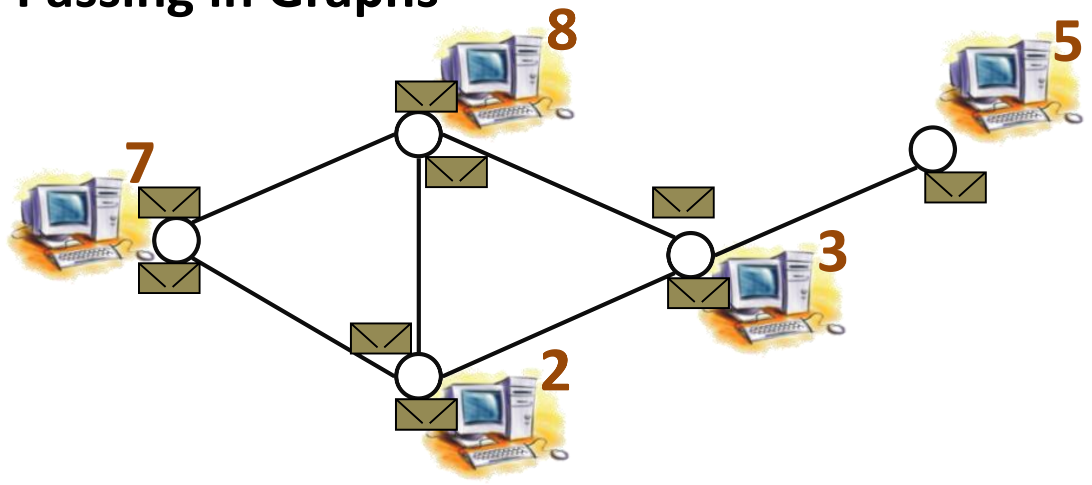
Chapter 11
Synchronization

Theory of Distributed Systems

Fabian Kuhn

Repetition: Message Passing Models

Message Passing in Graphs



Asynchronous Message Passing

- Messages can have arbitrary (but finite) delays
- Event-based (do something upon receiving a message)

Synchronous Message Passing

- Time is divided into synchronous rounds
- Each node can send a message to each neighbor in each round

Synchronous Systems

Synchronous systems:

- System runs in synchronous time steps (usually called **rounds**)
  - Discrete time $0, 1, 2, 3, 4, \ldots$
  - Round r takes place between time $r - 1$ and time r

Synchronous message passing:

- **Round r:**
  At time $r - 1$, each process sends out messages to its neighbors
- Messages are delivered and processed at time r

Time complexity:

- Total number of rounds until all nodes have terminated
  - Assumption: all nodes start at time 0

Asynchronous Systems

Asynchronous systems:

- Message delays are finite but otherwise completely unpredictable
- Assumption: message delays are determined in a worst-case way by an adversarial scheduler

Asynchronous message passing:

- Messages are always delivered (in failure-free executions)
- Message delays are arbitrary (chosen by an adversary)

Asynchronous time complexity:

- Time of a worst-case execution where the message delays are normalized to be at most 1 time unit
  - Typical assumption: all nodes start at time 0

Advantage of Synchronous Systems

Simplicity of Algorithms

- Algorithms are often easier to describe and analyze
  - e.g., the coloring and MIS algorithms from the last two lectures

Algorithm Efficiency

- Easier to get algorithms that are both time and message efficient
- Example from Chapter 2: Constructing a BFS tree
- **Synchronous BFS construction:**
  - Simple flooding, time complexity: $O(D)$, message complexity: $O(m)$
- **Asynchronous BFS construction:**
  - Level-by-level (distributed Dijkstra-like algorithm)
    Time complexity: $O(D^2)$, message complexity: $O(m + D \cdot n)$
  - Distributed Bellman-Ford
    Time complexity: $O(D)$, message complexity: $O(m \cdot D) = O(m \cdot n)$

Synchronizers: Idea

Motivation:

- synchronous algorithms are often simpler and more efficient than asynchronous ones
- however, often real systems are asynchronous

Goal: Run synchronous algorithms in asynchronous systems

Synchronizer:

- Locally simulates rounds at all nodes
- Needs to make sure that when running a synchronous algorithm using the locally simulated rounds:

The local schedules are the same as in the synchronous execution

Clock Pulses

- A synchronizer generates a clock pulse for each round at each node

Valid Clock Pulse:

- A clock pulse of round $i \geq 1$ at node v is valid if it is generated after v has received all the messages of the synchronous algorithm sent by its neighbors in rounds $< i$.

- If we have a mechanism that generates valid clock pulses, a synch. algorithm is turned into an asynch. algorithm in the obvious way
  - Node v sends messages of round i after i^{th} clock pulse at v is generated

Theorem: If all generated clock pulses are valid, the above method provides an asynchronous algorithm that behaves in exactly the same way as the given synchronous algorithm.

