Efficient Flooding in Ad Hoc Networks

Seminar: Pervasive Computing (SS 2004)

Frank Radmacher

References

[1] **Sze-Yao Ni, Yu-Chee Tseng, Yuh shyan Chen, and Jang-Ping Sheu.**
The Broadcast Storm Problem in a Mobile Ad Hoc Network.
***ACM MobiCom*, 1999.**

[2] **Jie Wu and Fei Dai.**
Broadcasting in Ad Hoc Networks Based on Self-Pruning.
***IEEE Infocom*, 2003.**

[3] Hyojun Lim and Chongkwon Kim. Flooding in Wireless Ad Hoc Networks. *Computer Communications 24(3-4)*, 2001.

[4] Yu-Chee Tseng, Sze-Yao Ni, and En-Yu Shih.
Adaptive Approaches to Relieving Broadcast Storms in a Wireless Multihop Mobile Ad Hoc Network. *IEEE Infocom*, 2001.

[5] Andrew S. Tanenbaum. *Computer Networks, Fourth Edition*.
Prentice Hall PTR, 2002.

Contents

- Introduction to Mobile Ad Hoc Networks
- The Broadcast Storm Problem
- Self-Pruning
- Simulation Results
- Conclusion

Mobile Ad Hoc Networks (MANETs)

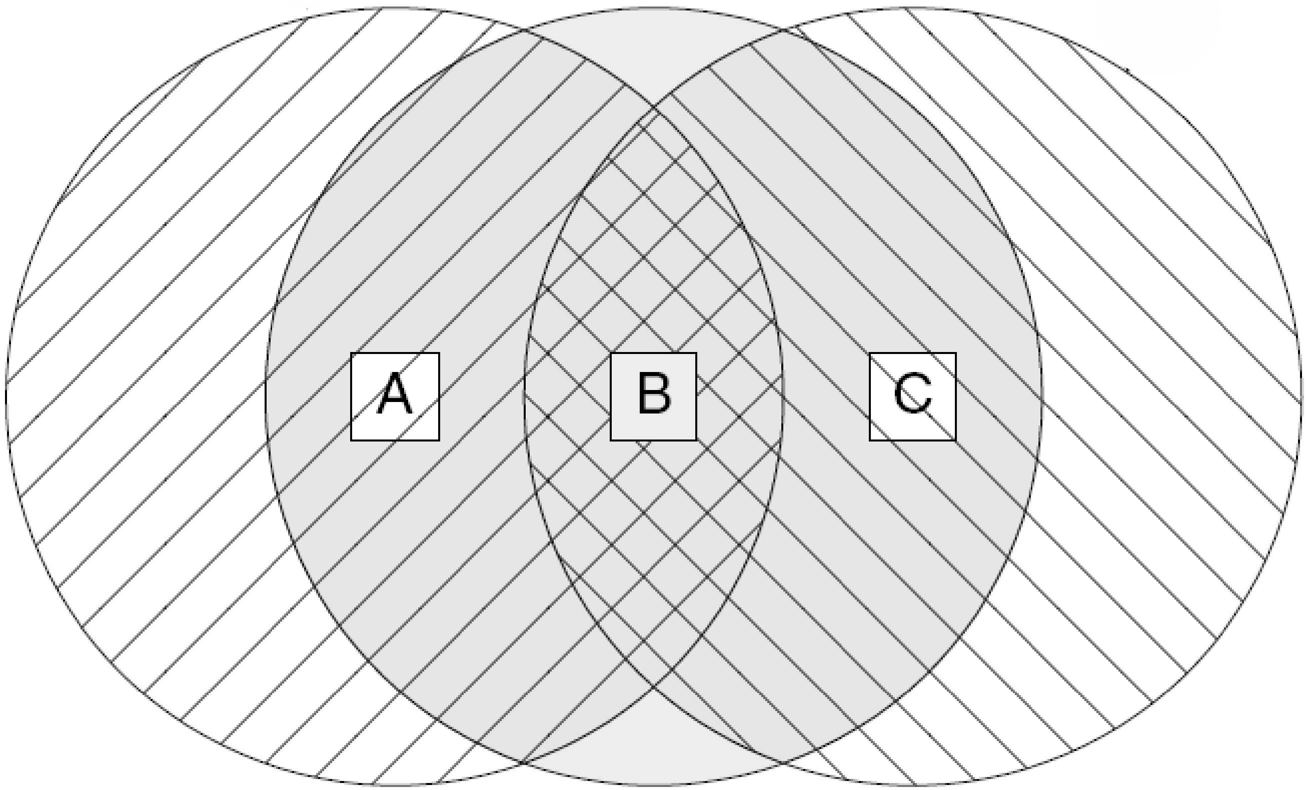
- Consist of wireless mobile hosts which form a temporary network
  - without the aid of established infrastructure (e. g. base stations)
  - without centralised administration (e. g. mobile switching centers)
- Every host in a MANET
  - can roam around freely
  - can only communicate with hosts which are currently in its transmission range
  - ➥ Multi-hop scenario: Packets must be forwarded to their destination

Mobile Ad Hoc Networks (MANETs)

- Consist of wireless mobile hosts which form a temporary network
  - without the aid of established infrastructure (e. g. base stations)
  - without centralised administration (e. g. mobile switching centers)

- Every host in a MANET
  - can roam around freely
  - can only communicate with hosts which are currently in its transmission range
  - ➥ Multi-hop scenario: Packets must be forwarded to their destination

Multi-Hop Scenario



The Broadcast Storm Problem

- Straightforward realisation of global broadcasting in a MANET
  - Simple Flooding:
    Every host retransmits a received broadcast message once.

- This leads to the so called *Broadcast Storm Problem* consisting of
  - Redundancy
  - Contention
  - Collision

The Broadcast Storm Problem

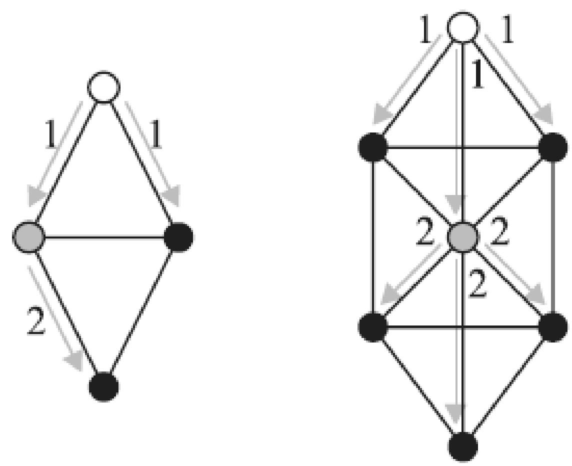
- Straightforward realisation of global broadcasting in a MANET
  - ➥ Simple Flooding:
    Every host retransmits a received broadcast message once.

