LIRIS

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
UMR 5205

3rd Workshop on CyberSecurity

A Self-Stabilizing Algorithm for Edge Monitoring Problem

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

[1] **Laboratoire d'InfoRmatique en Image et Systèmes d'information**
Université Claude Bernard Lyon (LIRIS/UCBL)

[2] **Institute of Telematics, Hamburg University of Technology,**
Hamburg, Germany (IT/TUHH)

Published in the proceeding of the 16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2014), Paderborn, Germany

INSA LYON

UNIVERSITÉ LUMIÈRE LYON 2

CENTRALE LYON

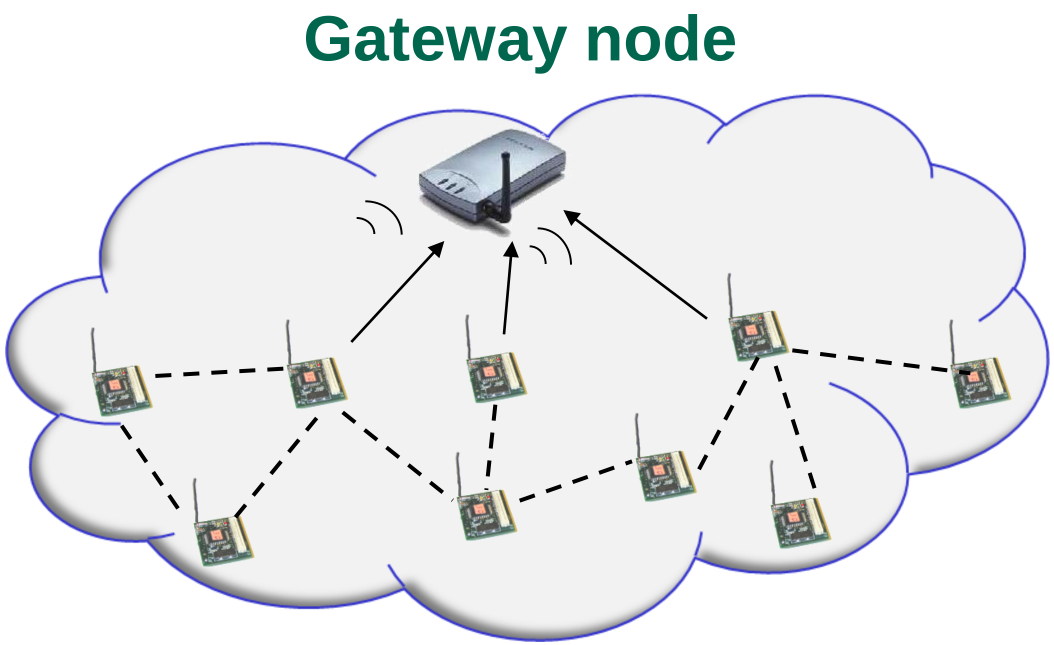
Université Claude Bernard Lyon 1

Lyon- 09/12/2014

Team: Graphs, Algorithms and Applications (GOAL)

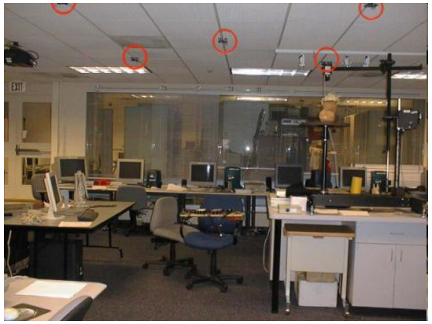
Wireless Sensor Network (WSN)

A Wireless Sensor Network (WSN) consists of distributed autonomous sensors to monitor Physical or environmental conditions such as temperature, sound, vibration, pressure.



WSN Applications

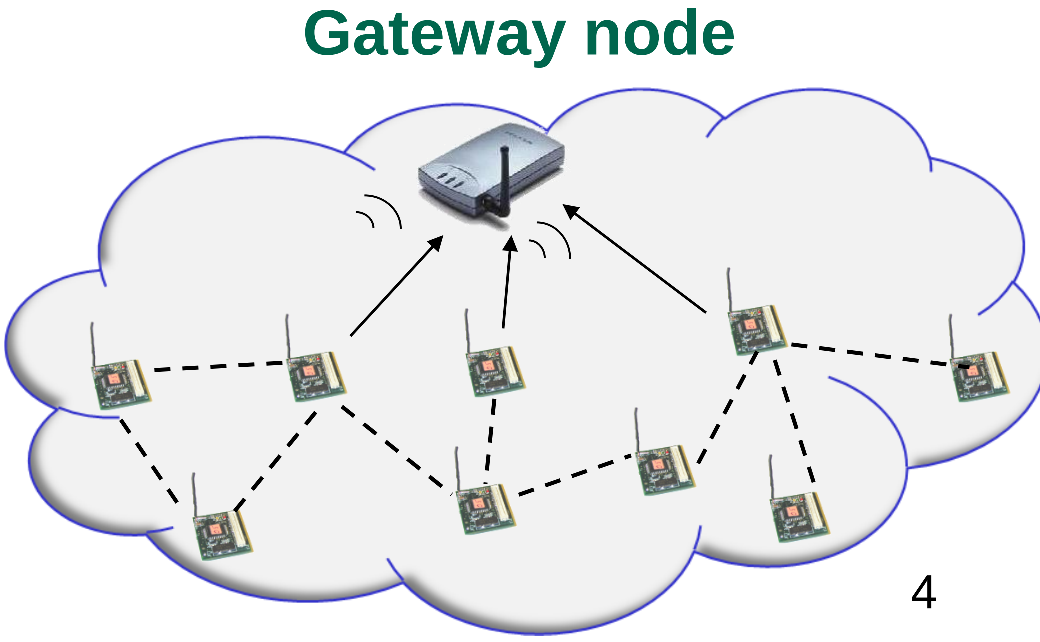
- Environmental/Habitat monitoring
- Acoustic detection
- Seismic Detection
- Military surveillance
- forest fire control
- Medical monitoring
- Industrial process control
- Process Monitoring
- …



Challenges in Running a WSN

Vulnerability of WSN due to :

- Wireless communication
- Hostile unattended environments
- Limited resources of sensors



Security of WSN is a real challenge

Security of WSN

Cryptography techniques are efficient for **data confidentiality** and **integrity**.

Are there sufficient for compromised nodes ?
NO

Security community develops **complementary** security techniques, based on the **self-monitoring**

Security of WSN

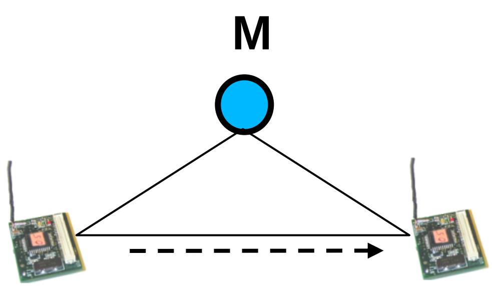
Self-monitoring : assigning monitoring roles to some of the nodes in the network.

Monitors are placed somewhere in the **intersection of the communication ranges** of the sending and the receiving nodes.

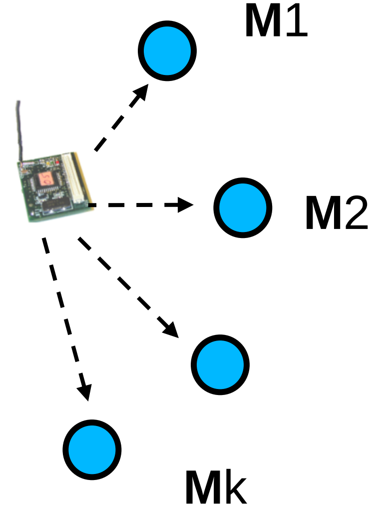
Security of WSN

We distinguish two types of **self-monitoring**

Edge-monitoring



Self-protection

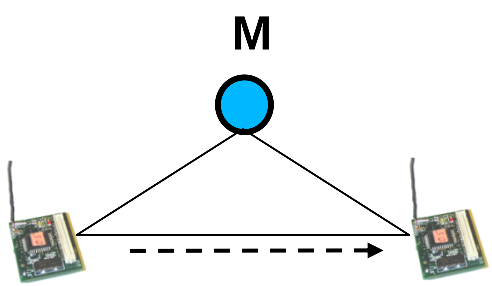


k-tuple total dominating set

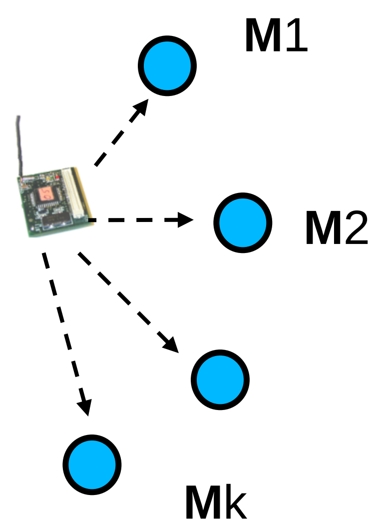
Security of WSN

We distinguish two types of **self-monitoring**

Edge-monitoring



Self-protection

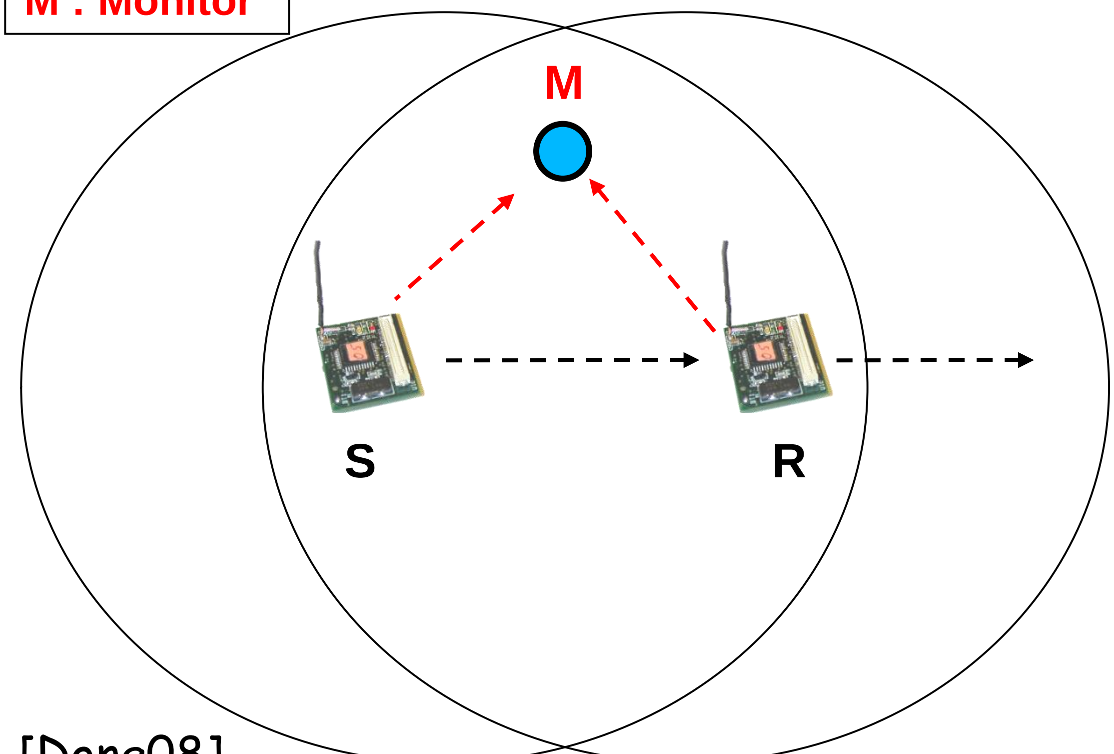


k-tuple total dominating set

Edge monitoring

This concept has been introduced in WSN by Marti et al. [Marti00]

