



SAPIENZA  
UNIVERSITÀ DI ROMA

# Ad hoc Network Routing - Clustering Internet of Things (ex. Reti Avanzate) a.a 2015/2016

Un. of Rome "La Sapienza"

Chiara Petrioli<sup>†</sup>

<sup>†</sup> *Department of Computer Science – University of Rome "Sapienza" – Italy*



- What happens to protocols when the number of network nodes grows?
  - Especially crucial in WSNs
- A traditional networking solution: Hierarchical organization of the nodes
- Network nodes are grouped into clusters
- Some nodes, locally the “best,” are selected to coordinate the clustering process: Clusterheads



- Independence of the clusterheads
- Dominance of the clusterheads
- Possibility to express "preferences"
- Distributed operations
- Fast and simple implementation



- Heuristics based on Independent Sets
  - Minimum ID approach (Gerla & al.)
  - Maximum degree (Ephremides & al.)
- Heuristics based on Dominating Sets
  - The concept of “spine”
  - Minimum connected dominating set



- A subset  $V'$  of the vertices  $V$  of a graph  $G=(V,E)$  is independent when for each  $u,v \in V'$  the edge  $\{u,v\} \notin E$
- MIS is an Optimization Problem
- Input: A Graph  $G=(V,E)$  with  $n$  vertices
- Output: A subset  $V'$  of  $V$  that is independent and has maximum size



- No known algorithm computes a MIS in polynomial time
- Need for approximate solutions
- And approximation algorithm is an algorithm that produces a solution that is not optimal, but that approximates it
- We sacrifice optimality in favor of a “good” solution that can be computed efficiently



- Bad news
  - Not only MIS is computationally hard
  - It is also hard to approximate:
    - ✓ Approximate solutions are not so good
    - ✓ They are “unboundedly” far from the optimum
- We consider the simple greedy heuristic for the MIS



- Select the vertex with minimum degree and put it in the MIS
  - The degree of a vertex is the number of its neighbors
    - ✓ Cardinality of its adjacency list
  - Keep going till all the vertices are either in the MIS or COVERED by a vertices in the MIS





MIS(V,E,d) // d is the vector of degrees

mis =  $\emptyset$

while  $V \neq \emptyset$  do

    v = vertex with min degree

    mis = mis  $\cup$  {v}

$V = V - \{v\} \cup N(v)$

return mis



- Bad news: Still computationally hard
- Better news: Minimum DS It is approximable “up to a constant”
  - It means that the ratio between the size of a DS computed by MIS greedy on UDGs and the size of a MDS is  $< c$ ,  $c$  a constant
- This constant is 5



- The greedy solution provides a maximal independent set
  - An independent set is maximal when, if you add a vertex, the set is no longer independent
    - ✓ You cannot make a maximal independent set bigger
- This solution is also a minimal dominating set
  - A dominating set  $D$  subset of  $V$  is a set such that a vertex  $v \in V$  is either in  $D$  or it has a neighbor in  $D$
- Solutions we will see are variant of this approach



- Key fact: In a UDG disk (radius 1) there are at most 5 independent nodes
- Consider an Optimal solution and a Greedy solution
- Since Opt is dominant, it dominates Greedy
- Assign every vertex of Greedy to one dominator in Opt (choose one if more)



- For each  $u$  in Opt consider its assigned vertices  $v_1(u)$ ,  $v_2(u)$ , ...,  $v_k(u)$  of Greedy
- How big is  $k$ ?
- Well, all  $v_i(u)$  must be distant 1 from  $u$  and they also have to be independent
- Greedy: at most 5 times bigger than Opt



- routing always through the clusterhead
- data aggregation at the clusterhead
- easy to locally synchronize nodes within the cluster, using TDMA MAC protocol for intra-cluster communication and different MAC protocols (e.g. CDMA) for inter-cluster communications
- Challenge: cluster maintenance



- MWIS = Maximal Weight Independent Set
- Clustering selection based on generic **weights** (real numbers  $> 0$ )
  - Mobility/node related parameters (e.g., energy, link quality ...)
  - Generalizes previous “Independent Set” solutions



- Distributed Clustering Algorithm (DCA)
  - Quasi-mobile networks, periodical reclustering. Allow complexity analysis, fast and simple
- Distributed and Mobility-Adaptive Clustering (DMAC) Algorithm
  - Same rules/procedures for clustering set up and maintenance, adaptive to nodes mobility and node/link failures





- Assumptions

- Knowledge of IDs and weights of one-hop neighbors
- Broadcast transmission of a packet in finite time (a "step")
- Nodes do not move during clustering



- (Only) Two messages:
  - CH(v): Sent by a clusterhead v
  - JOIN(u,t): Sent by ordinary node u when it joins the cluster of clusterhead t
- Three (simple) procedures:
  - Init (start up)
  - OnReceivingCH(v), OnReceivingJOIN(u,v) (message triggered)



- Each node knows its neighbors and their weight
- A node is init if it has the largest weight in its neighborhood
- Init nodes become clusterheads and invite their neighbors to join the cluster
- A node waits to receive messages from larger weight neighbors before it makes a decision on its role
  - If a neighbor with larger weight invites it to join its cluster then node x sends a JOIN and enters the cluster as ordinary node
  - Otherwise it becomes clusterhead and sends a CH message to invite its neighbors to join its cluster



- Two types of messages
  - $CH(v)$  used by node  $v$  to let its neighbors know it is acting as clusterhead
  - $JOIN(v,u)$  used by  $v$  to communicate to its neighbors that it will be part of the cluster whose clusterhead is node  $u$



- Variables at node  $v$ 
  - $\text{Cluster}(v)$  identifies the set of nodes which are part of node  $v$ 's cluster
  - $\text{Clusterhead}$  is a variable which identifies the clusterhead of the cluster I am affiliated to.
  - $\text{Ch}(u)$  is a boolean variable set to true if I have either sent a CH message (in this case the boolean variable is set to true for  $u==v$ ) or I have received a CH message from node  $u$ .
  - Boolean variable  $\text{Join}(u,t)$  is true if node  $v$  has received a  $\text{JOIN}(u,v)$  message from node  $u$



- Init

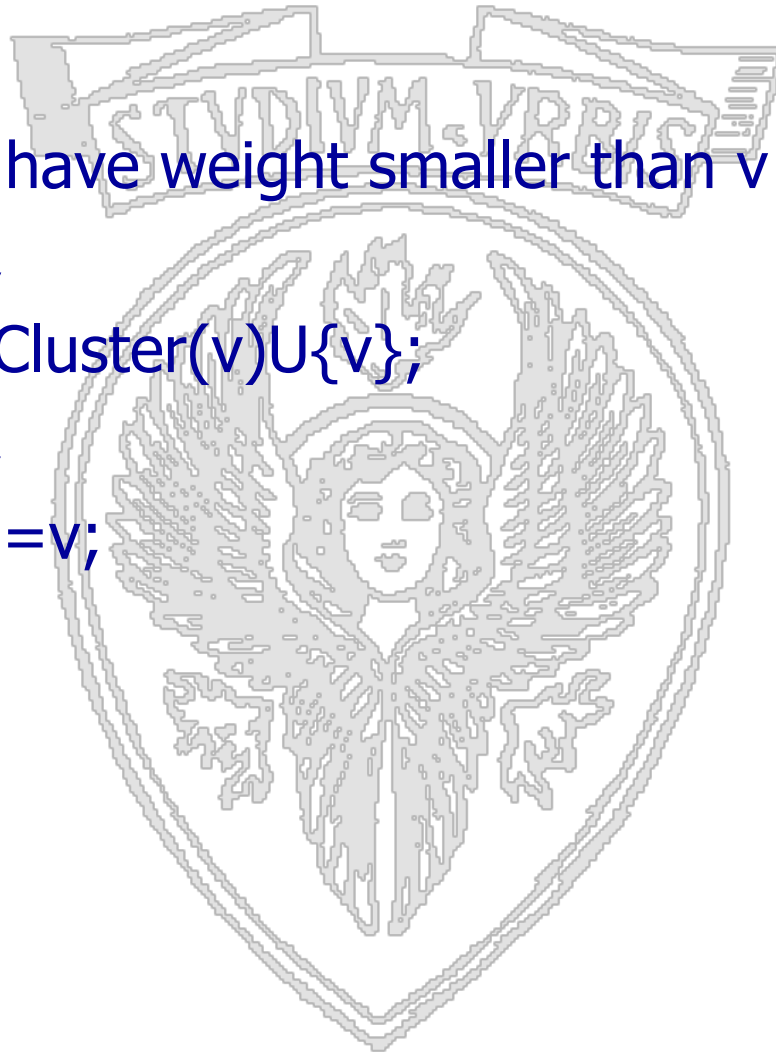
If all neighbors have weight smaller than v

send CH(v);

Cluster(v)=Cluster(v)U{v};

Ch(v)=true;

Clusterhead=v;





