SSS 2014 -- Paderborn, Germany

A Self-Stabilizing Algorithm for Edge Monitoring

Brahim NEGGAZI [1], Mohammed HADDAD [1] , Volker TURAU [2], Hamamache KHEDDOUCI [1]

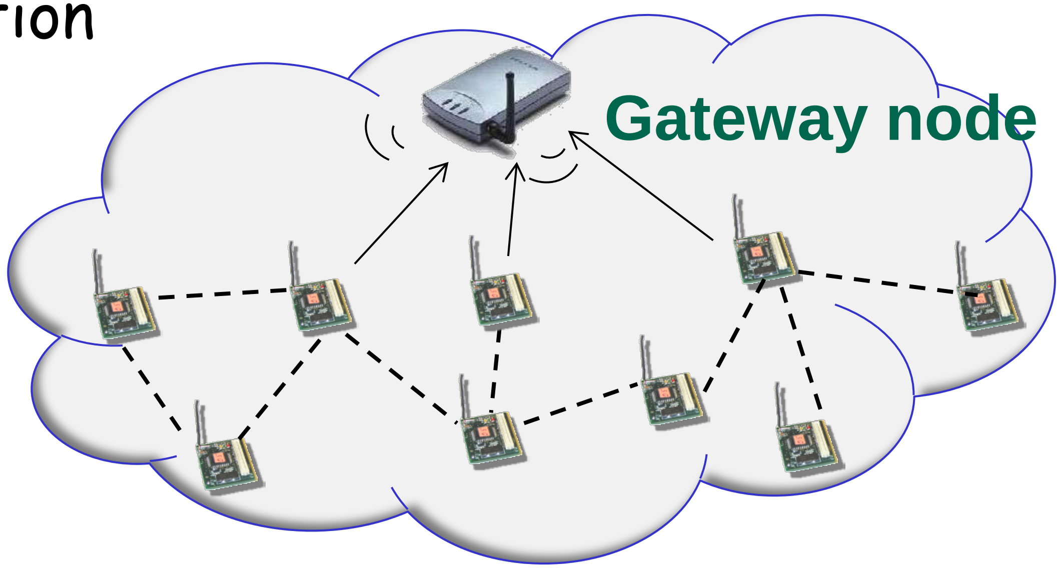
[1] **Laboratoire d'InfoRmatique en Image et Systèmes d'information**
LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon
43, boulevard du 11 novembre 1918 — F-69622 Villeurbanne Cedex

[2] **Hamburg University of Technology, Institute of Telematics,**
Schwarzenbergstraße 95, 21073 Hamburg, Germany

Challenges in Running a WSN

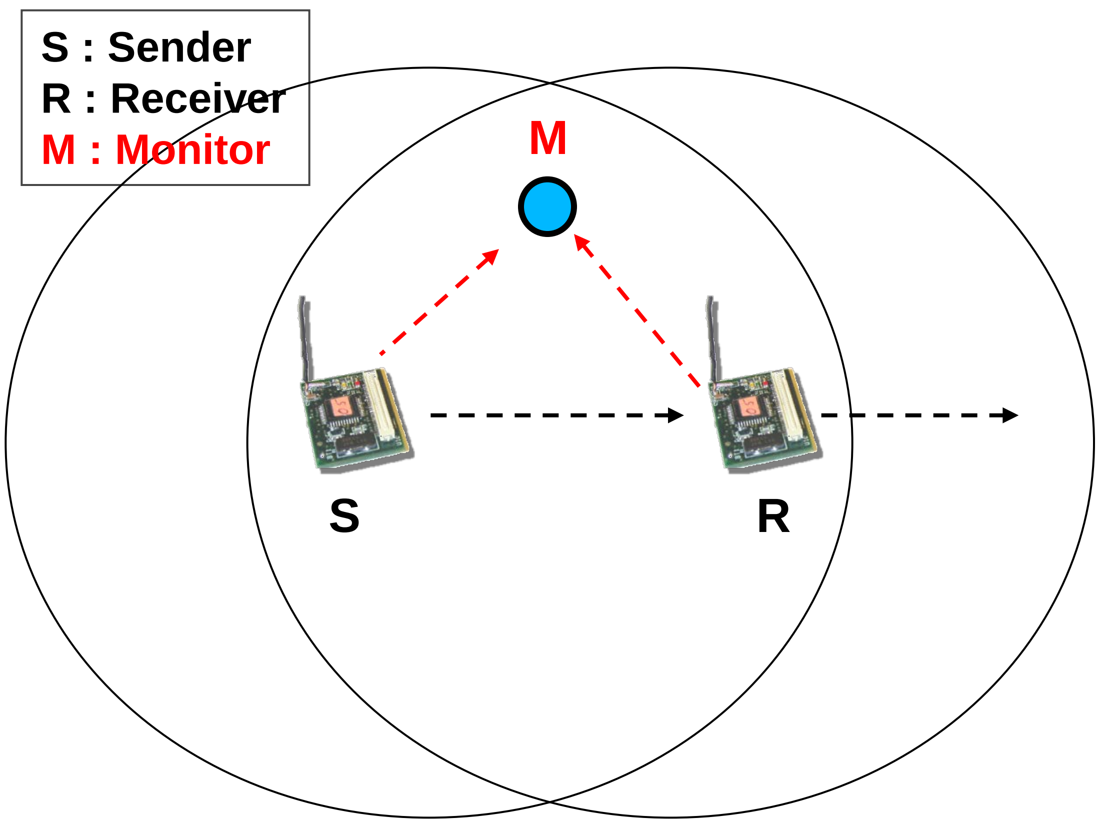
Vulnerability of WSN due to :

- Wireless communication
- Implementation errors
- Hardware faults
- Unattended operation



Local monitoring

One mechanisms to implement a watchdog concept is "local monitoring" Marti et al. [Marti00]

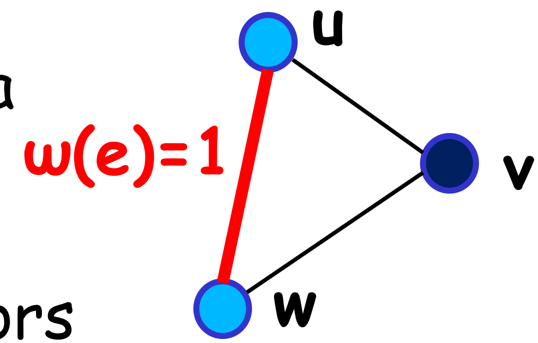


Node M monitors link from S to R by monitoring traffic that R receives from S and forwards out

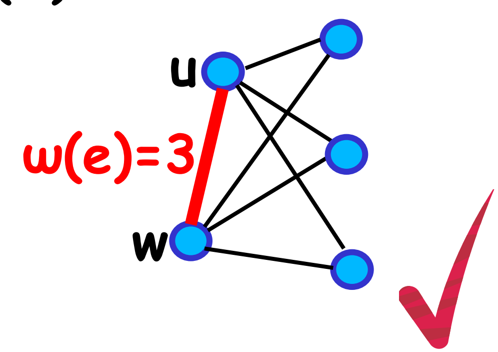
By analyzing traffic flows, monitoring nodes are able to detect behavior deviating from the specification caused by an implementation error or a fault, such as **delaying, dropping, modifying, or producing faulty packets**

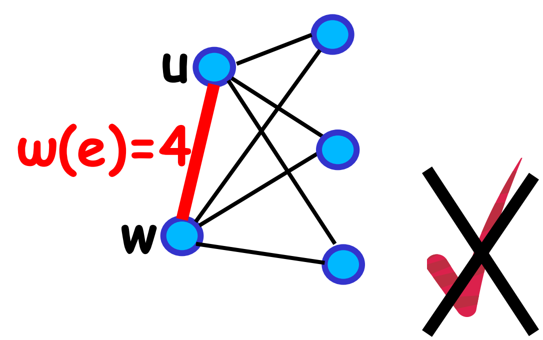
Edge monitoring

- Node *v* can monitor edge *e* = (u,w) if v is a neighbor of u and w
- Edges have monitoring constraints ω specifying the number of required monitors