- This leads to the so called *Broadcast Storm Problem* consisting of
  - Redundancy
  - Contention
  - Collision

Redundancy (1)

- Problem:
  When a mobile host retransmits a broadcast message, all its neighbors might already have received this message.

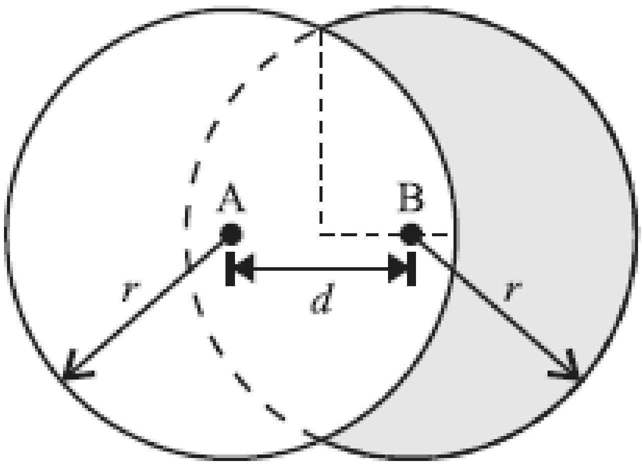
  ➥ The bandwidth of the network gets reduced by unnecessary broadcasts.



(a) (b)

Redundancy (2)

- We are interested in the additional coverage of a node (grey shaded area)



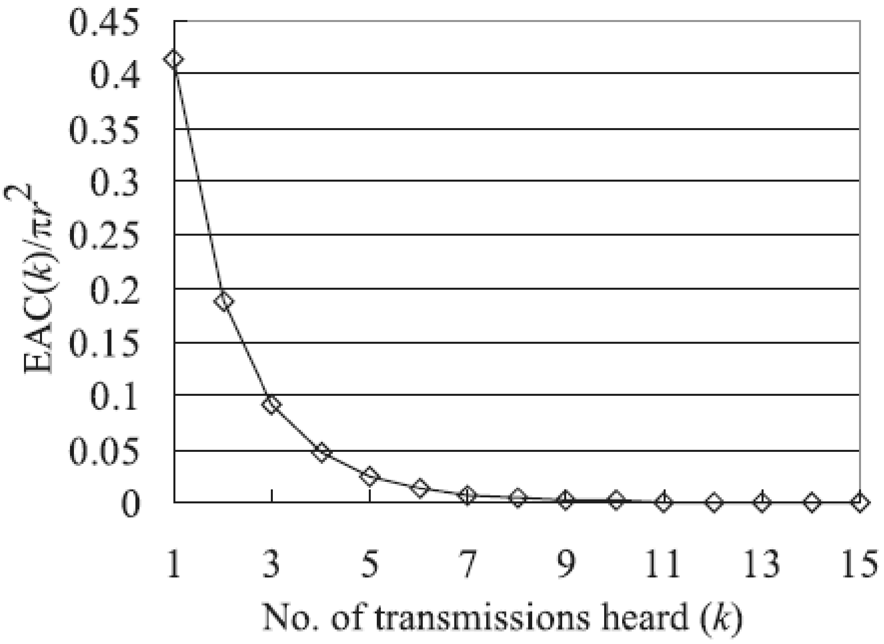
- The additional coverage of *B*: $\pi r^2 - \text{INTC}(d)$

  where $\text{INTC}(d) = 4 \int_{d/2}^{r} \sqrt{r^2 - x^2} dx$

- Expected additional coverage of a node: $\int_0^r \frac{2\pi x \cdot [\pi r^2 - \text{INTC}(x)]}{\pi r^2} dx \approx 0.41 \pi r^2$

Redundancy (3)

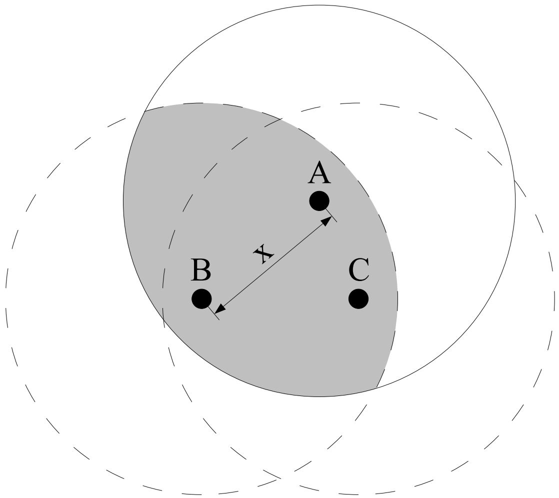
- If a host received a broadcast message from more than one host, the expected additional coverage decreases.
- Expected additional coverage $\text{EAC}(k)$ of a host after receiving a broadcast k times:



➥ Many rebroadcasts are superfluous in the case of simple flooding.

Contention (1)

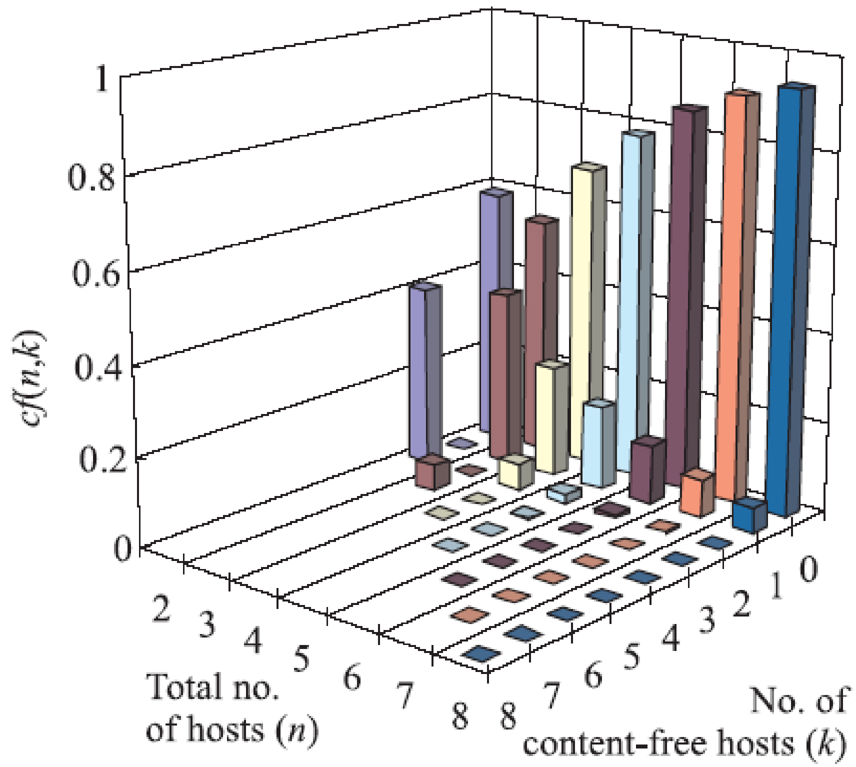
- Problem:
  If n nearby hosts try to rebroadcast a message nearly the same time, they are likely to compete with each other.
- Simple case of $n = 2$:



- The probability of contention is $\mathsf{INTC}(x)/\pi r^2$
- For arbitrarily located *B*'s: $\int_0^r \frac{2\pi x \cdot \mathsf{INTC}(x)/(\pi r^2)}{\pi r^2} dx \approx 59\%$

Contention (2)

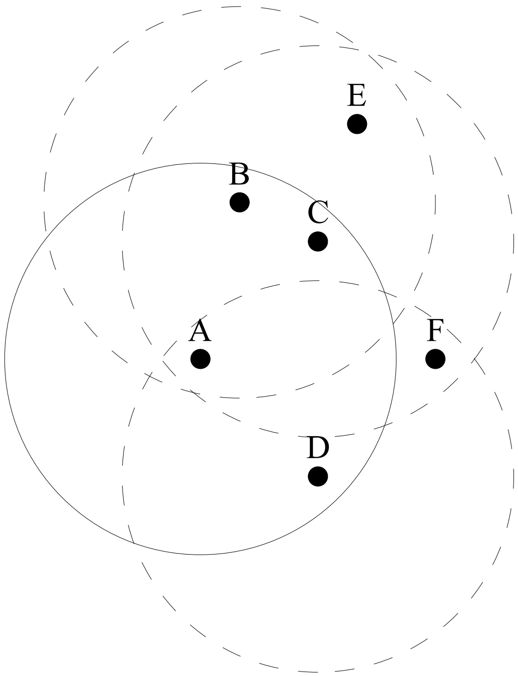
- The probability $cf(n,k)$ of having k contention-free host among n receiving hosts:



➥ Contention is likely to occur, especially in dense networks.

Collision

- Problem:
  Broadcast messages are rather sent simultaneously, such that collisions get more probable.



- Reason:
  CSMA/CA style communication
  - without RTS/CTS dialogues
  - without acknowledgement packets

- Two problems:
  - two hosts decide to transmit a message at around the same time
  - the hidden station problem

Observation

- Redundancy, Contention, Collision are serious problems.
- All problems have one cause in common:

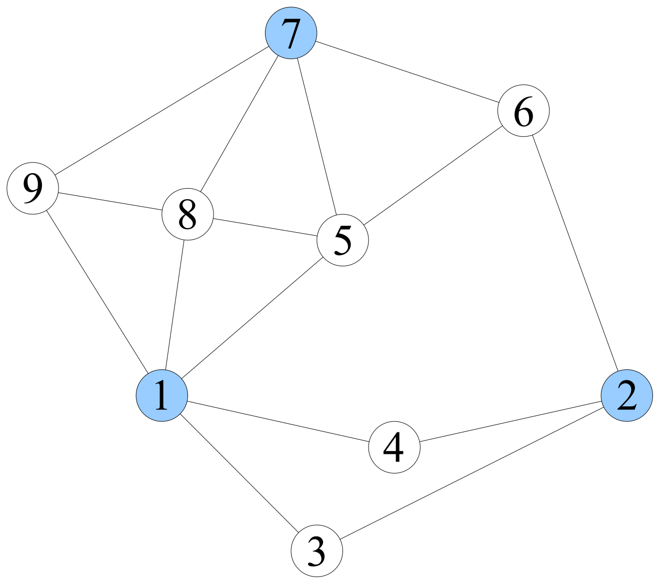
  They increase with the number of hosts which unnecessarily rebroadcast a message.

- Solution:
  Inhibit some nodes in the MANET from rebroadcasting.

➥ Select a *forward node set*

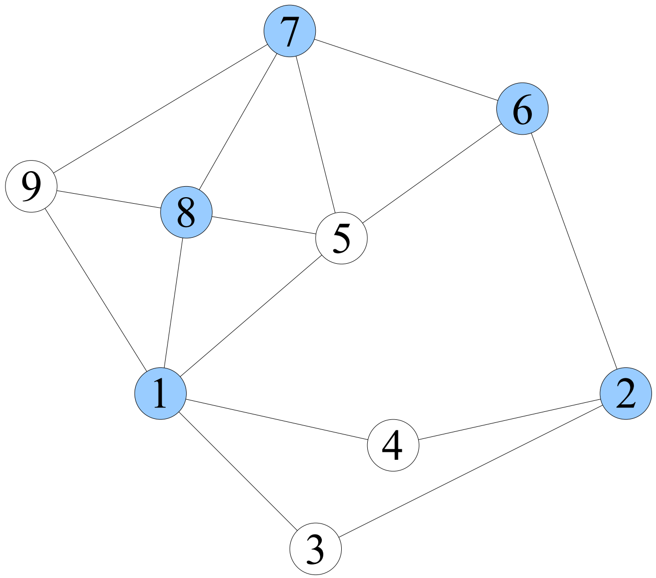
Introduction to Self-Pruning (1)

- Self-Pruning: Every node decides on its own whether to forward a message or not.
- A forward node set has to form a *connected dominating set*.
  - A set A of nodes is called *dominating set* of a graph G, if every node is either in the set or has a neighbor in the set.
  - dominating set:



Introduction to Self-Pruning (1)

- Self-Pruning: Every node decides on its own whether to forward a message or not.
- A forward node set has to form a *connected dominating set*.
  - A set A of nodes is called *dominating set* of a graph G, if every node is either in the set or has a neighbor in the set.
  - connected dominating set (CDS):

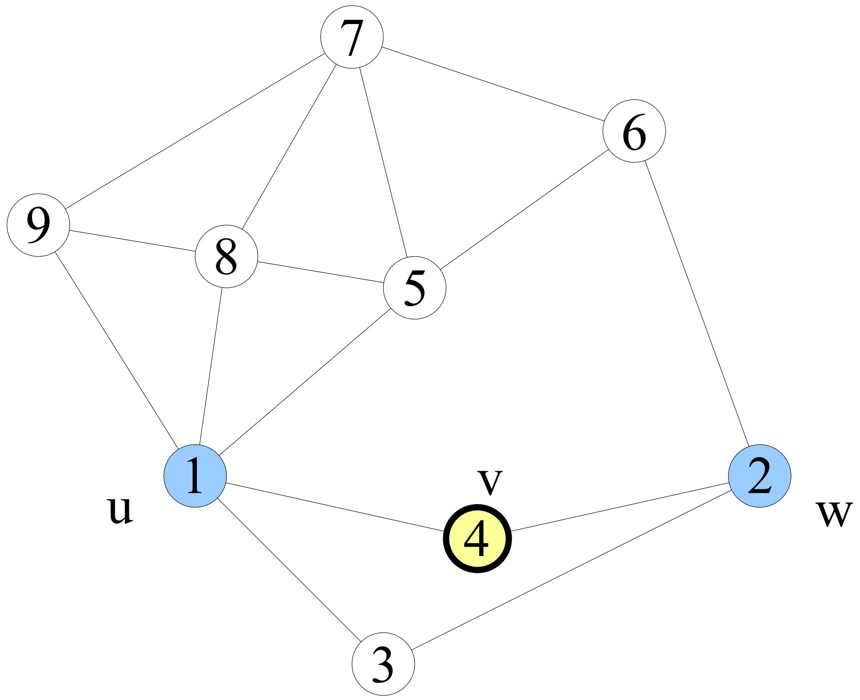


Introduction to Self-Pruning (2)