## *DCA-Procedure (executed at node v)*

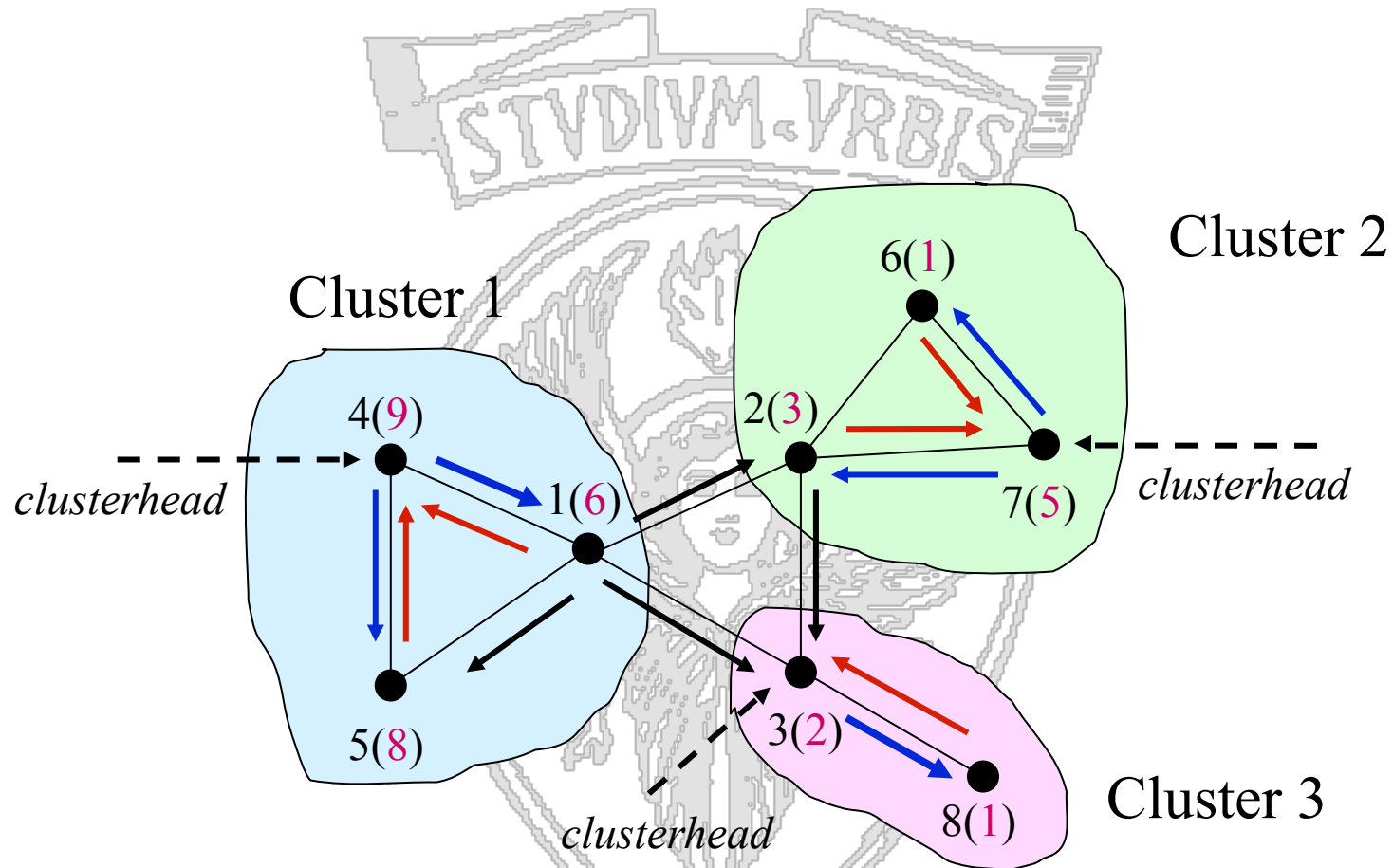
- On receiving CH(u)  
Ch(u)=true;  
If u has weight greater than mine and all my other neighbors with weight larger than his have sent a Join then:  
Clusterhead=u;  
send JOIN(v,Clusterhead);



## *DCA-Procedure (executed at node v)*

- On receiving JOIN(u,t)  
Join(u,t)=true;  
If v is a clusterhead then if t==v  
  {Cluster(v)=Cluster(v)U{u};  
  If I have received Join from all smaller weight neighbors EXIT}  
Otherwise if all bigger weight neighbors have made a decision on their role  
  {if they all have sent a JOIN  
    {send CH(v);  
    Cluster(v)=Cluster(v)U{v};  
    Clusterhead =v;  
    If JOIN have been received by all smaller weight neighbors EXIT.}  
  Otherwise (one or more bigger weight neighbors have sent a CH)  
    {Clusterhead= biggest weight neighbor among those who have  
    become clusterhead and have invited me to join their cluster sending a  
    send JOIN(v,Clusterhead);  
    EXIT;}  
  }  
}





I Step

II Step

III Step

IV Step

V Step



- Consider

$$\tau: V \rightarrow \{1, 2, 3, \dots, 2k\}$$

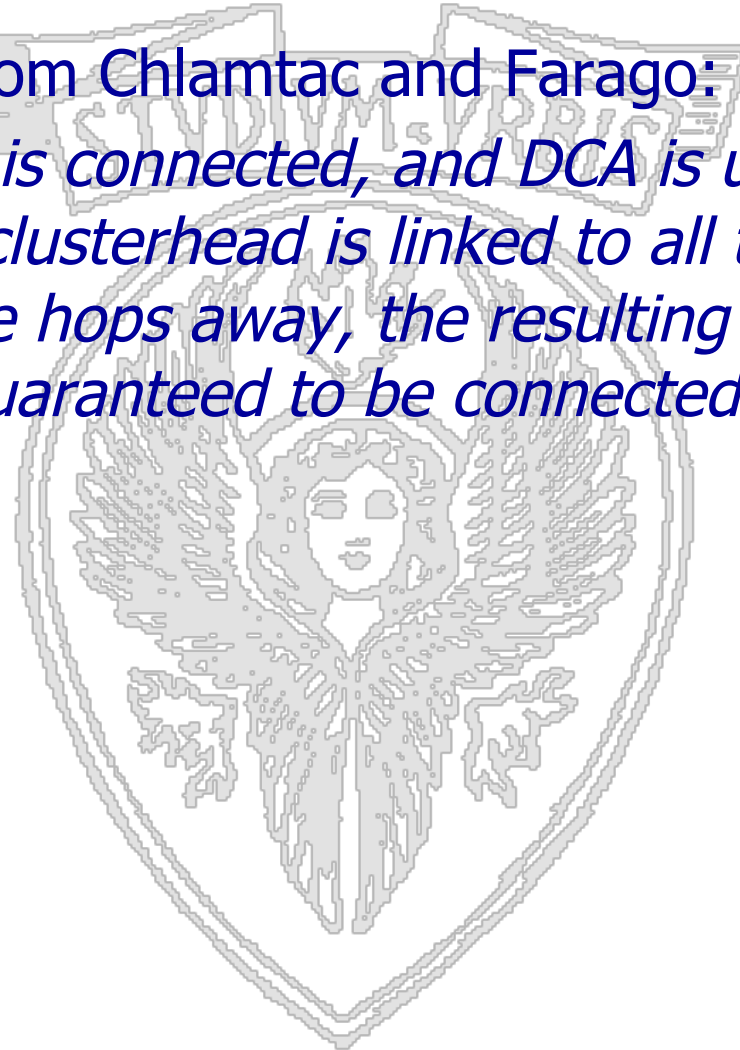
$V$  = set of network nodes,  $k$  = number of clusters

- **Proposition:** Each node  $v$  in  $V$  sends exactly one message by  $\tau(v)$  steps
- **Corollary 1:** DCA message complexity is  $n = |V|$
- **Corollary 2:** DCA terminates correctly in at most  $2k$  steps (  $\leq 2n$  )



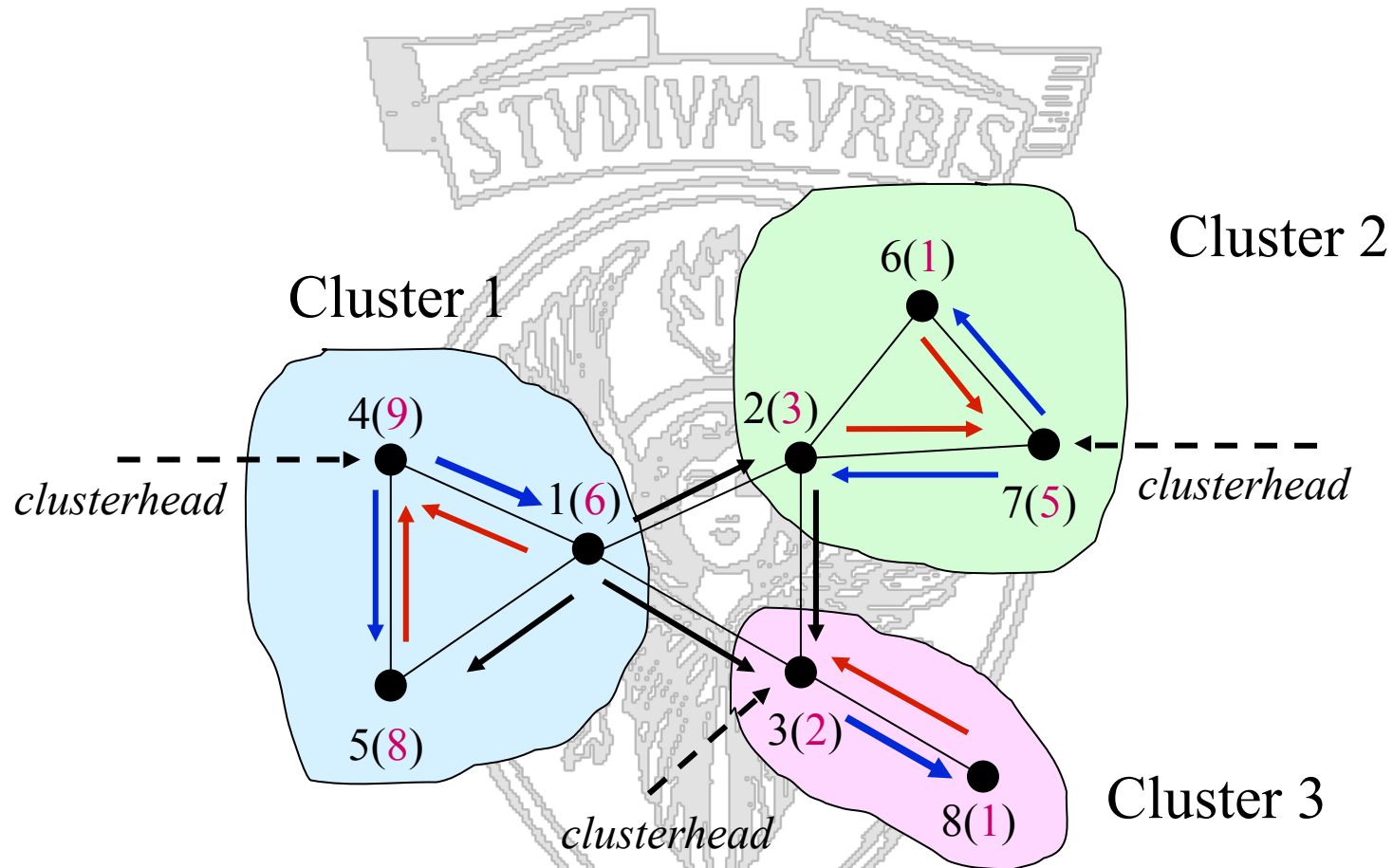
## Joining Clusterheads: Dynamic Backbone