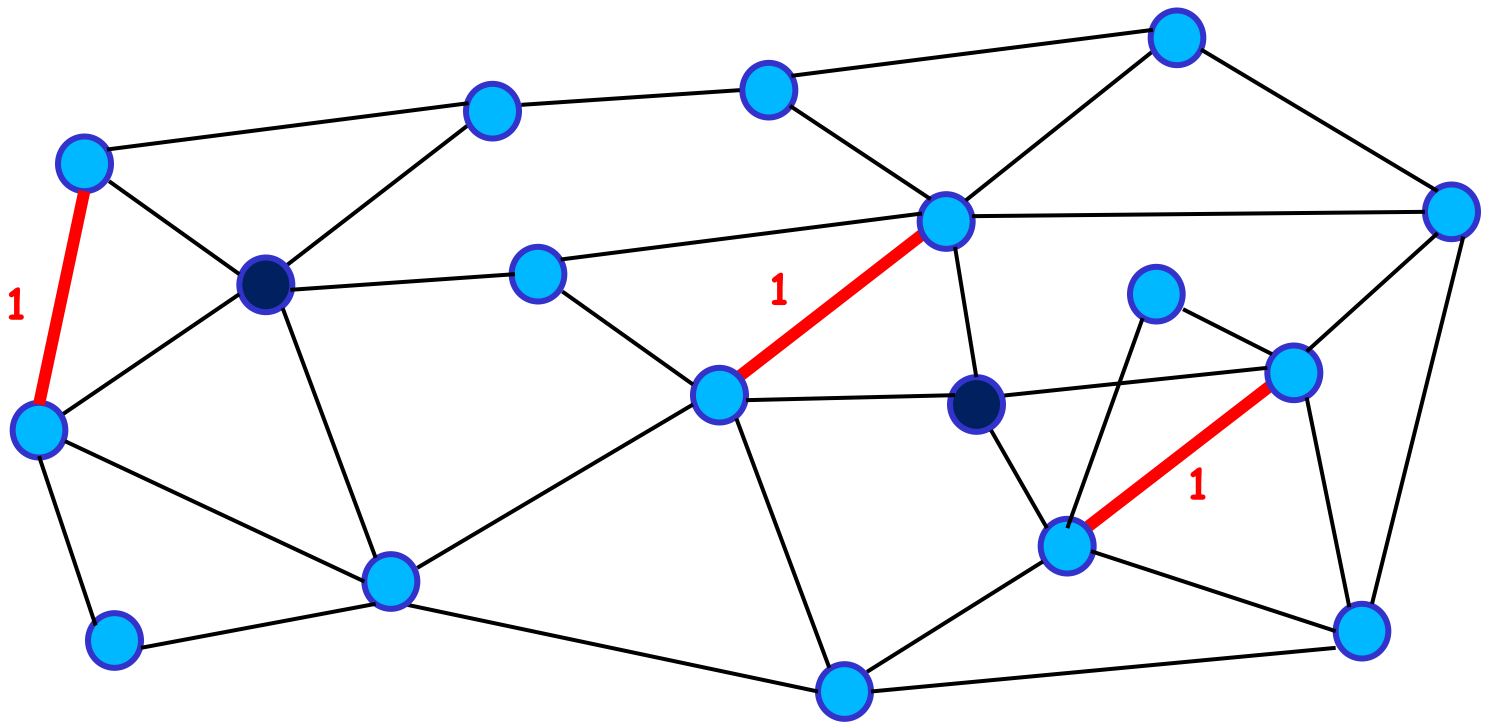


- Assumption: For each e = ‹u,w› ∈ E then $|N(u) \cap N(w)| \geq \omega(e)$





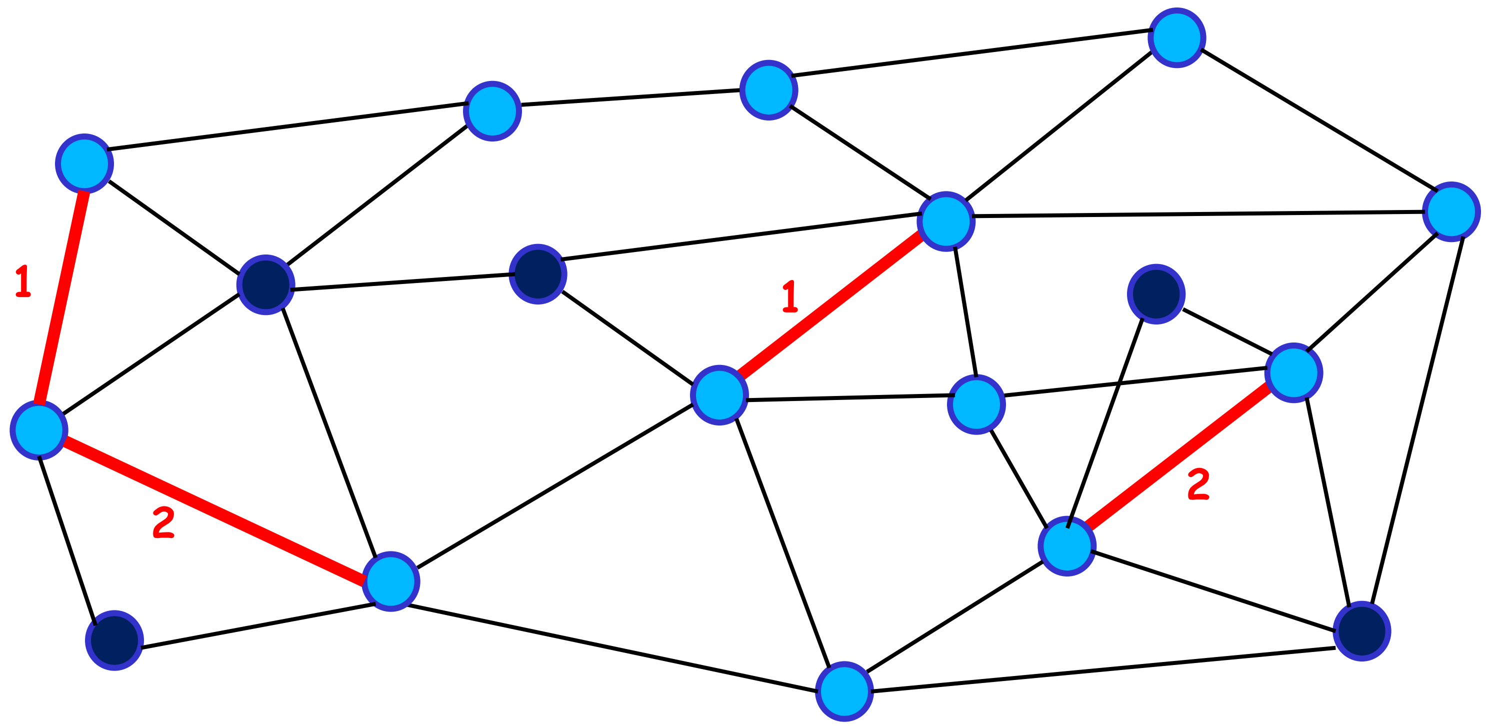
Example



red :: edges to be monitored

blue :: monitors

Example

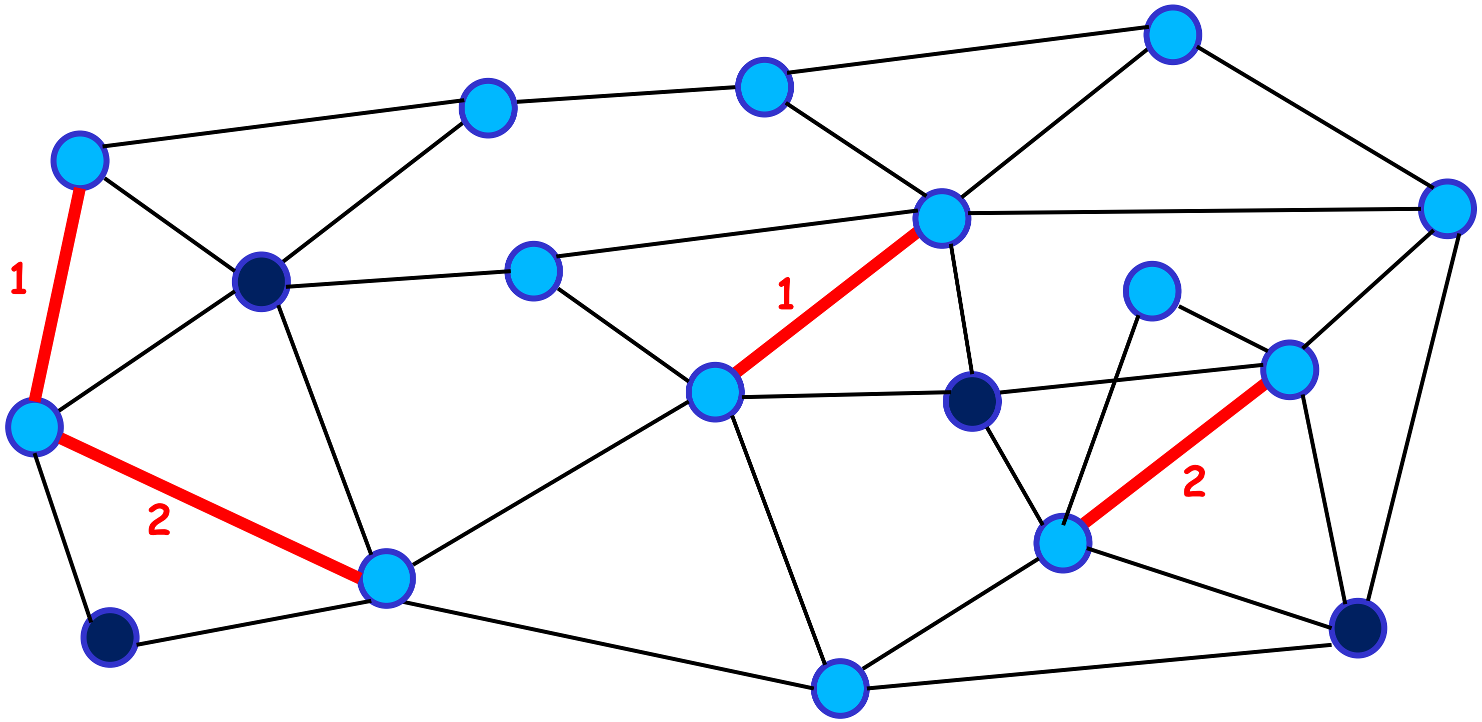


red :: edges to be monitored

blue :: monitors

5 monitors!

Example



red :: edges to be monitored

blue :: monitors

Only 4 monitors!

Edge monitoring

- Finding a **minimum set** of edge monitoring nodes is NP-hard
- Goal: Minimal edge monitoring sets
  - i.e. a subset D of nodes s.t. for each edge $e \in E$ there are at least $\omega(e)$ nodes in D that can monitor e and no proper subset of D satisfies this property
- Distributed algorithms with provable approximation ratios are known [Dong08]
- What about self-stabilizing algorithms?

Previous Work

- Hauck proposed the first self-stabilizing algorithm for minimal edge monitoring problem [Hauck12]
- $O(n^2m)$ moves under unfair distributed scheduler

Contribution

New self-stabilizing algorithm for computing minimal edge monitoring set: SEMS

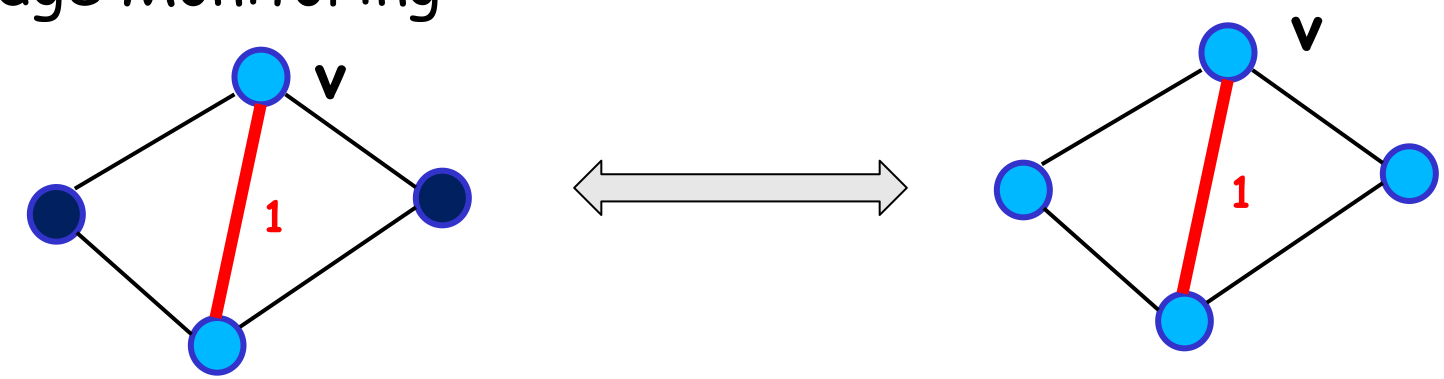
Algorithm SEMS operates under the unfair distributed scheduler and converges in $O(\Delta^2 m)$ moves

Algorithm

- Self-Stabilization = Closure + Convergence
- Example: Maximal independent set
    - Nodes have state IN or OUT
    - Two simple rules
    - Livelocks under distributed scheduler
- Solution:
    - Mutual exclusion
    - Often to restrictive
    - Nodes do not know next move of a neighbor
    - Introduce new state indicating move (WAIT)
    - Symmetry breaking with ids

Algorithm

- Edge Monitoring



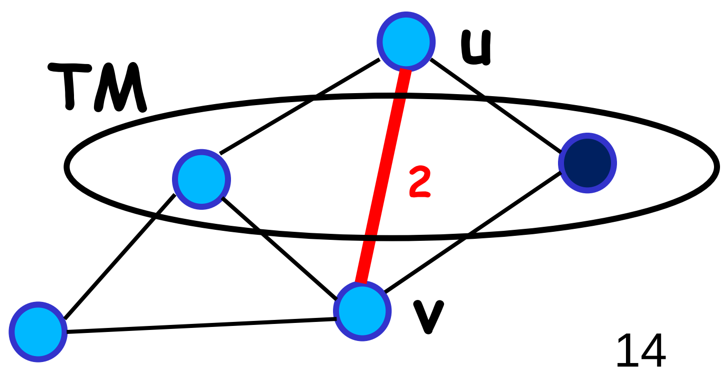
- Problem: Critical nodes are not neighbors
- Solution: Intermediate nodes give permission to a single neighbor to make a move
- Problem: Deadlocks may arise
- Solution: Enforce ordering (based on ids)

SEMS

- Each node maintains a variable state with range
  {IN, OUT,WAIT}
- Nodes with state IN are **monitors**
- State WAIT is an intermediate state from IN to OUT required for symmetry breaking

SEMS

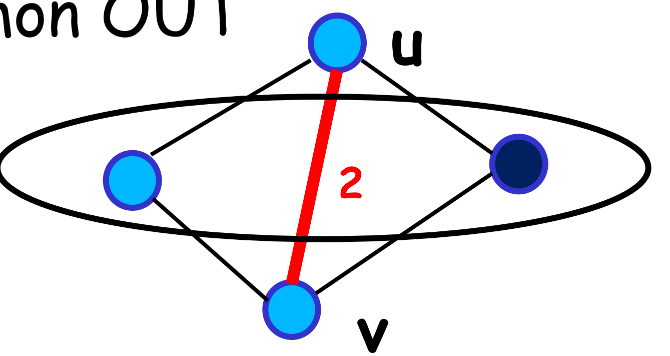
- Monitors of an edge are administered by end node of edge with smaller identifier
- Neighbors of v that do or could monitor an edge adjacent to v are called **target monitors**
- A node maintains for each edge it is responsible for a set of target monitors (TM)



