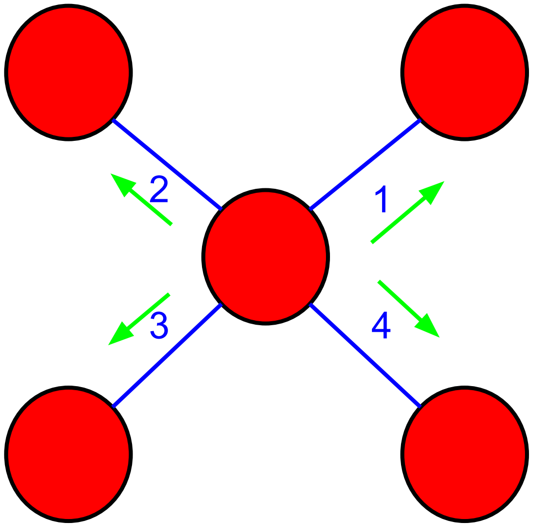
NODE
FAILURE
DETECTION

RESOURCE DISCOV

IBUTED COMPUTATION

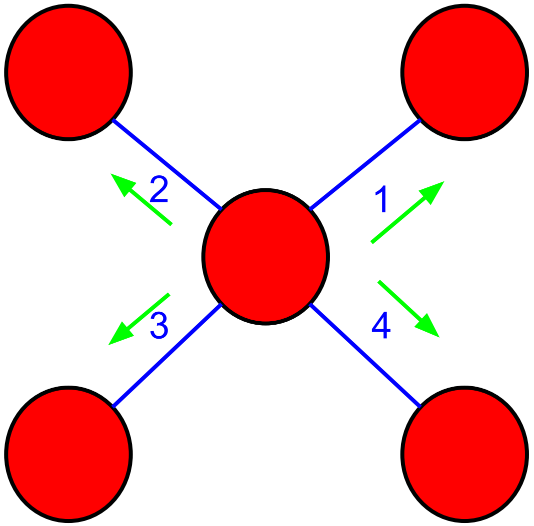
Basic algorithms

Naive solution: simulated flooding



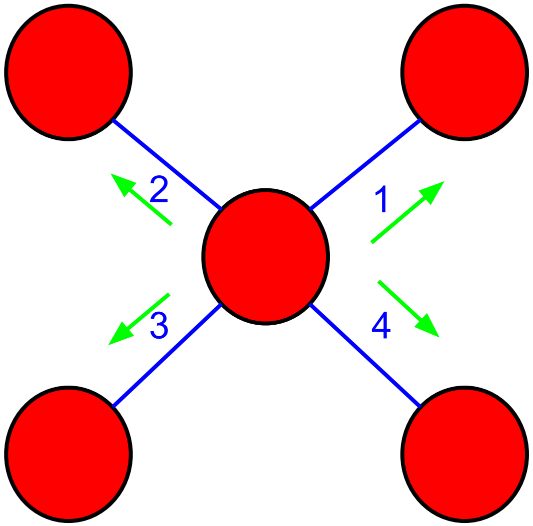

```
R[v] = rumor of v
REPEAT D times
   R′ = ∅
   FOR t = 1 to Δ
         exchange rumors in R[v] with n[v][t]
   add all received rumors to R′
   R[v] = R[v]  ∪ R′
```

Naive solution: simulated flooding



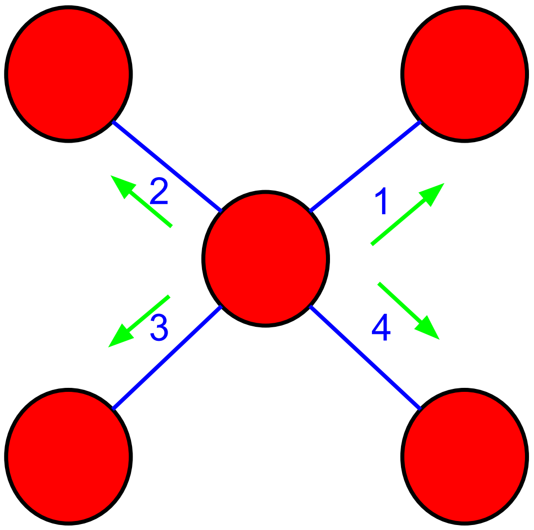

```
R[v] = rumor of v
REPEAT D times
  R′ = ∅
  FOR t = 1 to Δ
      exchange rumors in R[v] with n[v][t]
  add all received rumors to R′
  R[v] = R[v] ∪ R′
```

Naive solution: simulated flooding



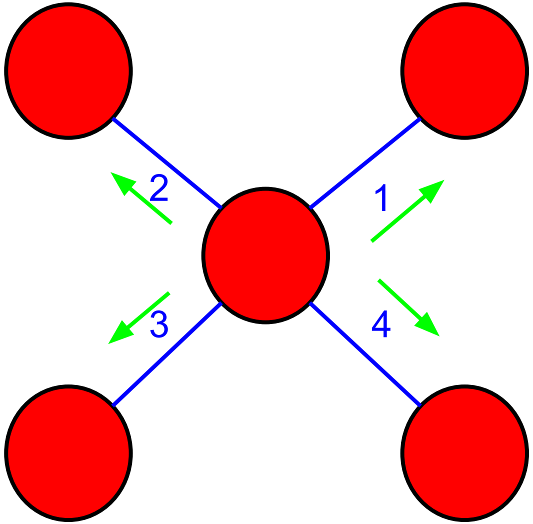

```
R[v] = rumor of v
REPEAT D times
   R′ = ∅
   FOR t = 1 to Δ
         exchange rumors in R[v] with n[v][t]
   add all received rumors to R′
   R[v] = R[v] ∪ R′
```

Naive solution: simulated flooding




```
R[v] = rumor of v
REPEAT D times
    R′ = ∅
    FOR t = 1 to Δ
        exchange rumors in R[v] with n[v][t]
    add all received rumors to R′
    R[v] = R[v] ∪ R′
```

Naive solution: simulated flooding

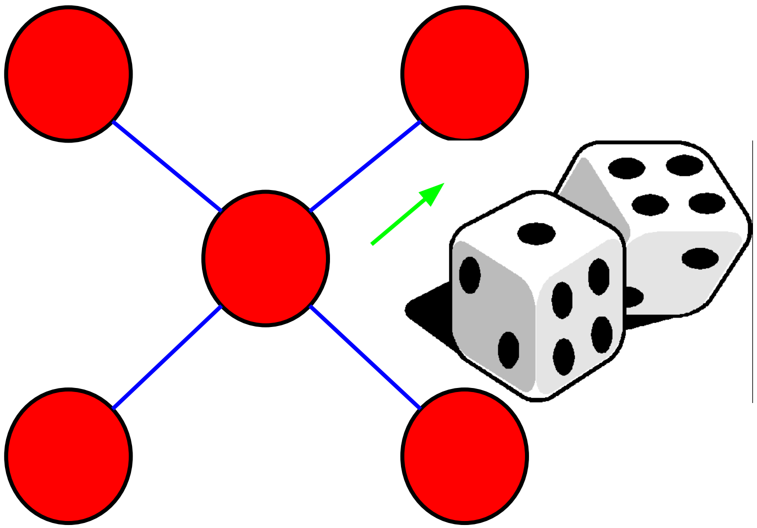


COMPLEXITY

TIME: **O(Δ*D)**

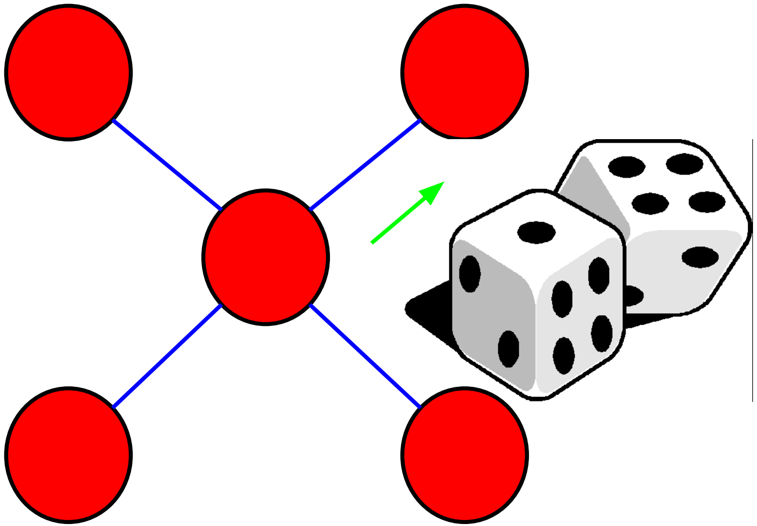
MESSAGE: **O(m*D)**

Classic solution: uniform gossip



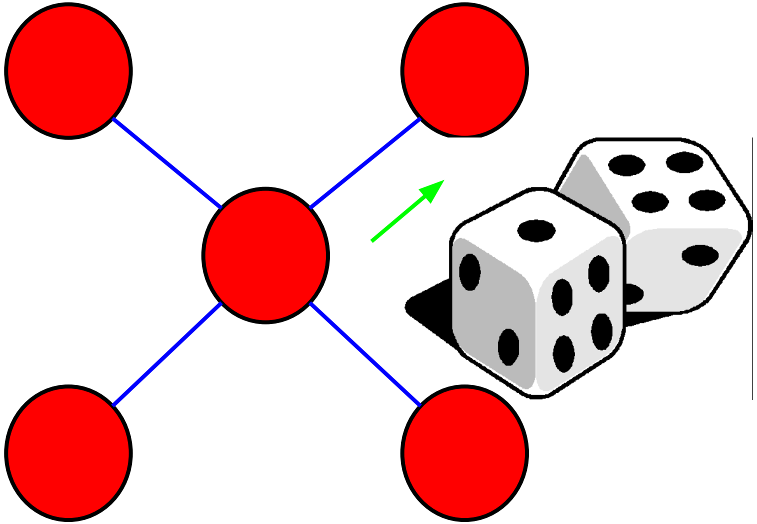
```
REPEAT ? times
  choose a uniformly random neighbor
  PUSH-PULL rumors in R[v] with n[v][t]
  add received rumors to R[v]
```