- A theorem from Chlamtac and Farago:  
*If a network is connected, and DCA is used, then if and only if each clusterhead is linked to all the clusterheads at most three hops away, the resulting backbone network is guaranteed to be connected*





# Example



I Step

II Step

III Step

IV Step

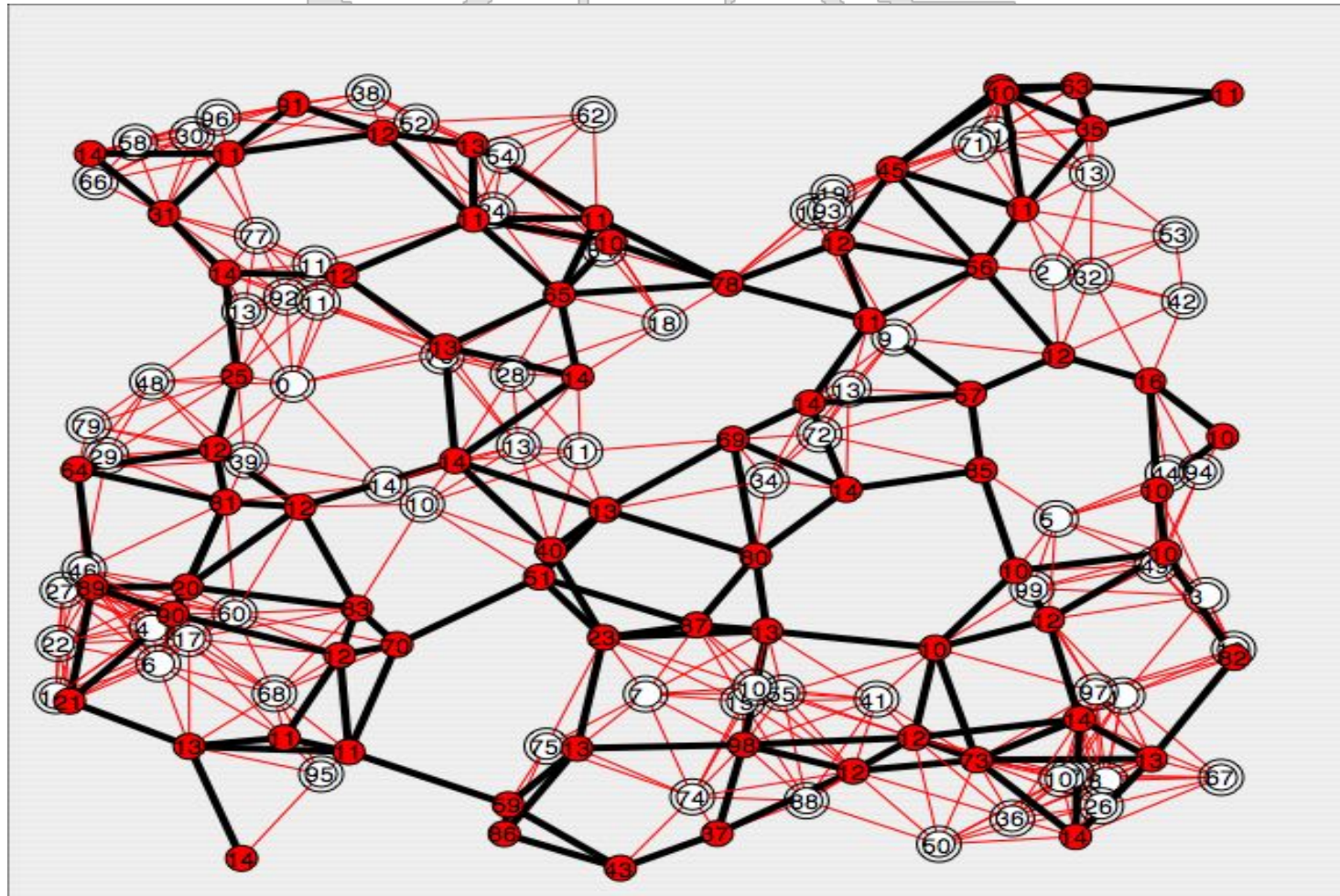
V Step



- 3 representatives of major approaches
  - Selection of independent set of nodes and backbone construction (**DCA**)
  - Rich dominating set formation and pruning (**WuLi**)
  - Two-phase algorithm with theoretical guarantees (**WAF**)
- 1 proposal after the performance comparison (**DCA-S**)



- Distributed and localized implementation of the greedy for independent set
- Takes node status into account for node selection
- Independent nodes are joined into a connected backbone (connectivity is guaranteed) via gateways
- Low degree of parallelism (“dependency chains”)





- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism (“all localized”)





- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism (“all localized”)

If a vertex  $v$  has two neighbors which are not in visibility range it enters the set  $C$



- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism (“all localized”)

What is needed is, from the neighbors, whether they are in C and their list of neighbors.

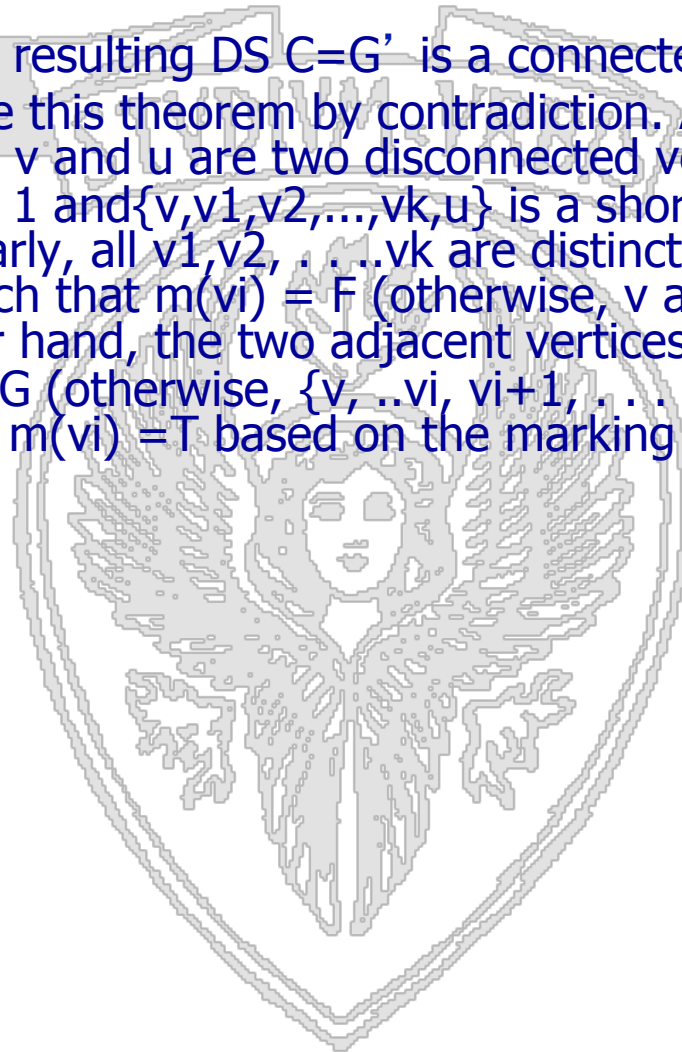


Assume  $V'$  is the set of vertices that are marked T in  $V$ , i.e., the set of vertices which initially enter the CDS since they have at least two neighbors which are not neighbors of each other. The reduced graph  $G'$  is the subgraph of  $G$  induced by  $V'$ . The following two theorems show that  $G'$  is a dominating set of  $G$  and it is connected.

- THEOREM 1: Given a  $G = (V, E)$  that is connected, but not completely connected, the vertex subset  $V'$ , derived from the marking process, forms a dominating set of  $G$ .
- PROOF: Randomly select a vertex  $v$  in  $G$ . We show that  $v$  is either in  $V'$  (a set of vertices in  $V$  that are marked T) or adjacent to a vertex in  $V'$ . Assume  $v$  is marked F, if there is at least one neighbor marked T, the theorem is proved. When all its neighbors are marked F, we consider the following two cases: (1) All the other vertices in  $G$  are neighbors of  $v$ . Based on the marking process and the fact that  $m(v)=F$ , all these neighbors must be pairwise connected, i.e.,  $G$  is completely connected. This contradicts to the assumption that  $G$  is not completely connected. (2) There is at least one vertex  $u$  in  $G$  that is not adjacent to vertex  $v$ . Construct a shortest path,  $\{v, v_1, v_2, \dots, u\}$ , between vertices  $v$  and  $u$ . Such a path always exists since  $G$  is a connected graph. Note that  $v_2$  is  $u$  when  $v$  and  $u$  are 2-distance apart in  $G$ . Also,  $v$  and  $v_2$  are not directly connected; otherwise,  $\{v, v_2, \dots, u\}$  is a shorter path between  $v$  and  $u$ . Based on the marking process, vertex  $v_1$ , with both  $v$  and  $v_2$  as its neighbors, must be marked T. Again this contradicts the assumption that  $v$ 's neighbors are all marked F. CVD



- THEOREM 2: The resulting DS  $C=G'$  is a connected graph.
- PROOF: We prove this theorem by contradiction. Assume  $G'$  is disconnected and  $v$  and  $u$  are two disconnected vertices in  $G'$ . Assume  $\text{dis}_G(v,u) = k+1 > 1$  and  $\{v, v_1, v_2, \dots, v_k, u\}$  is a shortest path between vertices  $v$  and  $u$  in  $G$ . Clearly, all  $v_1, v_2, \dots, v_k$  are distinct and among them there is at least one  $v_i$  such that  $m(v_i) = F$  (otherwise,  $v$  and  $u$  are connected in  $G'$ ). On the other hand, the two adjacent vertices of  $v_i$ ,  $v_{i-1}$  and  $v_{i+1}$ , are not connected in  $G$  (otherwise,  $\{v, \dots, v_i, v_{i+1}, \dots, v_k, u\}$  would be a shorter path). Therefore,  $m(v_i) = T$  based on the marking process.