SEMS

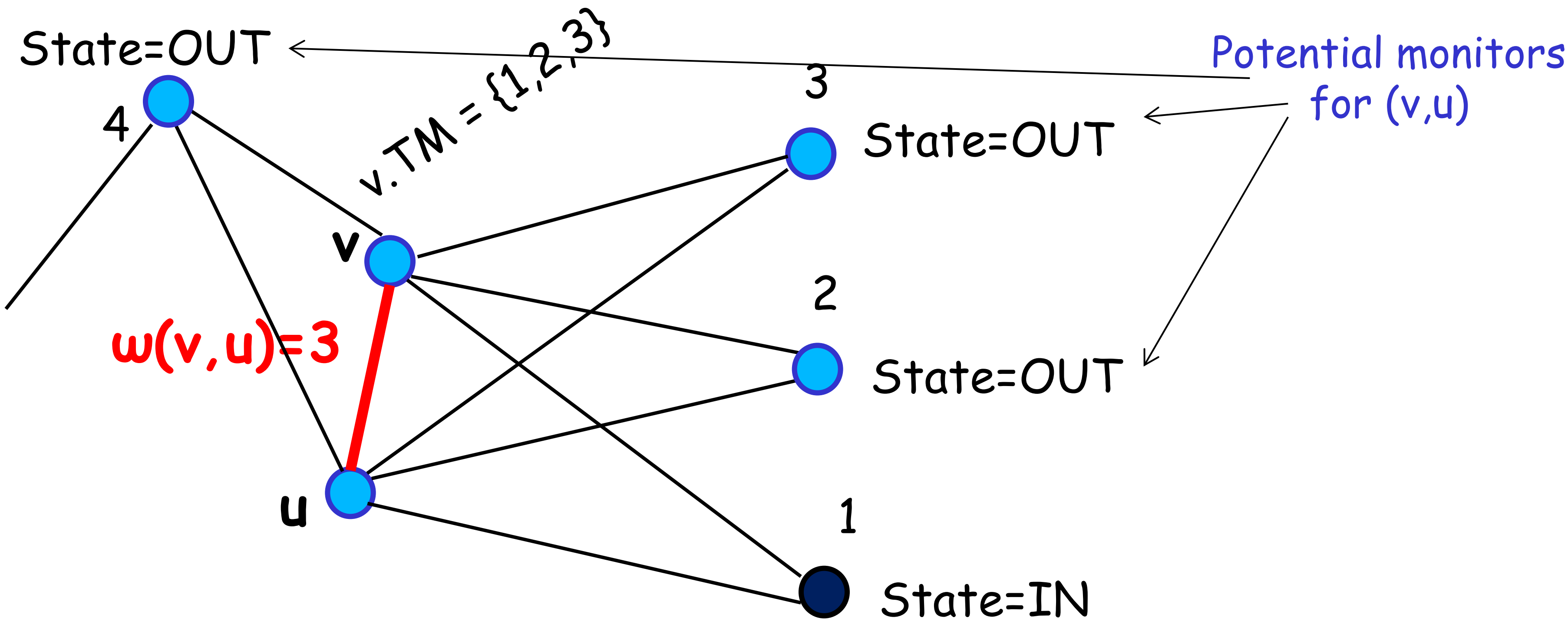
Rule to maintain TM of edge e = (v,u)

1. If number of common neighbors of v and u with state IN or WAIT is larger than ω(e) then let TM = ∅
2. Otherwise TM consists of common neighbors of v and u with state IN or WAIT. If this number is less than ω(e) then smallest common OUT neighbors are added

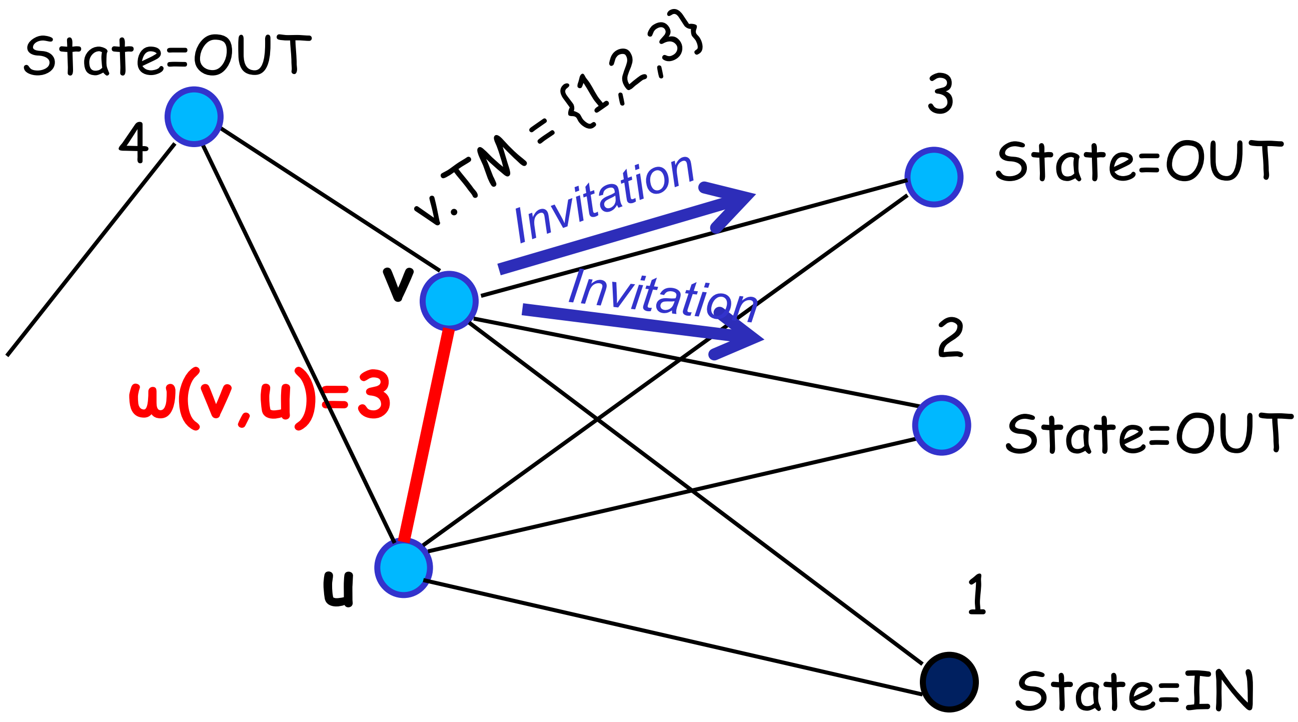


SEMS

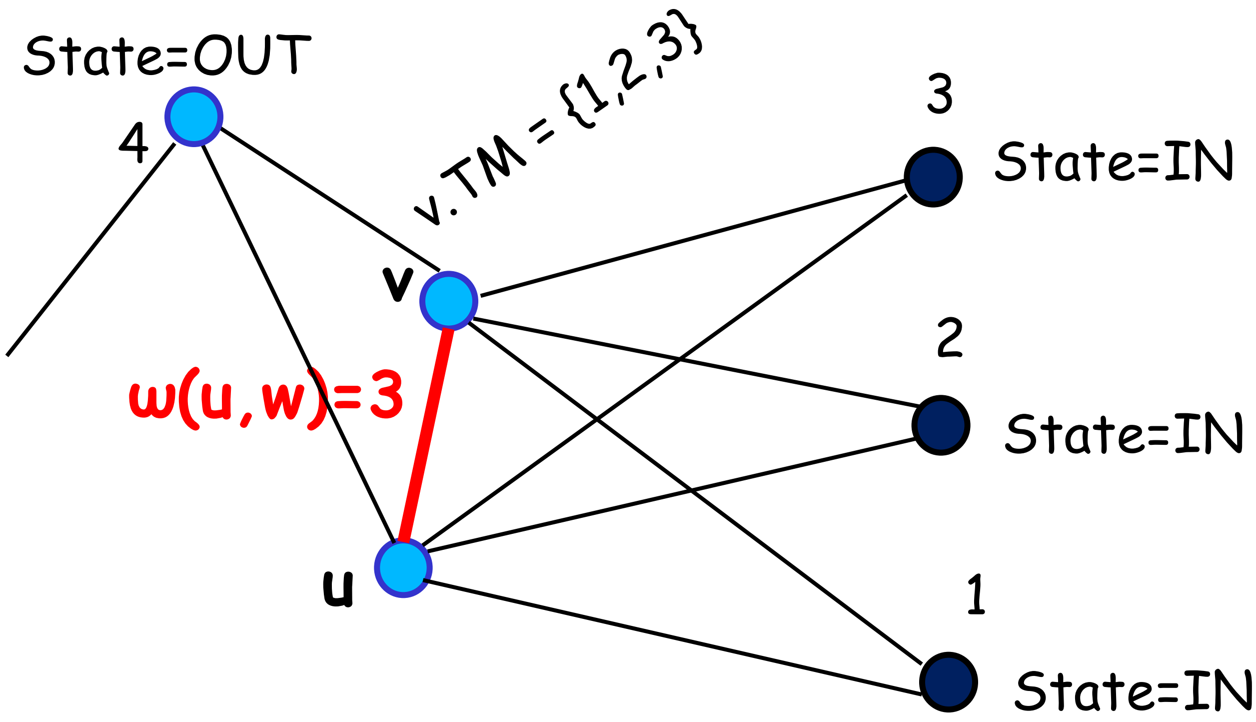
If an OUT node discovers that it is contained in TM of a neighbor it regards this as an invitation to change to IN



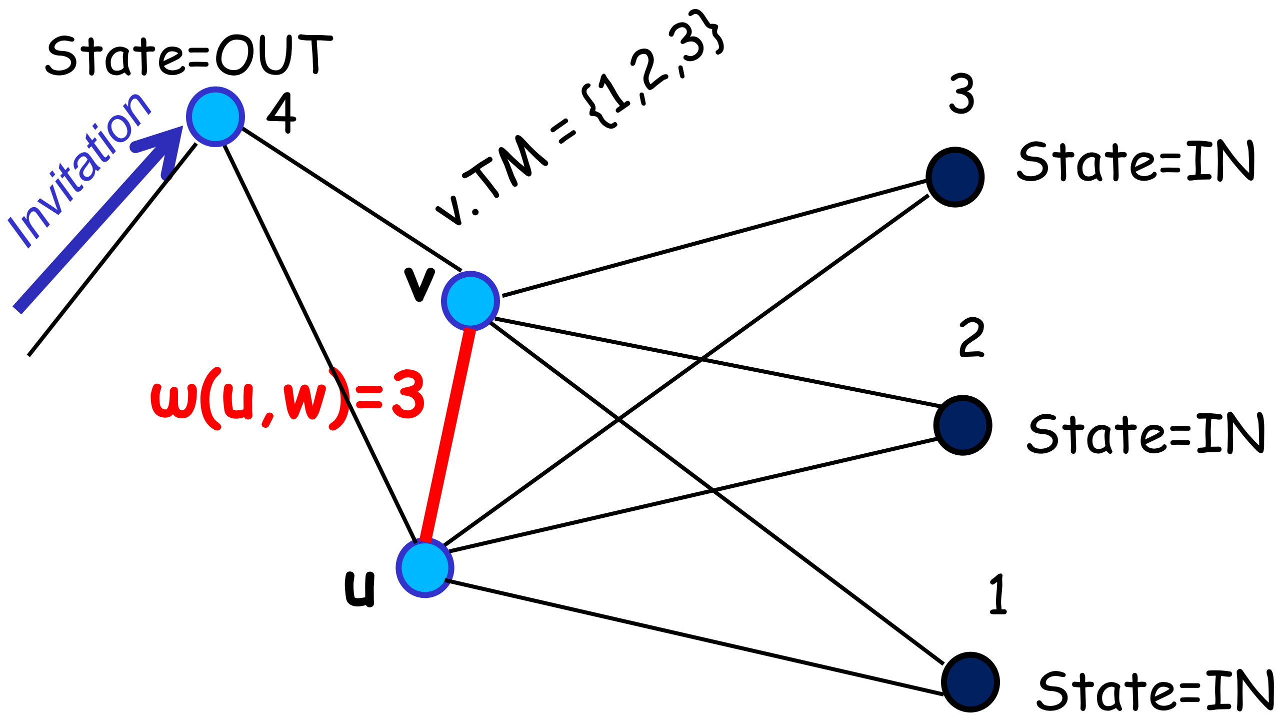
SEMS


State=OUT
4
v.TM = {1,2,3}
Invitation
Invitation
v
ω(v,u)=3
u
3
State=OUT
2
State=OUT
1
State=IN