Classic solution: uniform gossip



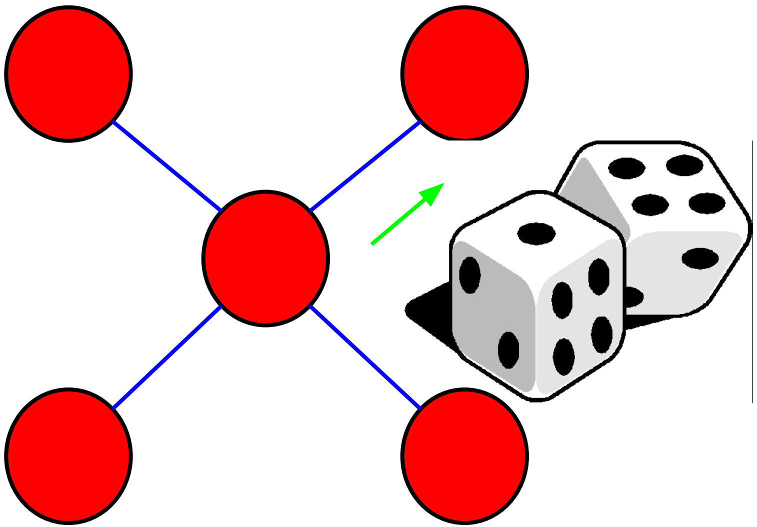
```
REPEAT ? times
   choose a uniformly random neighbor
   PUSH-PULL rumors in R[v] with n[v][t]
   add received rumors to R[v]
```

Classic solution: uniform gossip



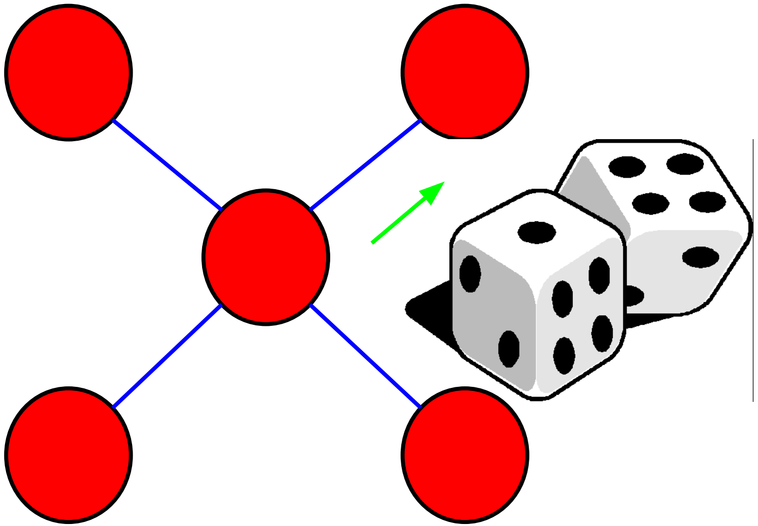
```
REPEAT ? times
  choose a uniformly random neighbor
  PUSH-PULL rumors in R[v] with n[v][t]
  add received rumors to R[v]
```

Classic solution: uniform gossip



```
REPEAT ? times
   choose a uniformly random neighbor
   PUSH-PULL rumors in R[v] with n[v][t]
   add received rumors to R[v]
```

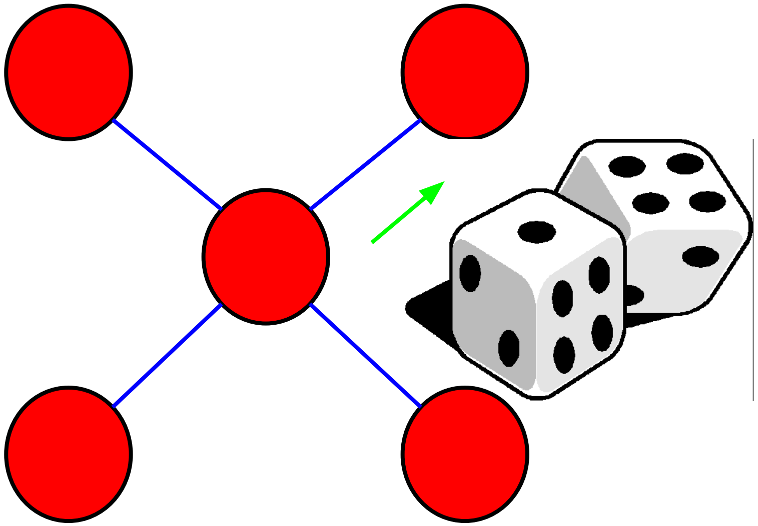
Classic solution: uniform gossip



GIAKKOUPIS ‘12

TIME: **O(log n / φ)**

Classic solution: uniform gossip



GIAKKOUPIS ‘12

TIME: **O(log n / φ)**

φ ?!

Graph conductance

$$a(S) = \sum_{i \in S} \sum_{j \in V} a_{ij}$$

VOLUME

$$\varphi(S) = \frac{\sum_{i \in S, j \in \bar{S}} a_{ij}}{min(a(S), a(\bar{S}))}$$

CUT CONDUCTANCE

$$\varphi_G = \min_{S \subseteq V} \varphi(S)$$

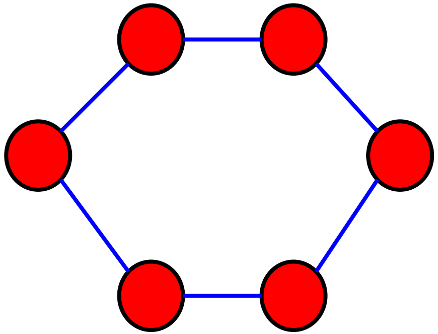
GRAPH CONDUCTANCE

Graph conductance

It measures how much
the network is bottlenecked

Graph conductance

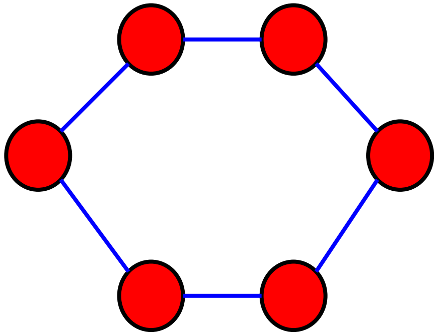
It measures how much the network is bottlenecked



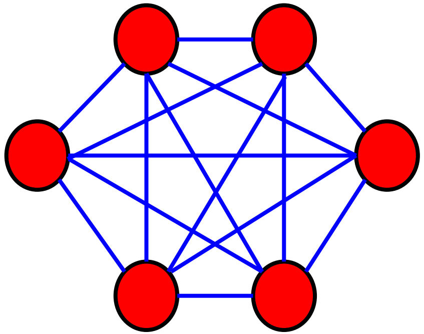
$\theta(1/n)$

Graph conductance

It measures how much the network is bottlenecked



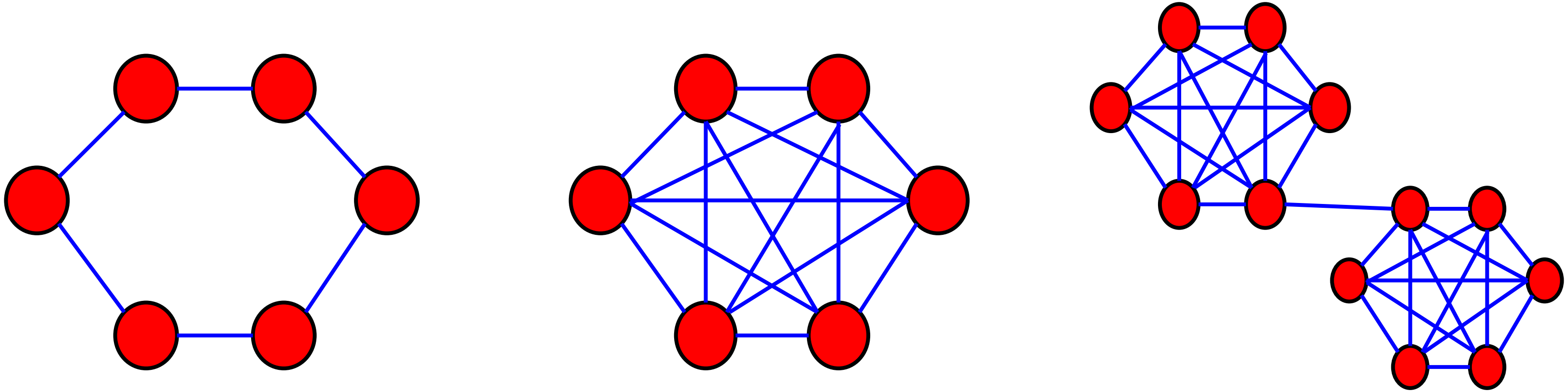
θ(1/n)



θ(1)

Graph conductance

It measures how much the network is bottlenecked

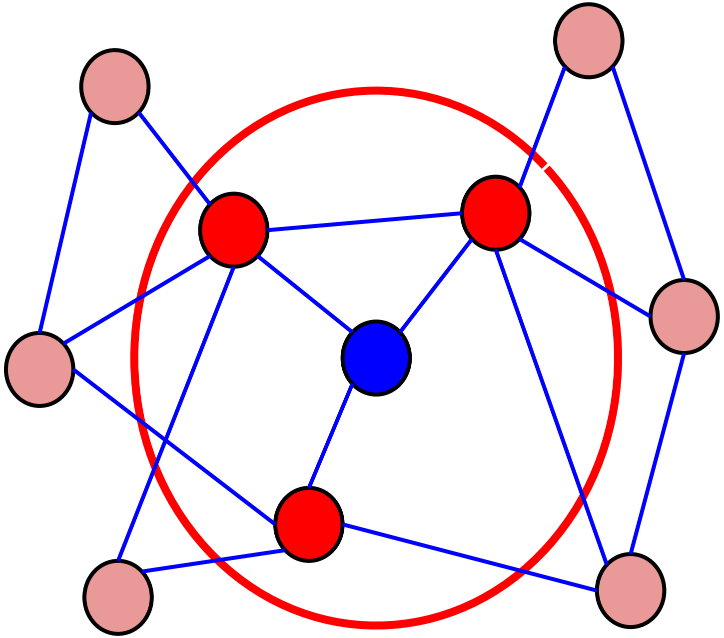


θ(1/n) θ(1) θ(1/n^2)