Proof: When the i^{th} pulse is generated at a node v, v has sent and received exactly the same messages and performed the same local computations as in the first $i-1$ rounds of the synchronous algorithm. ■

Synchronizer Complexity

Synchronizer $\mathcal{S}$:

- Algorithm that generates *valid clock pulses* in an *asynchronous network*
  - w.r.t. a given synchronous alg. $\mathcal{A}$
  - Assumption: synchronous time and message complexity of $\mathcal{A}$: $T(\mathcal{A})$ and $M(\mathcal{A})$

$T(\mathcal{S})$: time complexity of synchronizer $\mathcal{S}$

- (Asynch.) time complexity per clock pulse (i.e., for simulating 1 round)

$M(\mathcal{S})$: message complexity of synchronizer $\mathcal{S}$

- Number of synchronization messages per clock pulse
  - Synchronization messages are all messages that are not sent by $\mathcal{A}$ anyway

Initialization cost of $\mathcal{S}$: time compl. $T_{\text{init}}(\mathcal{S})$, msg. compl. $M_{\text{init}}(\mathcal{S})$

  - Synchronizer might require some initialization
    (e.g., a leader, a spanning tree, or some other graph structure)

Synchronizer Complexity

- Assume that a synchronous alg. $\mathcal{A}$ with time compl. $T(\mathcal{A})$ and msg. compl. $M(\mathcal{A})$ is run in an asynch. network by using synchronizer $\mathcal{S}$

Total cost of the resulting asynchronous algorithm $\mathcal{A}'$:

Time complexity:

$$T_{ASYNC}(\mathcal{A}') = T_{init}(S) + \boxed{\underline{T(S)} \cdot T(\mathcal{A})}$$

$2 \cdot$ for ack.

Message complexity:

$$M_{ASYNC}(\mathcal{A}') = M_{init}(S) + \boxed{\underline{M(S)} \cdot T(\mathcal{A}) + M(\mathcal{A})}$$

Remark: Since the initialization of $\mathcal{S}$ in a given network G only needs to be done once, we are mostly interested in minimizing the per round costs $T(\mathcal{S})$ and $M(\mathcal{S})$ of a synchronizer $\mathcal{S}$

Node Safety

Challenge: How to generate valid clock pulses?

- A node v needs to know when it has received all messages of its neighbors from a previous round
  - v does not know which of its neighbors want to send it a message

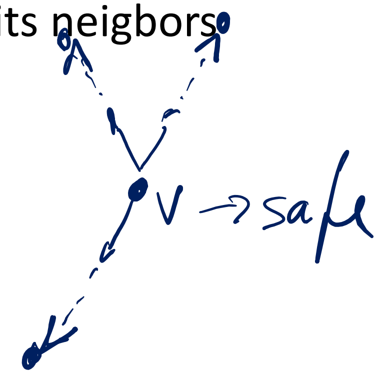
Safe Node: A node v is called safe w.r.t. clock pulse (round) i if all its messages of round i have been received by the neighbors

Observation: A node u can generate clock pulse $i + 1$ when all its neighbors v are safe w.r.t. clock pulse i

- This in particular implies that u has received the messages of its neigbors

How can a node v detect that it is safe?

send acknowledgements!



Detecting Node Safety

Protocol to detect node safety:

- Assume that we send an acknowledgement for each message of the algorithm $\mathcal{A}$
- After receiving all acknowledgements, a node v is certainly safe

Cost of sending acknowledgements:

- The total additional message cost is at most $M(\mathcal{A})$
- The additional time cost per clock pulse (round) is 1
- Hence, acknowledgements do not increase the asymptotic cost

In the following, we will assume that our algorithm sends acknowledgements for all messages of $\mathcal{A}$ without explicitly analyzing the cost of these messages.

Synchronizers: Ideas?

A node v can generate the next clock pulse when all neighbors are safe.

How can a node v detect that all neighbors are safe?

every node u sends msg. to all neighbors when u is safe

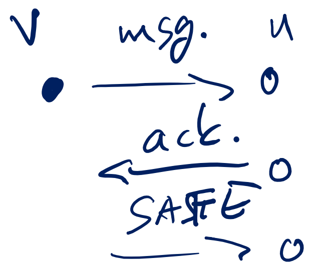
Synchronizer α

Synchronizer α at node v (for each clock pulse)

1. **wait** until v is safe (until all ack. have been received)
2. **send** SAFE to all neighbors
3. **wait** until v receives SAFE from all neighbors
4. start next clock pulse

Theorem: The time and message complexities of synchronizer α per synchronous round (clock pulse) are

$$T(\alpha) = O(1) \quad \text{and} \quad \underline{M(\alpha) = O(m)}$$



← pulse i gen. at v

$T(\alpha) = 3$

Simpler Version of Synchronizer α

- Synchronizer α was presented by using the general framework
- There is an easier and slightly more efficient way to achieve the same
- In each round, each node u sends a message to each neighbor v containing the following data
  - The data of the message u sends to v in algorithm $\mathcal{A}$ (if there is such a msg.)
  - The current round number
    (If in $\mathcal{A}$, u does not send a msg. to v, u sends a dummy message that just contains the round number)
- A node can then move on to the next round when it has received the messages of the previous round from all neighbors

Synchronizer α: Discussion

Advantages of synchronizer α:

- does not require any precomputation
- very time-efficient $T(\alpha) = O(1)$

$\Longrightarrow$ time complexity of resulting asynchronous alg. is $O\big(T(\mathcal{A})\big)$

– i.e., α allows to turn a synchronous algorithm into an asynchronous algorithm with the same asymptotic time complexity

Disadvantages of synchronizer α:

- very high message cost: $M(\alpha) = O(m)$

$\Longrightarrow$ msg. complexity of resulting asynch. alg. is $O\big(\underline{M(\mathcal{A})} + \underline{m} \cdot \underbrace{T(\mathcal{A})}_{D}\big)$

– Example: Synchronous BFS construction has $T(\mathcal{A}) = D$ and $M(\mathcal{A}) = O(m)$, using α, we get an asynch. alg. with msg. compl. $O(m \cdot D)$ (same as distr. Bellman-Ford)

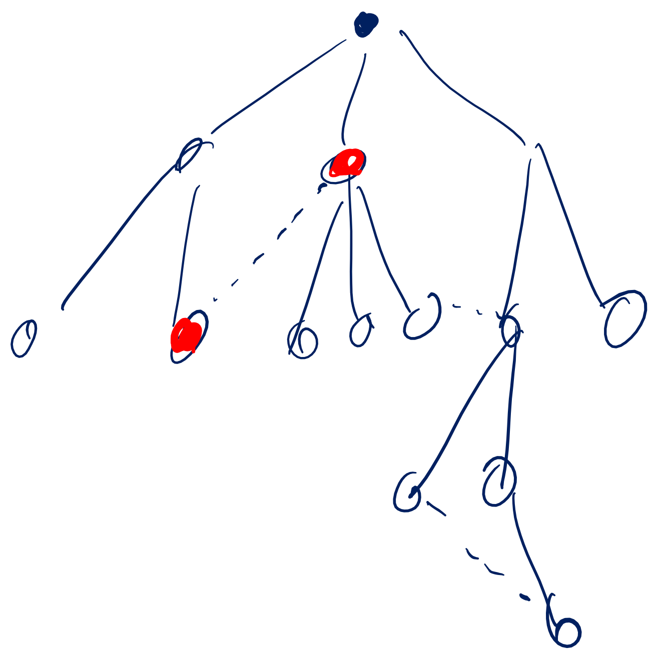
Is it possible to be more message-efficient?

– maybe at the cost of a higher time complexity…

Message-Efficient Synchronization

Ideas?

build a spanning tree T



Synchronizer β

- As a precomputation, compute a rooted spanning tree T

Synchronizer $\boldsymbol{\beta}$ at node $\boldsymbol{v}$ (for each clock pulse)

```
1.  **wait** until $v$ is safe (until all ack. have been received)
2.  **wait** until $v$ receives SAFE message from all children in $T$
3.  **if** $v \neq \mathrm{root}$ **then**
4.      **send** SAFE to parent in $T$
5.      **wait** until PULSE message received from parent in $T$
6.  **send** PULSE message to all children in $T$
7.  start next clock pulse
```

Synchronizer β

Theorem: The time and message complexities of synchronizer β per synchronous round (clock pulse) are

$$T(\beta) = O(n) \quad \text{and} \quad M(\beta) = O(n)$$

$T(\beta) = O(\text{depth of tree}) = O(n)$

$M(\beta) = O(\text{\# edges of tree}) = O(n)$

Cost of initialization

- There is an asynchronous distributed algorithm that computes a spanning tree in time $O(n)$ with message complexity $\underline{O(m + n \log n)}$.
  - Does not require a leader, optimal with some natural assumptions
  - The diameter of this tree might be linear in n even if the diameter of G is small

Synchronizer β: Discussion

Advantages of synchronizer β:

- relatively message-efficient $M(\beta) = O(\underline{n})$

  $\Longrightarrow$ msg. complexity of resulting asynch. alg. is $O\big(\underline{M(\mathcal{A})} + n \cdot T(\mathcal{A})\big)$
  - Example: Synchronous BFS construction $\underline{\text{has } T(\mathcal{A}) = D}$ and $M(\mathcal{A}) = \underline{O(m)}$, using β, we get an asynch. alg. with msg. compl. $O(\underline{m} + \underline{D \cdot n})$ (same as distr. Dijkstra)

Disadvantages of synchronizer β:

- Needs a leader and a spanning tree (precomputation)
- very high time cost: $T(\beta) = O(\text{tree diameter}) = O(n)$

  $\Longrightarrow$ time complexity of resulting asynch. alg. is $O\big(n \cdot T(\mathcal{A})\big)$
  - Example: Synchronous BFS construction has $T(\mathcal{A}) = D$ and $M(\mathcal{A}) = O(m)$, using β, we get an asynch. alg. with time complexity $O(D \cdot n)$
    (if tree has diameter $O(D)$, we time $O(D^2)$ as in the distr. Dijkstra alg.)

Can be we efficient w.r.t. time and message complexity?