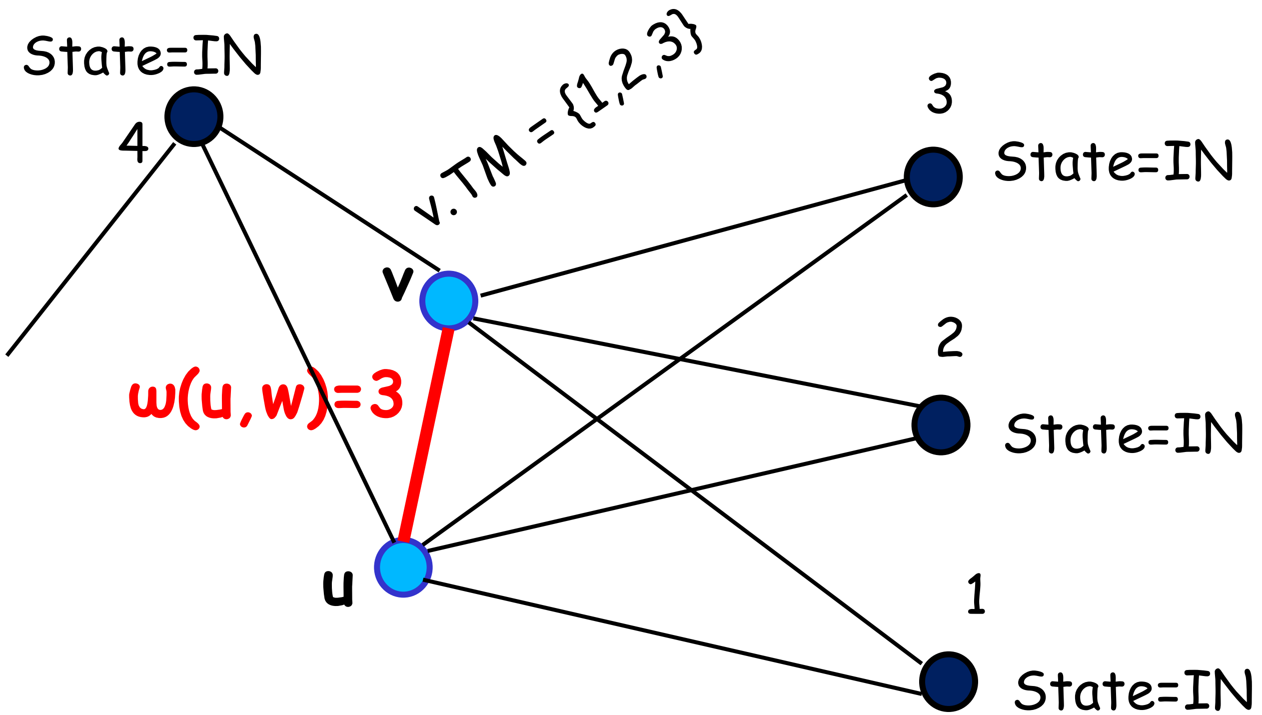
SEMS



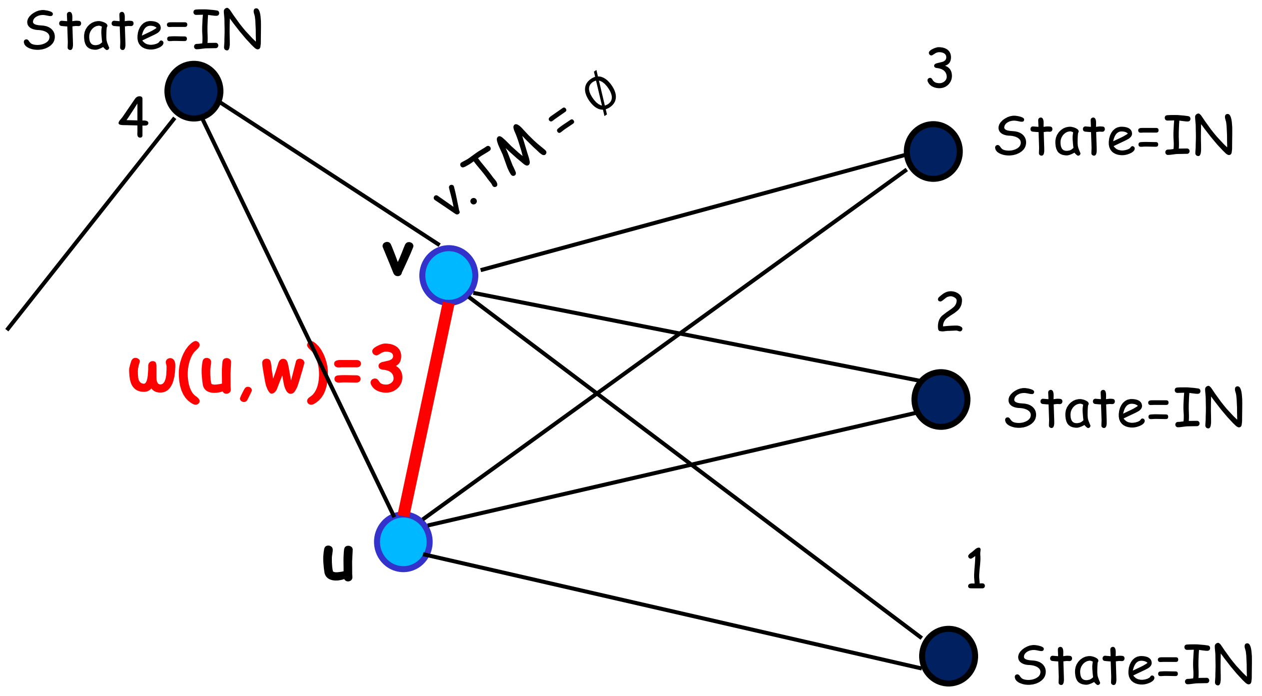
SEMS


State=OUT
4
Invitation
v.TM = {1,2,3}
v
ω(u,w)=3
u
3
State=IN
2
State=IN
1
State=IN

SEMS



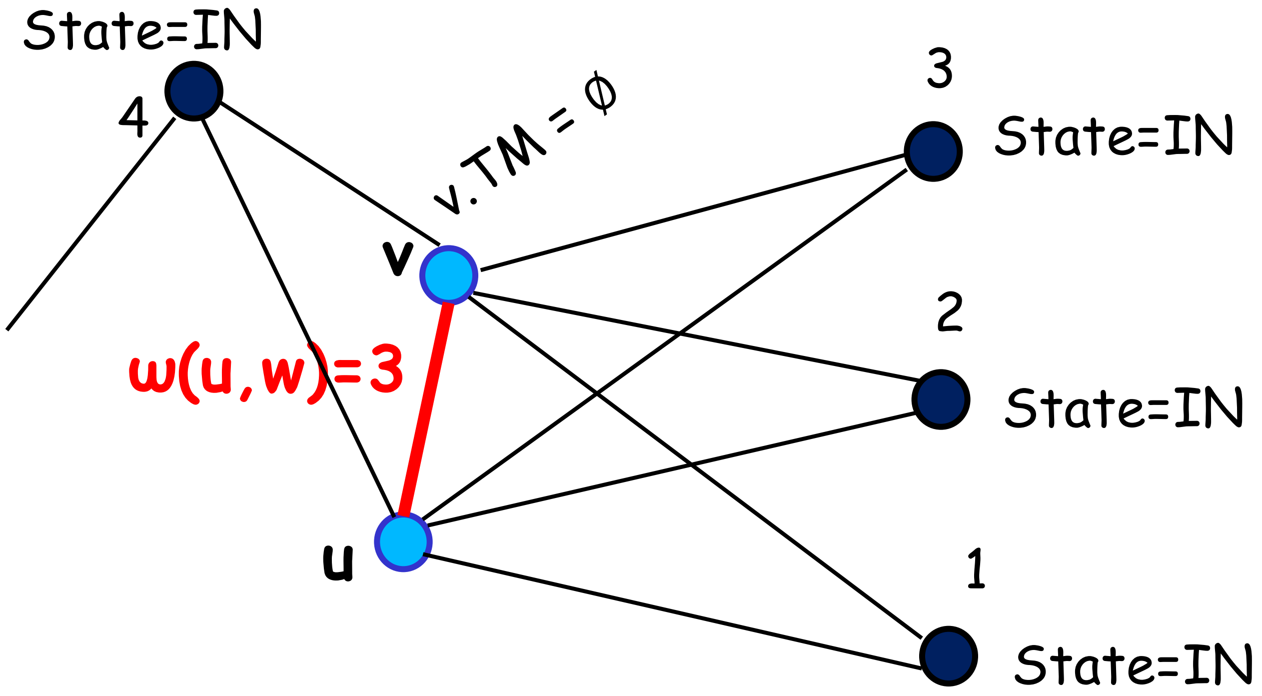
SEMS



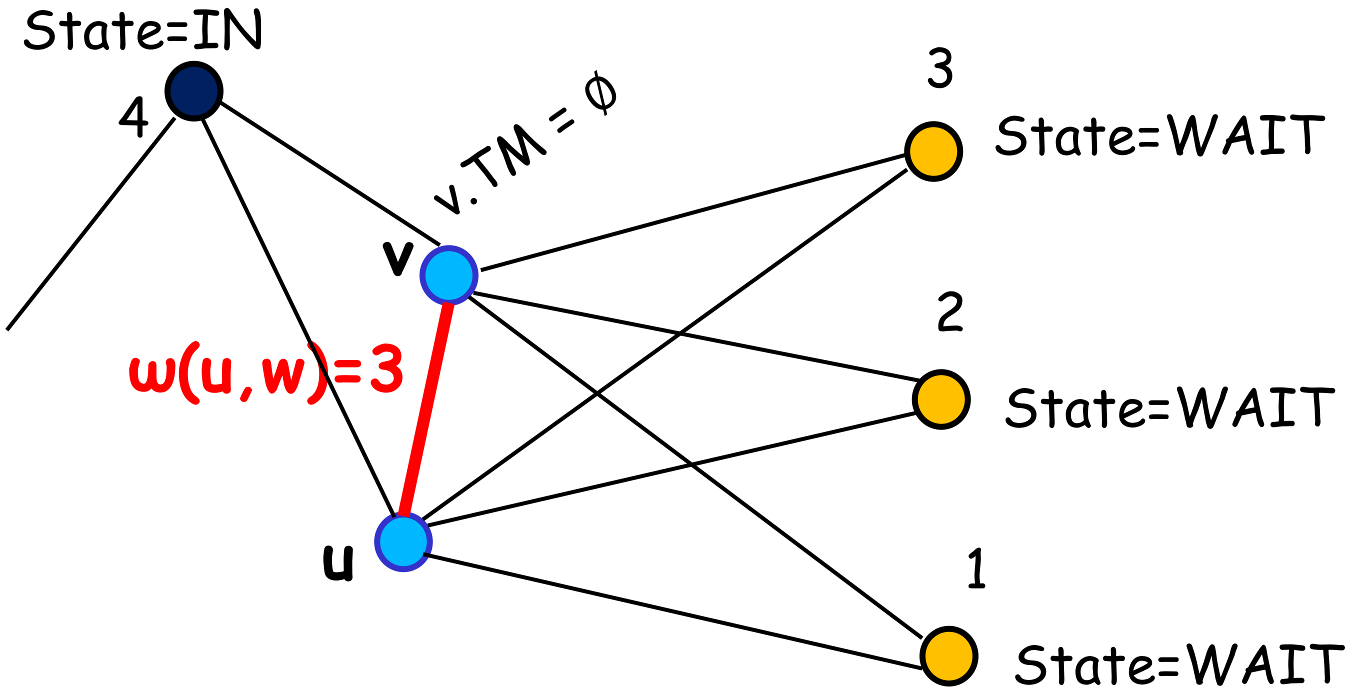
SEMS

- Nodes with state IN that are not target monitor for any neighbor changes from IN to WAIT
- To transit from WAIT to OUT, all neighbors must give permission
- A node gives this permission (variable PO) to neighbor with state WAIT with smallest identifier