## **WuLi: Wu and Li protocol**

- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism ("all localized")



- Distr
- conn
- Build
- Appl
- links

Rule 1: for each pair of nodes  $u$  and  $v$  in  $C$  the one with the smallest ID, say  $v$ , can be removed from  $C$  if  $v$  and all its neighbors are covered by  $u$

Rule 2: Assume nodes  $u, v$ , and  $w$  are in  $C$  and assume that  $v$ 's ID is the smallest. If  $u$  and  $w$  are neighbors of  $v$  and are in each other transmission range and if each neighbor of  $v$  is covered by  $u$  and  $w$ , then  $v$  can be removed from  $C$ .

- High degree of parallelism ("all localized")

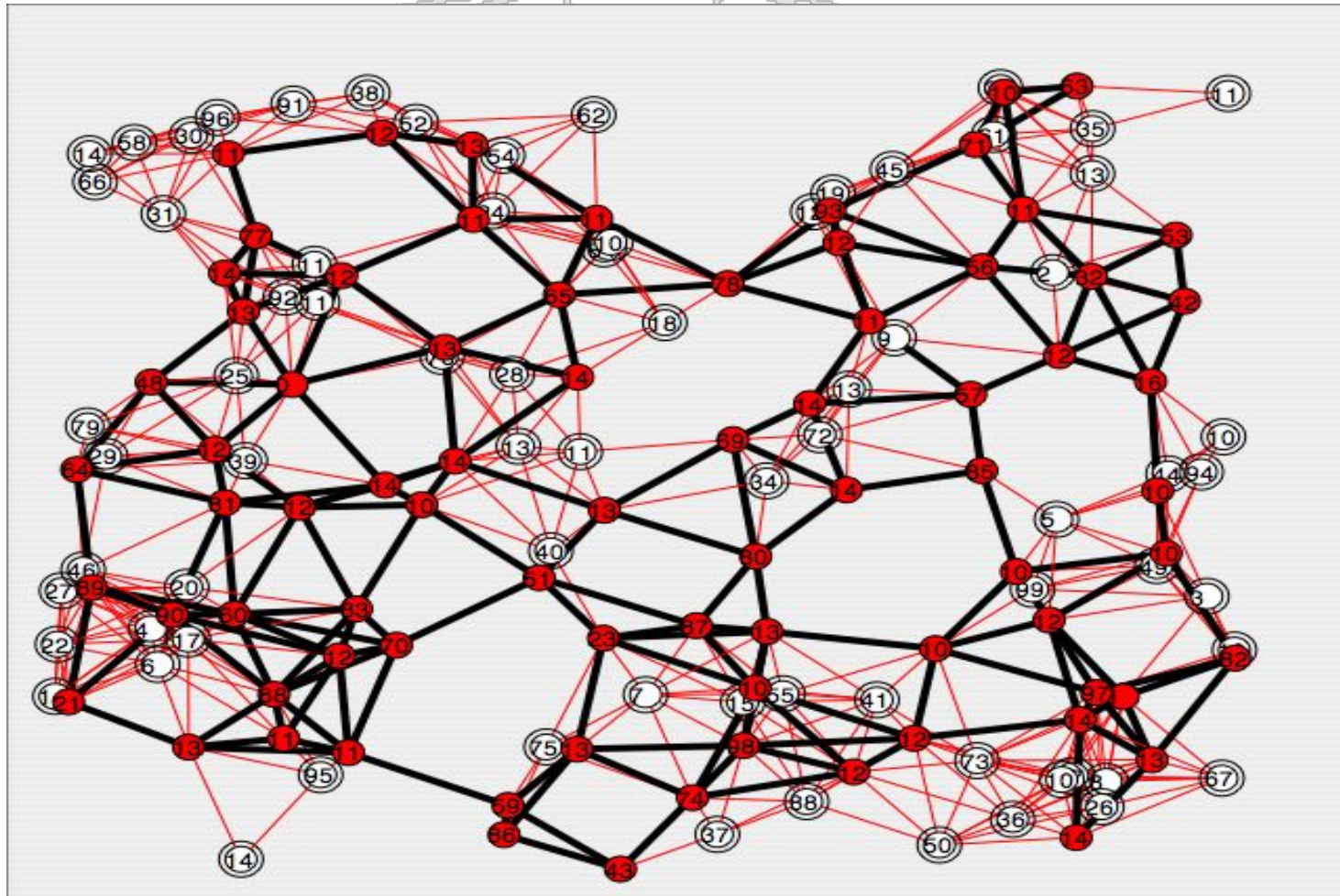


Connectivity and dominance properties are maintained



- Distributed and localized protocols for forming a connected dominating set
- Build a rich connected dominating set
- Applies localized rules for pruning unnecessary nodes/links
- High degree of parallelism (“all localized”)

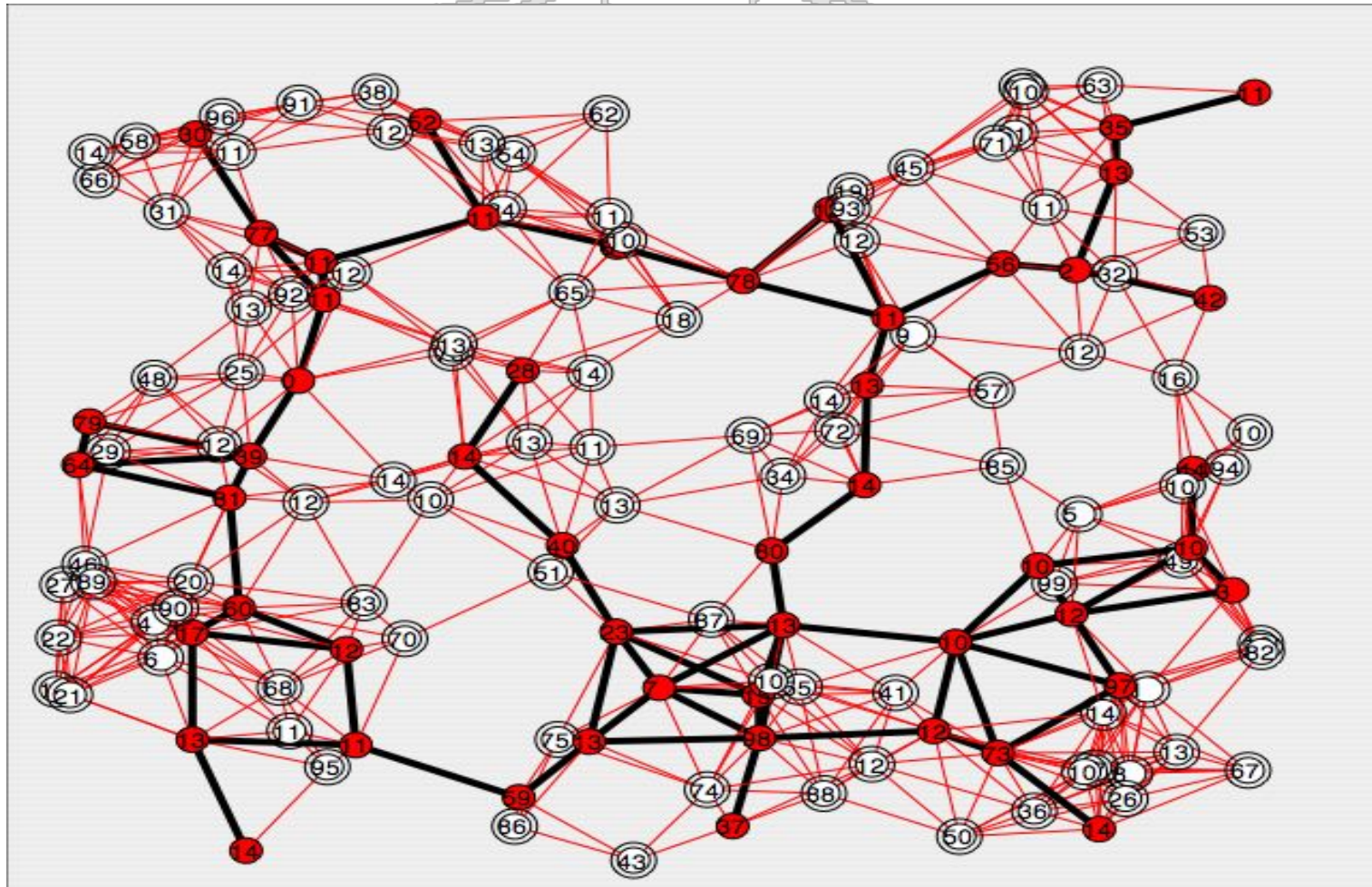
What is needed is, from the neighbors, whether they are in C and their list of neighbors.