- Ideal forward node set:
  minimum connected dominating set (MCDS).
- A *minimum connected dominating set* (MCDS) is a connected dominating set (CDS) with a minimal number of nodes.
- But:
  - MCDS problem is NP complete.
  - Global network information is needed for computation.
  - ➥ Define *coverage condition* which only results in a nearly optimal CDS but is suitable for computation.

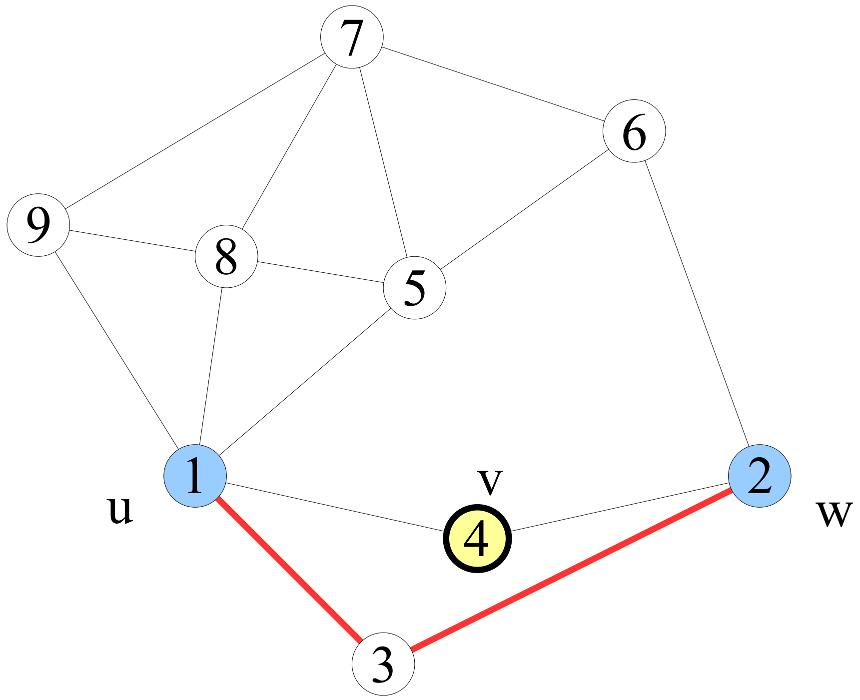
Coverage Condition I

- **Coverage Condition I:**
  Node v has a non-forward node status if for any two neighbors u and w, a *replacement path* exists that connects u and w via several intermediate nodes (if any) with higher priority values than the priority value of v.



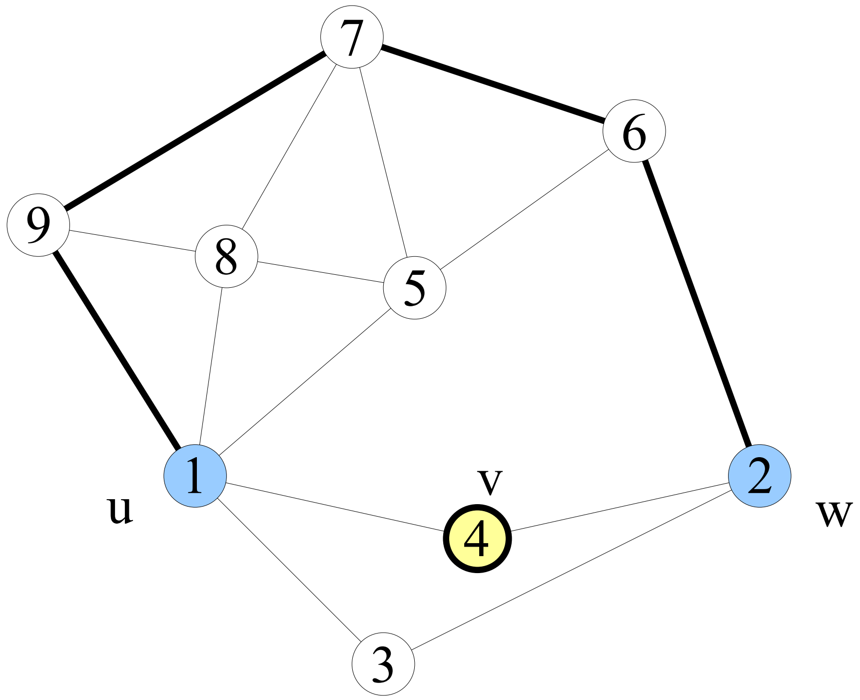
Coverage Condition I

- **Coverage Condition I:**
  Node v has a non-forward node status if for any two neighbors u and w, a *replacement path* exists that connects u and w via several intermediate nodes (if any) with higher priority values than the priority value of v.



Coverage Condition I

- **Coverage Condition I:**
  Node v has a non-forward node status if for any two neighbors u and w, a *replacement path* exists that connects u and w via several intermediate nodes (if any) with higher priority values than the priority value of v.



Coverage Condition I

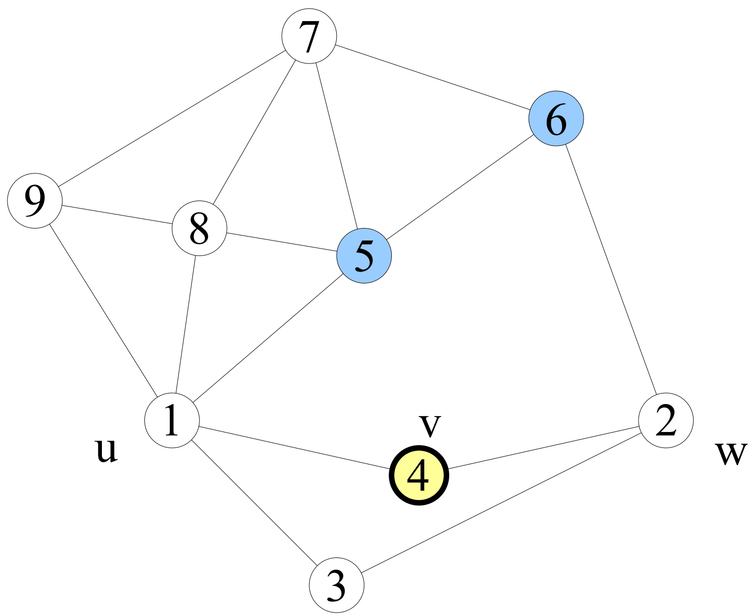
- Disadvantage of Coverage Condition I:
  - Every node has to check the condition for every pair of neighbors.
  - There are $\binom{\deg(v)}{2} \in O(\deg(v)^2)$ such pairs
  - ➥ Overall computation complexity: $O(n\Delta^2)$

    n – number of nodes
    Δ – maximum vertex degree

Coverage Condition II

- **Coverage Condition II:**
  Node v has a non-forward node status if it has a coverage set. In addition the coverage set belongs to a connected component of the subgraph induced from nodes with higher priority values than the priority value of v.