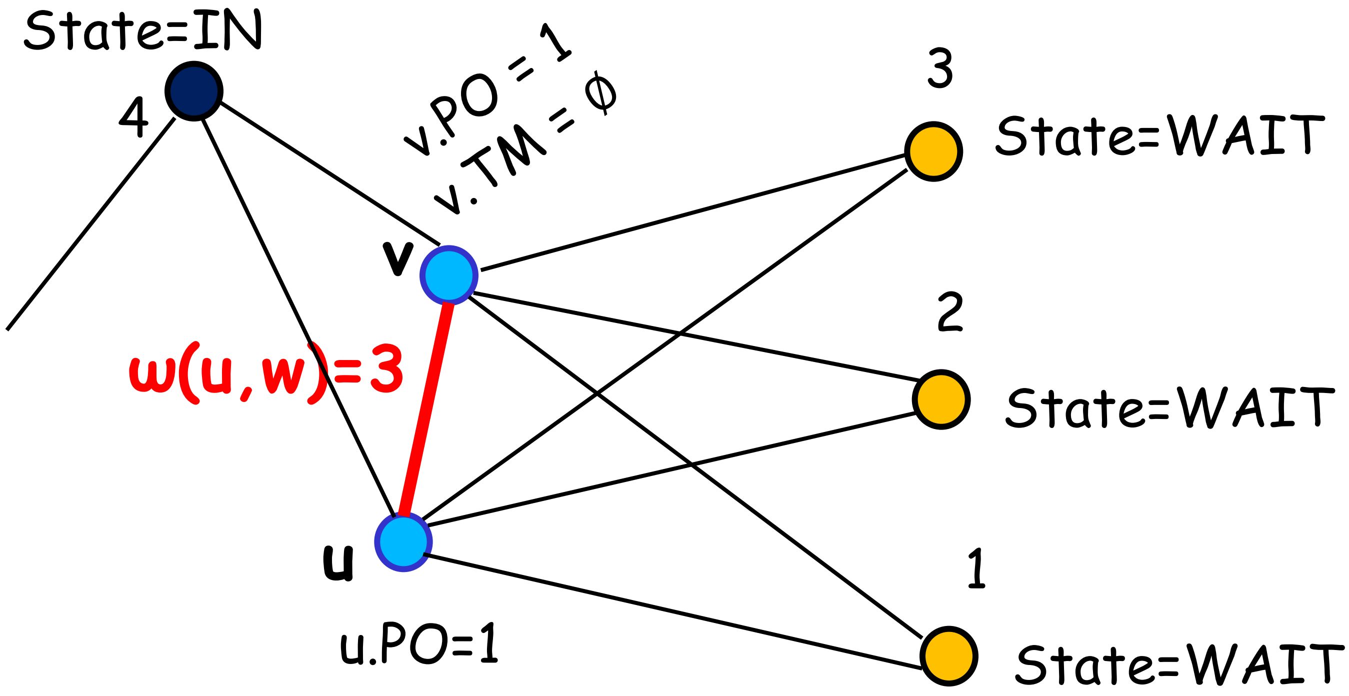
SEMS



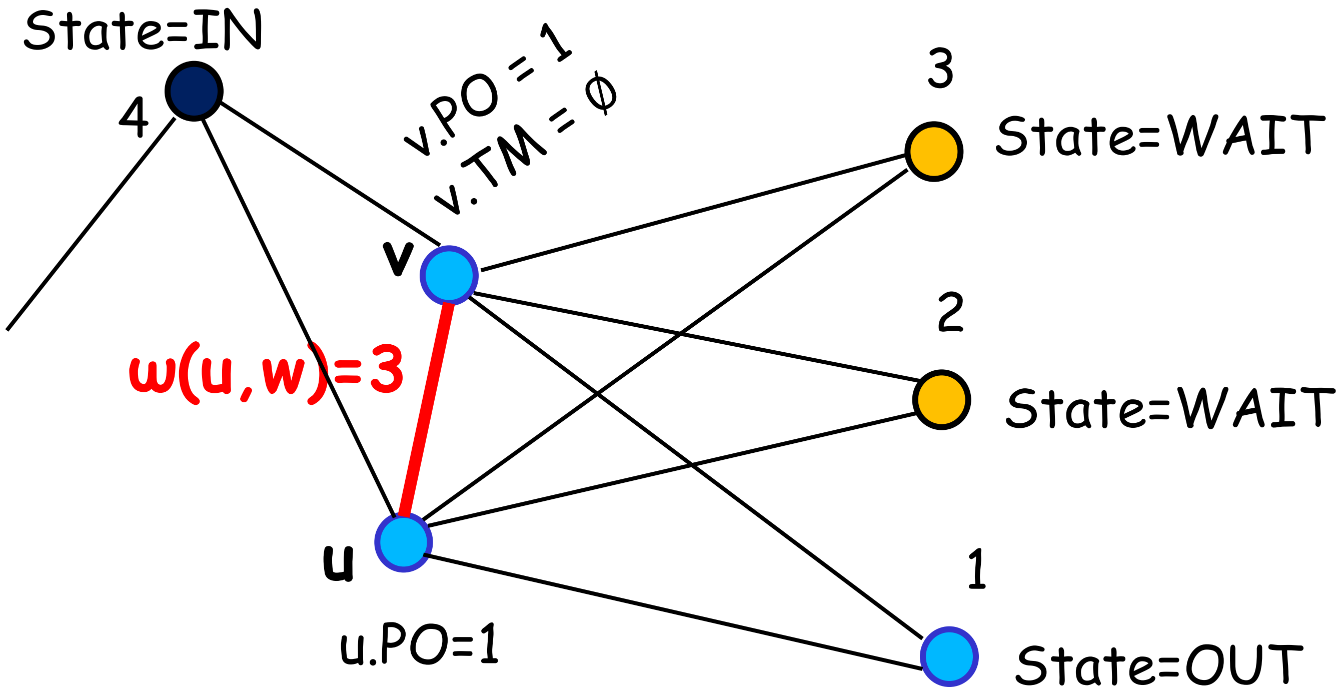
SEMS



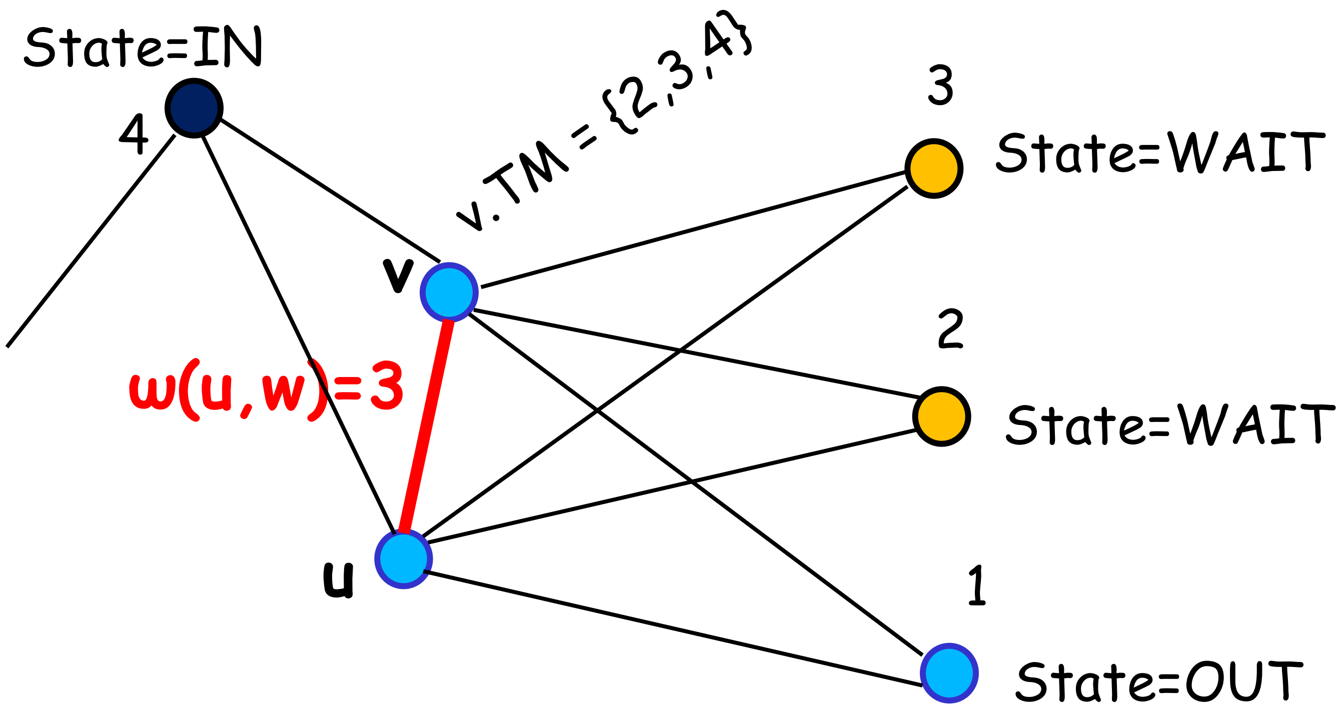
SEMS


State=IN
4
v.PO = 1
v.TM = ∅
v
ω(u,w)=3
u
u.PO=1
3
State=WAIT
2
State=WAIT
1
State=WAIT

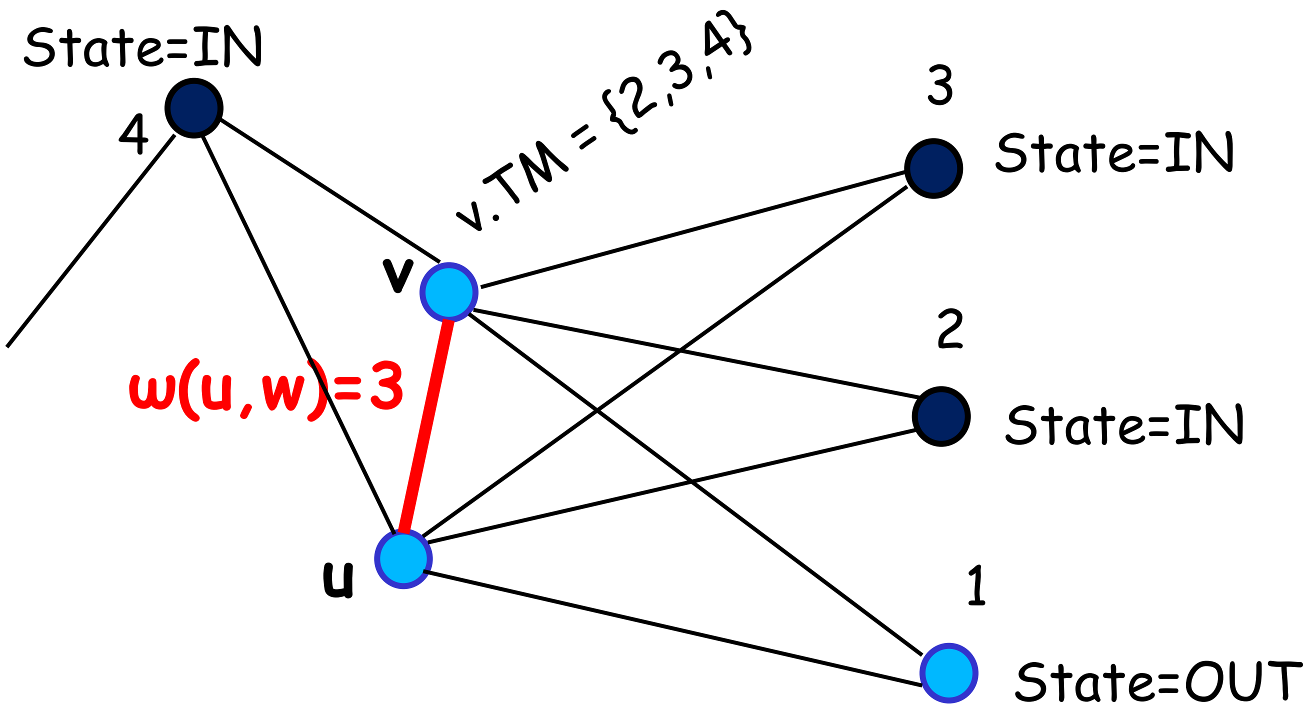
SEMS


State=IN
4
v.PO = 1
v.TM = ∅
3
State=WAIT
v
2
ω(u,w)=3
State=WAIT
u
1
u.PO=1
State=OUT