Advanced algorithms

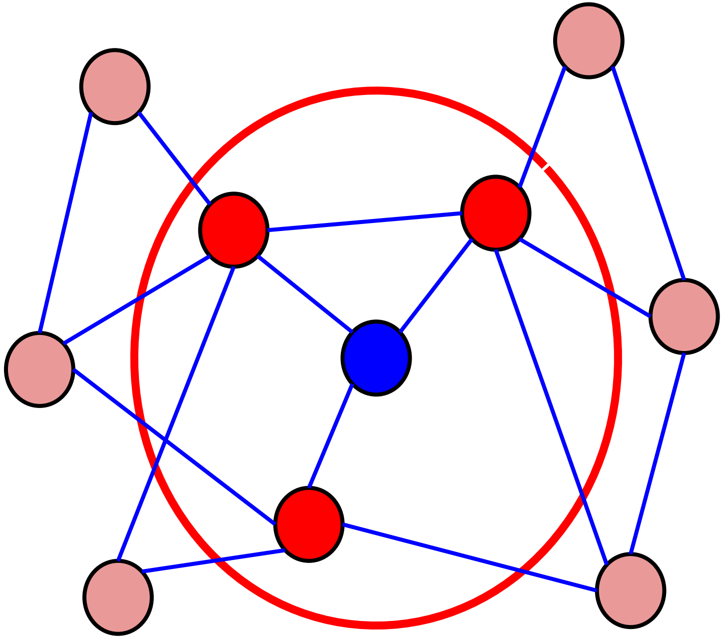
Conductance Independent results

NEIGHBOR EXCHANGE PROBLEM



Conductance Independent results

NEIGHBOR EXCHANGE PROBLEM



COMMON IDEA

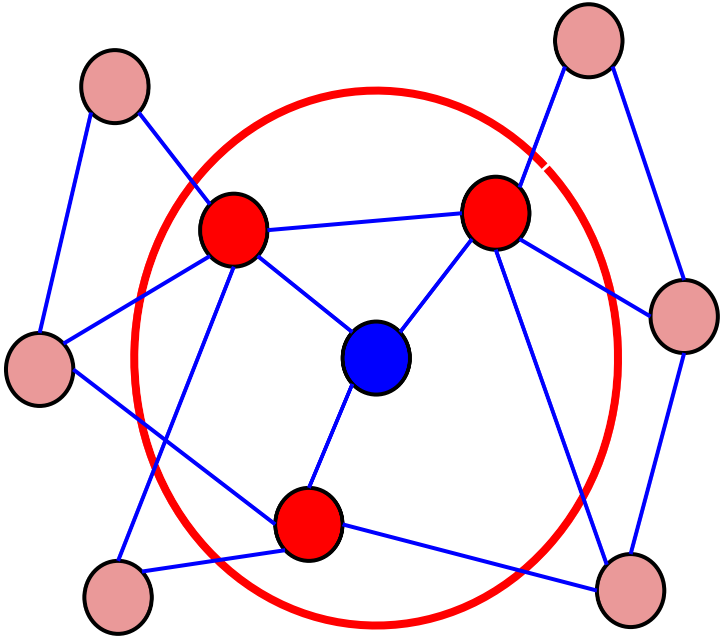
Solve NEP

+

compose it D times

Conductance Independent results

NEIGHBOR EXCHANGE PROBLEM



COMMON IDEA

Solve NEP
+
compose it D times

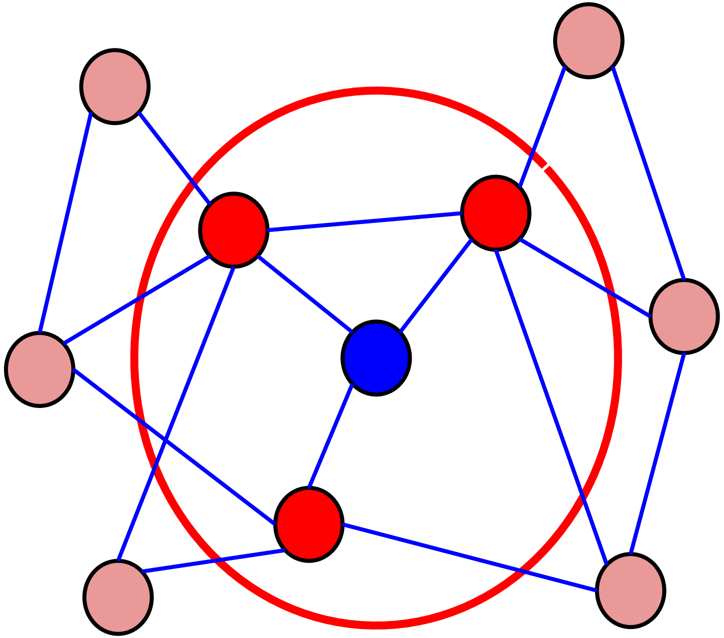
RESULTS (global)

RANDOMIZED O(D*log^3 n)

DETERMINISTIC O(D*log n + log^2 n)

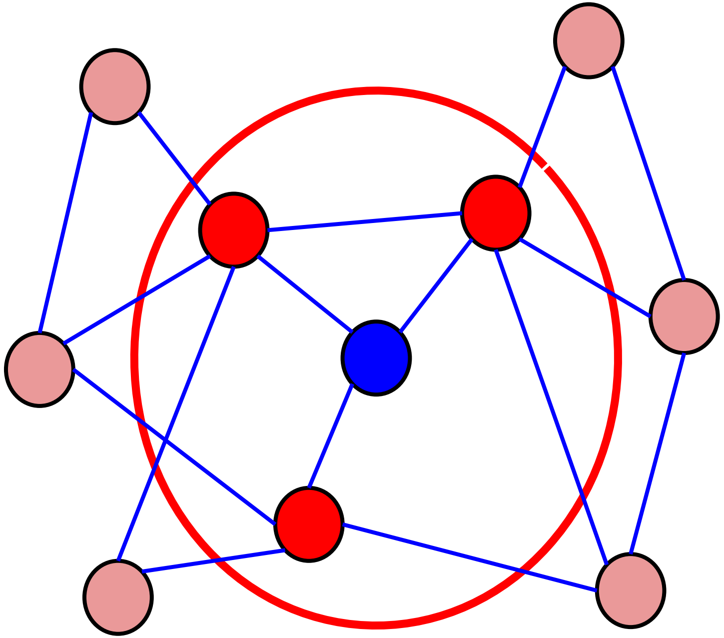
NEP: randomized (1)

NEIGHBOR EXCHANGE PROBLEM



NEP: randomized (1)

NEIGHBOR EXCHANGE PROBLEM



QUALITATIVE IDEA

Run uniform gossip for a while…

+

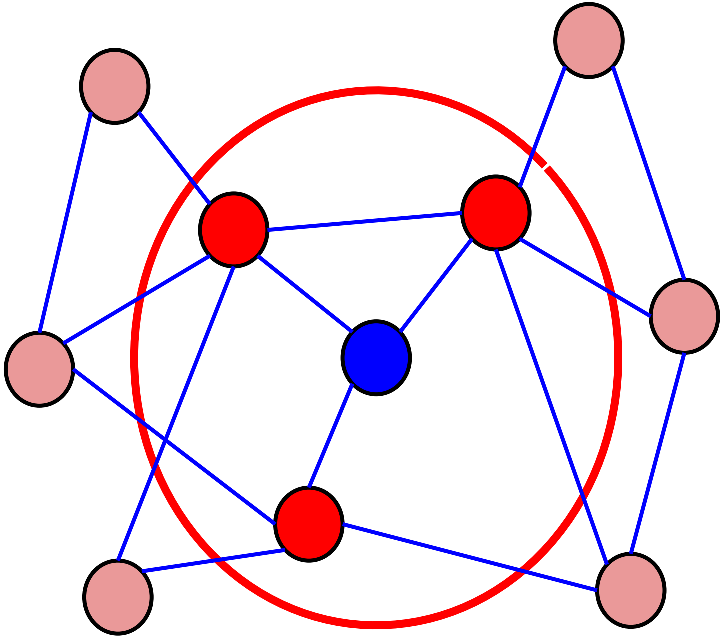
… remove some edges …

+

do it again

NEP: randomized (1)

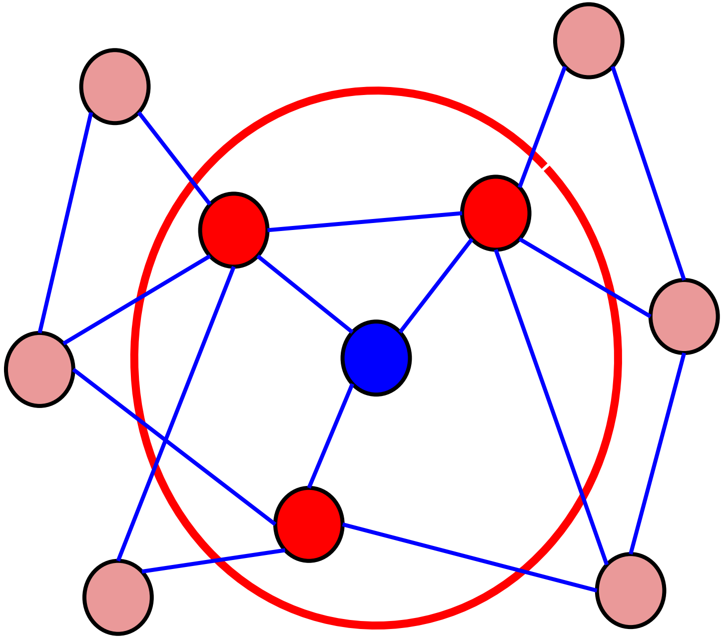
NEIGHBOR EXCHANGE PROBLEM



```
Superstep(G, τ):
1. UniformGossip algorithm with respect to F
[i] for τ rounds. K[i]: order of the random
activated edges
2. UniformGossip Krev[i], the reverse
process of the one realized in Step 1
3. (Pruning) Set of pruned directed edges P
[i] = (u, w) : u received from v
4. Set F[i+1] := F[i] − P[i] and i := i + 1
```

NEP: randomized (1)

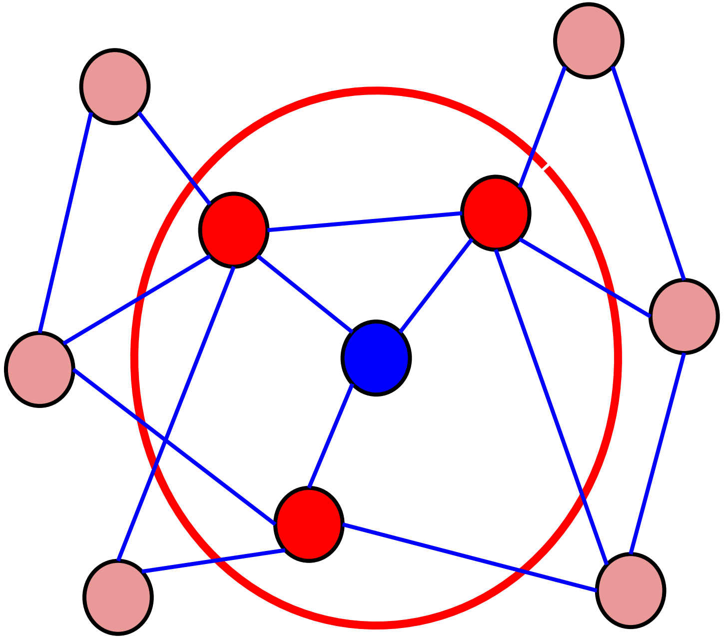
NEIGHBOR EXCHANGE PROBLEM



Superstep(G, τ):
1. UniformGossip algorithm with respect to F[i] for τ rounds. K[i]: order of the random activated edges
2. UniformGossip Krev[i], the reverse process of the one realized in Step 1
3. (Pruning) Set of pruned directed edges P[i] = (u, w) : u received from v
4. Set F[i+1] := F[i] – P[i] and i := i + 1