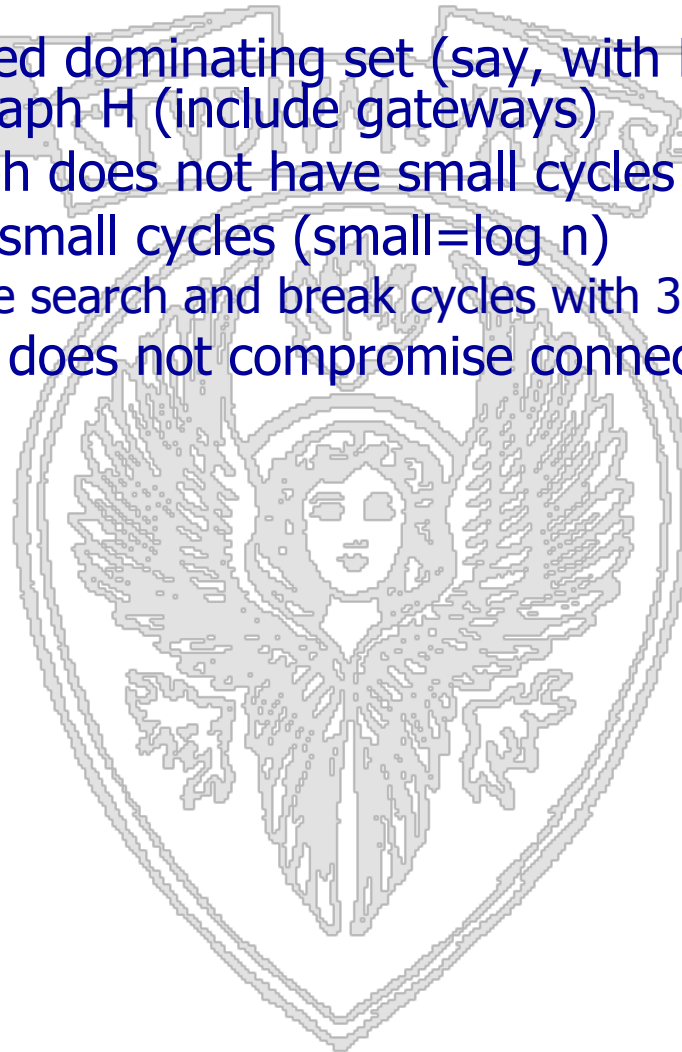


- Two phases
  - Leader election: One node is chosen among all network nodes to be the root of a tree
  - Nodes at different levels of the trees can be chosen to form a connected dominating set
- The “leader election tree” is quite expensive
- Very low degree of parallelism





- Build a connected dominating set (say, with DCA) and consider its spanned sub-graph  $H$  (include gateways)
- Erdős: If a graph does not have small cycles then it is sparse
- Find and break small cycles (small= $\log n$ )
  - In practice we search and break cycles with 3 and 4 links
- Breaking cycles does not compromise connectivity





- Metrics (all averages)
  1. Protocol duration
  2. Operation overhead (in bytes)
  3. Energy consumption (per node)
  4. Backbone size
  5. Route length
  6. Backbone robustness (node deaths for disconnections)



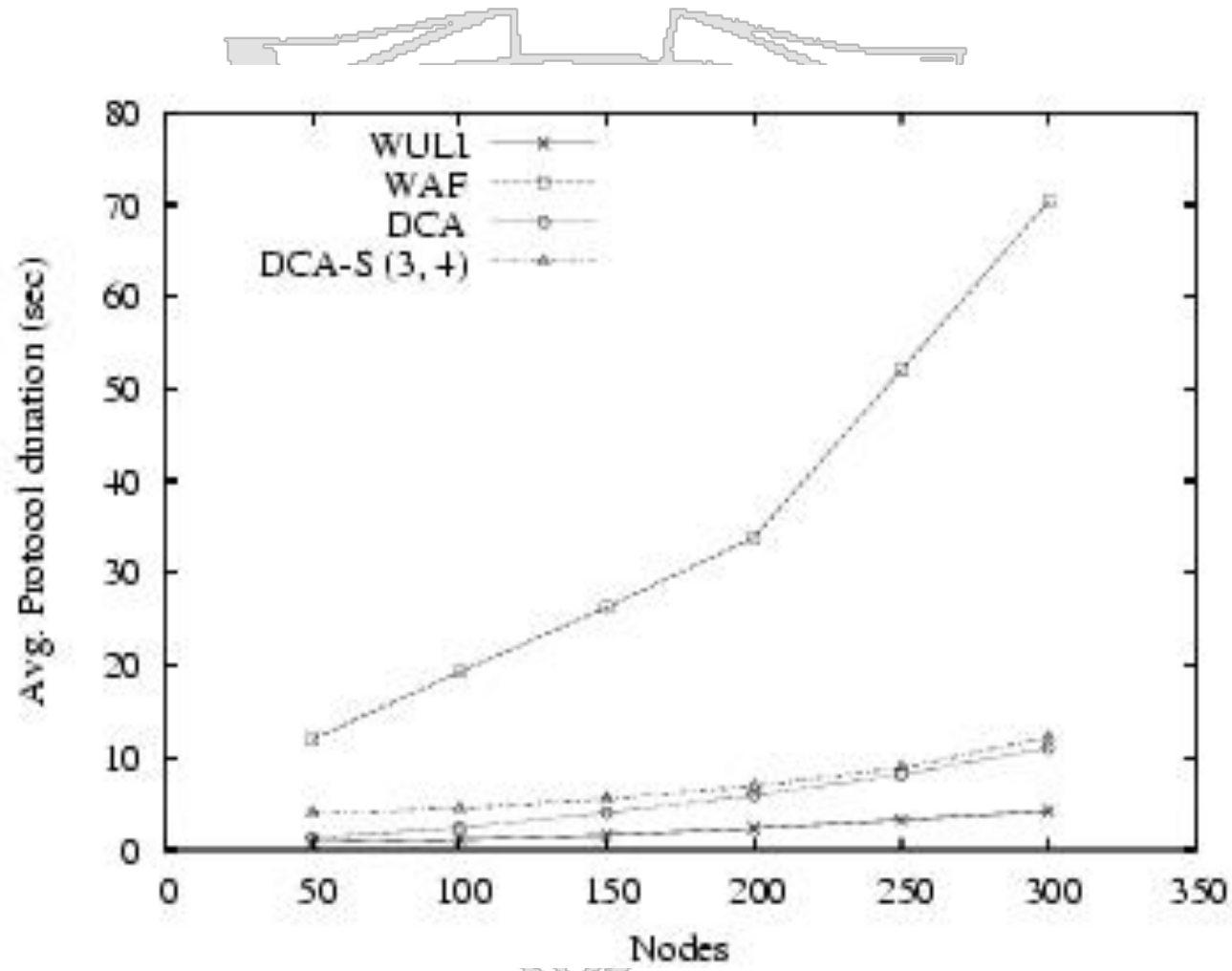
- Parameters of ns2-based simulations
  - Nodes:  $\leq 300$ , IST EYES prototype
    - ✓ Tx range: 30m
    - ✓ Initial (residual) energy: 1J
    - ✓ Tx, Rx, idle power: 24, 14.4, 0.015 (mW)
  - Area: 200 x 200m
  - Six scenarios with increasing densities (avg. degrees: 3.5 to 20)



- WuLi is fastest
  - Simple operation; parallelism
- DCA: Reasonably fast
  - Possible dependencies and gateway selection
- DCA-S: As DCA
  - The sparsification phase is executed by fewer nodes and requires little info exchange
- WAF: Slower
  - Non-trivial leader election