SEMS



SEMS



SEMS: Formal Definition

Variables for each node **v**:

- **S** :: contains N(v)
- **TM** :: the set of target monitors (Note that |TM| ≤ Δ)
- **PO** :: contains the smallest id of all neighbors in state WAIT not contained in TM or null – used to give permission to change state to OUT

SEMS: Formal Definition

Two groups of rules:
Management of invitations and permissions
Management of state

Algorithm *SEMS*: Maintaining TM, PO and S

Nodes: v is the current node

$S \neq N(v) \quad \longrightarrow S := N(v);$ [R1]

$TM \neq \bigcup_{u \in N(v)} TM_e(v,u) \vee PO \neq min\{u \in N(v) \mid u.state = Wait \wedge u \notin TM\}$

$\longrightarrow TM := \bigcup_{u \in N(v)} TM_e(v,u);$

$PO := min\{u \in N(v) \mid u.state = Wait \wedge u \notin TM\}$; [R2]

SEMS: Formal Definition

Algorithm *SEMS*: Maintaining *state*

Nodes: v is the current node

$state = Out \wedge \exists u \in N(v) : v \in u.TM \wedge \forall w \in N(v) : v \neq w.PO$

$\longrightarrow state := In;$ [R3]

$state = In \wedge \forall u \in N(v) : v \notin u.TM \longrightarrow state := Wait;$ [R4]

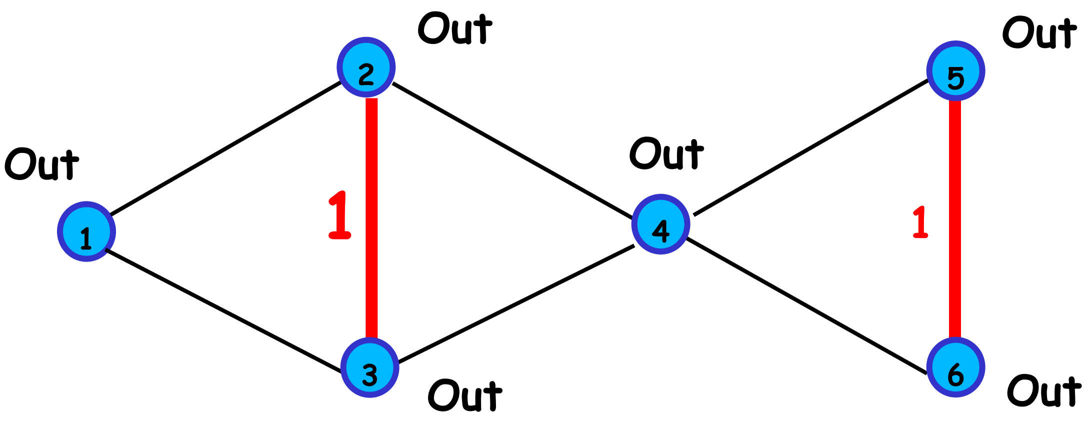
$state = Wait \wedge \exists u \in N(v) : v \in u.TM \longrightarrow state := In;$ [R5]

$state = Wait \wedge \forall u \in N(v) : v = u.PO \longrightarrow state := Out;$ [R6]

Examples

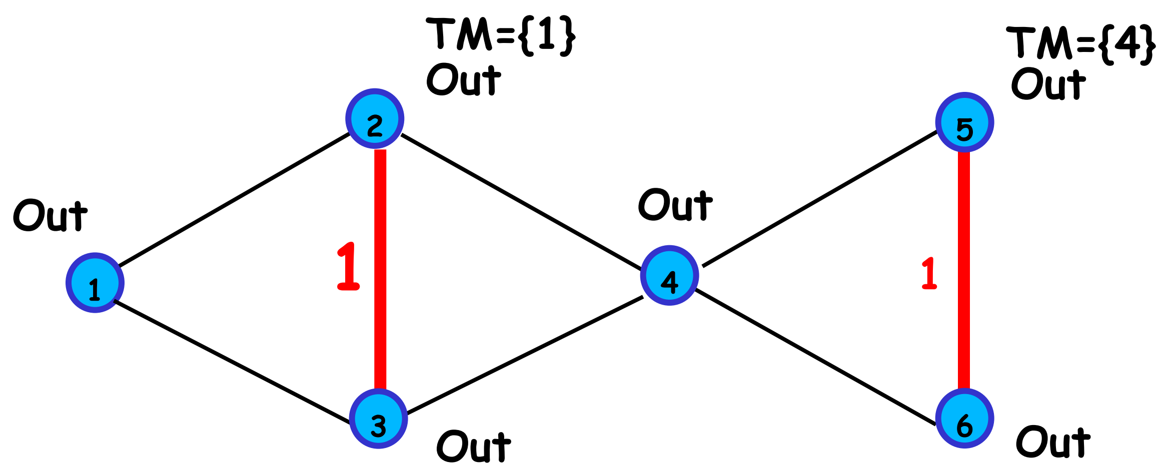
To simplify examples, we consider the synchronous scheduler

SEMS



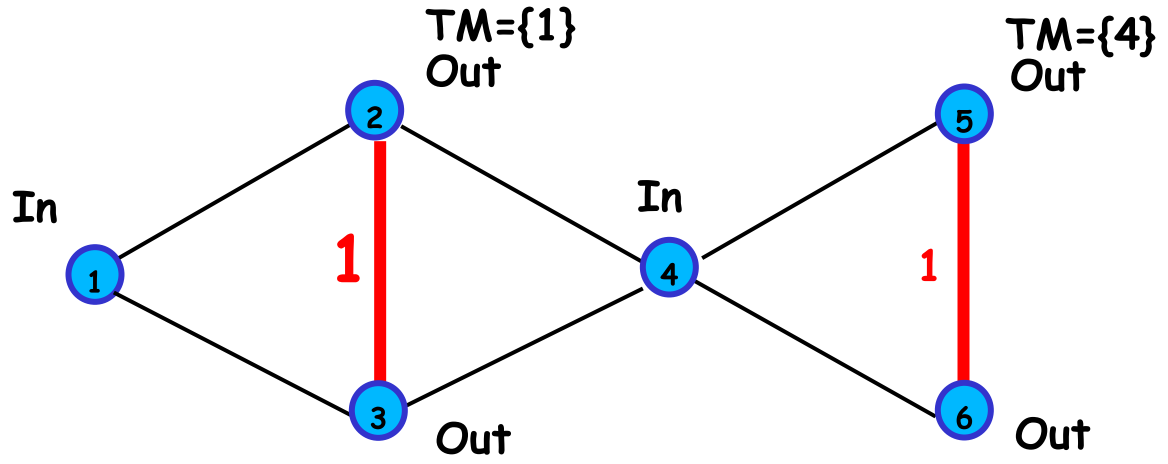
Consider a situation where each node has state=Out and TM=∅