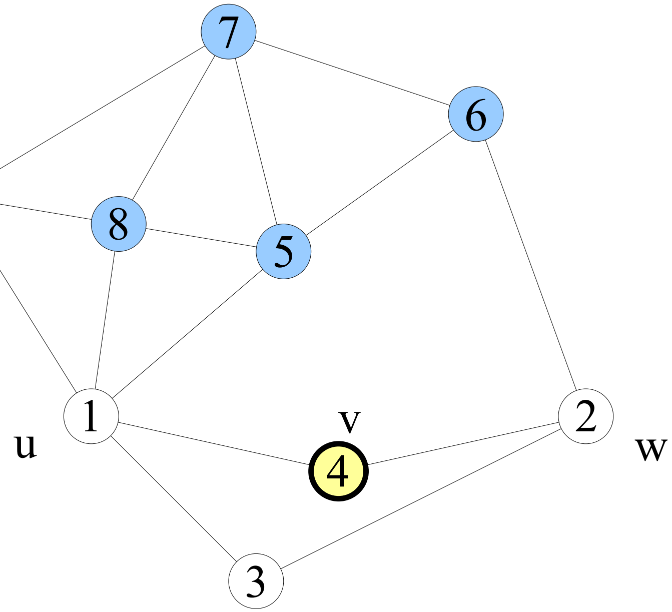
- A set $C(v)$ is called a *coverage set* of v if the neighbor set of v can be *covered* by nodes in $C(v)$.



Coverage Condition II

- **Coverage Condition II:**
  Node v has a non-forward node status if it has a coverage set. In addition the coverage set belongs to a connected component of the subgraph induced from nodes with higher priority values than the priority value of v.

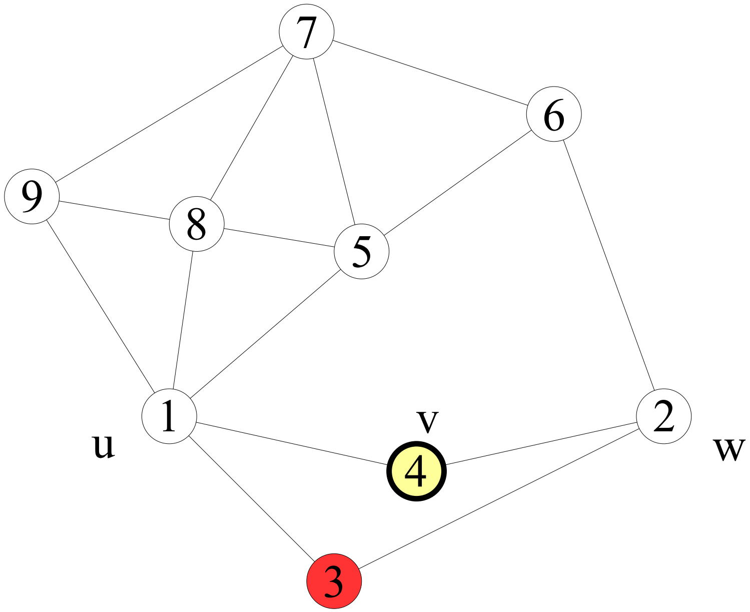
- A set $C(v)$ is called a *coverage set* of v if the neighbor set of v can be *covered* by nodes in $C(v)$.



Coverage Condition II

- **Coverage Condition II:**
  Node v has a non-forward node status if it has a coverage set. In addition the coverage set belongs to a connected component of the subgraph induced from nodes with higher priority values than the priority value of v.

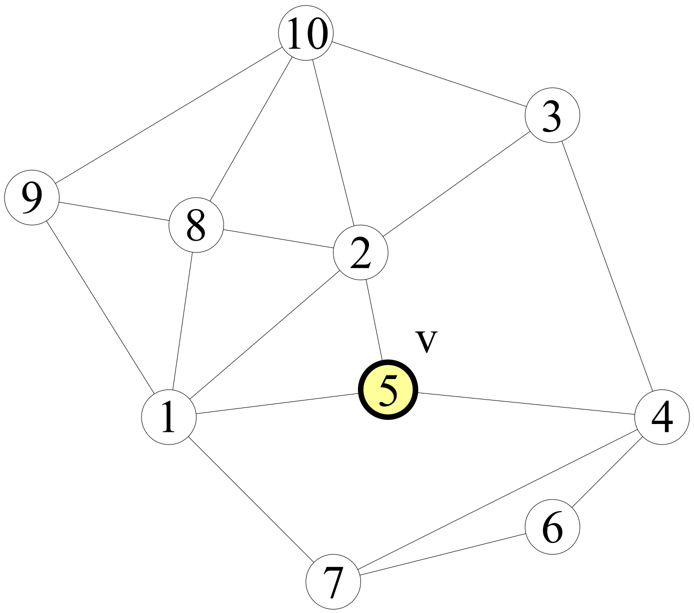
- A set $C(v)$ is called a *coverage set* of v if the neighbor set of v can be *covered* by nodes in $C(v)$.