NEP: randomized (1)

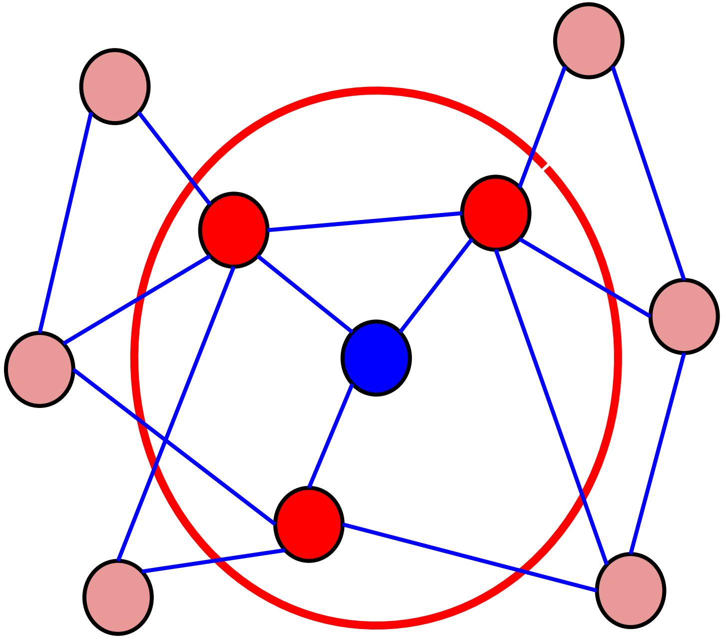
NEIGHBOR EXCHANGE PROBLEM



Superstep(G, τ):
1. UniformGossip algorithm with respect to F[i] for τ rounds. K[i]: order of the random activated edges
2. UniformGossip Krev[i], the reverse process of the one realized in Step 1
3. (Pruning) Set of pruned directed edges P[i] = (u, w) : u received from v
4. Set F[i+1] := F[i] − P[i] and i := i + 1

NEP: randomized (1)

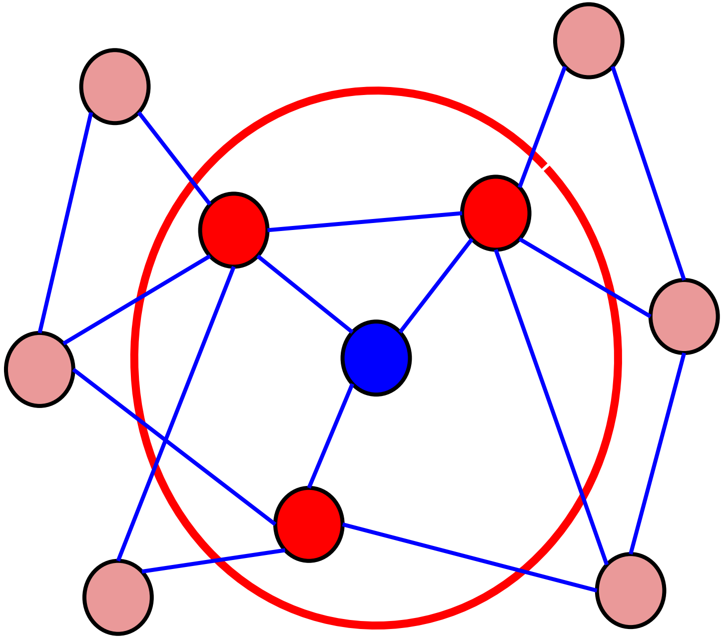
NEIGHBOR EXCHANGE PROBLEM



```
Superstep(G, τ):
1. UniformGossip algorithm with respect to F
[i] for τ rounds. K[i]: order of the random
activated edges
2. UniformGossip Krev[i], the reverse
process of the one realized in Step 1
3. (Pruning) Set of pruned directed edges P
[i] = (u, w) : u received from v
4. Set F[i+1] := F[i] − P[i] and i := i + 1
```

NEP: randomized (1)

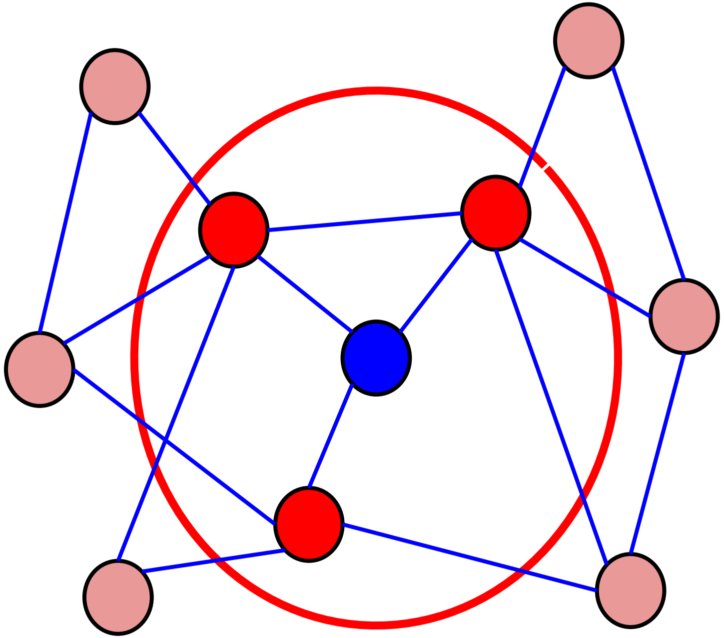
NEIGHBOR EXCHANGE PROBLEM



Superstep(G, τ):
1. UniformGossip algorithm with respect to F[i] for τ rounds. K[i]: order of the random activated edges
2. UniformGossip Krev[i], the reverse process of the one realized in Step 1
3. (Pruning) Set of pruned directed edges P[i] = (u, w) : u received from v
4. Set F[i+1] := F[i] − P[i] and i := i + 1

NEP: randomized (1)

NEIGHBOR EXCHANGE PROBLEM

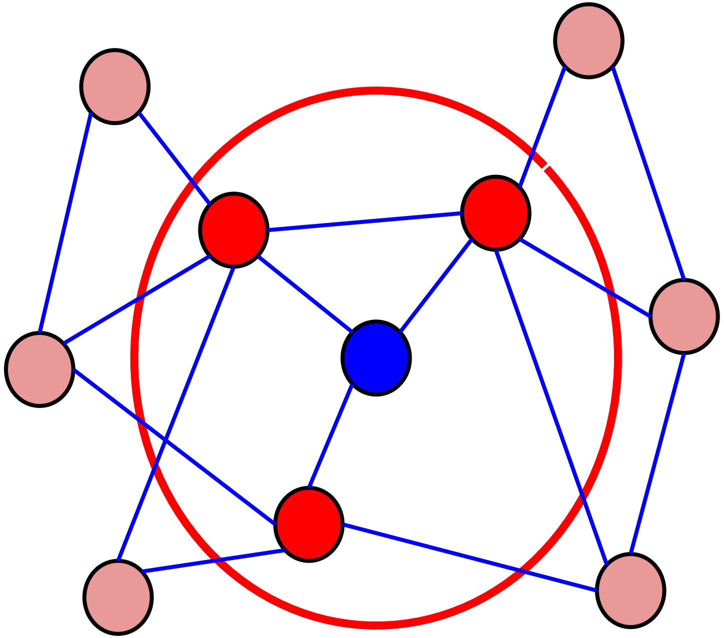


RESULT

If τ = θ(log^2 n),
we need only θ(log n) cycles

NEP: randomized (1)

NEIGHBOR EXCHANGE PROBLEM



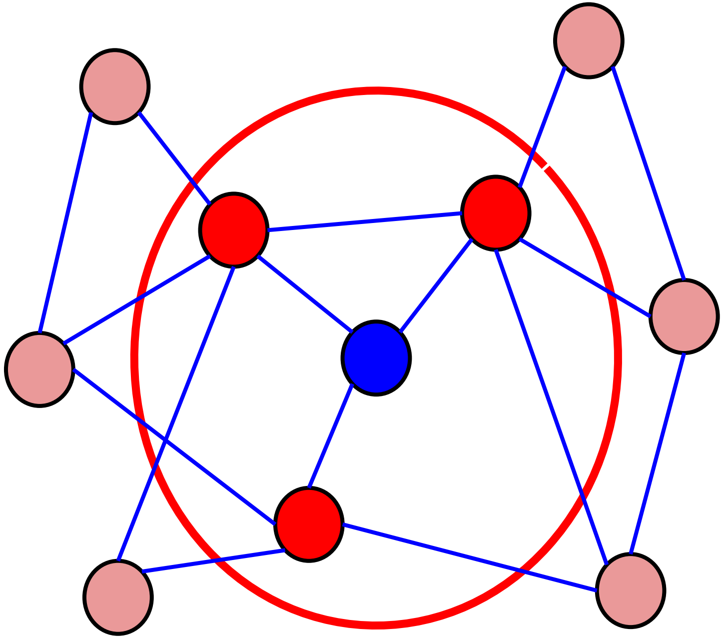
RESULT

If τ = θ(log^2 n),
we need only θ(log n) cycles

NEP: θ(log^3 n)

NEP: randomized (1)

NEIGHBOR EXCHANGE PROBLEM



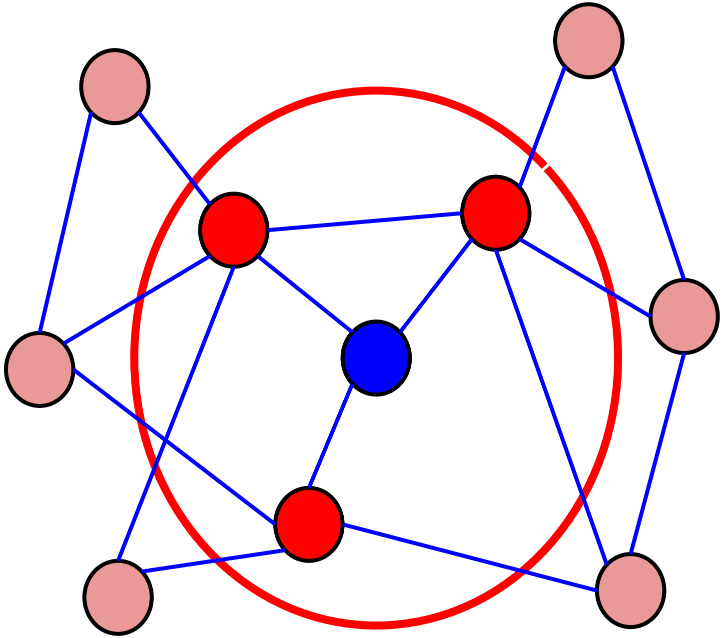
RESULT

If $\tau = \theta(\log^2 n)$,
we need only $\theta(\log n)$ cycles

PROOF

NEP: randomized (1)