SEMS



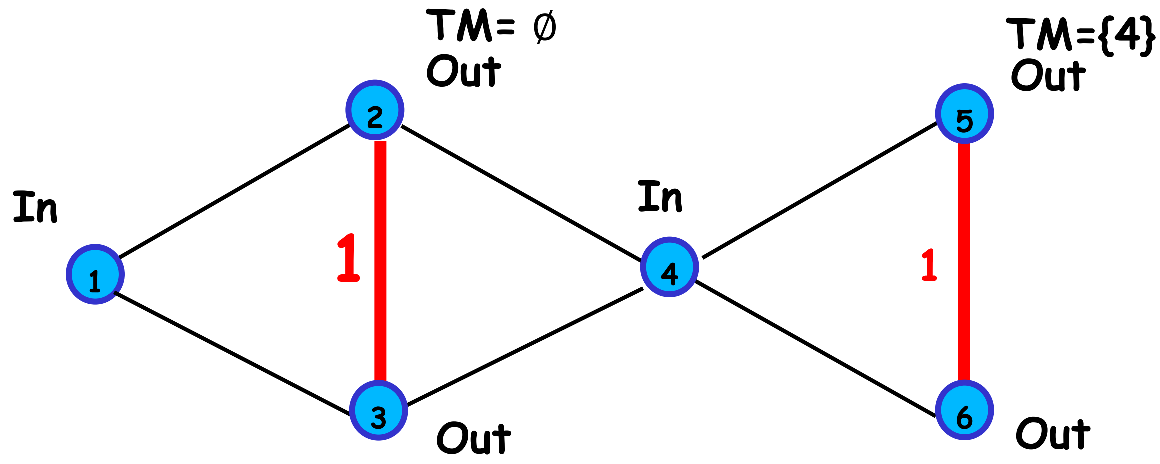
Step 1: Nodes 2 and 5 execute R2

SEMS



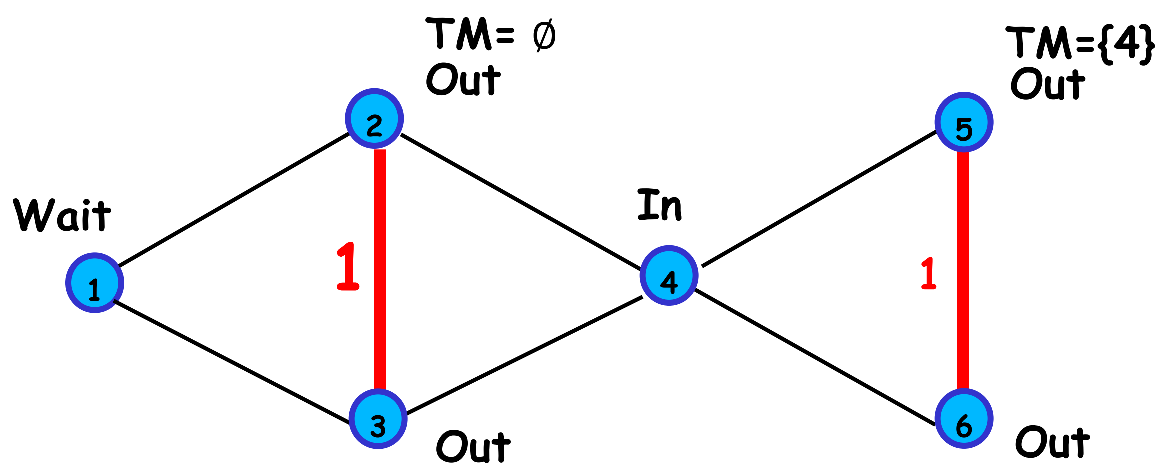
Step 2: Nodes 1 and 4 execute R3

SEMS



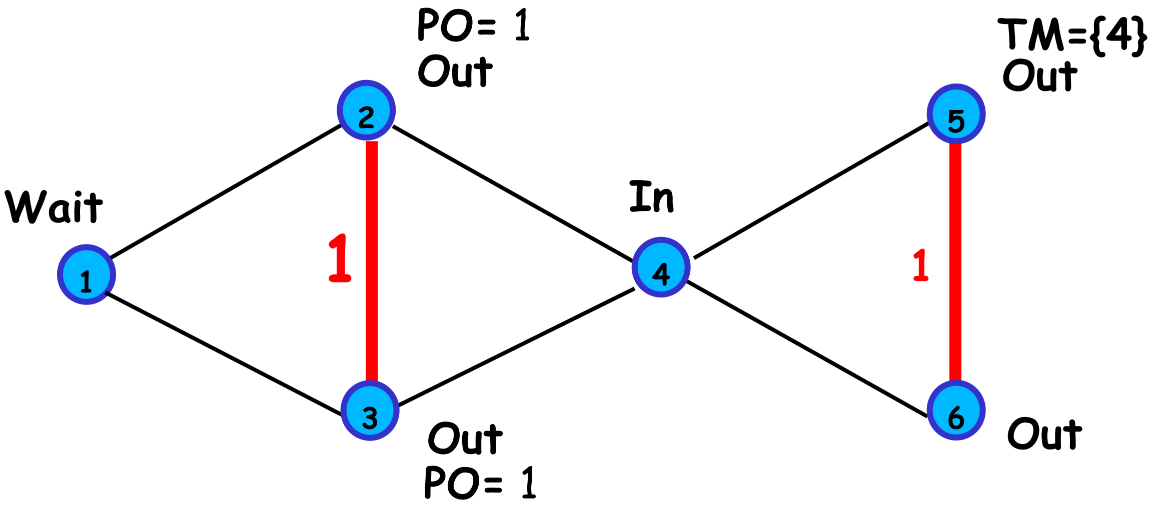
Step 3: Node 2 executes R2

SEMS



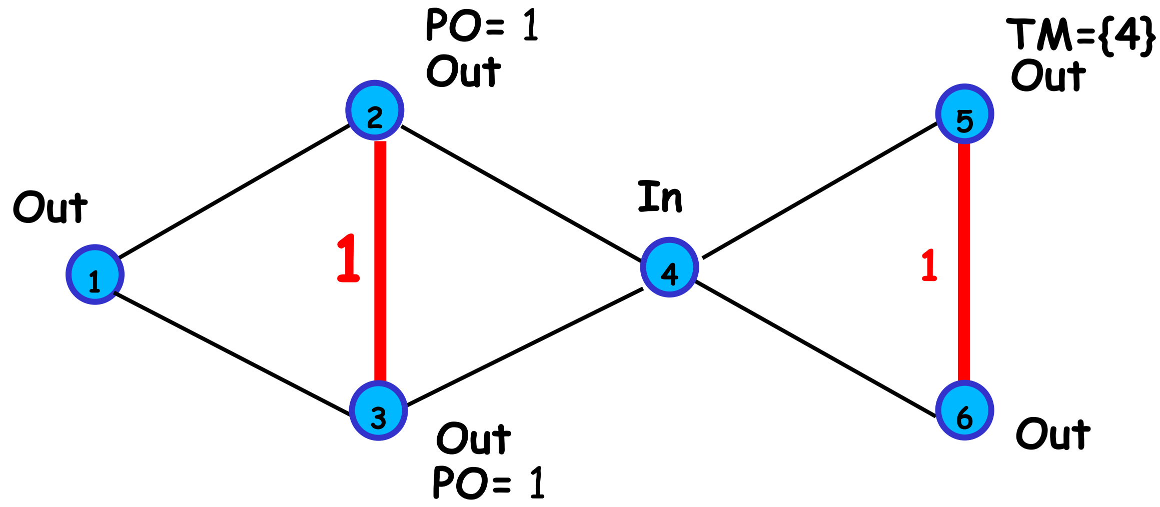
Step 4: Node 1 executes R4

SEMS



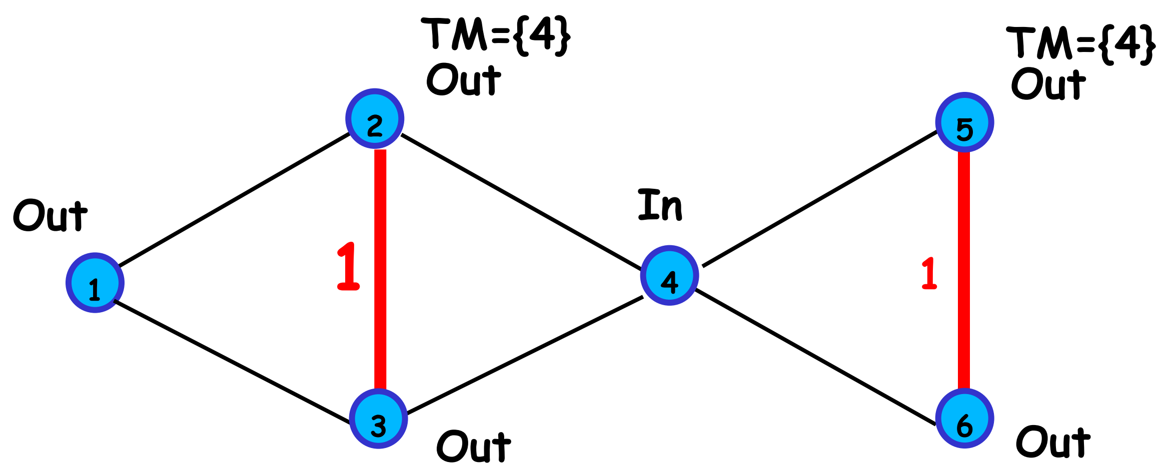
Step 5: Nodes 2 and 3 execute R2

SEMS



Step 6: Node 1 executes R6

SEMS

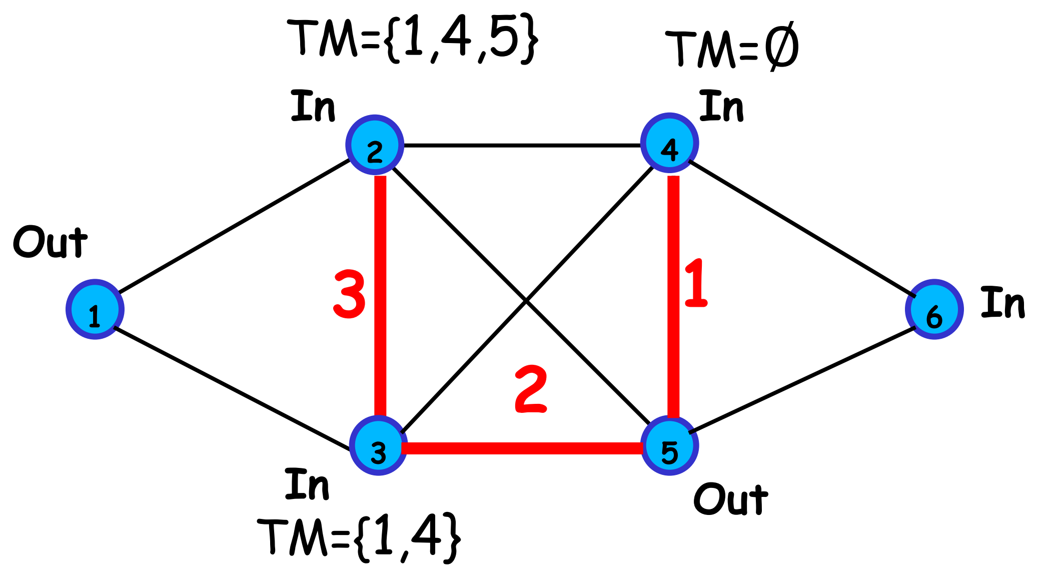


Step 7: Nodes 2 and 3 execute R2

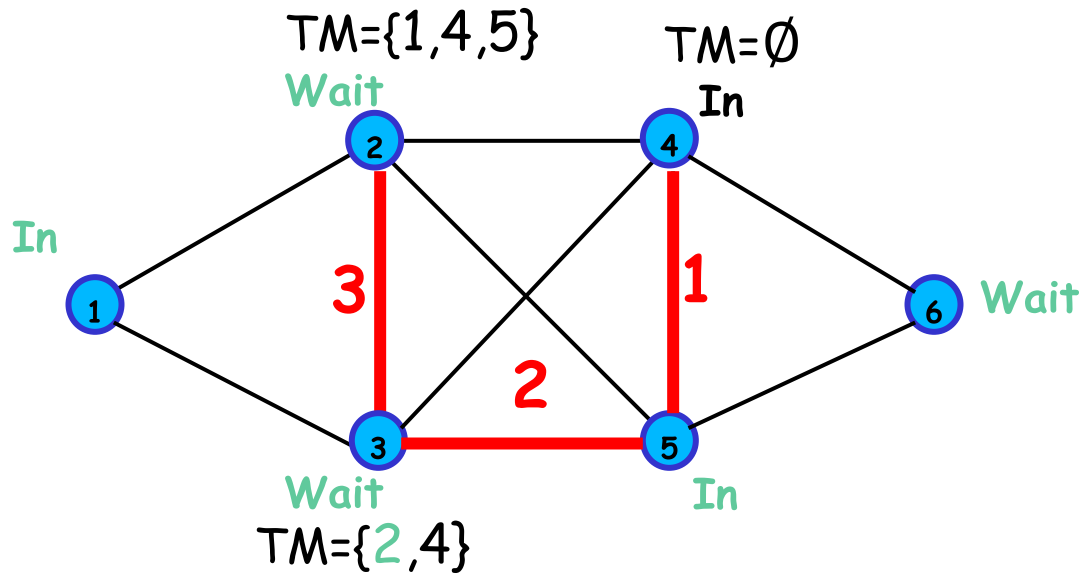
SEMS

Example with corrupted state

SEMS

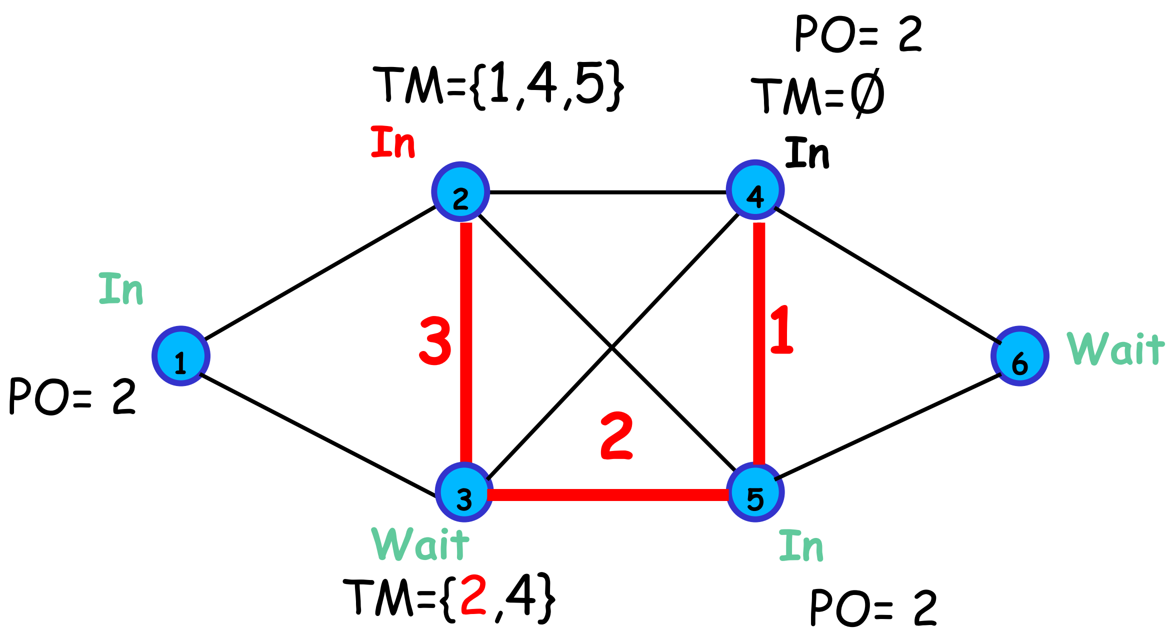


SEMS



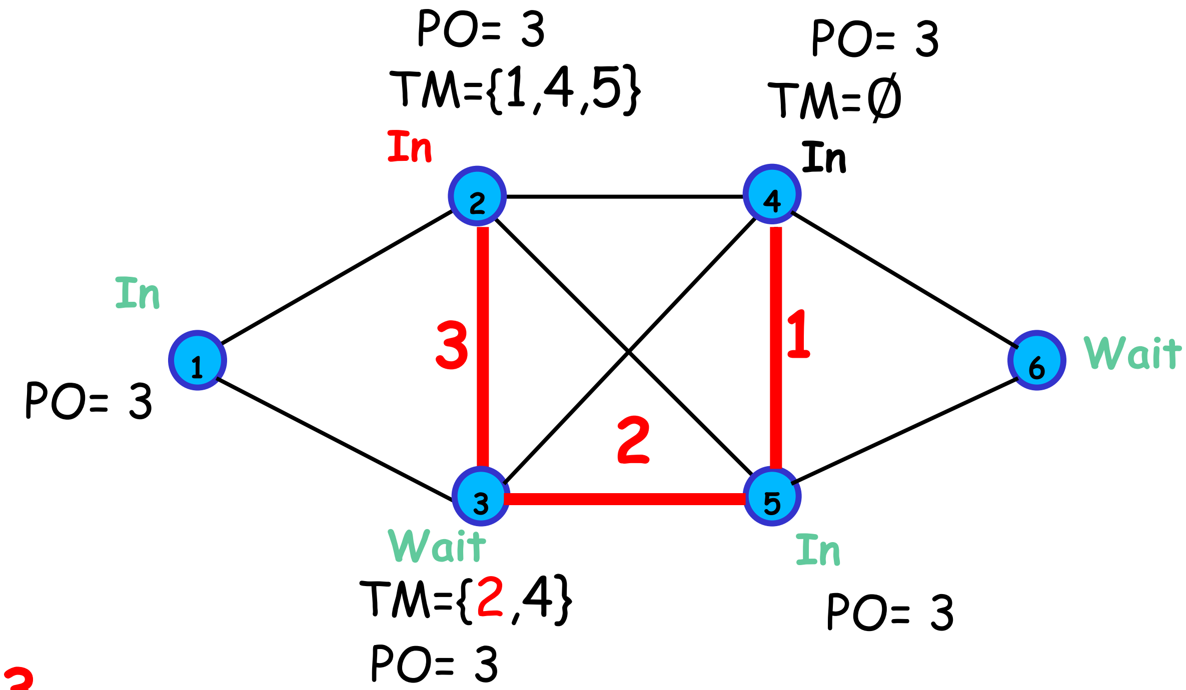
Step 1

SEMS



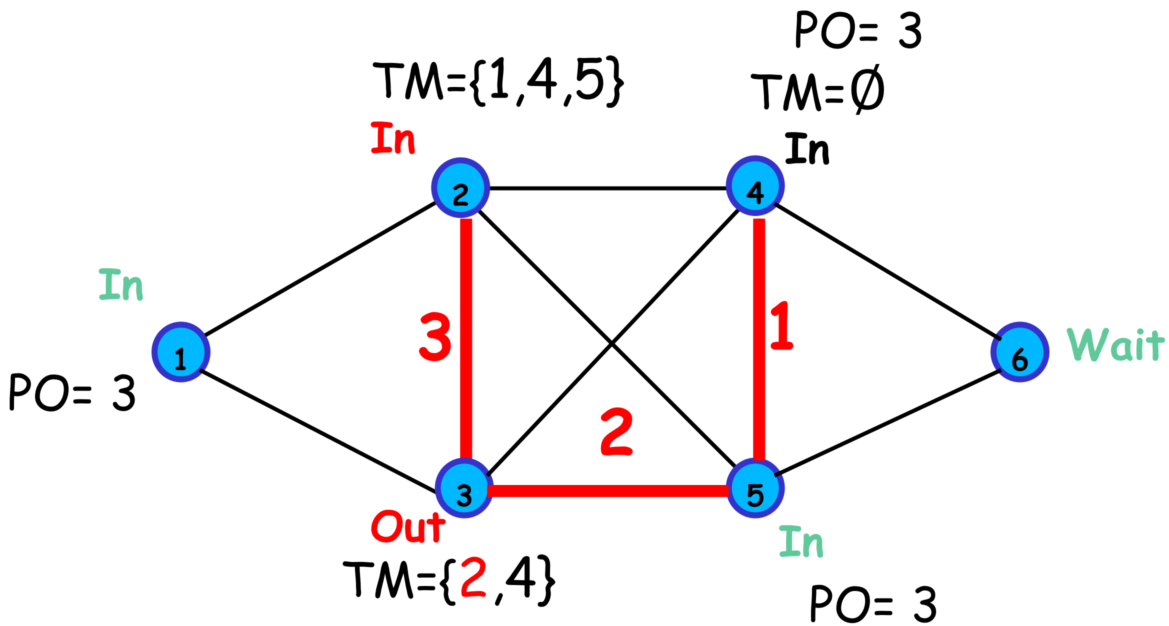
Step 2