S : Sender
R : Receiver
M : Monitor



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**

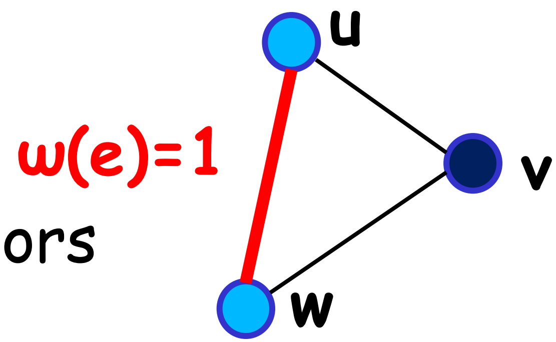
[Dong08]

Edge monitoring

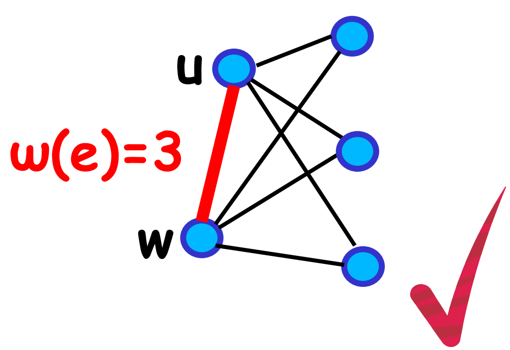
Since it is natural to model a WSN by a graph **G=(V,E)**

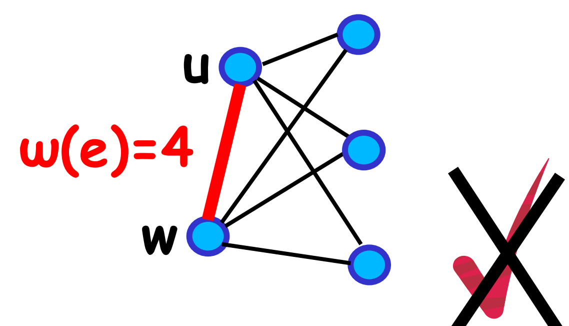
- **V:** set of nodes that represents the sensors
- **E:** set of edges that represents their communications
- We denote
  - **N(v)**={u ∈ V / <v,u> ∈ E}
  - **n**=|V|
  - **m**=|E|
  - **Δ** max node degree in G

Edge monitoring

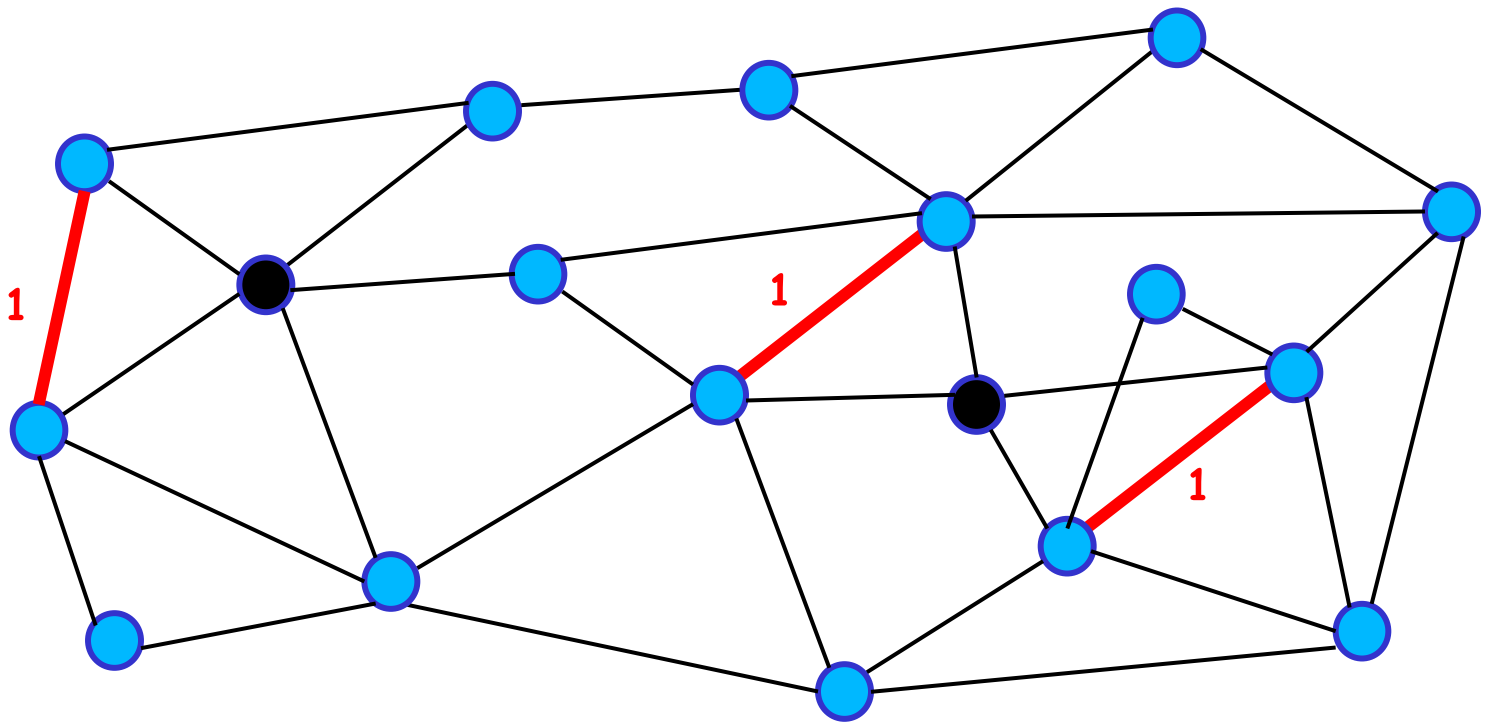


- Edges have monitoring constraints ω specifying the number of required monitors
- Assumption: For each e = ‹u,w› ∈ E then |N(u) ∩ N(w)| ≥ ω(e)