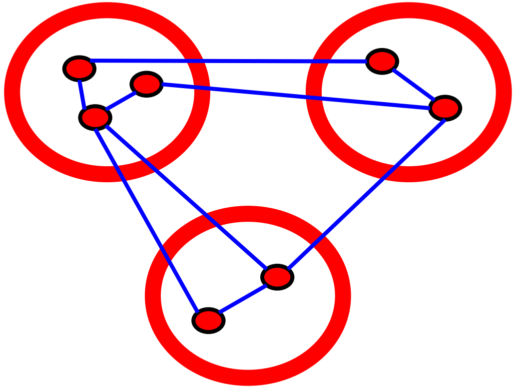
NEIGHBOR EXCHANGE PROBLEM



RESULT

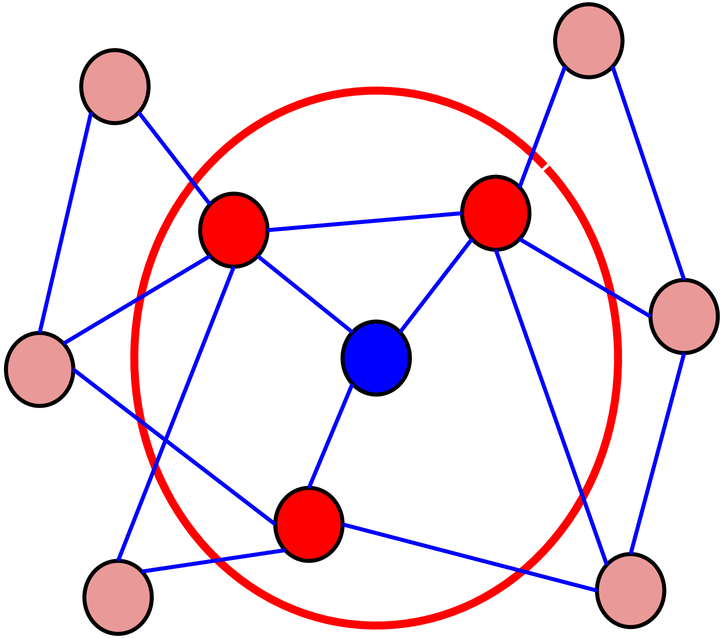
If τ = θ(log^2 n),
we need only θ(log n) cycles

PROOF



NEP: randomized (1)

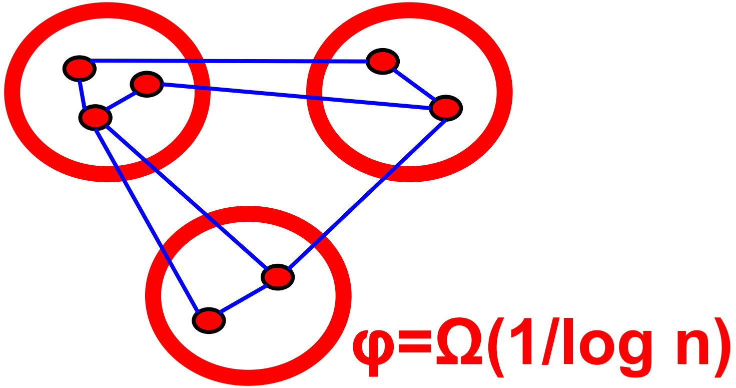
NEIGHBOR EXCHANGE PROBLEM



RESULT

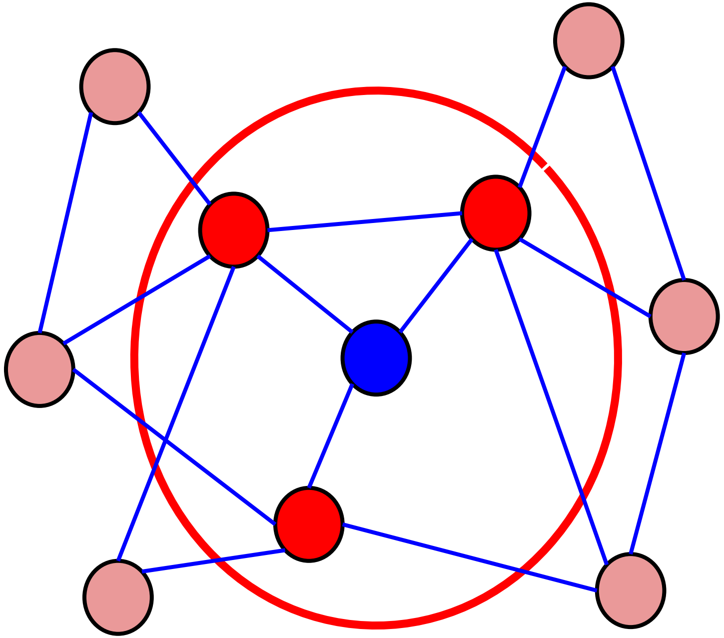
If $\tau = \theta(\log^2 n)$,
we need only $\theta(\log n)$ cycles

PROOF



NEP: randomized (1)

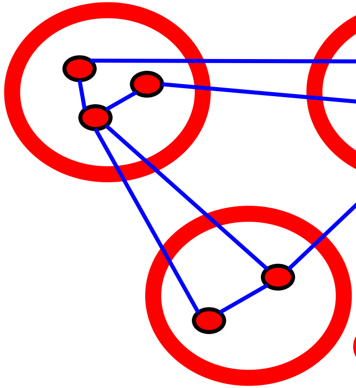
NEIGHBOR EXCHANGE PROBLEM



RESULT

If τ = θ(log^2 n),
we need only θ(log n) cycles

PROOF

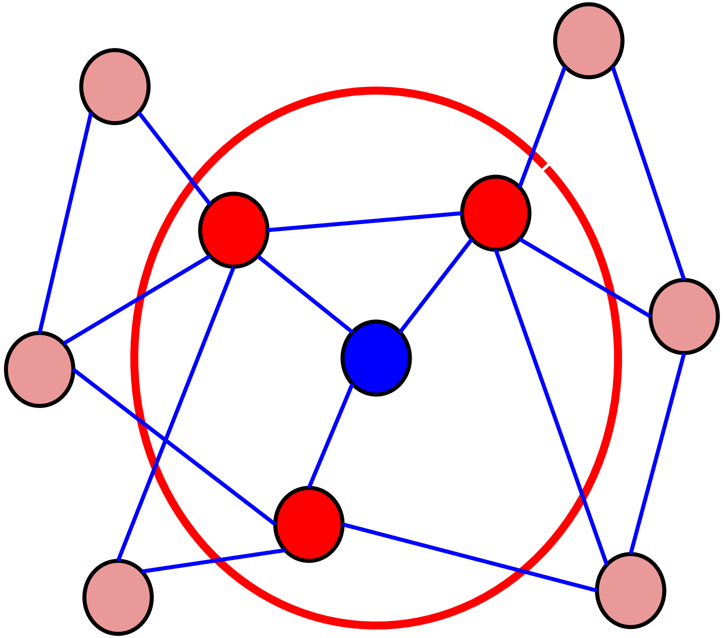


$F \leftarrow F/2$

$\varphi=\Omega(1/\log n)$

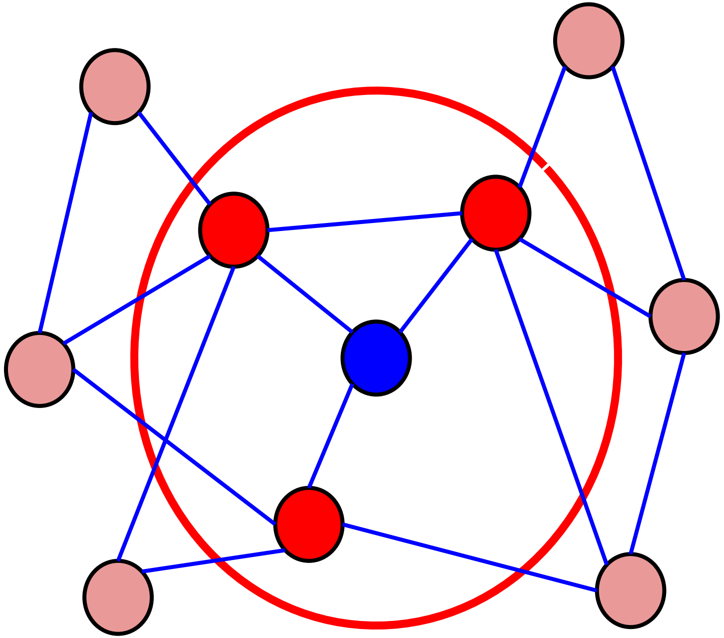
NEP: randomized (2)

NEIGHBOR EXCHANGE PROBLEM



NEP: randomized (2)

NEIGHBOR EXCHANGE PROBLEM



QUALITATIVE IDEA

Run simulated flooding for a while…

\+

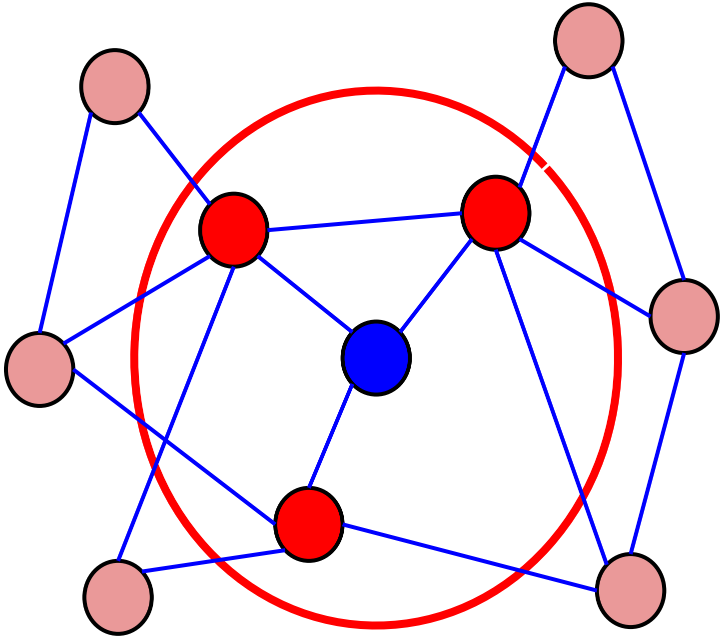
… add some edges …

\+

do it again

NEP: randomized (2)