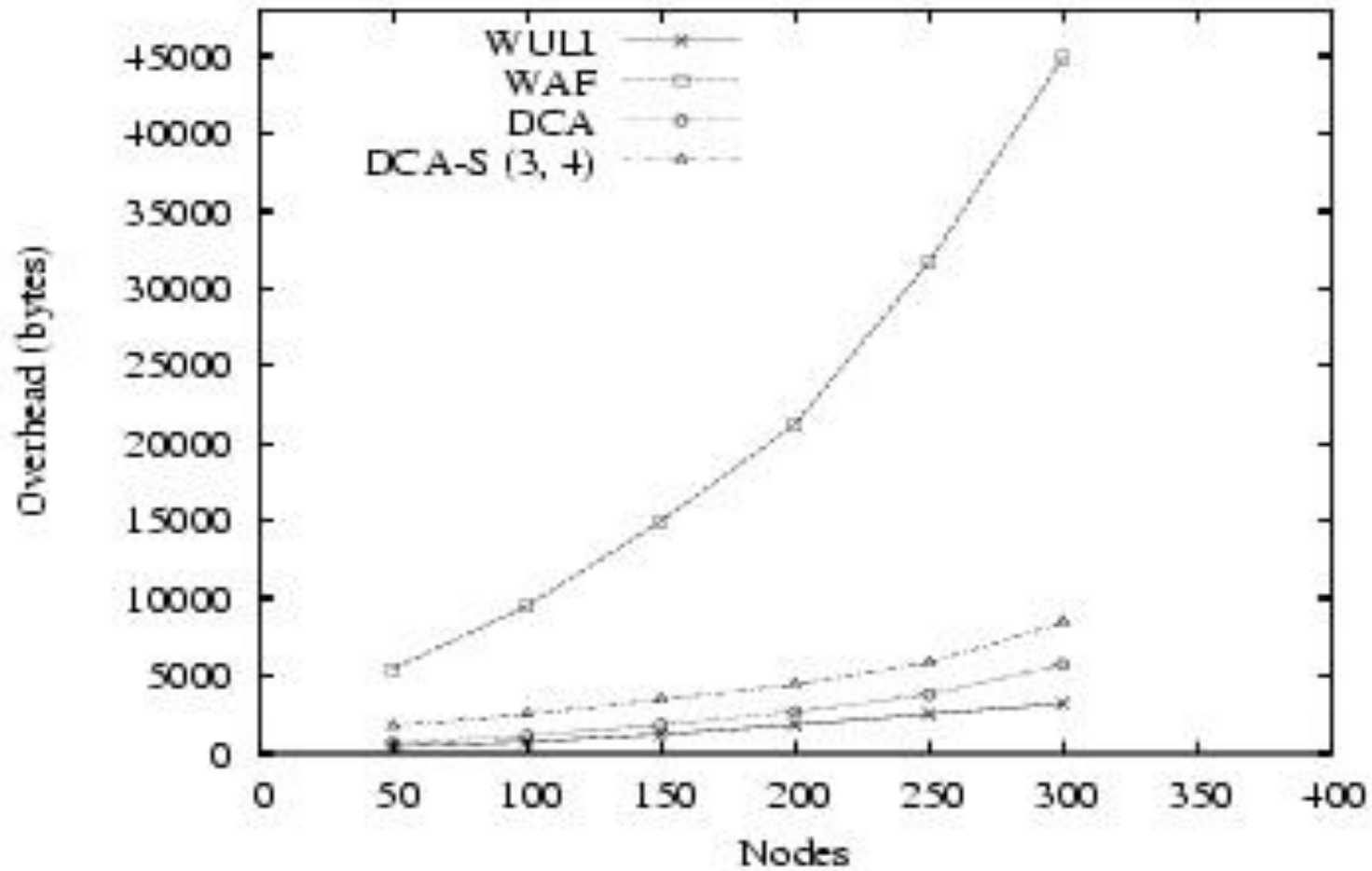


# Protocol Duration, 2





# Protocol Overhead







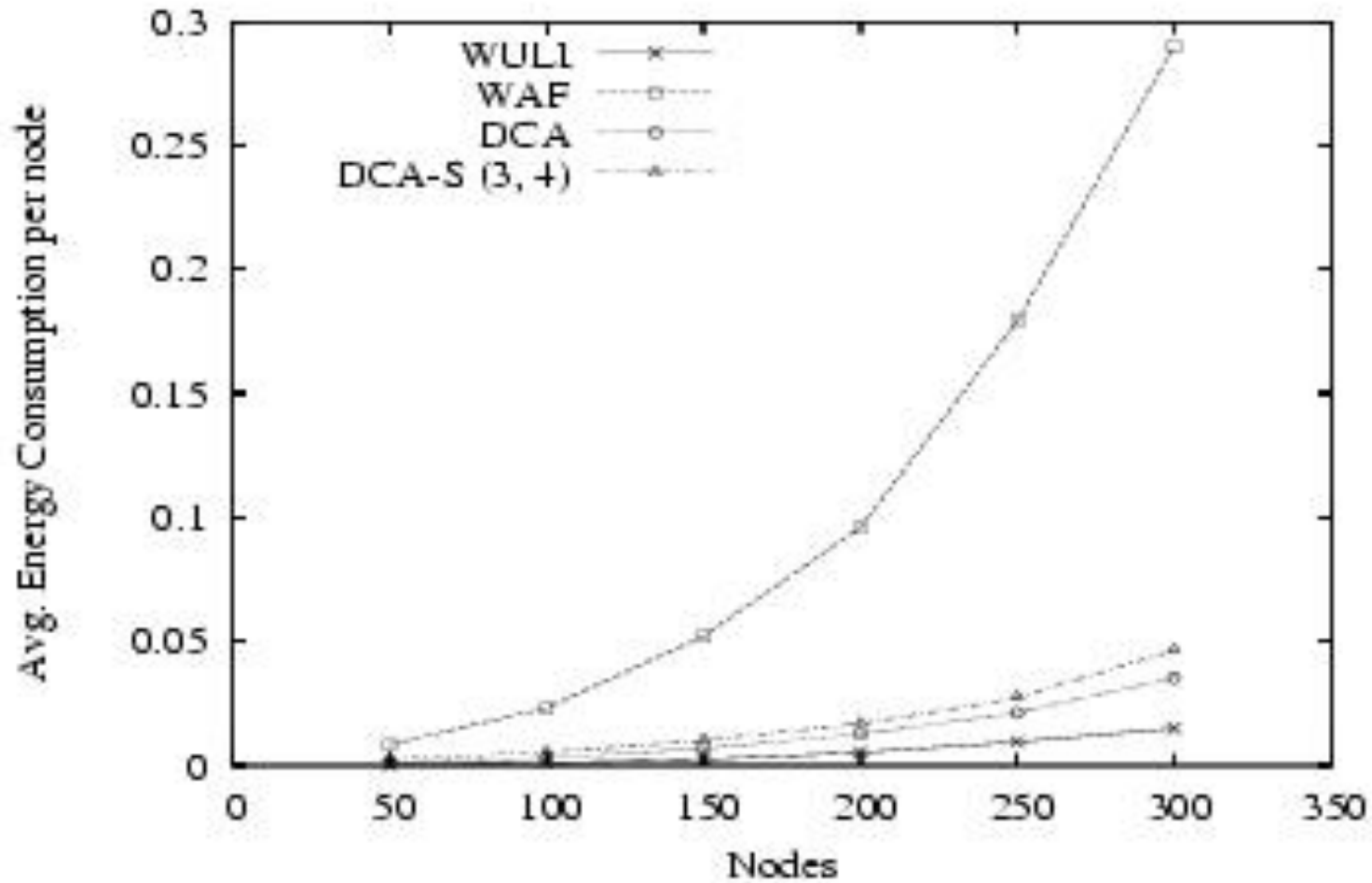
- Average number of protocol bytes per node
- WuLi: Best performing
  - Simple list exchange
- DCA(-S): Almost twice as much
  - Bit more info needed (weight, IDs, ...)
- WAF
  - Leader election complexity



- Important metric per backbone set up and maintenance
- Similar to overhead results
- WuLi and DCA perform quite well
- DCA-S performs similarly: No difference in breaking cycles with 3 or 4 links
- WAF: High consumption due to first phase



# Energy Consumption, 2

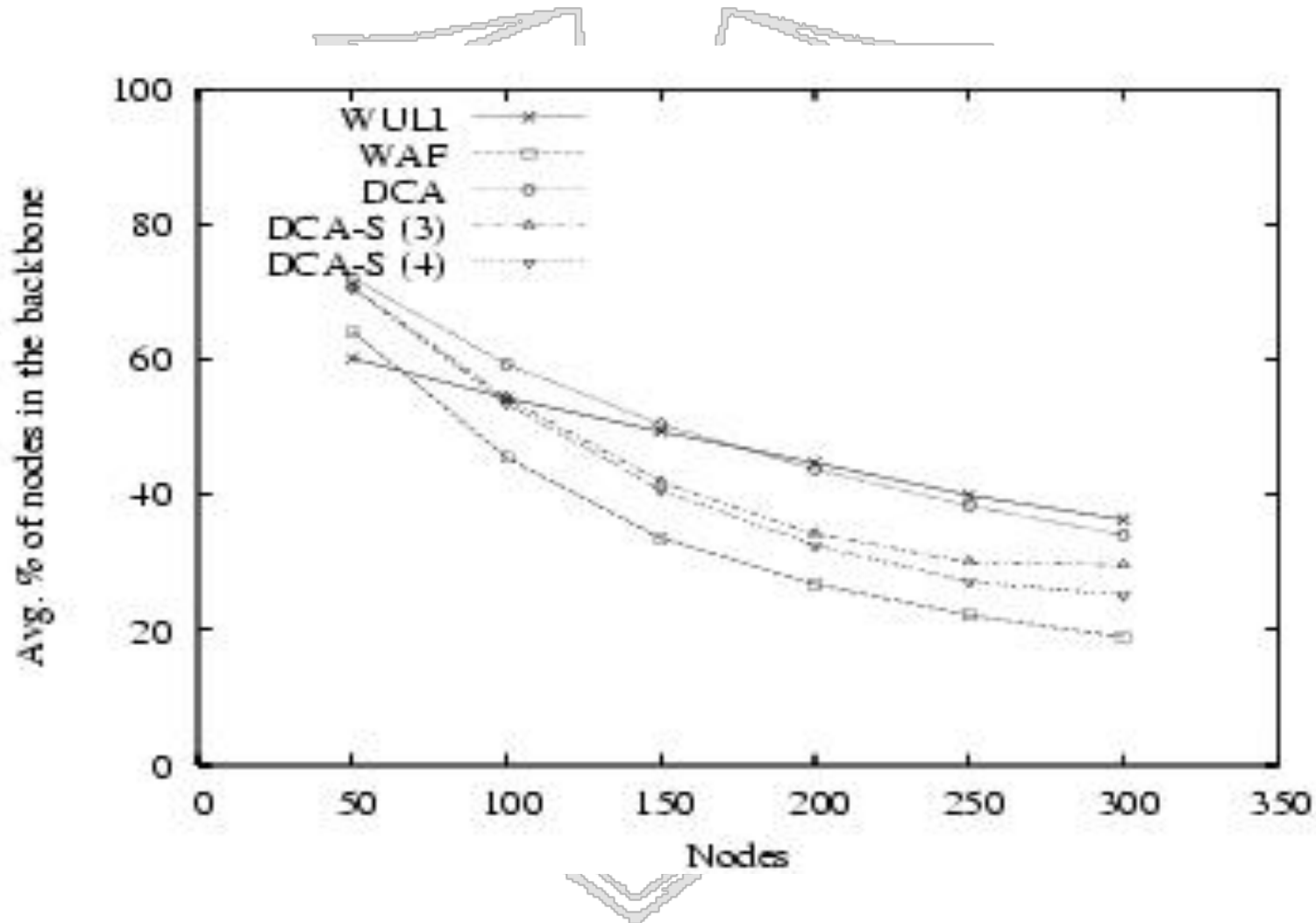




- Important metric: Routing info and awake/asleep cycles
  - Small backbone + role rotation: key for WSNs
- Decrease with n increasing (bigger clusters)
- WAF: “Slimmer” backbone (tree like)
- $DCA-S, 4 < DCA-S, 3 < DCA < WuLi$