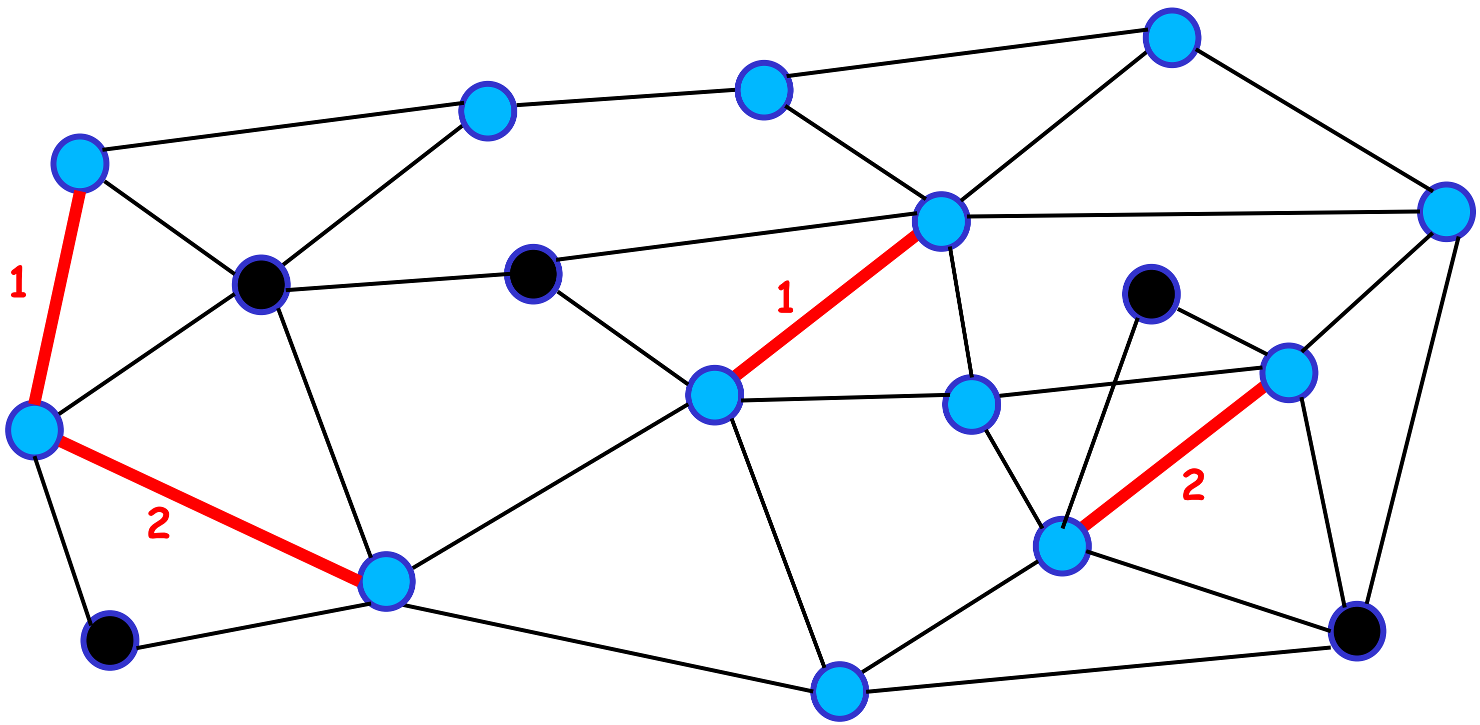
Example



red :: edges to be monitored

black :: monitors

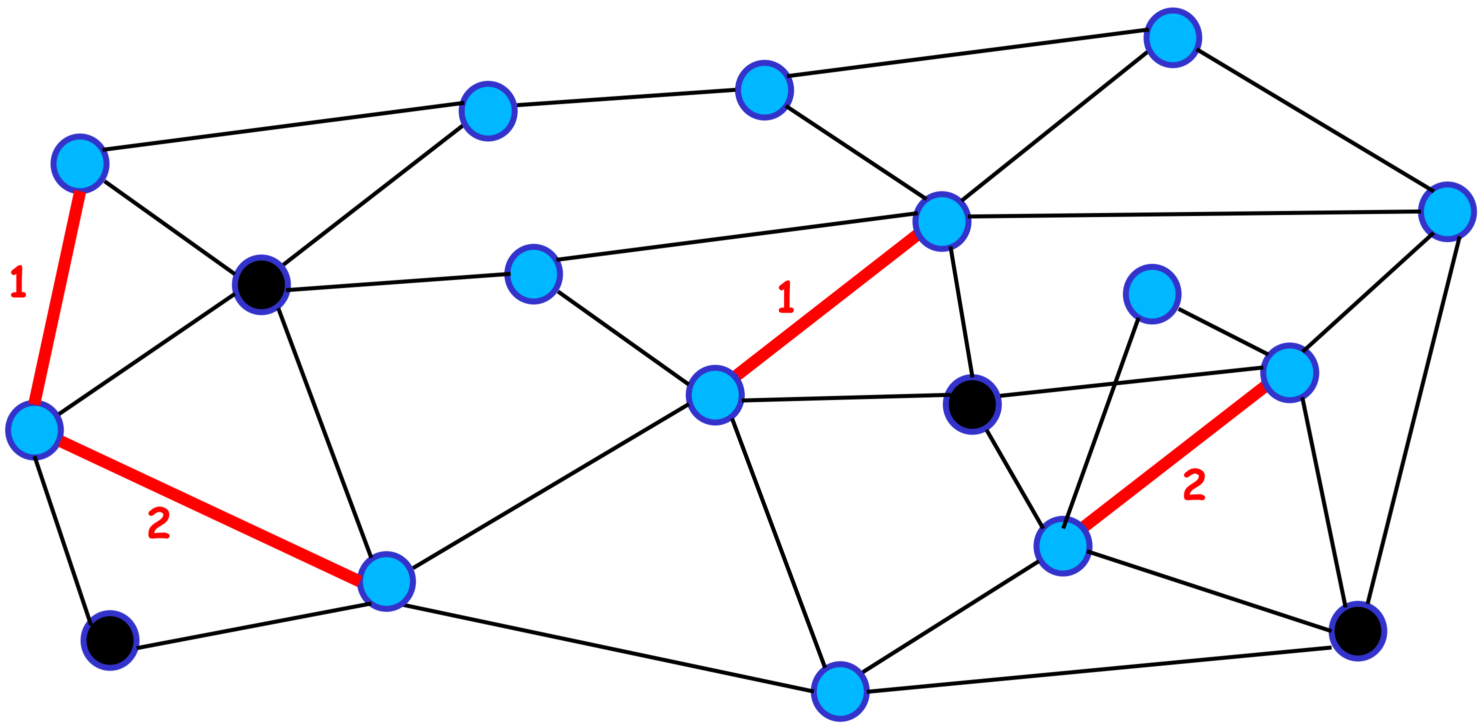
Example



red :: edges to be monitored
black :: monitors

5 monitors!

Example



red :: edges to be monitored
black :: 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** ∈ E there are at least **ω(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, Dong11]
- 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

Reference	Dist. knowledge	Transformer	Com. model	Self-stab. ?	Complexity.
[Dong 08] [Dong 11]	Distance-two	Yes	Synchronous	No	$O(\Delta)$
[Hauck 12]	Expression model	Yes	**Asynchronous**	**Yes**	$O(n^2 m)$
Our paper	-	-	-	-	-

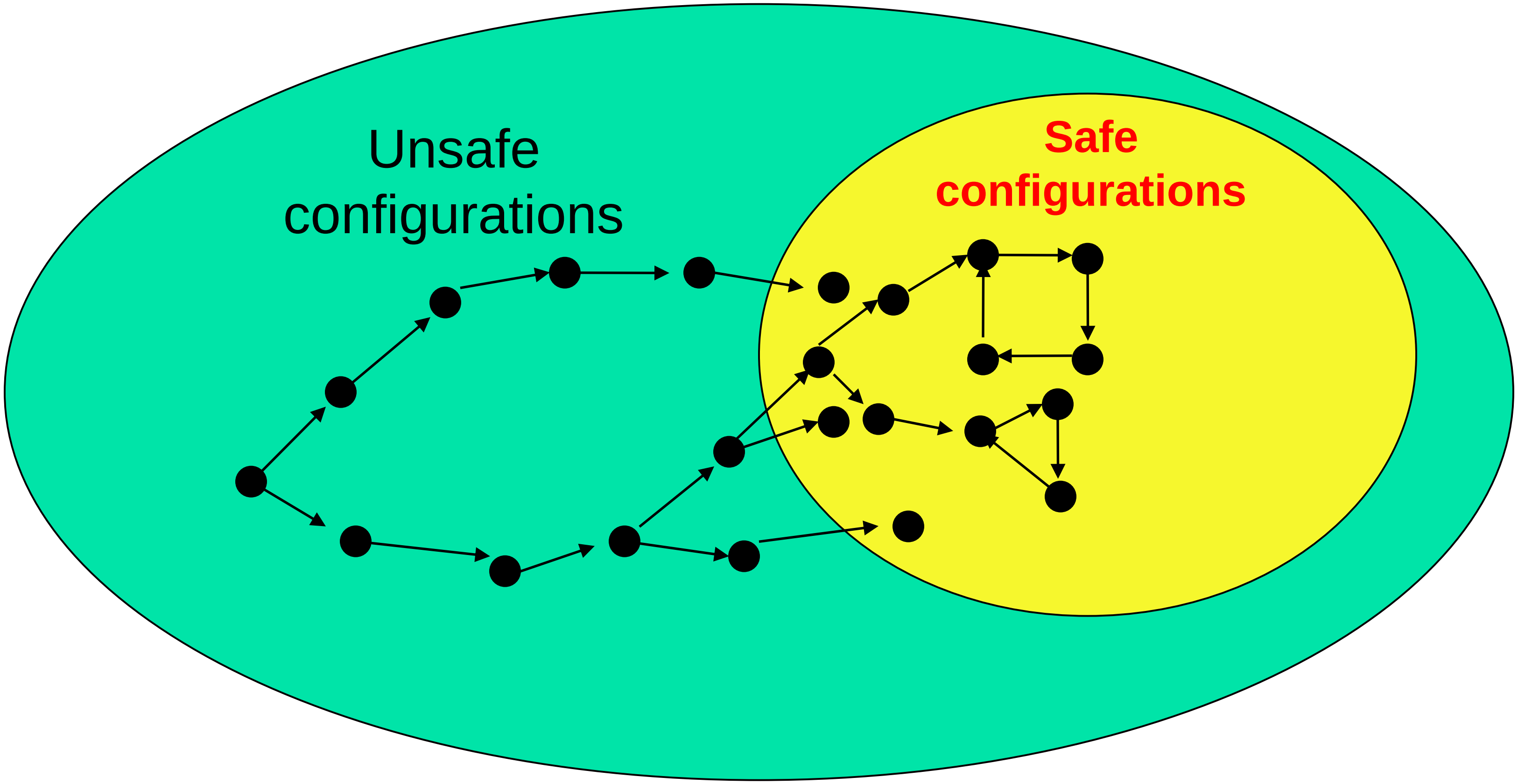
Contribution

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

Reference	Dist. knowledge	Transformer	Com. model	Self-stab. ?	Complexity.
[Dong 08] [Dong 11]	Distance-two	Yes	Synchronous	No	$O(\Delta)$
[Hauck 12]	Expression model	Yes	**Asynchronous**	**Yes**	$O(n^2 m)$
Our paper	**Distance-one**	**No**	**Asynchronous**	Yes	$\mathbf{O(\Delta^2 m)}$

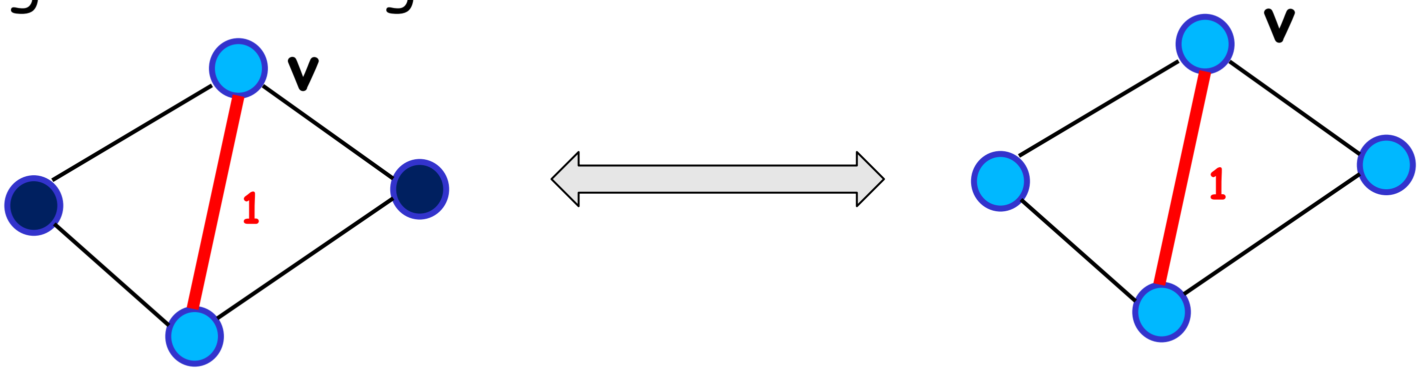
Self-stabilizing algorithm



- Self-Stabilization = Closure + Convergence

Algorithm for minimal Edge monitoring set (SEMS)

- Edge Monitoring



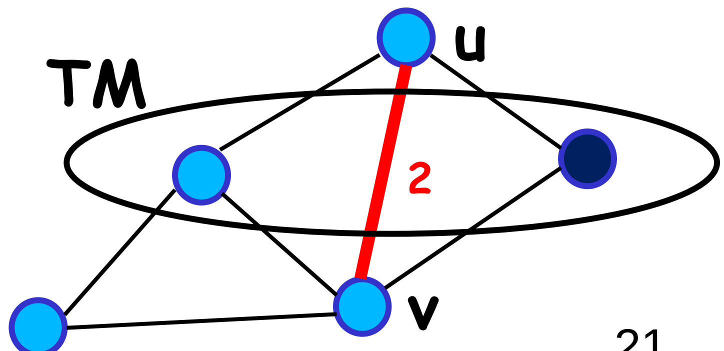
- **Problem:** Critical nodes are not neighbors
- **Solution:** Intermediate nodes give permission to a single neighbor to make a move
- **Problem:** Deadlocks may arise
- **Solution:** Enforce ordering (based on *ids*)

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)