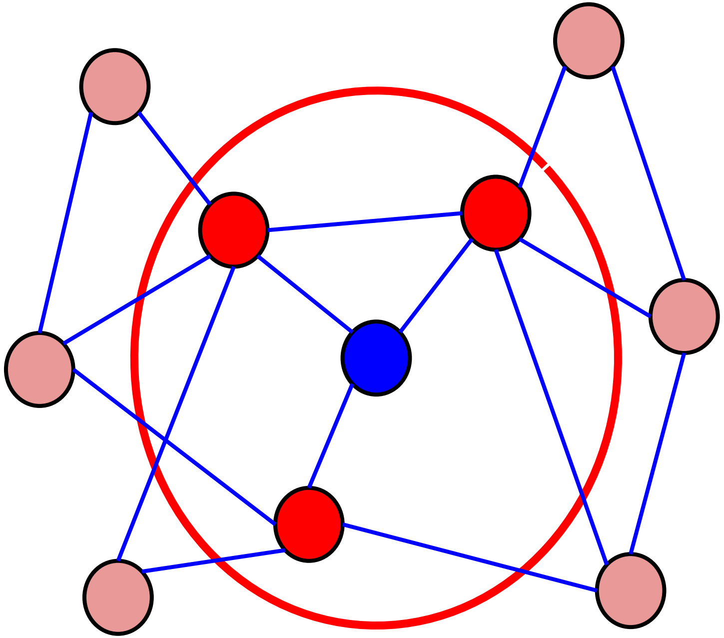
NEIGHBOR EXCHANGE PROBLEM



```
R[v] = v
WHILE Γ[v]\R[v] = ∅
  pick Θ(log^2 n) random edges in Γ[v]\R[v]
  d = Θ(log^2 n);
  E′ = all newly picked edges
  Flood in R[v] along E′-edges for d-hops
  add all received rumors to R[v]
```

NEP: randomized (2)

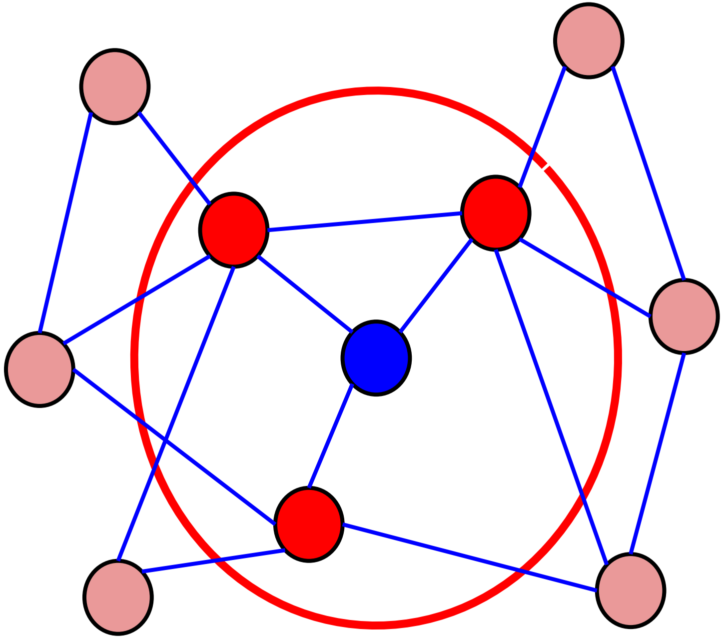
NEIGHBOR EXCHANGE PROBLEM



```
R[v] = v
WHILE Γ[v]\R[v] = ∅
  pick Θ(log^2 n) random edges in Γ[v]\R[v]
  d = Θ(log^2 n);
  E’ = all newly picked edges
  Flood in R[v] along E’-edges for d-hops
  add all received rumors to R[v]
```

NEP: randomized (2)

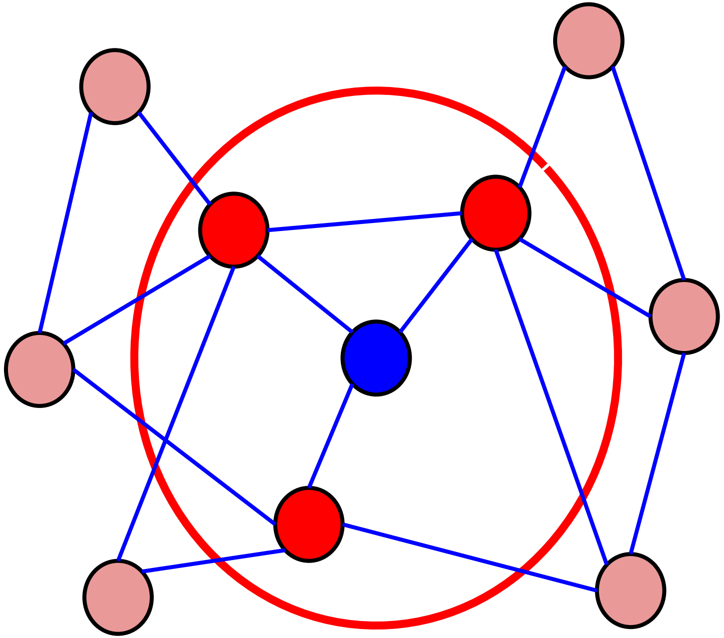
NEIGHBOR EXCHANGE PROBLEM



```
R[v] = v
WHILE Γ[v]\R[v] = ∅
   pick Θ(log^2 n) random edges in Γ[v]\R[v]
   d = Θ(log^2 n);
   E’ = all newly picked edges
   Flood in R[v] along E’-edges for d-hops
   add all received rumors to R[v]
```

NEP: randomized (2)

NEIGHBOR EXCHANGE PROBLEM

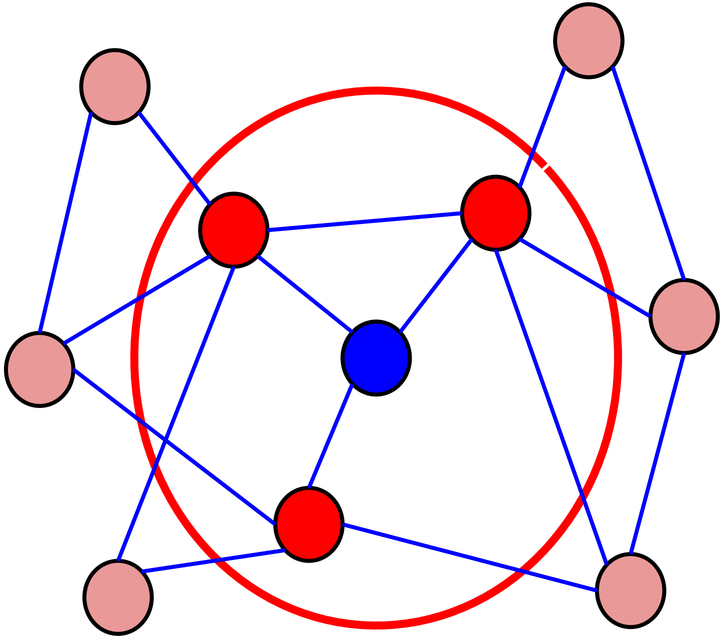


```
R[v] = v
WHILE Γ[v]\R[v] = ∅
  pick Θ(log^2 n) random edges in Γ[v]\R[v]
  d = Θ(log^2 n);
  E’ = all newly picked edges
  Flood in R[v] along E’-edges for d-hops
  add all received rumors to R[v]
```

O(log^6 n)

NEP: deterministic

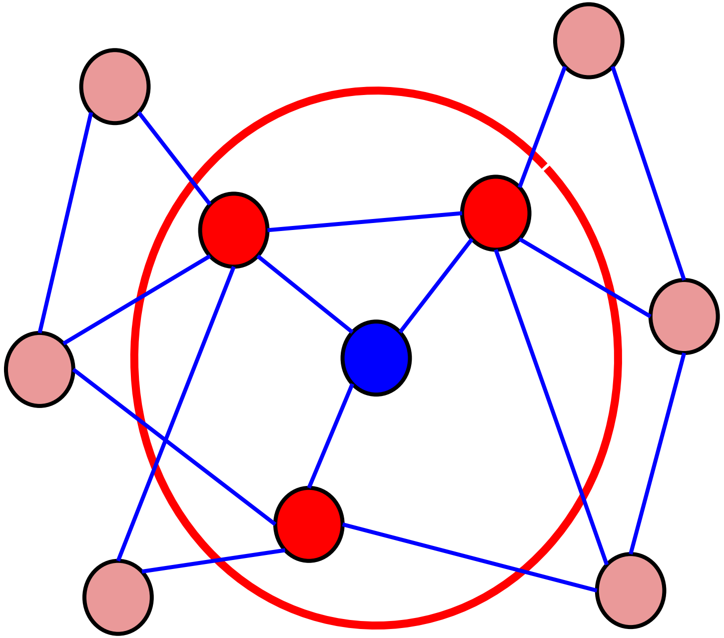
NEIGHBOR EXCHANGE PROBLEM



```
R[v] = v
WHILE Γ[v]\R[v] = ∅
  pick Θ(log^2 n) random edges in Γ[v]\R[v]
  d = Θ(log^2 n);
  E′ = all newly picked edges
  Flood in R[v] along E′-edges for d-hops
  add all received rumors to R[v]
```

NEP: deterministic

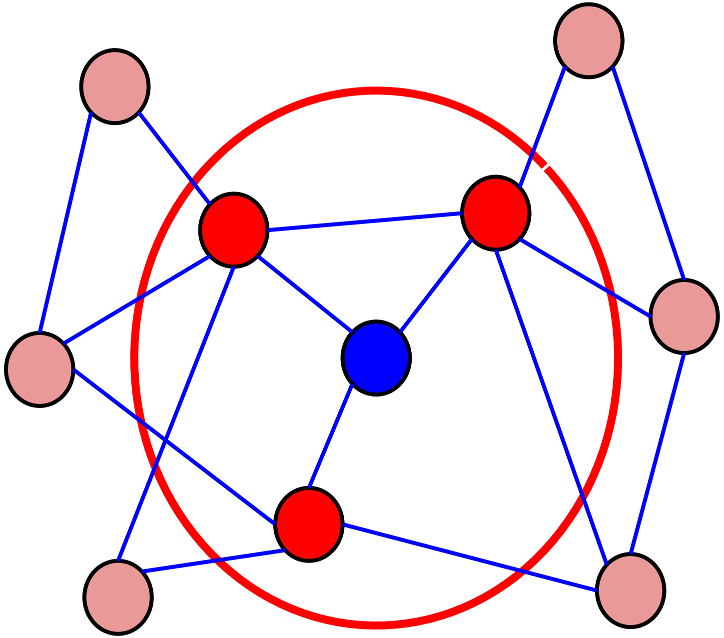
NEIGHBOR EXCHANGE PROBLEM



```
R[v] = v
WHILE Γ[v]\R[v] = ∅
   arbitrarily pick one edge in Γ[v]\R[v]
   d = 2log n;
   E’ = all newly picked edges
   Flood in R[v] along E’-edges for d-hops
   add all received rumors to R[v]
```