Time and Message-Efficient Sychronization

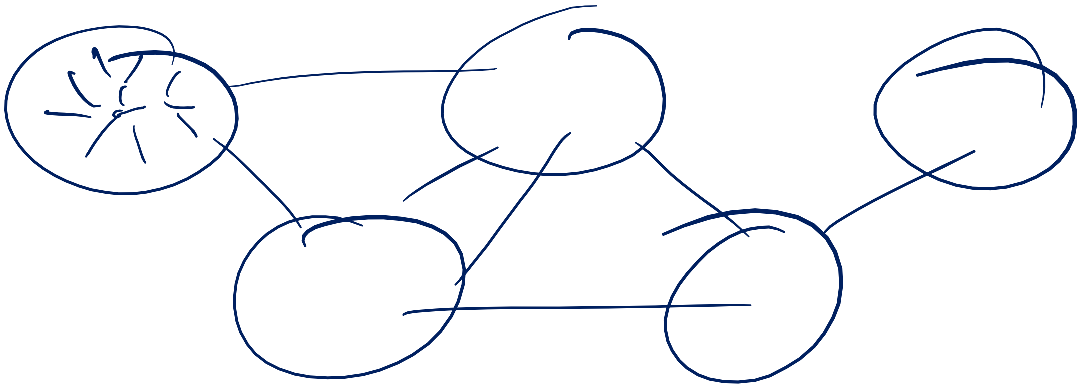
Ideas?