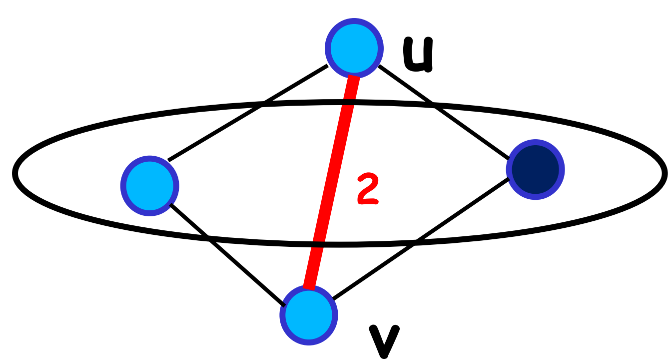


SEMS

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

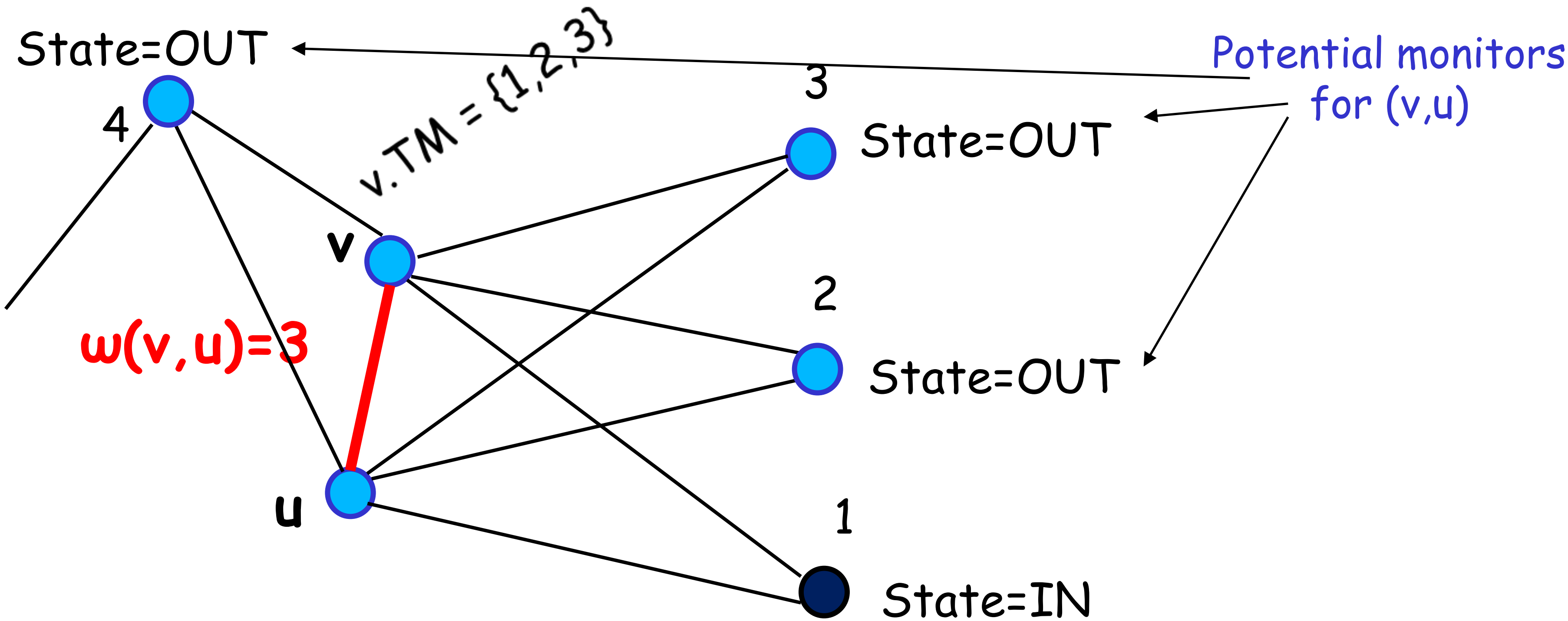
1.If number of common neighbors of v and u with state IN or WAIT is **larger than ω(e)** then let **TM = ∅**

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**

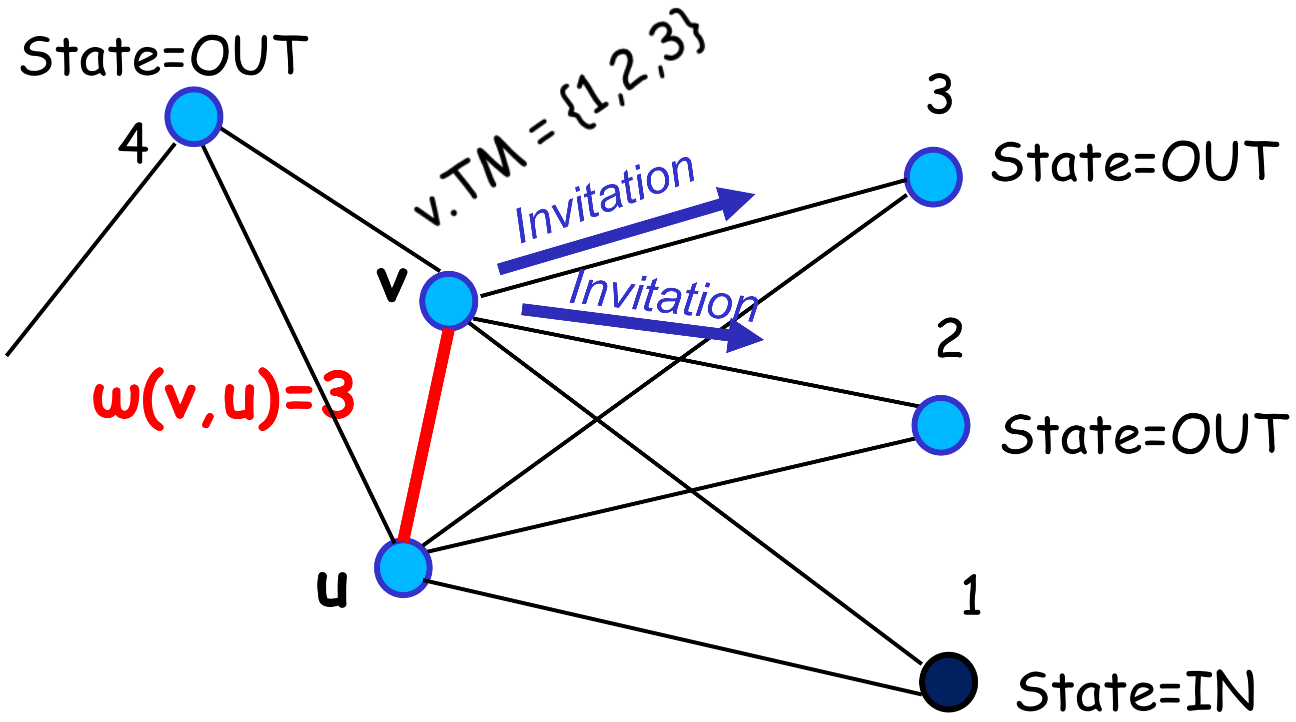


SEMS

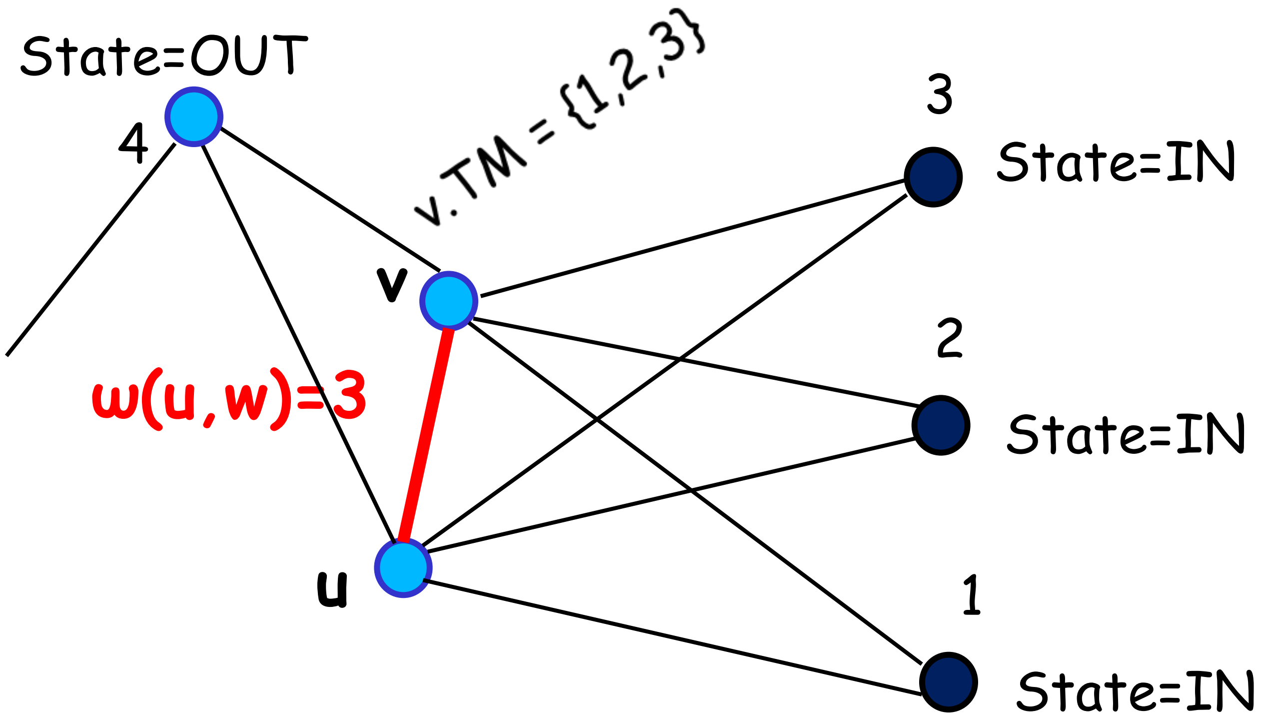
If an OUT node discovers that it is contained in TM of a neighbor it regards this as an invitation to change to IN