Coverage Condition II

- Computation:
  - Decompose the graph into connected components $V_1, V_2, \ldots, V_l$ that only contain nodes with a higher priority than v via *depth-first search*. ($O(n\Delta)$)
  - Compute for each V_i the set of covered neighbors $N(V_i) := \bigcup_{w \in V_i} N(w)$ and check if there exists a V_i such that $N(v) \subseteq N(V_i)$. ($O(n\Delta)$)

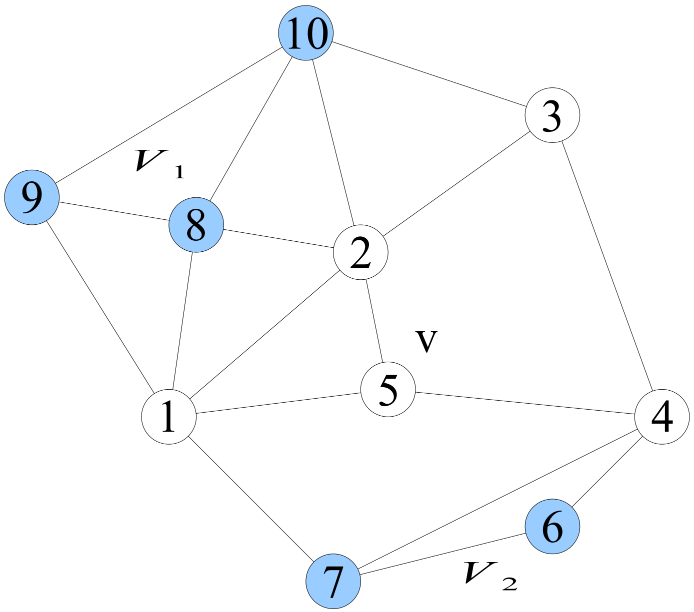
➥ Overall computation complexity: $O(n\Delta)$



Coverage Condition II

- Computation:
  - Decompose the graph into connected components $V_1, V_2, \ldots, V_l$ that only contain nodes with a higher priority than v via *depth-first search*. ($O(n\Delta)$)
  - Compute for each V_i the set of covered neighbors $N(V_i) := \bigcup_{w \in V_i} N(w)$ and check if there exists a V_i such that $N(v) \subseteq N(V_i)$. ($O(n\Delta)$)

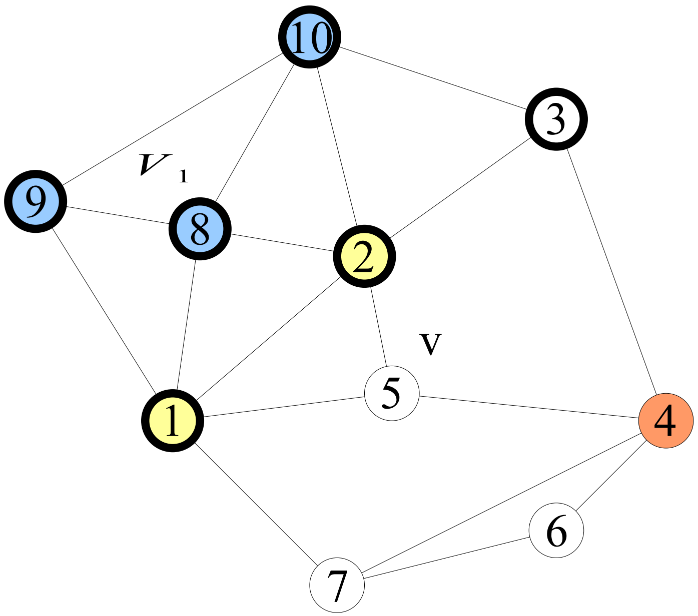
➥ Overall computation complexity: $O(n\Delta)$



Coverage Condition II

- Computation:
  - Decompose the graph into connected components $V_1, V_2, \ldots, V_l$ that only contain nodes with a higher priority than v via *depth-first search*. ($O(n\Delta)$)
  - Compute for each V_i the set of covered neighbors $N(V_i) := \bigcup_{w \in V_i} N(w)$ and check if there exists a V_i such that $N(v) \subseteq N(V_i)$. ($O(n\Delta)$)

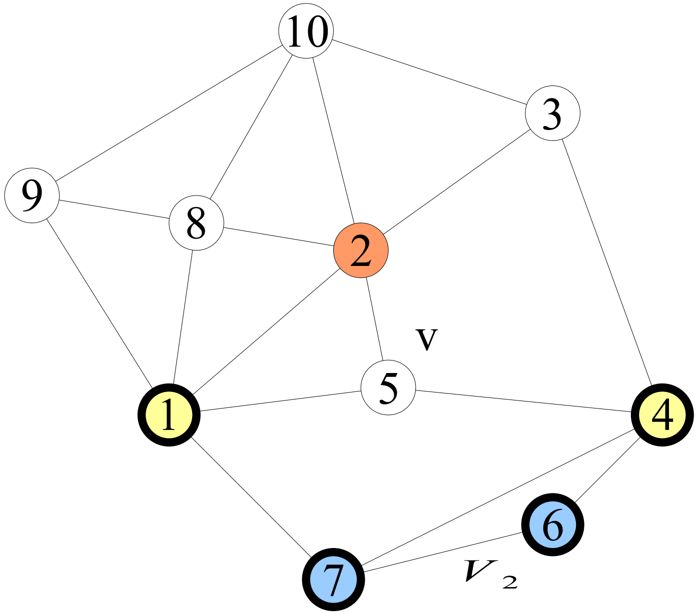
➥ Overall computation complexity: $O(n\Delta)$



Coverage Condition II

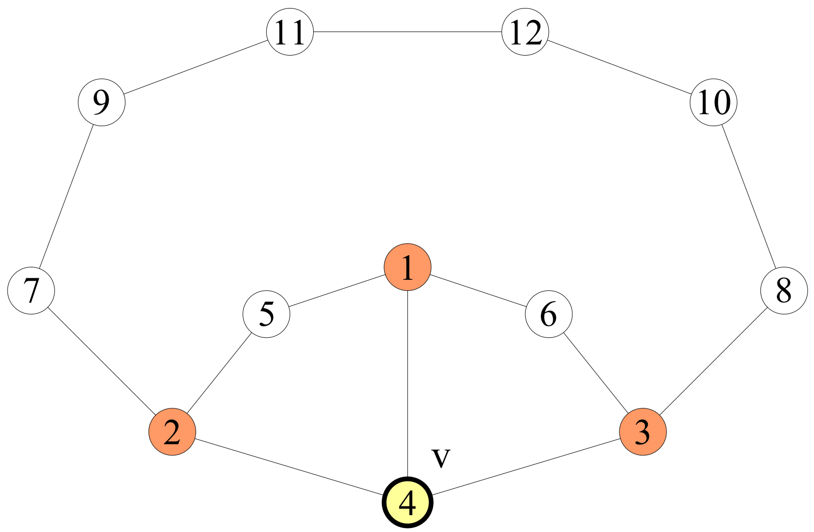
- Computation:
  - Decompose the graph into connected components $V_1, V_2, \ldots, V_l$ that only contain nodes with a higher priority than v via *depth-first search*. $(O(n\Delta))$
  - Compute for each V_i the set of covered neighbors $N(V_i) := \bigcup_{w \in V_i} N(w)$ and check if there exists a V_i such that $N(v) \subseteq N(V_i)$. $(O(n\Delta))$