NEP: deterministic

NEIGHBOR EXCHANGE PROBLEM

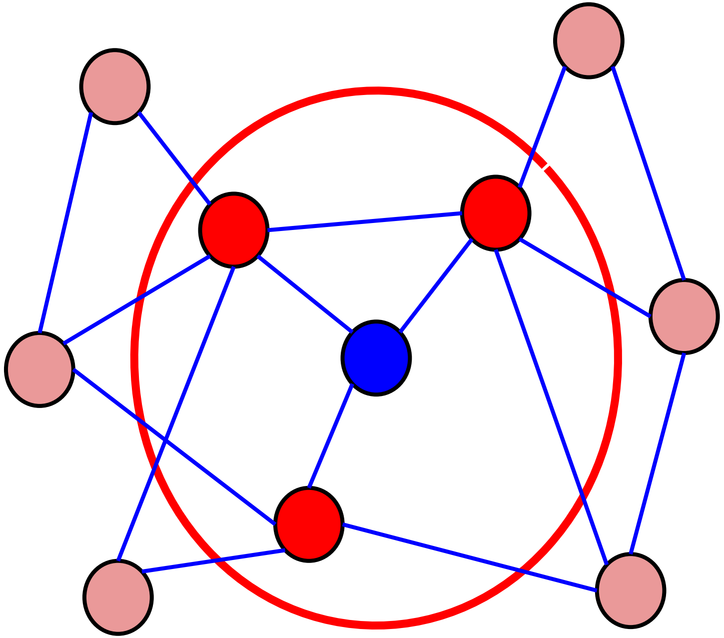


RESULT

We need only log n cycles

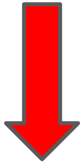
NEP: deterministic

NEIGHBOR EXCHANGE PROBLEM



RESULT

We need only log n cycles



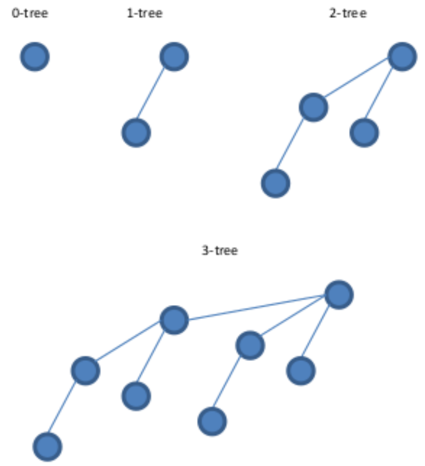
NEP: 2log^3 n

NEP: deterministic

RESULT

We need only log n cycles

PROOF



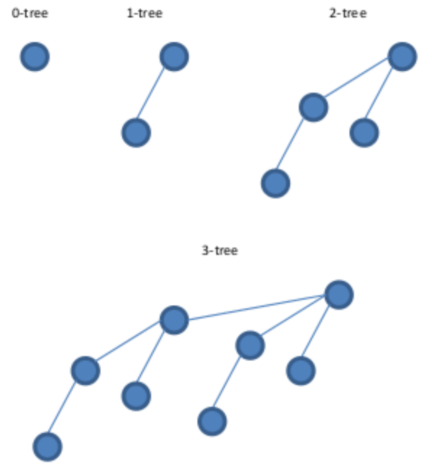
- in cycle *i*, vertex v creates a binomial i-tree and floods for *2i* hops

NEP: deterministic

RESULT

We need only log n cycles

PROOF



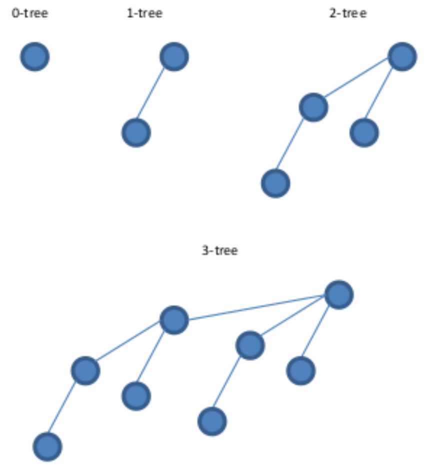
- in cycle *i*, vertex v creates a binomial i-tree and floods for *2i* hops
- in each cycle (at least) all the nodes in the binomial tree get the information from the root

NEP: deterministic

RESULT

We need only log n cycles

PROOF



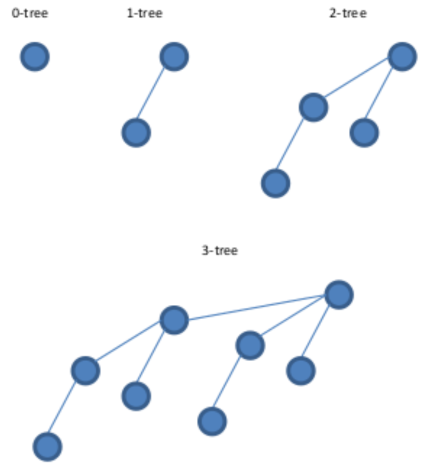
- in cycle *i*, vertex v creates a binomial i-tree and floods for *2i* hops
- in each cycle (at least) all the nodes in the binomial tree get the information from the root
- if 2 neighbours are strangers, their current trees must be disjoint

NEP: deterministic

RESULT

We need only log n cycles

PROOF



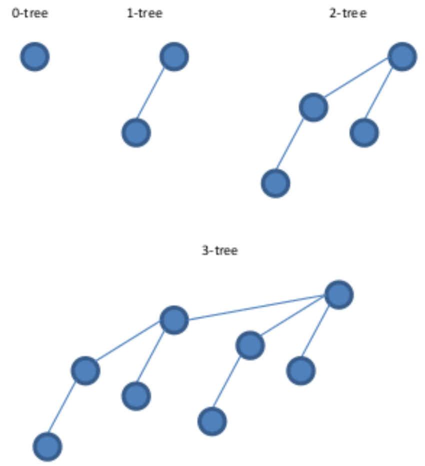
- in cycle *i*, vertex v creates a binomial i-tree and floods for *2i* hops
- in each cycle (at least) all the nodes in the binomial tree get the information from the root
- if 2 neighbours are strangers, their current trees must be disjoint
- a tree at step i contains 2^i nodes

NEP: deterministic

RESULT

We need only log n cycles

PROOF



- in cycle *i*, vertex v creates a binomial i-tree and floods for *2i* hops
- in each cycle (at least) all the nodes in the binomial tree get the information from the root
- if 2 neighbours are strangers, their current trees must be disjoint
- a tree at step i contains 2^i nodes

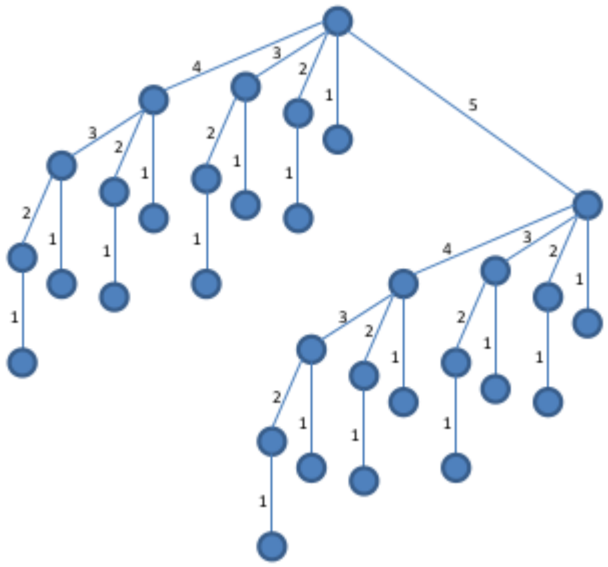
→ **at most log n cycles**

NEP: faster deterministic

Can we exploit the structure of the binomial tree?

NEP: faster deterministic

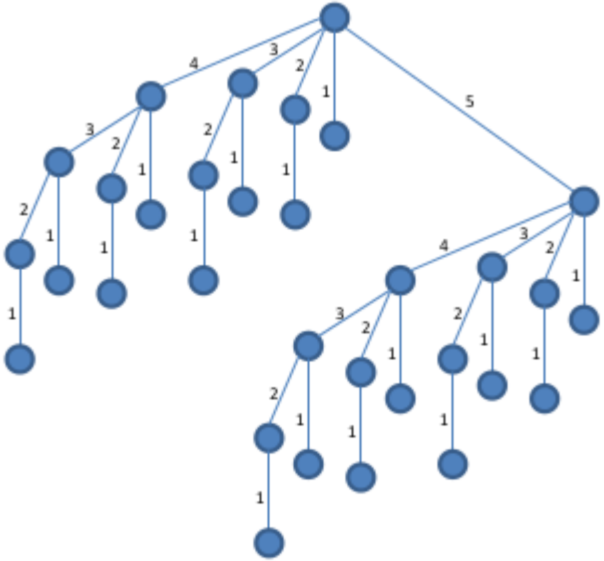
Can we exploit the structure of the binomial tree?



PUSH in inverse order
+
PULL in order
+
symmetric PULL & PUSH

NEP: faster deterministic

Can we exploit the structure of the binomial tree?