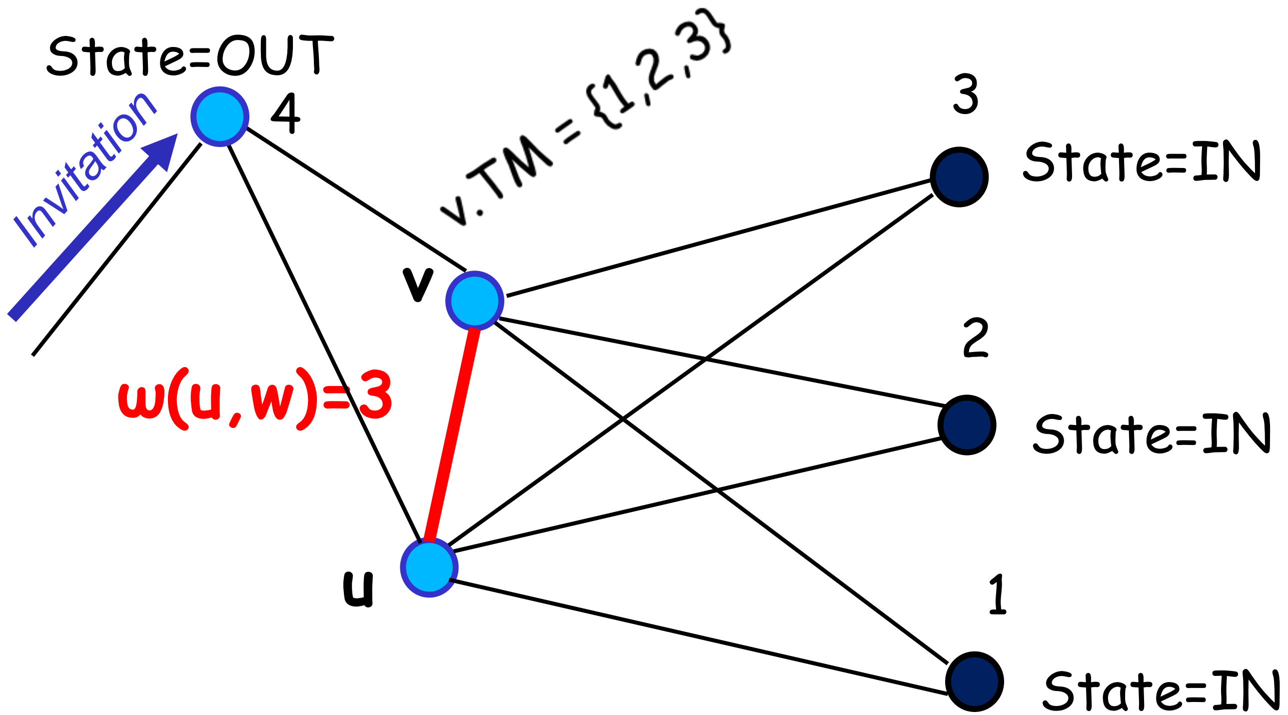
SEMS


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

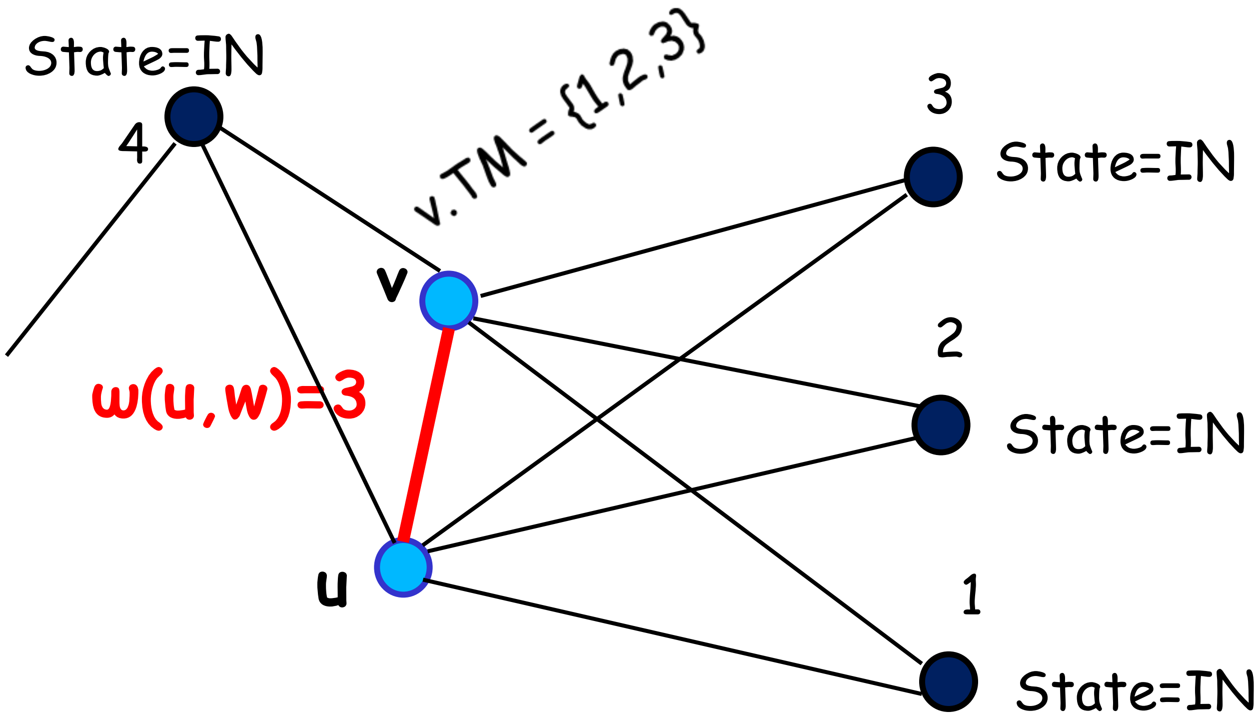
SEMS



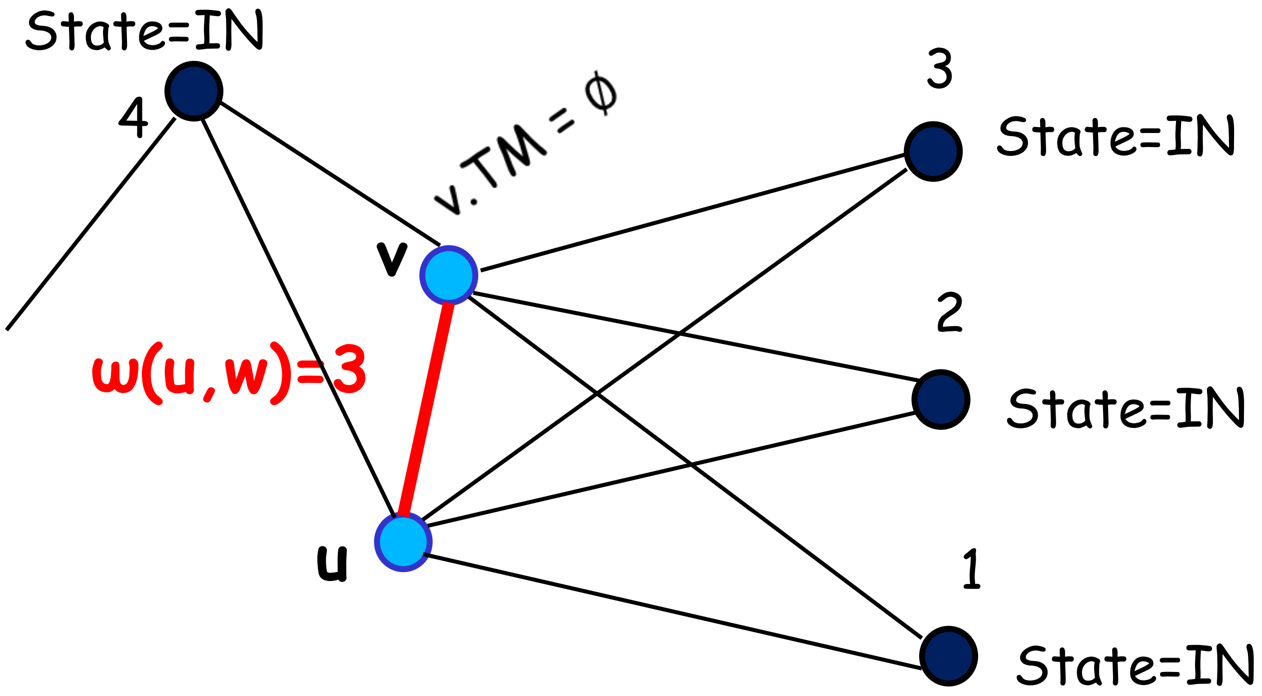
SEMS


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

SEMS



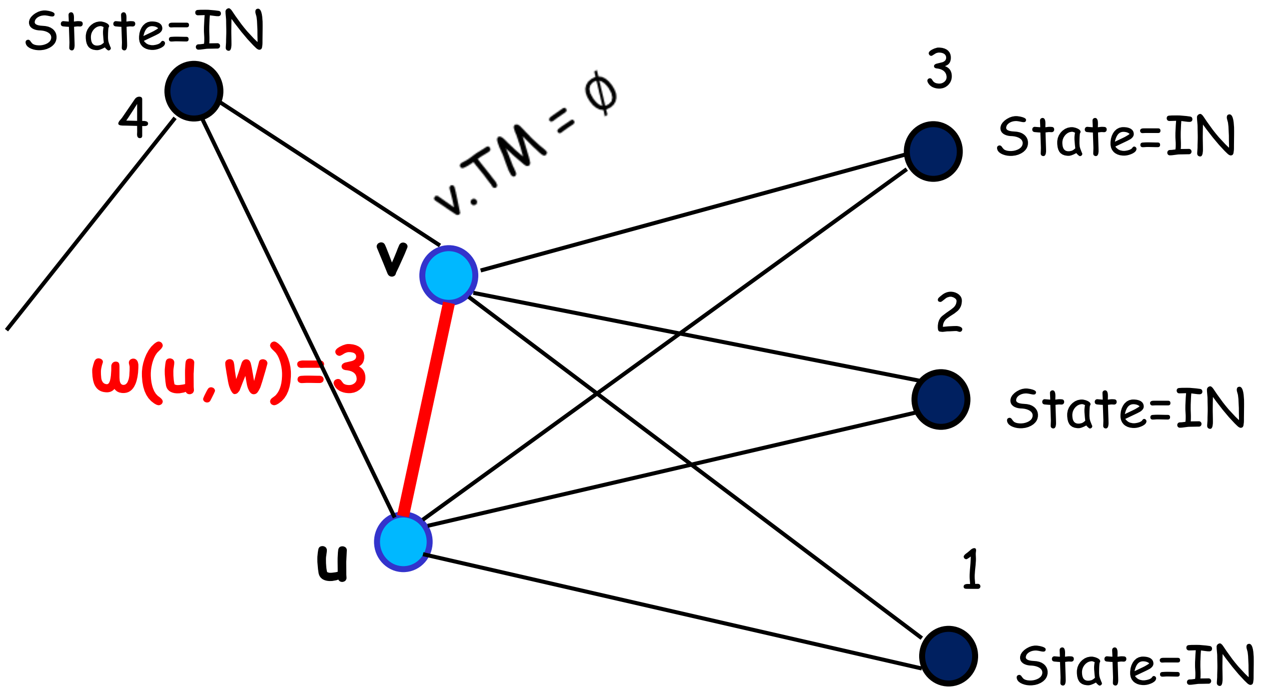
SEMS