Combine α and β

build clusters of small diameter

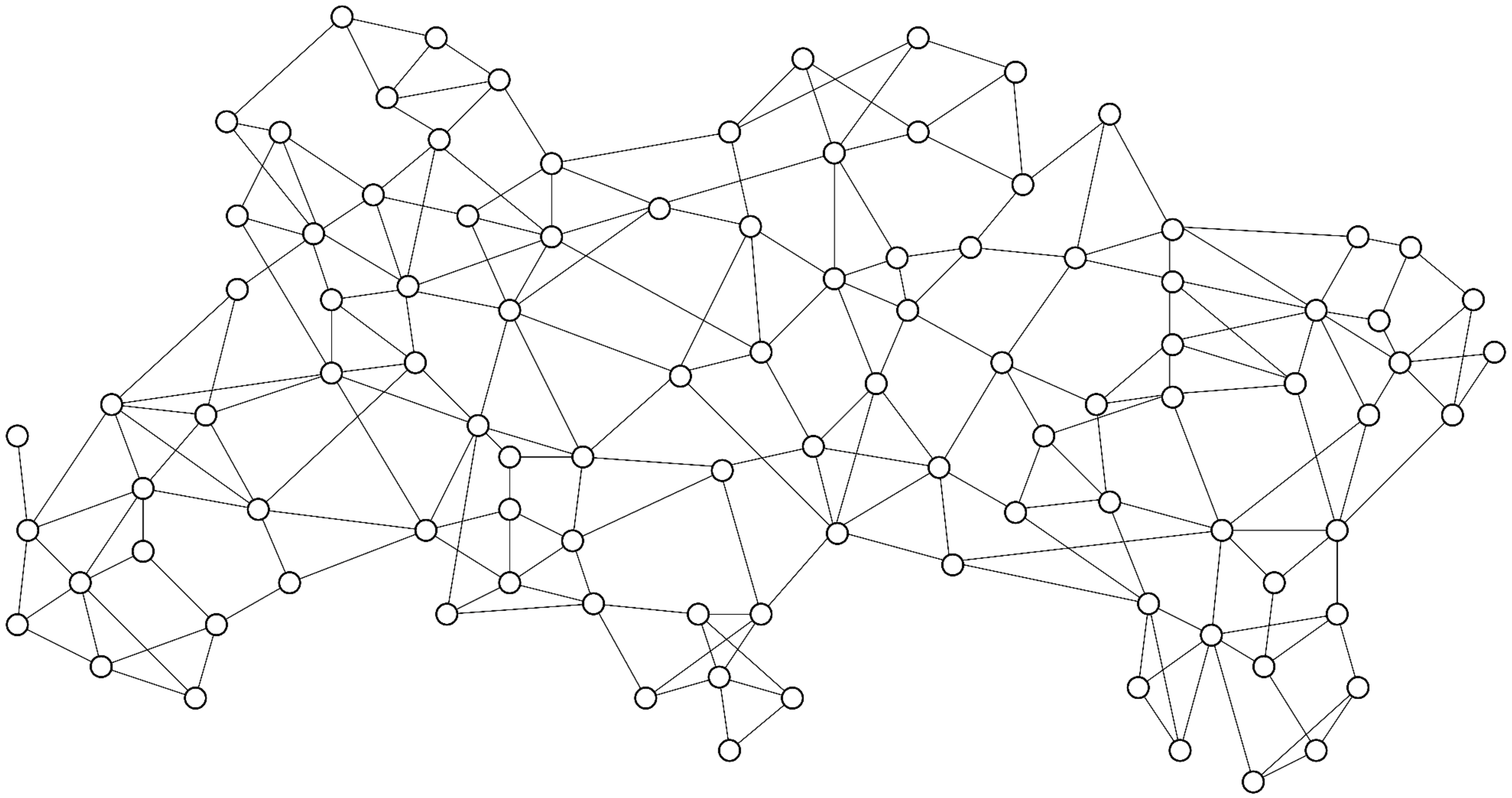
+ spanning trees in all clusters

$\rightarrow$ use β inside clusters and α between clusters



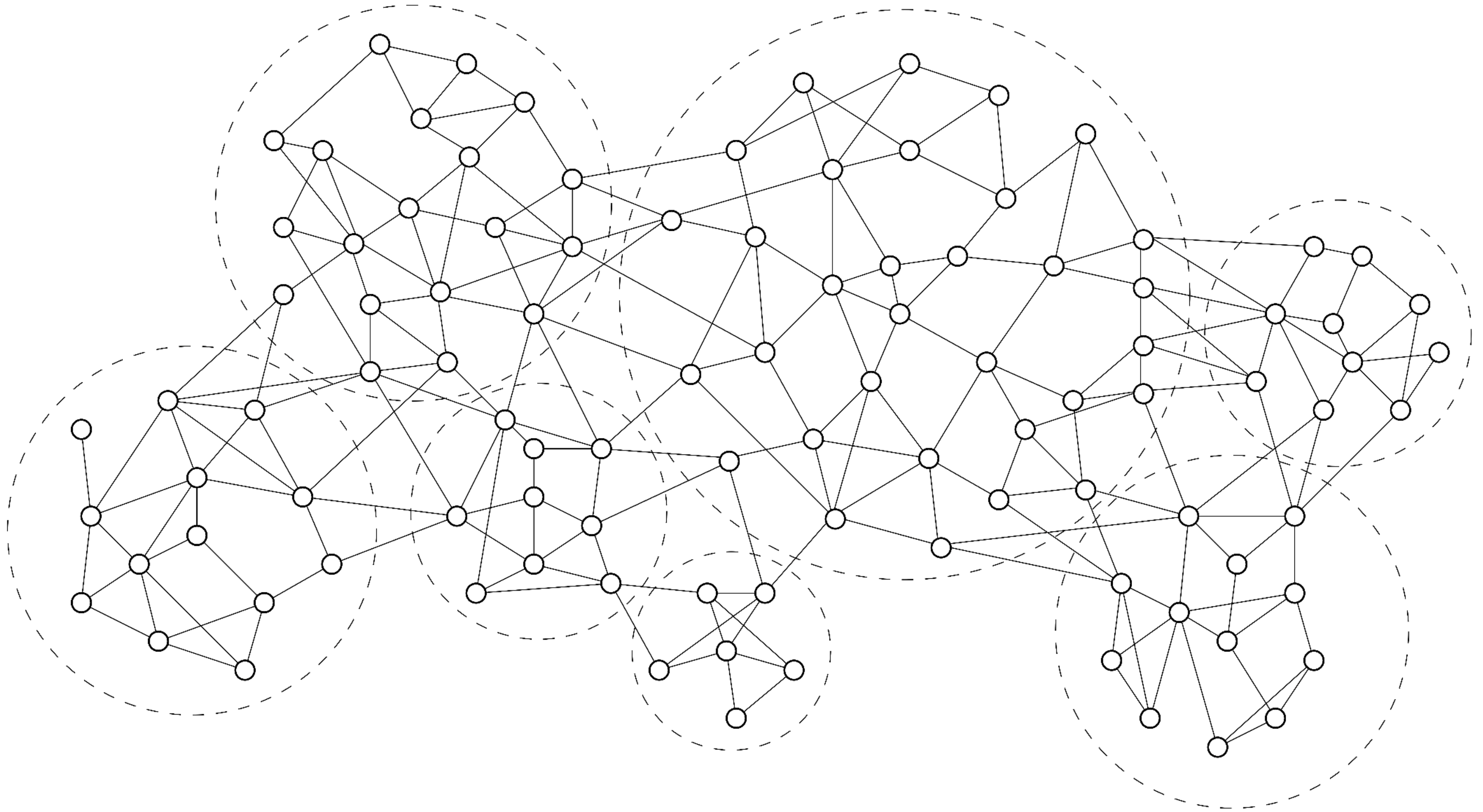
Clustering of Network

Precomputation: A partition into clusters of small diameter



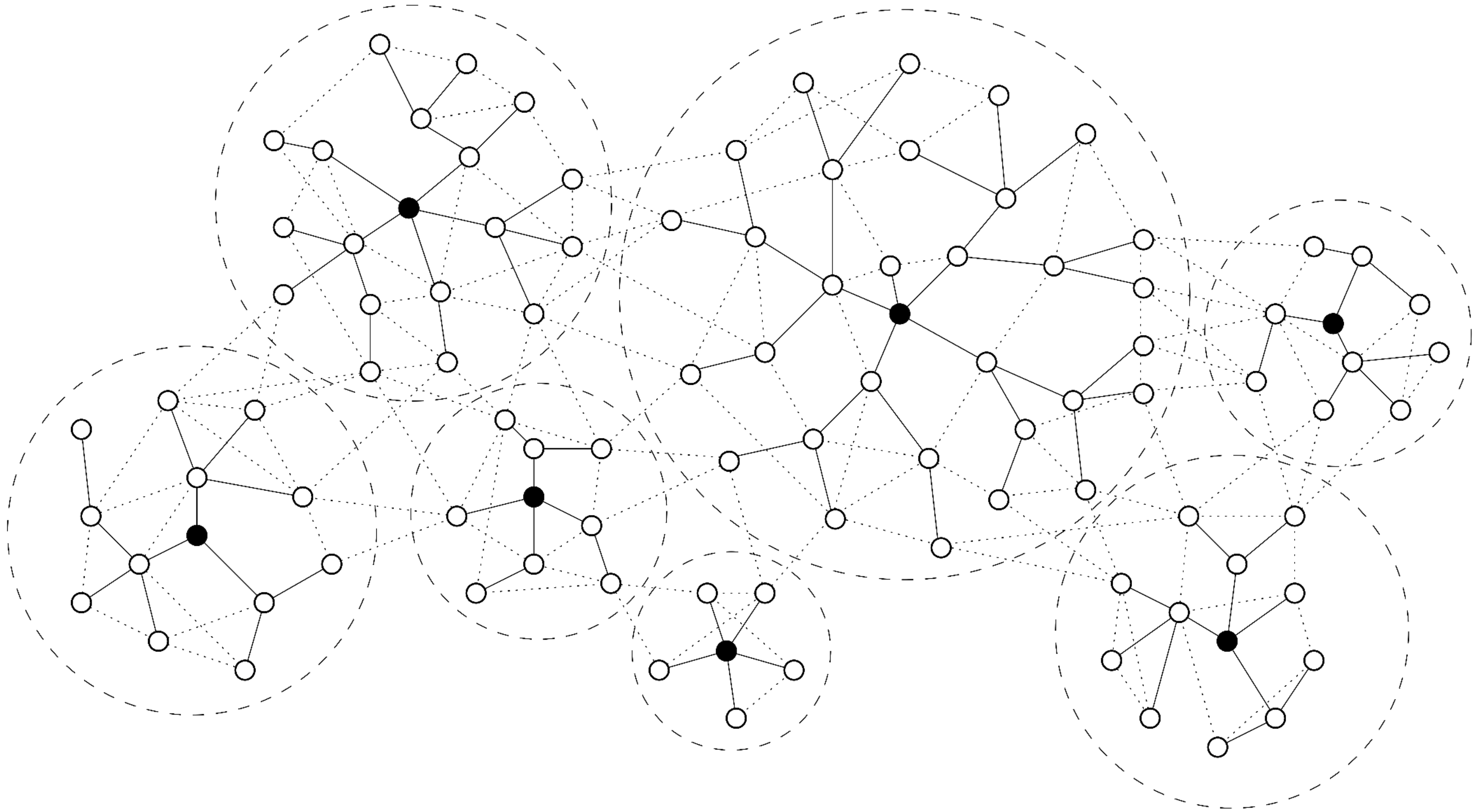
Clustering of Network

Precomputation: A partition into clusters of small diameter



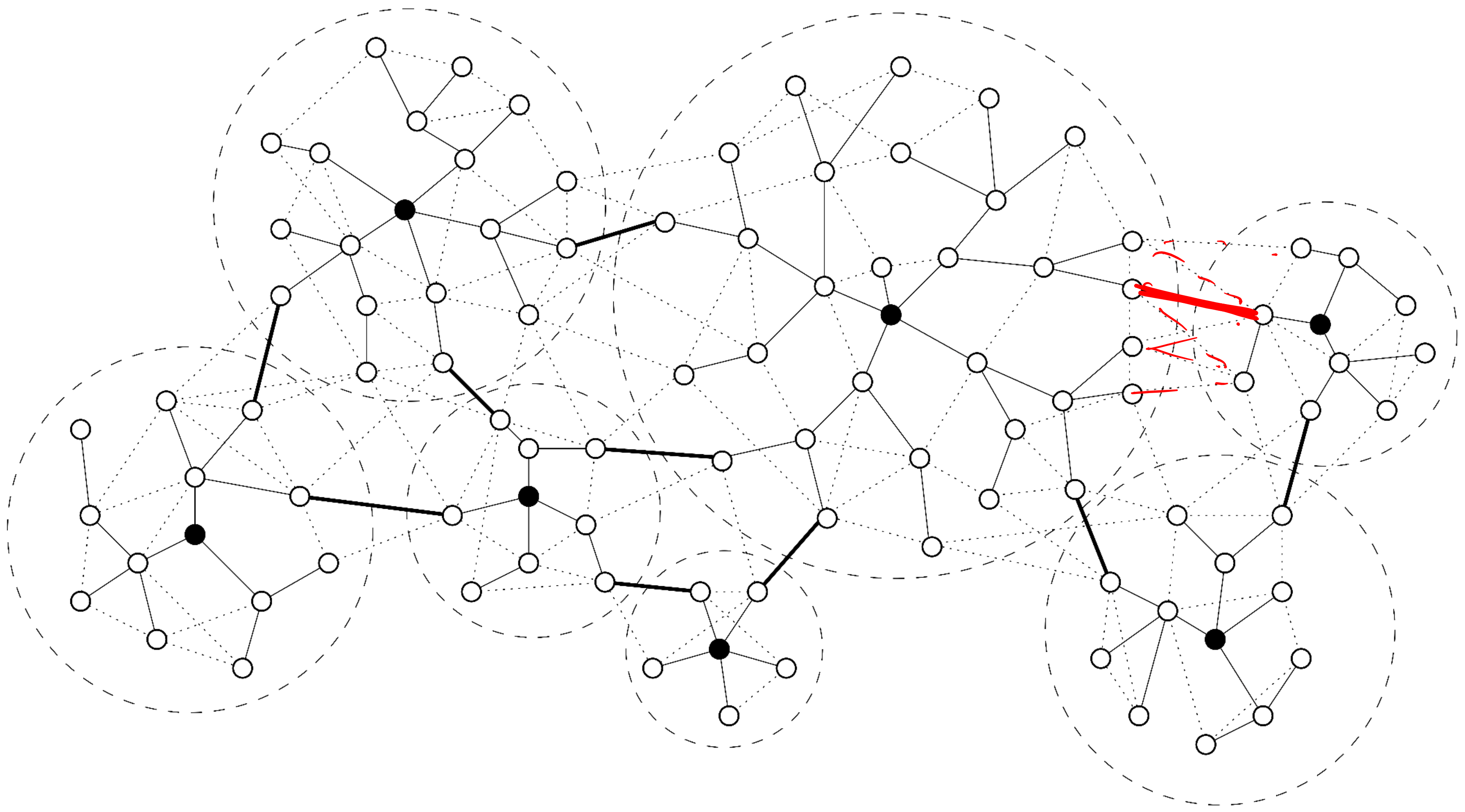
Clustering of Network

Precomputation: A partition into clusters of small diameter



Clustering of Network

Precomputation: A partition into clusters of small diameter



Clustering of Network

Precomputation: A partition into clusters of small diameter

Properties of clustering:

- Maximum cluster radius R