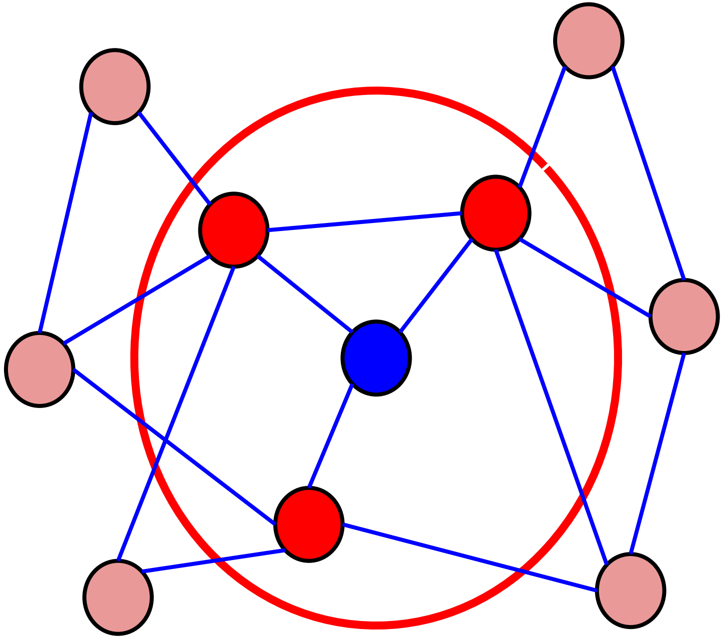


PUSH in inverse order
+
PULL in order
+
symmetric PULL & PUSH

NEP: 2log n(log n + 1)

NEP composition

NEIGHBOR EXCHANGE PROBLEM

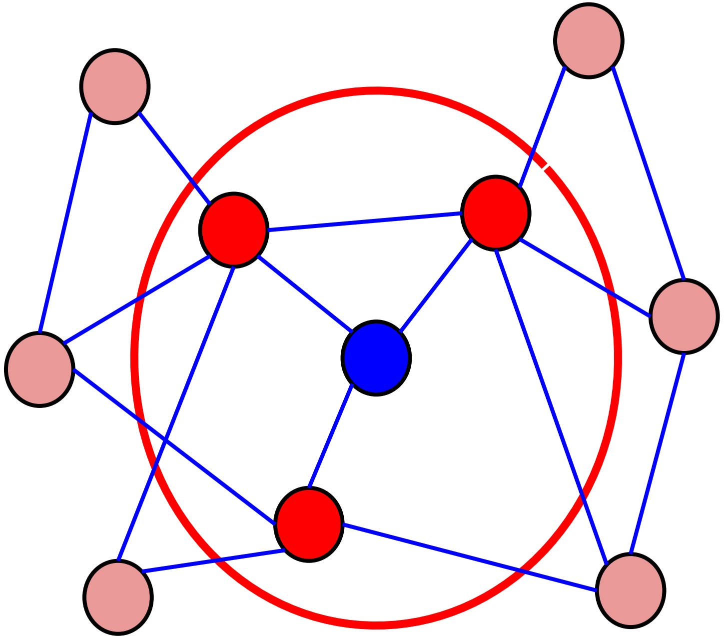


NAIVE COMPOSITION

O(D*log^2 n)

NEP composition

NEIGHBOR EXCHANGE PROBLEM



NAIVE COMPOSITION

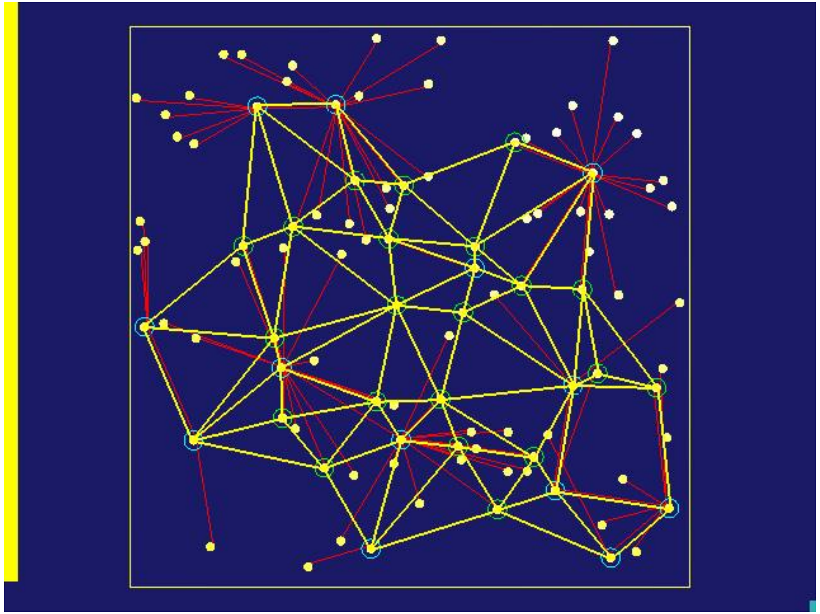
O(D*log^2 n)

REUSING TREE

O(D*log n + log^2 n)

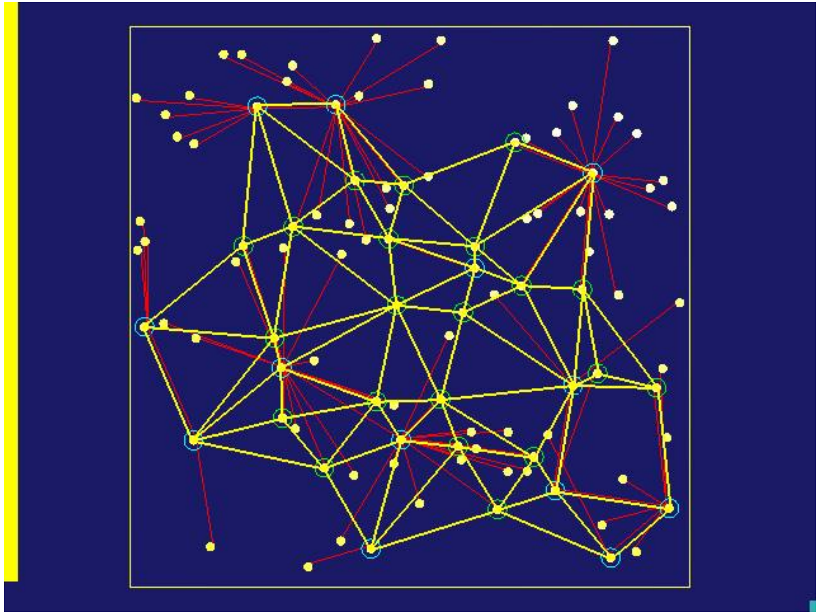
Other results
&
Current research

Other results & Current research



SPANNERS & HEREDITARY DENSITY

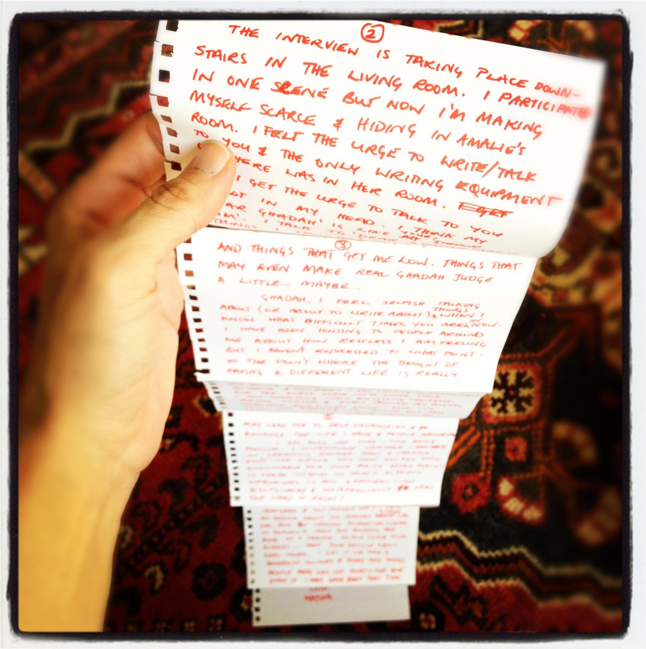
Other results & Current research



SPANNERS & HEREDITARY DENSITY

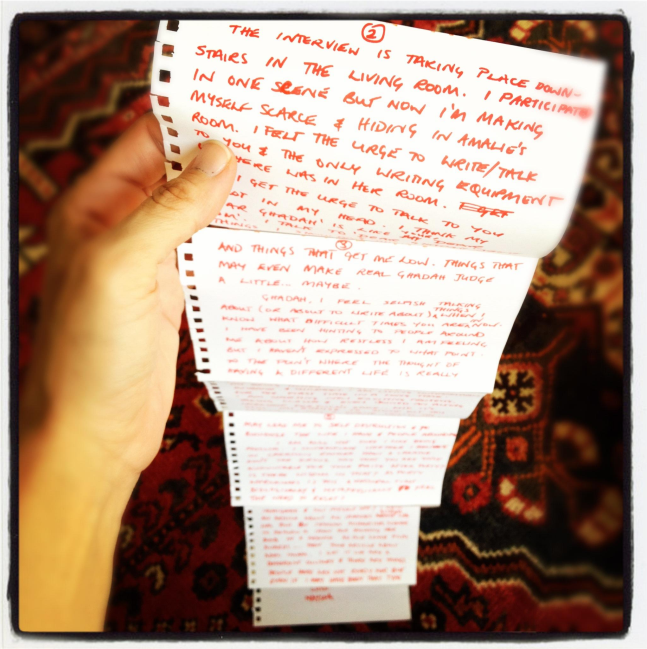
ROBUSTNESS & ASYMMETRY

Other results & Current research

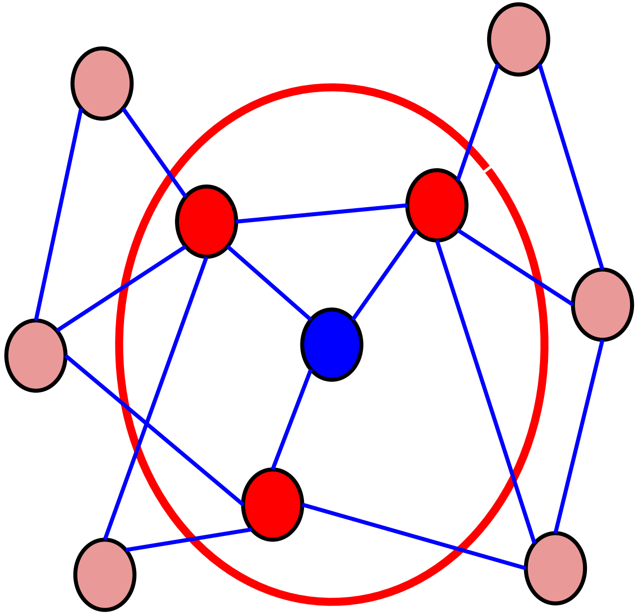


MAXIMUM MESSAGE SIZE

Other results & Current research



MAXIMUM MESSAGE SIZE



NEP LOWER BOUND

References

- ***Simple, Fast, and Deterministic Gossip and Rumor Spreading*, B. Haeupler**
- *Global Computation in a Poorly Connected World*, K. Censor-Hillel et al.

- *Tight bounds for rumor spreading in graphs of a given conductance*, G. Giakkoupis
- *Epidemic Algorithms for Replicated Database Maintainance*, A. Demers et al.
- *Resource Discovery in Distributed Networks*, M. Harchol-Balter et al.
- *Gossip Algorithms: Design, Analysis and Applications*, S. Boyd et al.
- *A Gossip-Style Failure Detection Service*, R. van Renesse et al.

Q & A