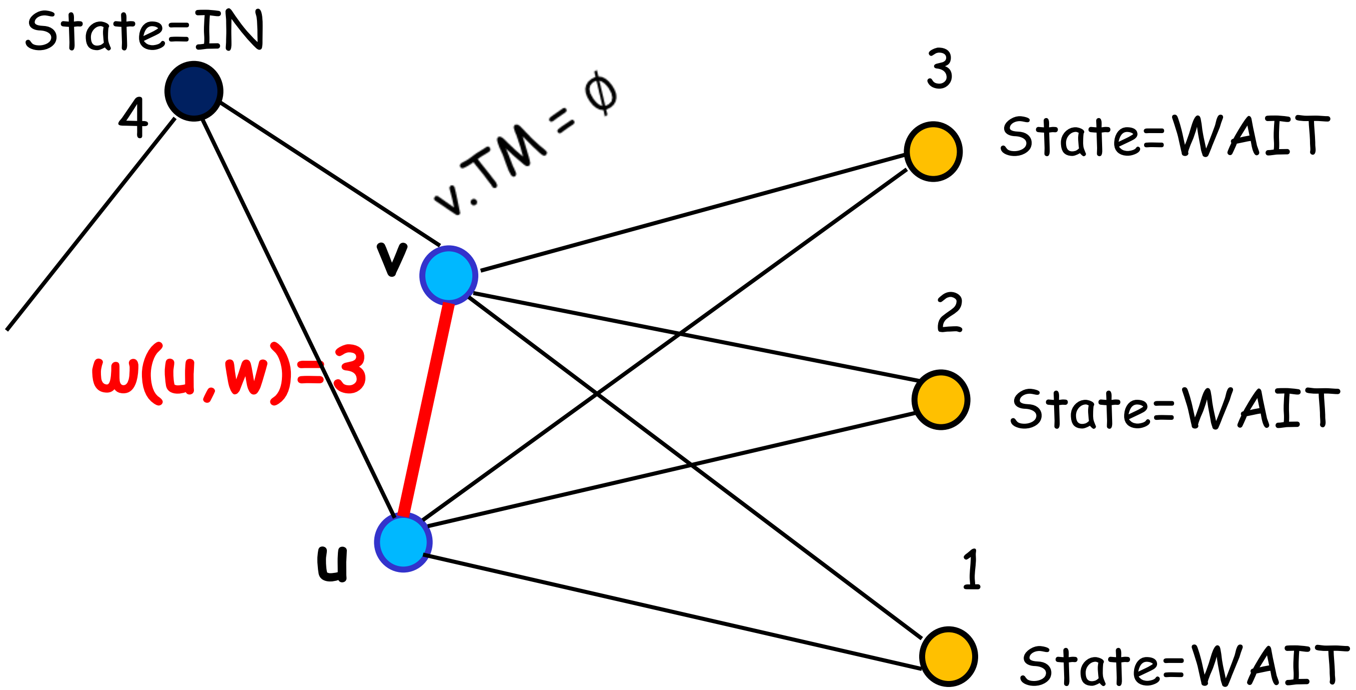
SEMS

- 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

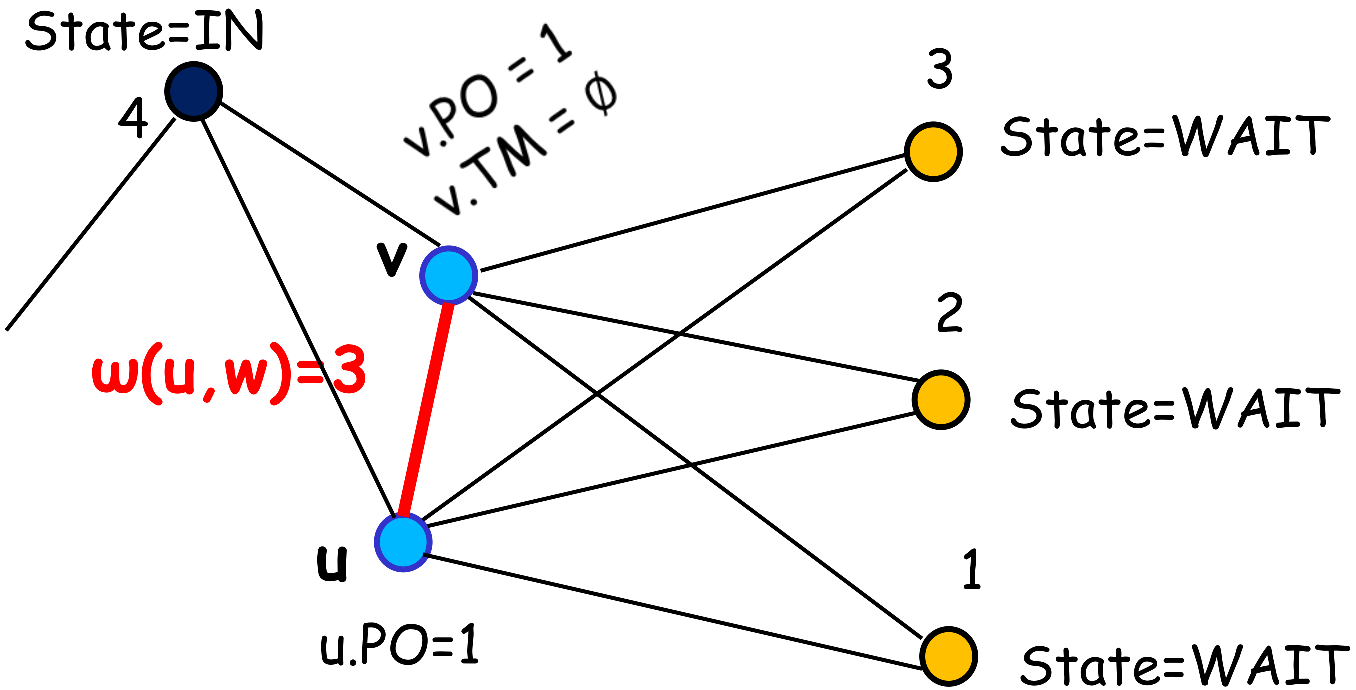
SEMS



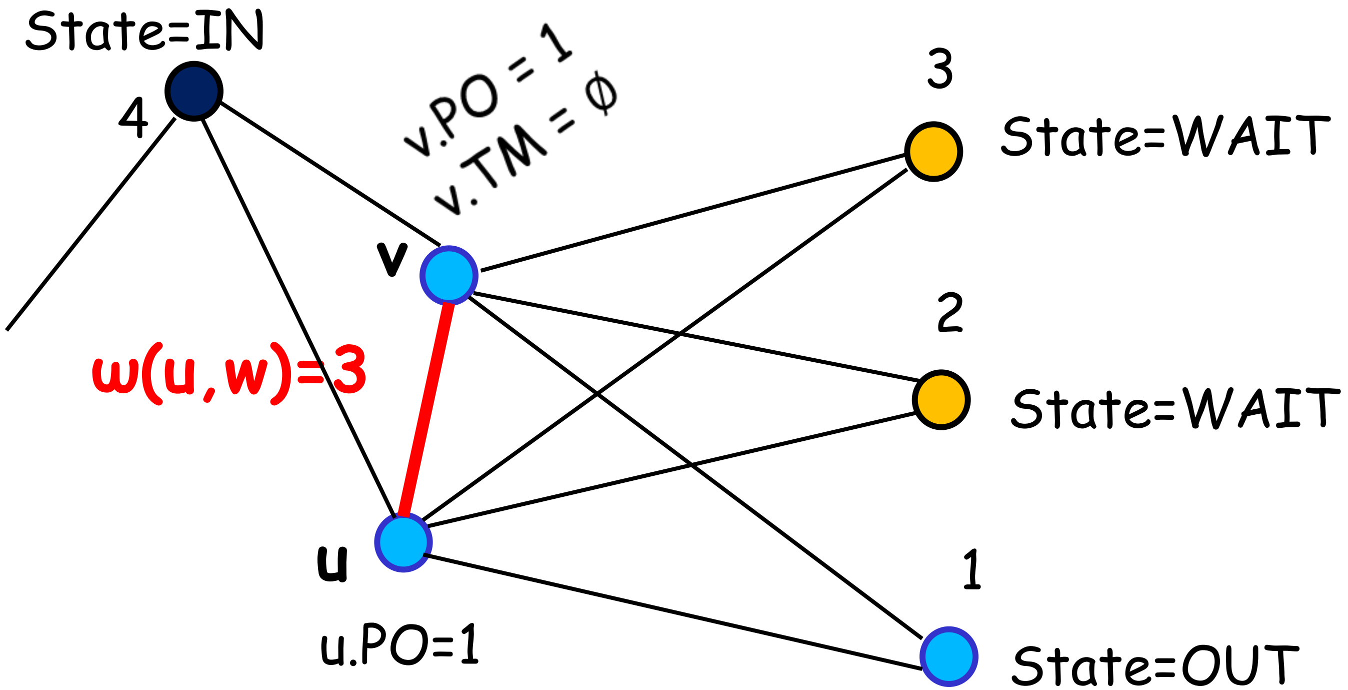
SEMS



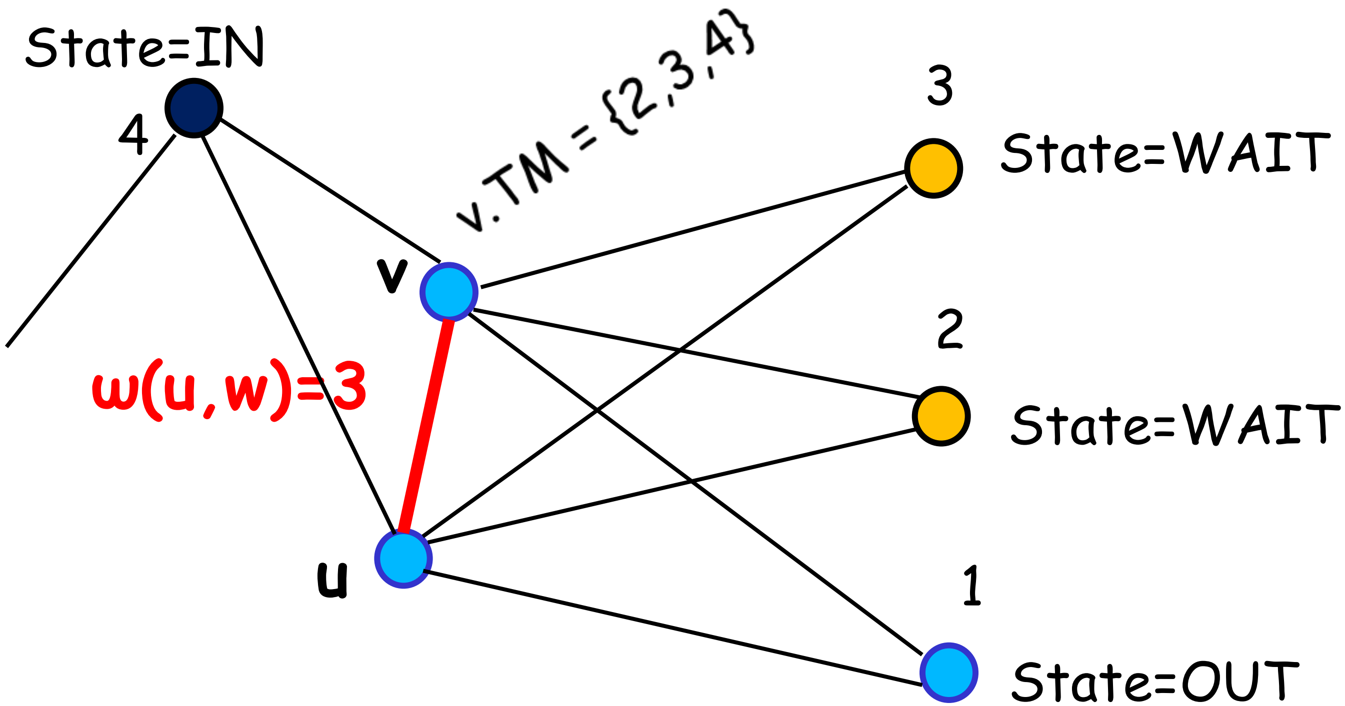
SEMS


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