- Number of intercluster edges m_C

Synchronizer γ Ideas:

- Use synchronizer β to synchronize within each cluster
- Use synchronizer α to synchronize between cluster

Synchronizer γ at node v

```
1.  wait until $v$ is safe (until all ack. have been received)
2.  wait until $v$ receives SAFE message from all children in cluster tree
3.  if $v \neq \text{cluster root}$ then
4.      send SAFE message to parent in cluster tree
5.      wait until CLUSTERSAFE msg. received from parent in cluster tree
6.  send CLUSTERSAFE message to all children in cluster tree
7.  send NEIGHBORSAFE msg. over all intercluster edges of $v$
8.  wait until $v$ receives NEIGHBORSAFE msg. over all intercluster edges
        of $v$ and from all children in cluster tree
9.  if $v \neq \text{cluster root}$ then
10.     send NEIGHBORSAFE message to parent in cluster tree
11.     wait until PULSE message received from parent in cluster tree
12. send PULSE message to all children in cluster tree
13. start next clock pulse
```

Synchronizer γ

Theorem: The time and message complexities of synchronizer γ per synchronous round (clock pulse) are

$$T(\gamma) = O(R) \quad \text{and} \quad M(\gamma) = O(m_C + n)$$

- where $R =$ max. cluster radius and $\underline{m_C = \#}$ of intercluster edges

$T(\gamma)$: 2 conv.casts ~~+ 2 flooding~~ in each cluster tree + $O(1)$ time for exchange on intercl. edges

$\hookrightarrow$ time: $O(\text{max. cluster tree radius}) = O(R)$

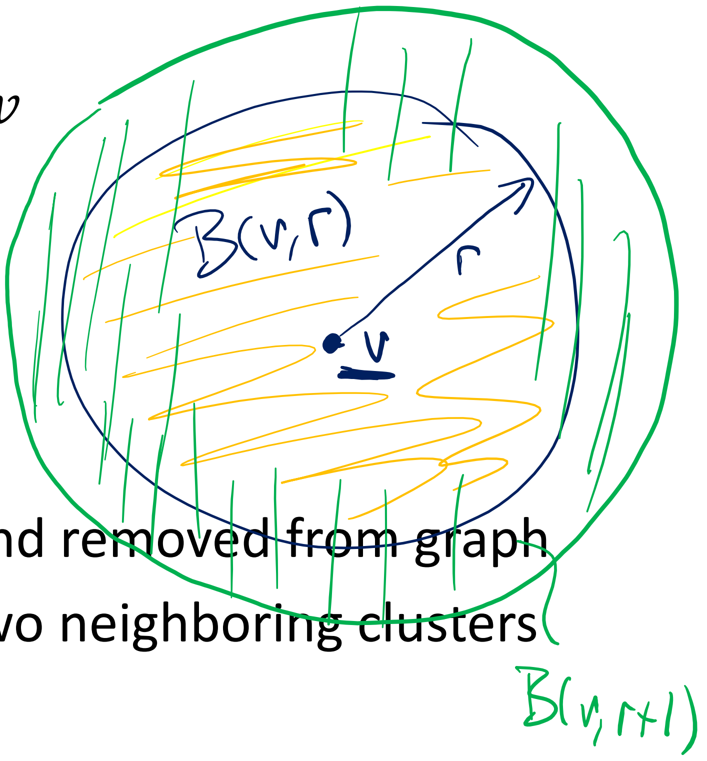
$M(\gamma)$: msg. inside clusters : $O(n)$

msg. between clusters: $O(m_C)$

Construction of Clustering (Sequential Alg.)