# Backbone Size, 2

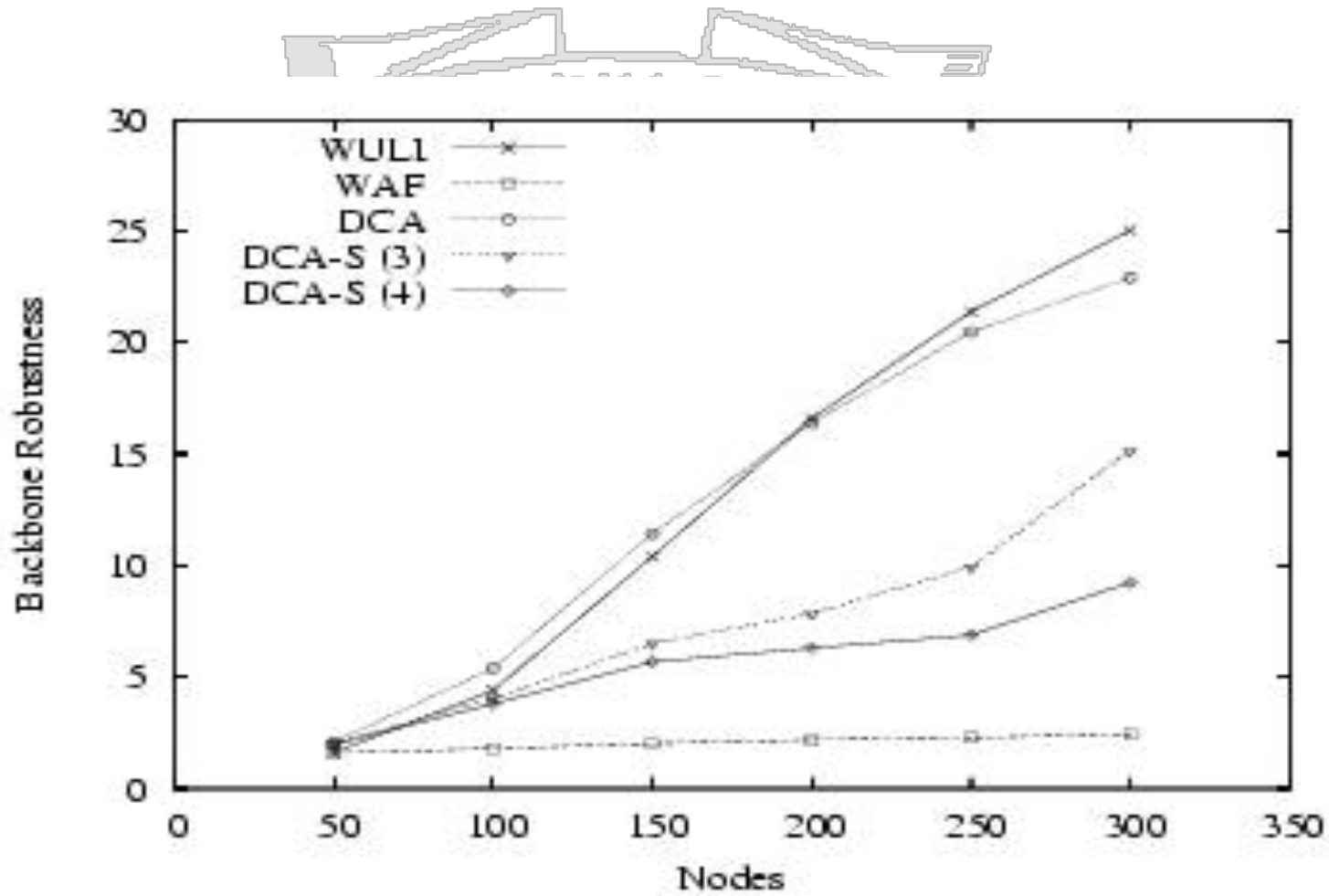




- Number of nodes needed to disconnect the backbone
- Useful for planning backbone re-orgs
- Increases with network density
- WuLi and DCA: More robust
  - Resilient to up to 25 "death" when  $n = 300$
- WAF: Quite a disaster (tree-like topologies)
- DCA-S: In the middle



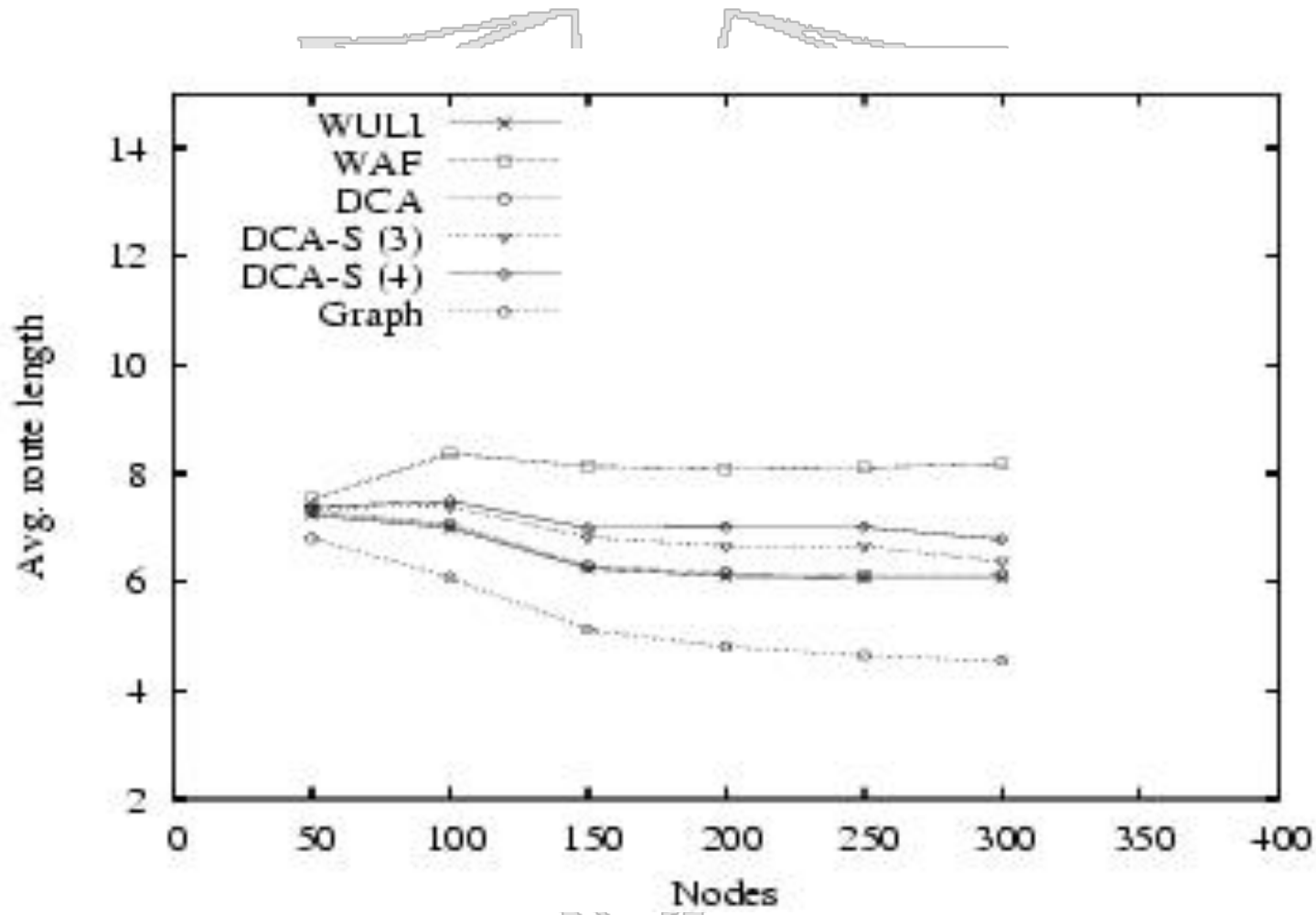
# Backbone Robustness, 2





- Flat topology (“visibility graph”) as a base
- Expected increase: Hierarchy routes are longer
- DCA & WuLi: 7 to 34.7% longer routes
- DCA-S: Up to 9% more than DCA
- WAF: Up to 33.4% longer than DCA







- Hierarchical organization is effective for prolonging network lifetime
- Four protocols for backbone formation:
  - DCA, WuLi, WAF and DCA-S
- Nice theoretical features → hard to implement
- Simple solutions (WuLi, DCA): Good starting point for efficient implementations
- DCA-S: "Slimmer" backbone at a reasonable cost



- DMAC is for clustering set up AND maintenance
- Nodes can move during the clustering
- Each node reacts to
  - Reception of a message
  - Presence of a new link
  - Link failure
- Same assumptions of DCA, plus knowledge of neighbors' roles (no role = ordinary role)



- DMAC is for clustering set up AND maintenance

When an ordinary node "dies" we simply need to keep track  
When a gw node "dies" we have to select a new gw to  
interconnect two adjacent cluster