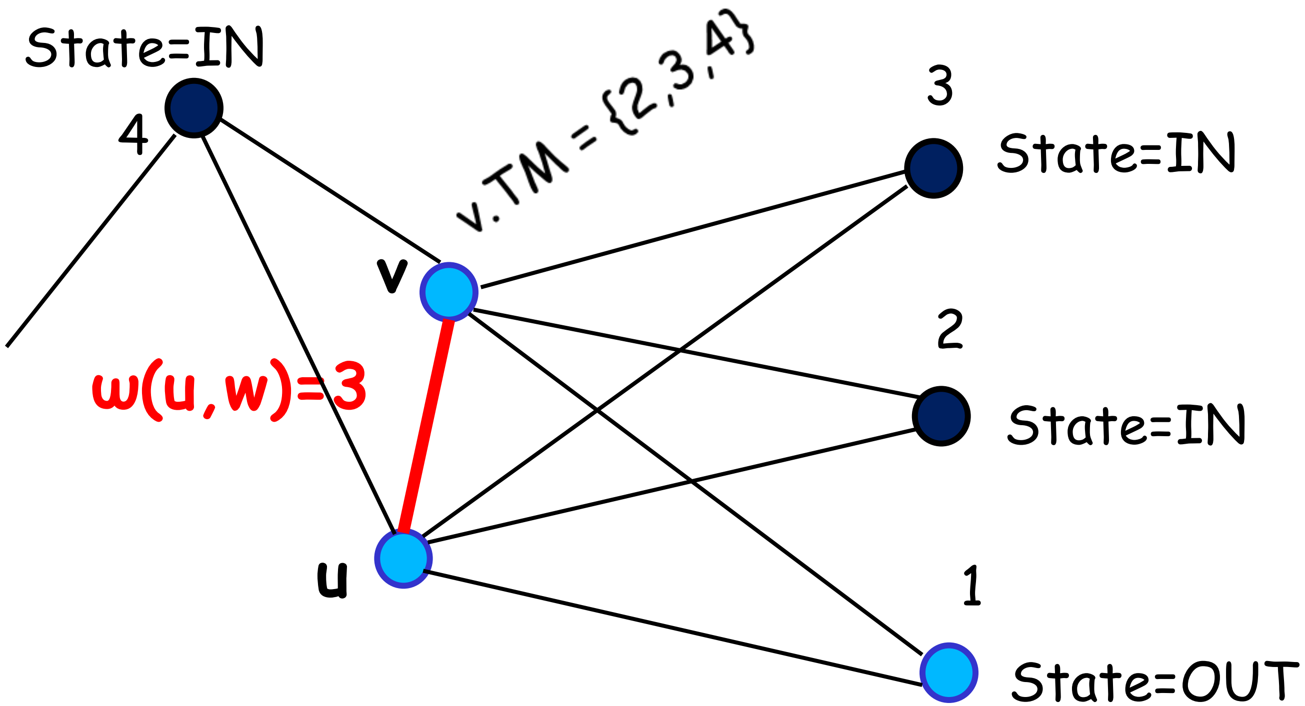
SEMS


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

SEMS



SEMS



SEMS

Variables for each node **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]

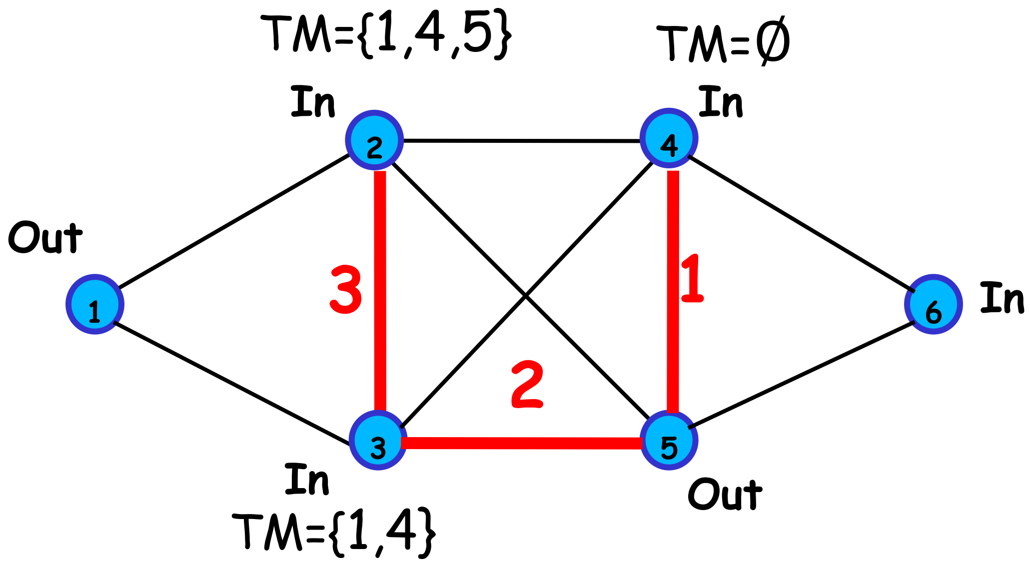
$state = Wait \wedge \exists u \in N(v) : v \in u.TM \longrightarrow state := In;$ [R5]