- Construction has a parameter $\rho > 1$ (think of $\rho = 2$)
- $B(v,r)$ means ball of radius r around v
  - Set of nodes at distance at most r from v

```
1.  while there are unprocessed nodes do
2.      select an arbitrary unprocessed node $v$
3.      $r := 0$
4.      while $\boxed{|B(v,r+1)|} > \underset{2}{\rho} \cdot \boxed{|B(v,r)|}$ do     ($\leq$ ← stop)
5.          $r := r + 1$
6.      makeCluster$(B(v,r))$
7.      nodes in $B(v,r)$ are now processed and removed from graph
8.  add one intercluster edge between any two neighboring clusters
```



Properties of Clustering

Theorem: The radius of each cluster is at most $R = O(\log_\rho n)$

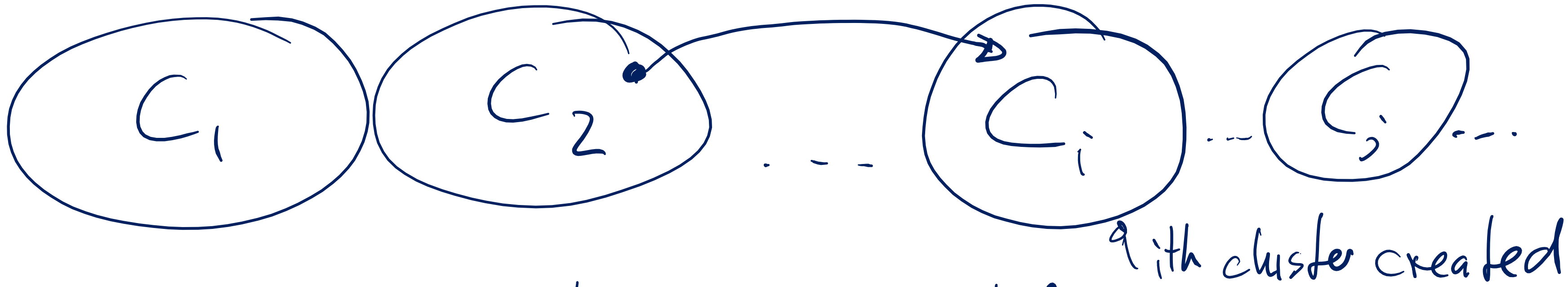
grow r until $|B(v,r+1)| \leq \rho |\underset{\substack{\uparrow \\ \text{cluster}}}{B(v,r)}|$

$n \geq |B(v,r)| > \rho^r$

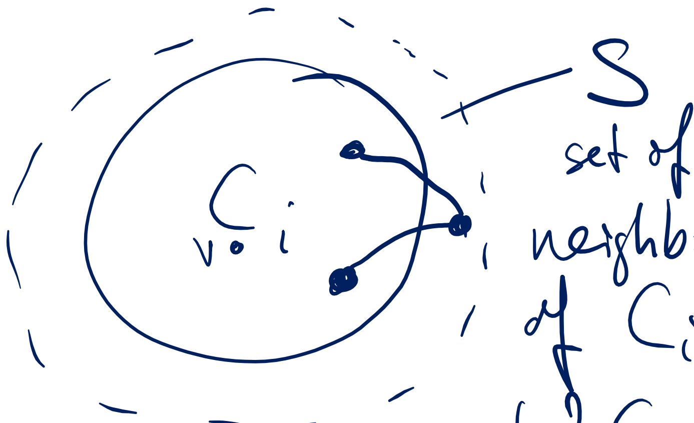
$\hookrightarrow r < \log_\rho n$

Properties of Clustering

Theorem: The number of intercluster edges is $m_c \leq (\rho - 1) \cdot n$



$$\underline{(\#\text{ intercl. edges } \text{between } C_i \text{ and } \bigcup_{j>i} C_j)} \leq |S|$$



set of neighbors of C_i in $\bigcup_{j>i} C_j$

$$\underbrace{|S \cup C_i|}_{B(v,r+1)} \leq \rho \cdot \underbrace{|C_i|}_{B(v,r)}$$

$$|S| \leq \underline{(\rho - 1)|C_i|}$$

$$m_c \leq (\rho - 1) \cdot n$$

Synchronizer γ

Theorem: The time and message complexities of synchronizer γ per synchronous round (clock pulse) are

$$\underline{T(\gamma) = O(\log_\rho n)} \quad \text{and} \quad \underline{M(\gamma) = O(\rho \cdot n)}$$

$$M(\gamma) = \Theta(n + \underbrace{(\rho-1)n}_{m_c})$$

$\underline{\rho = 2}$: $T(\gamma) = \Theta(\log n)$, $M(\gamma) = \Theta(n)$

almost as good as α — same as for β

$\underline{\text{BFS}}$: $\underline{\text{time}}$: $\Theta(D \cdot \log n)$, $\underline{\text{msg}}$: $\Theta(m + D \cdot n)$

(without precomp.)

$\rho = n^{1/k}$: $T(\gamma) = \Theta(k)$, $M(\gamma) = O(n^{1+1/k})$

Discussion of Synchronizer γ

- Precomputation can be done in time $O(n)$ with $O(m + n \log n)$ msg.
- When applying to computing a BFS tree, we get

  time compl.: $O(n + D \cdot \log n)$, msg. compl.: $O(m + n \log n + D \cdot n)$
- α, β, and γ achieve global synchronization (every node generates every clock pulse). Often (e.g., BFS construction) each node only participates in a few synchronous rounds. In such cases, it is possible to achieve a synchronizer with time and message complexity $O(\log^3 n)$ (no initialization)
- The time/msg. trade-off of synchronizer γ is optimal if all nodes need to generate all clock pulses
- Partitions/coverings of networks with clusters of small diameter come in different variants and have many applications in distributed computations, e.g., for routing, constructing sparse spanning subgraphs, distributed data structures, and also computations of local graph structures such as colorings or maximal independent sets