SEMS


PO= 3
TM={1,4,5}
In
PO= 3
TM=∅
In
In
PO= 3
1
2
3
4
5
6
3
1
2
Wait
Wait
TM={2,4}
PO= 3
In
PO= 3

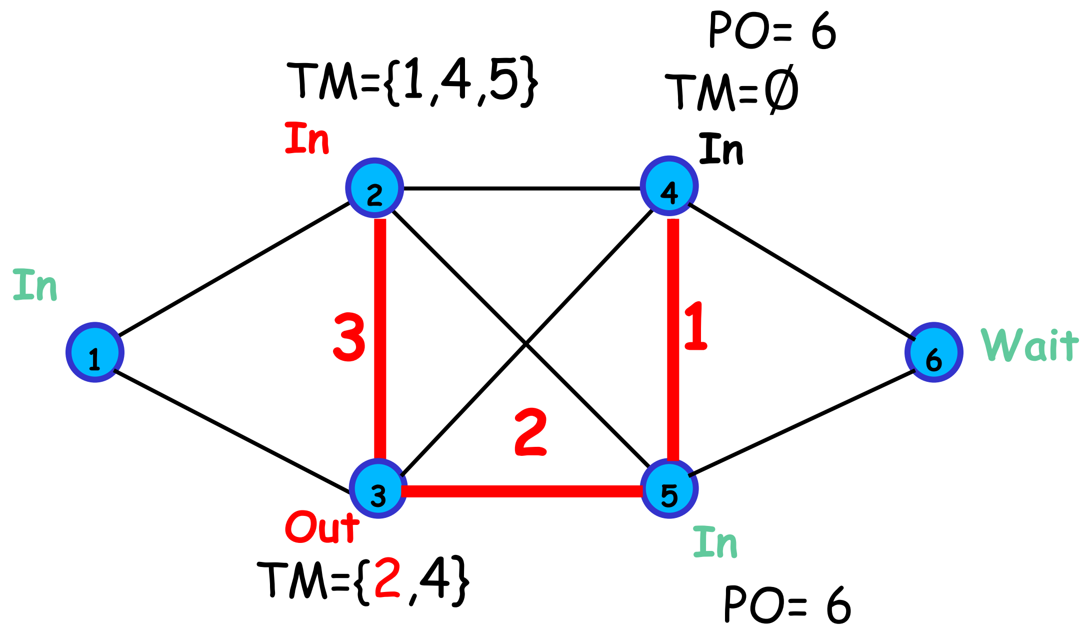
Step 3

SEMS



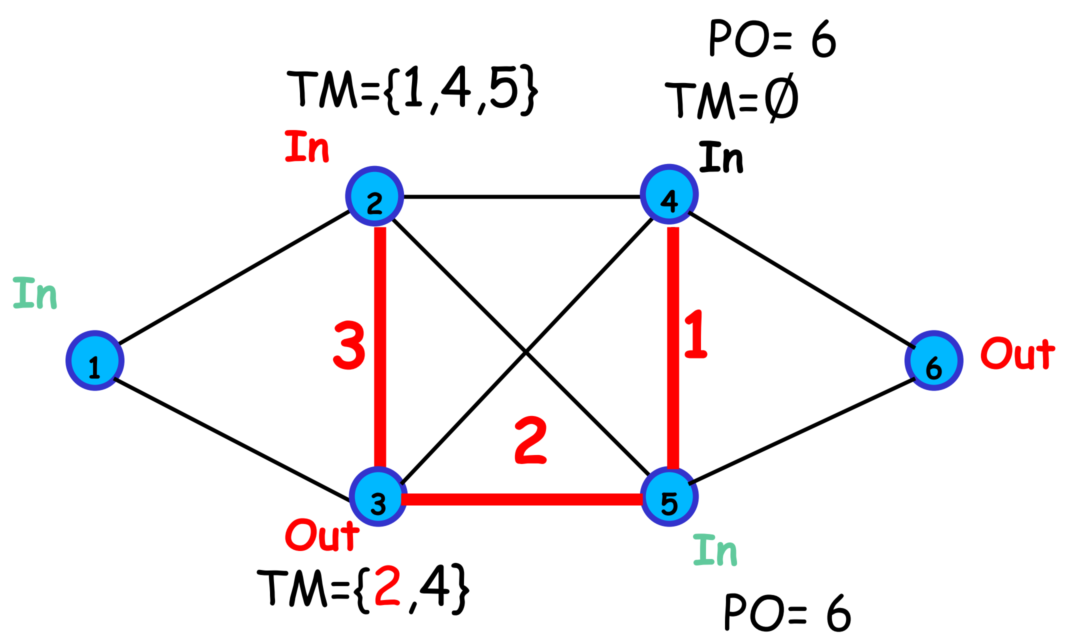
Step 4

SEMS



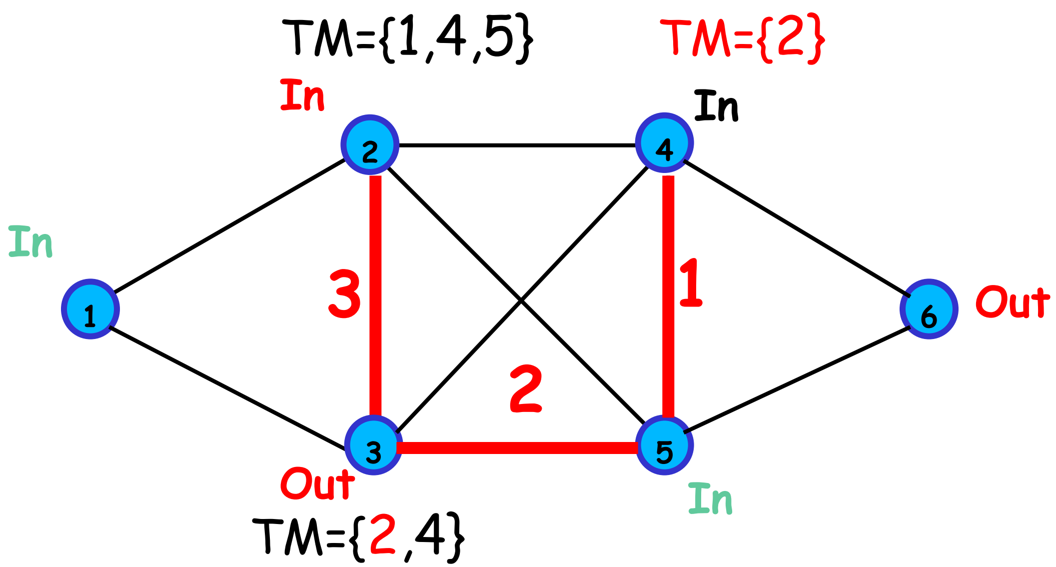
Step 5

SEMS



Step 6

SEMS



Step 7

Conclusions & future work

Contribution:

- SEMS: A self-stabilizing algorithm for computing a minimal edge monitoring set

- SEMS converges in $O(\Delta^2 m)$ moves under unfair distributed scheduler

- Improving on previous work (Hauck $O(n^2 m)$ moves)

Conclusions & future work

Future work

1. We believe that complexity of algorithm is lower than $O(\Delta^2 m)$. Conjecture: $O(\Delta m)$

2. Study lower bounds of the problem for distributed scheduler