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

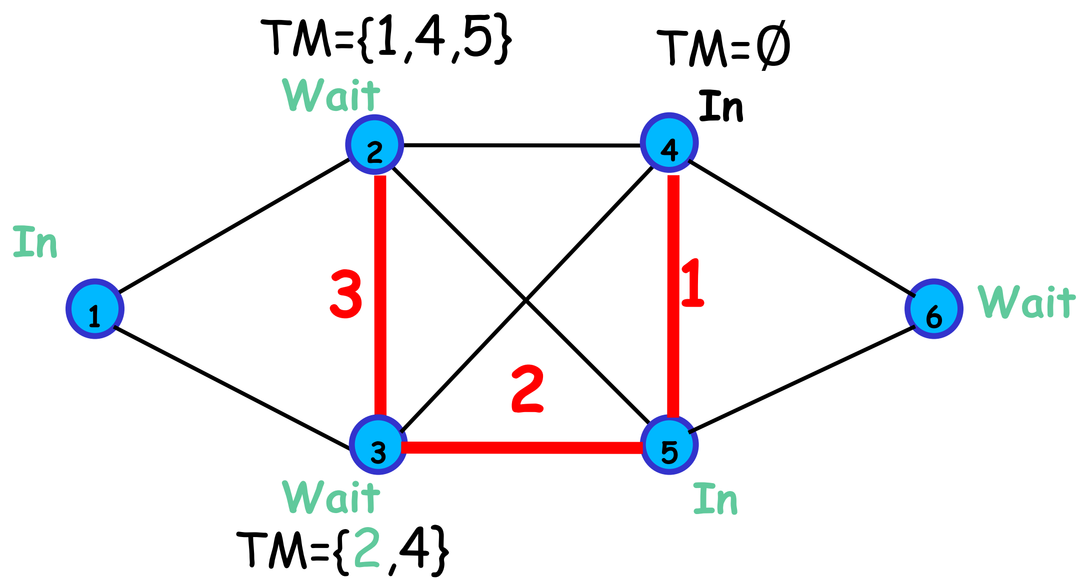
Example with corrupted state

To simplify the example, we consider the synchronous scheduler

SEMS

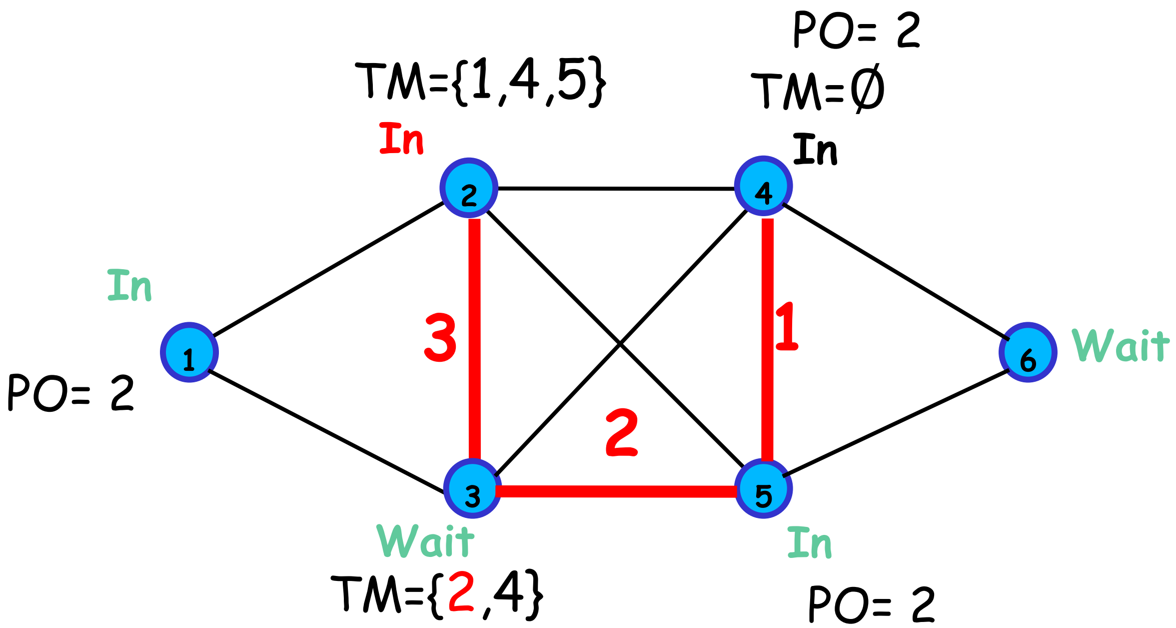


SEMS



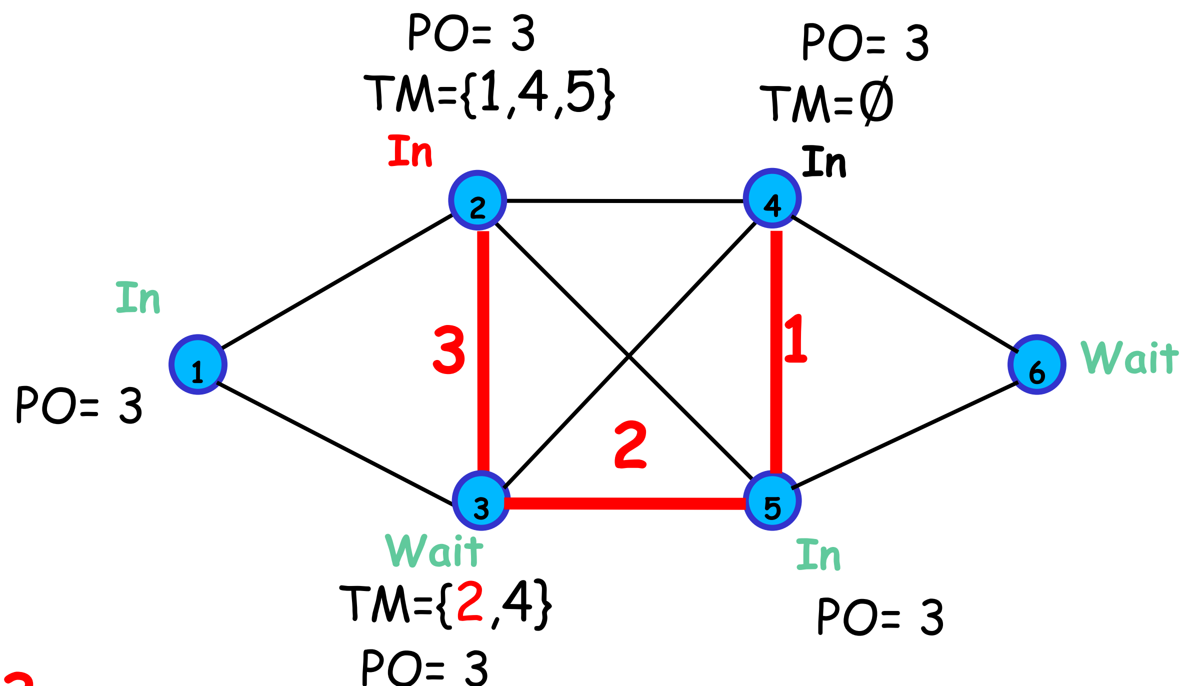
Step 1

SEMS



Step 2

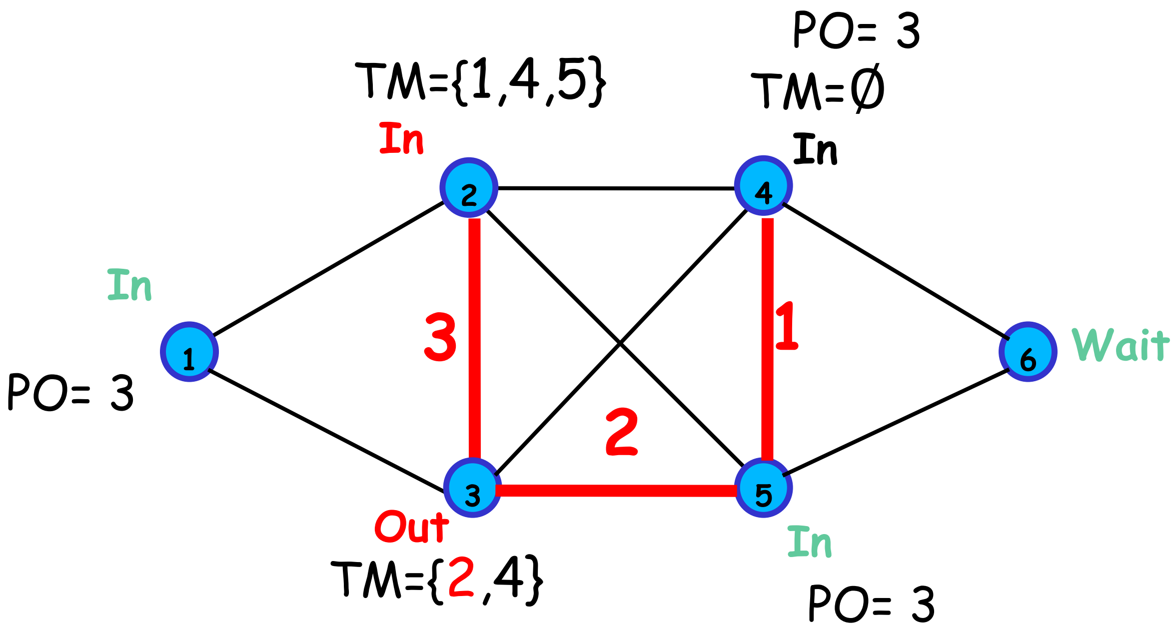
SEMS



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

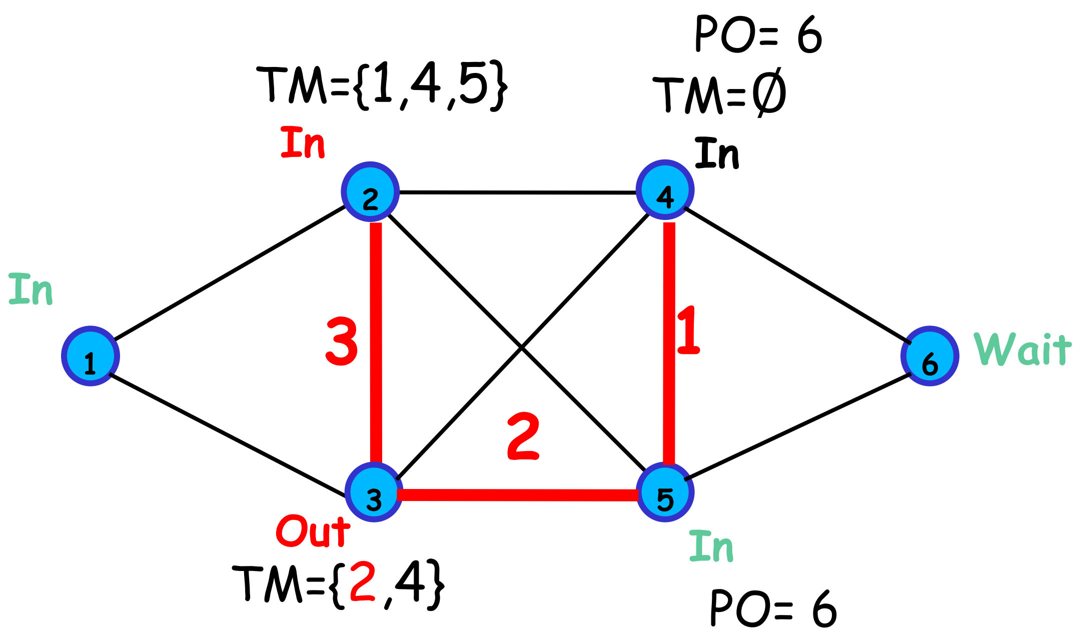
Step 3

SEMS



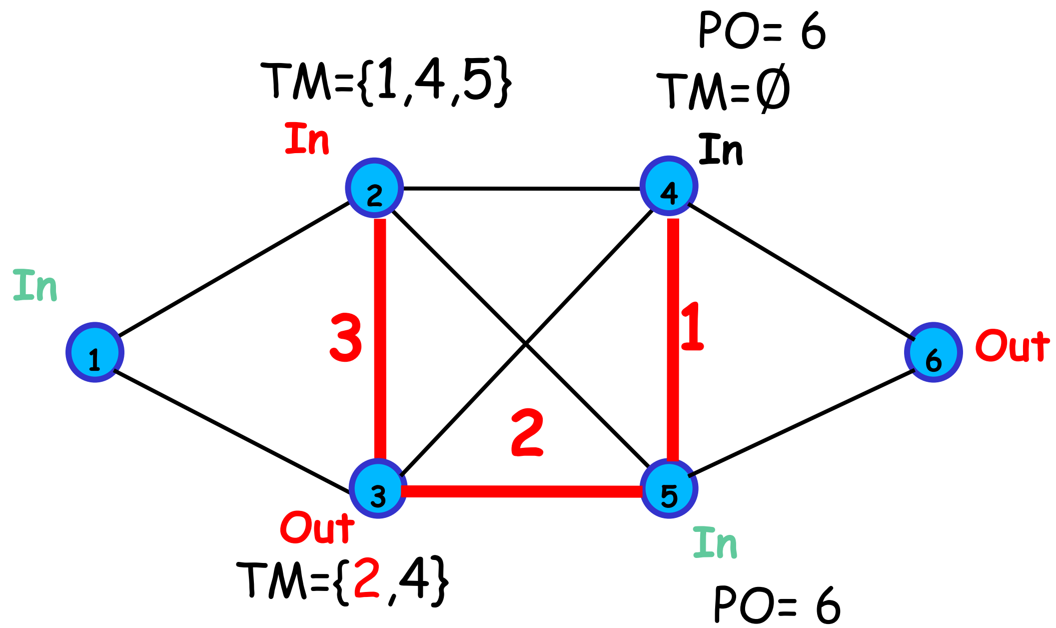
Step 4

SEMS



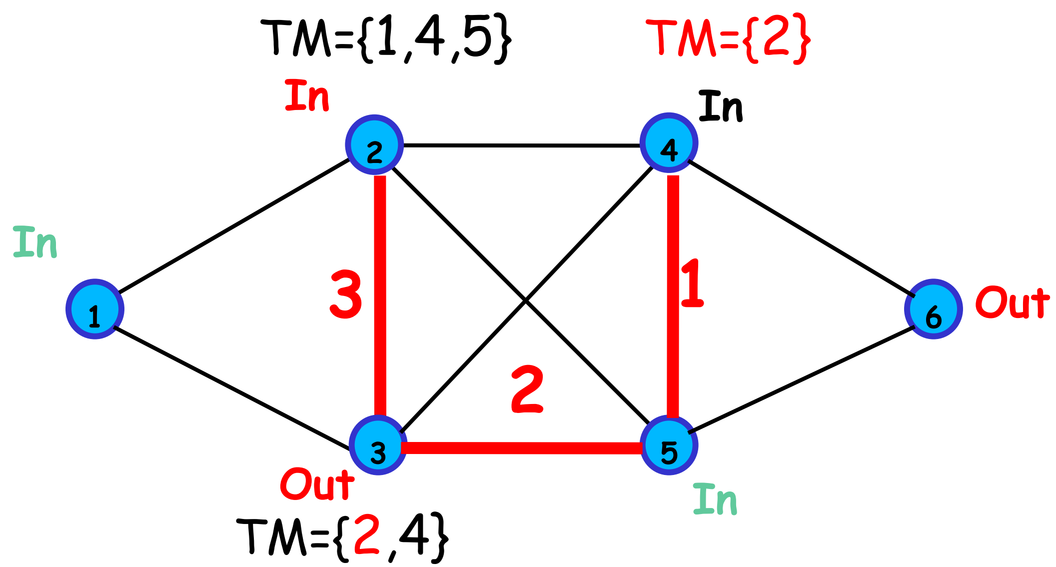
Step 5

SEMS



Step 6

SEMS



Step 7

Final configuration

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)

•No transformer

Conclusions & future work

Future work

- We believe that complexity of algorithm is lower than $O(\Delta^2 m)$. Conjecture: $O(\Delta m)$
- Study lower bounds of the problem for distributed scheduler