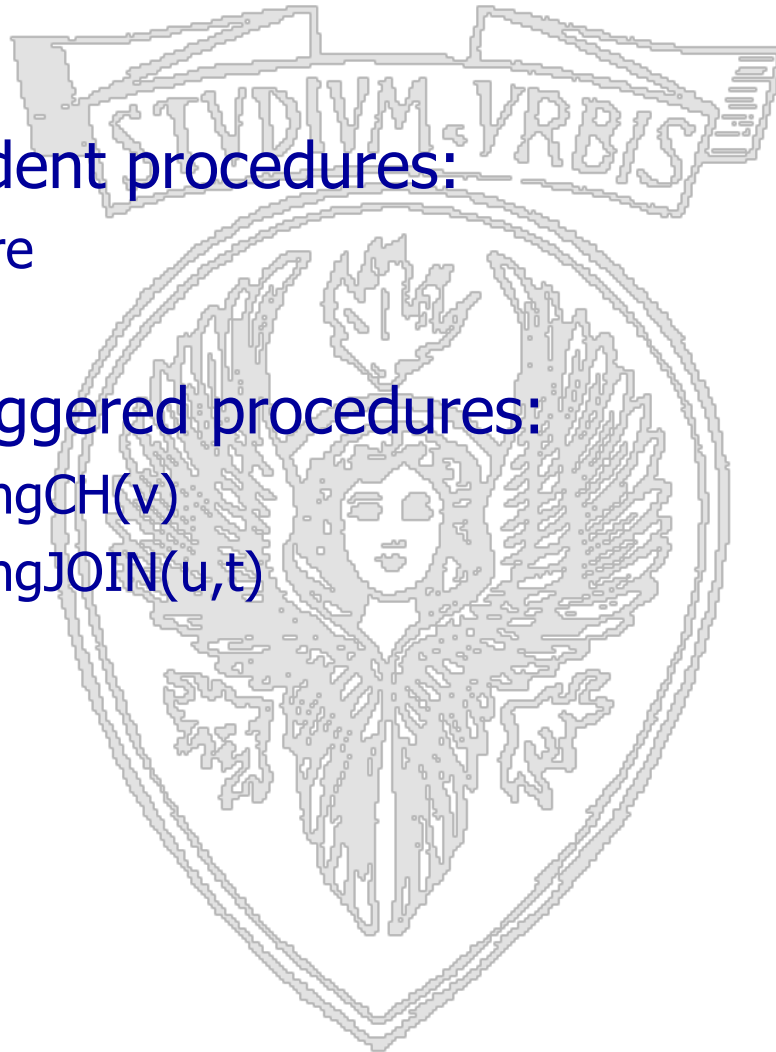
When a CH node "dies" an ordinary node in its cluster looks  
for other CH neighbors, selecting the one with biggest  
weight, if one or more are available.

Otherwise, if there are no CH neighbors and the ordinary  
node has the biggest weight it becomes CH.

Nodes removal/arrival, changes in link quality may result in  
need to maintain the backbone (which significantly  
increases the control overhead of clustering).

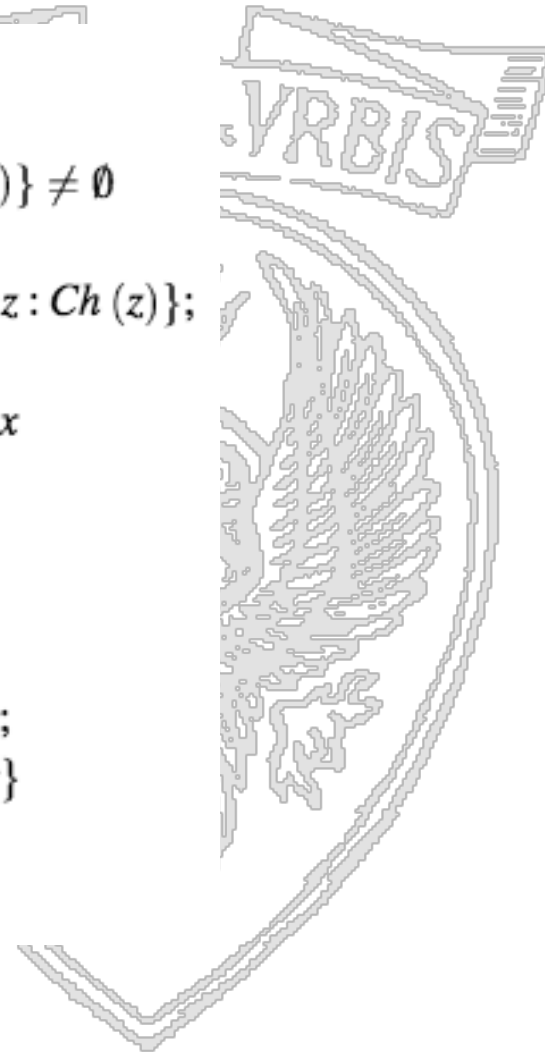


- INIT
- Link-dependent procedures:
  - Link\_Failure
  - New\_Link
- Message-triggered procedures:
  - OnReceivingCH(v)
  - OnReceivingJOIN(u,t)





```
PROCEDURE Init;  
begin  
  if  $\{z \in \Gamma(v) : w_z > w_v \wedge Ch(z)\} \neq \emptyset$   
    then begin  
       $x := \max_{w_z > w_v} \{z : Ch(z)\};$   
      send JOIN( $v, x$ );  
       $Clusterhead := x$   
    end  
  else begin  
    send CH( $v$ );  
     $Ch(v) := \mathbf{true};$   
     $Clusterhead := v;$   
     $Cluster(v) := \{v\}$   
  end  
end;
```





```
PROCEDURE Link_failure (u);  
begin  
  if Ch (v) and (u  $\in$  Cluster (v))  
    then Cluster (v) := Cluster (v) \ {u}  
    else if Clusterhead = u then  
      if {z  $\in$   $\Gamma$ (v) : wz > wv  $\wedge$  Ch (z)}  $\neq$   $\emptyset$   
        then begin  
          x :=  $\max_{w_z > w_v}$  {z : Ch (z)};  
          send JOIN(v,x);  
          Clusterhead := x  
        end  
      else begin  
        send CH(v);  
        Ch (v) := true;  
        Clusterhead := v;  
        Cluster (v) := {v}  
      end  
end;
```





```
PROCEDURE New_link(u);
```

```
begin
```

```
  if Ch(u) then
```

```
    if ( $w_u > w_{Clusterhead}$ )
```

```
      then begin
```

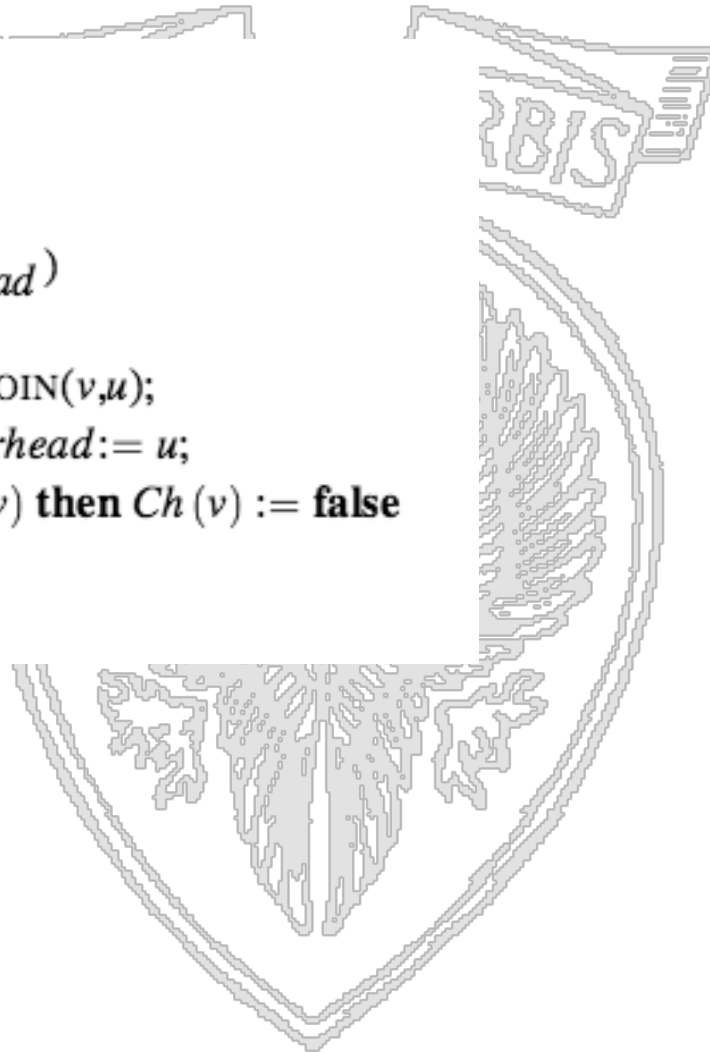
```
        send JOIN(v,u);
```

```
        Clusterhead := u;
```

```
        if Ch(v) then Ch(v) := false
```

```
      end
```

```
end;
```







On receiving  $CH(u)$ ;

**begin**

**if** ( $w_u > w_{Clusterhead}$ ) **then begin**

**send**  $JOIN(v,u)$ ;

$Clusterhead := u$ ;

**if**  $Ch(v)$  **then**  $Ch(v) := false$

**end**

**end;**





```
On receiving JOIN( $u, z$ );
begin
  if  $Ch(v)$ 
    then if  $z = v$  then  $Cluster(v) := Cluster(v) \cup \{u\}$ 
         else if  $u \in Cluster(v)$  then  $Cluster(v) := Cluster(v) \setminus \{u\}$ 
    else if  $Clusterhead = u$  then
      if  $\{z \in \Gamma(v) : w_z > w_v \wedge Ch(z)\} \neq \emptyset$ 
        then begin
           $x := \max_{w_z > w_v} \{z : Ch(z)\}$ ;
          send JOIN( $v, x$ );
           $Clusterhead := x$ 
        end
      else begin
        send CH( $v$ );
         $Ch(v) := \mathbf{true}$ ;
         $Clusterhead := v$ ;
         $Cluster(v) := \{v\}$ 
      end
    end
end;
```