➥ Overall computation complexity: $O(n\Delta)$



Coverage Condition I & II Comparison

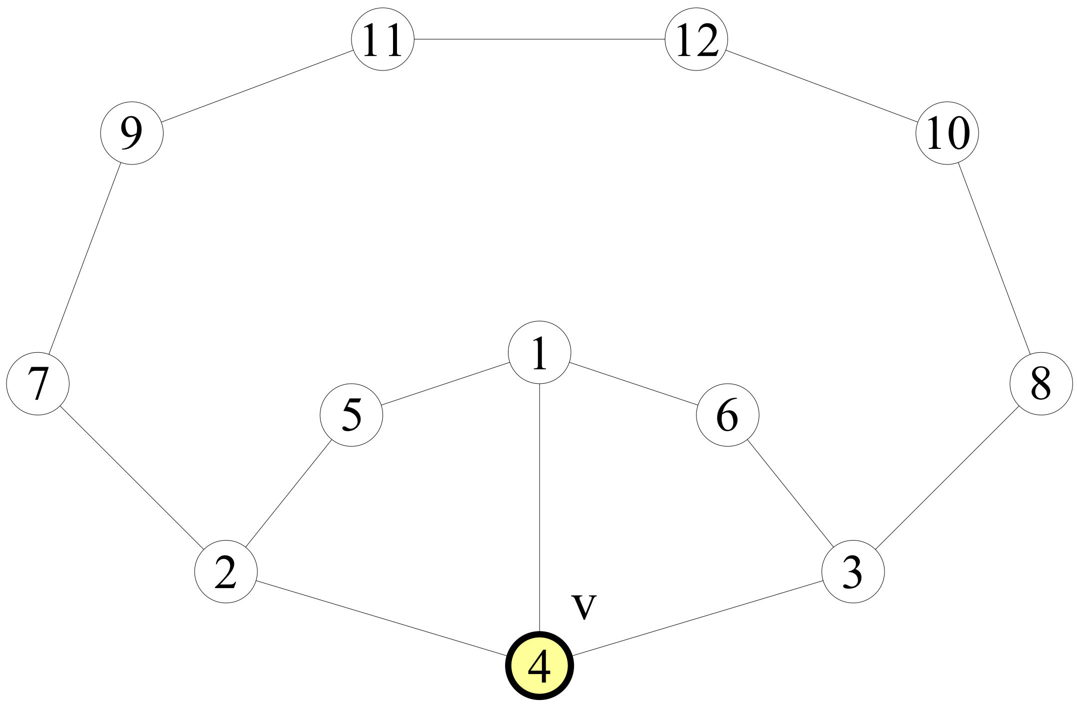
- Coverage condition I is stronger than coverage condition II.
  - The existence of a *connected coverage set* for v implies the existence of a *replacement path* for any pair of v's neighbors.
  - But generally the reverse situation does not hold:



➥ Coverage condition II has a lower computation complexity than coverage condition I but may result in larger forward node sets.

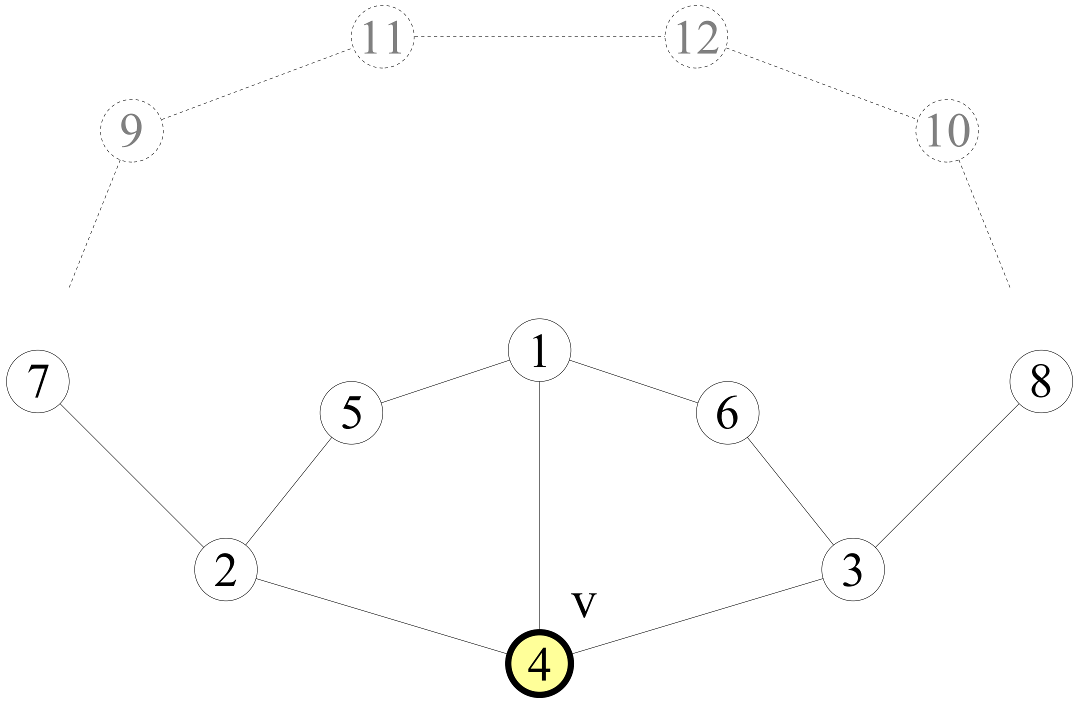
k-Hop Neighbor Set $N_k(v)$

- For deciding whether to be a *forward node* or a *non-forward node*, a node can only use small neighborhood information:
  - ➥ The k-hop neighbor set $N_k(v)$
- $k \geq 5$:



k-Hop Neighbor Set $N_k(v)$

- For deciding whether to be a *forward node* or a *non-forward node*, a node can only use small neighborhood information:
  - ➥ The k-hop neighbor set $N_k(v)$
- $k = 2$:



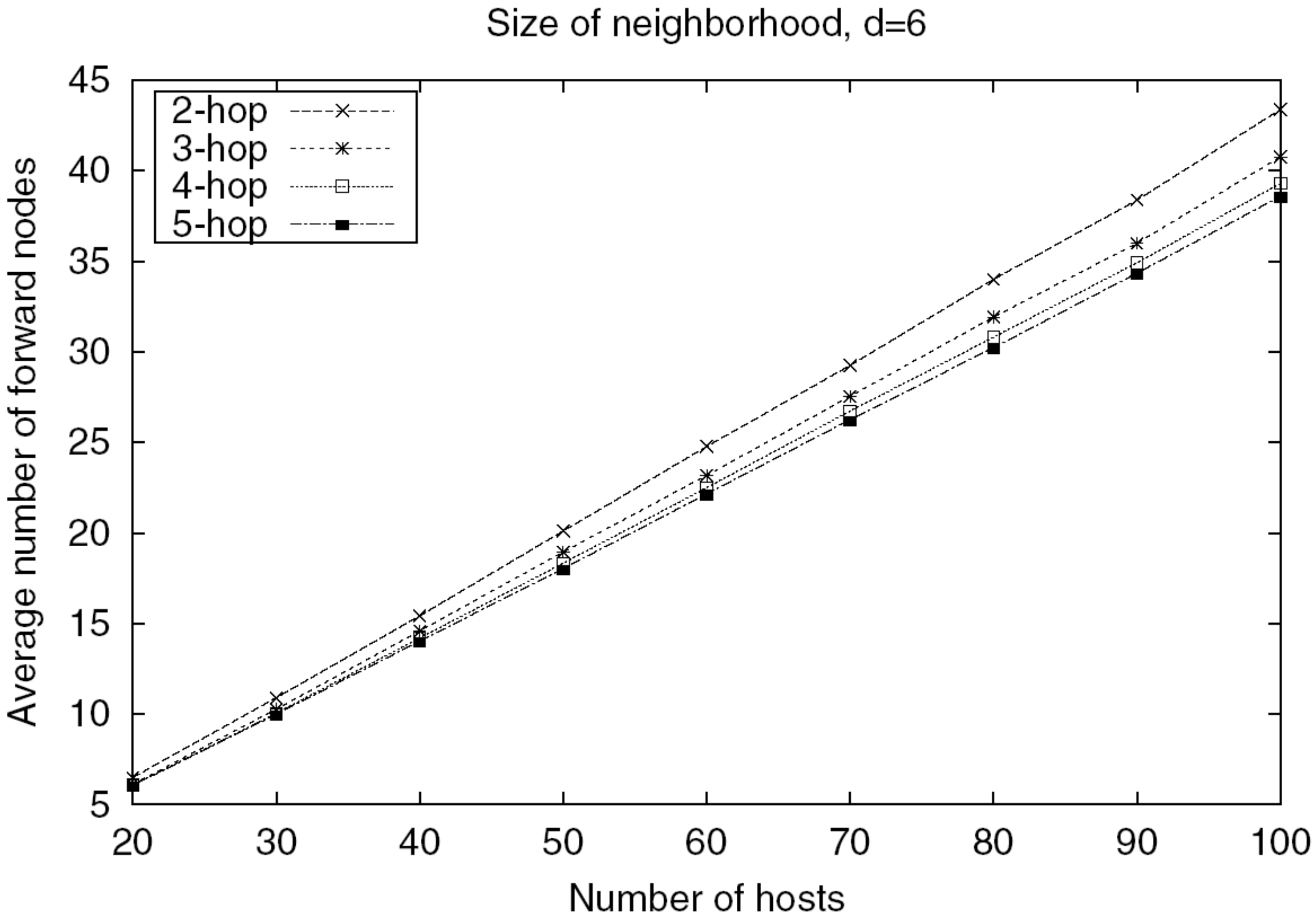
Simulation Setup & Parameters

- Because we are mainly interested in the size of the forward node set, we are assuming an ideal MAC layer without contention or collision.

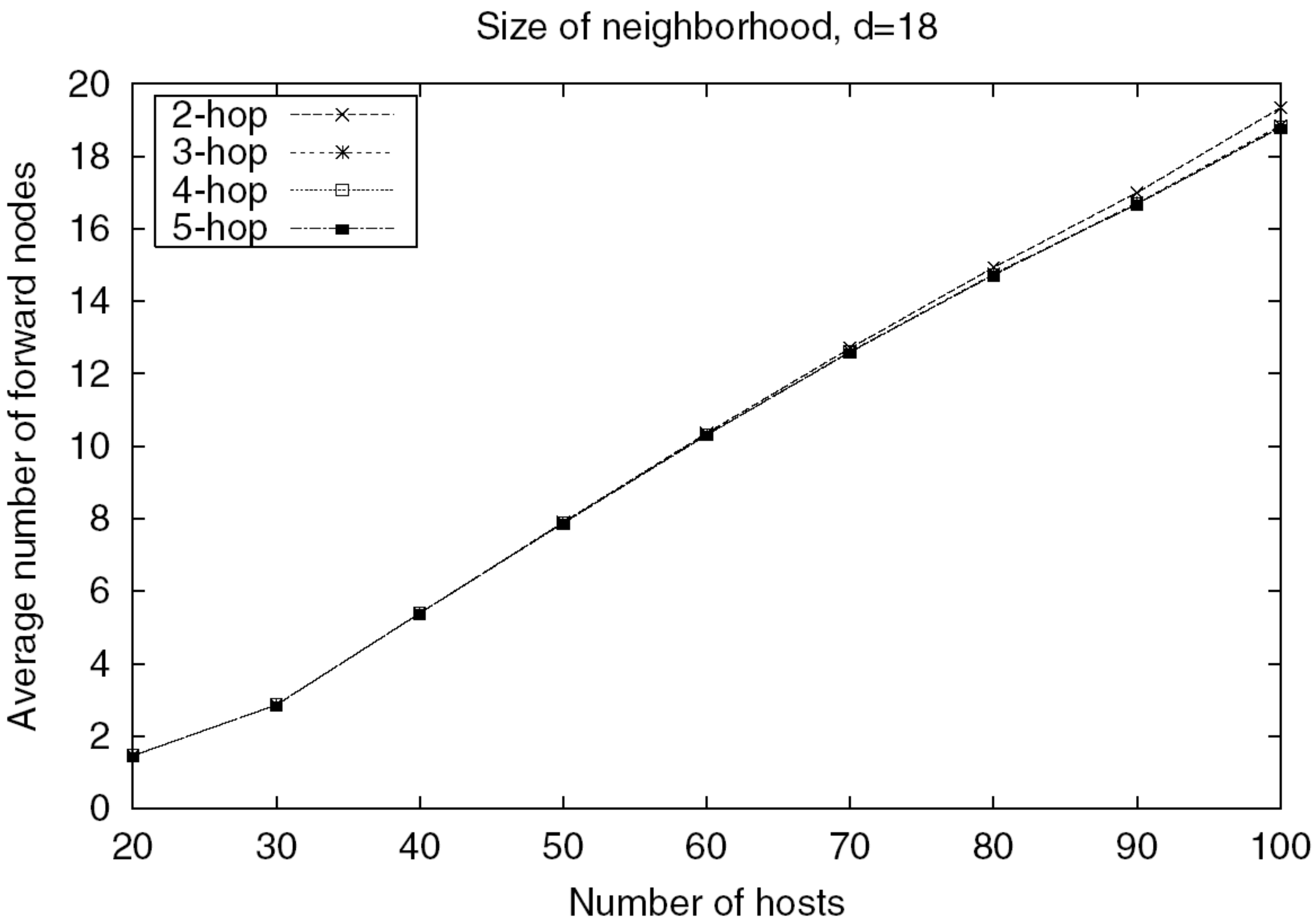
- Simulation parameters:
  - number of hosts n
  - average node degree d (density of the network)

- n hosts placed randomly in a 100×100 area.
- The transmission range r has been adjusted to produce $\frac{nd}{2}$ links.

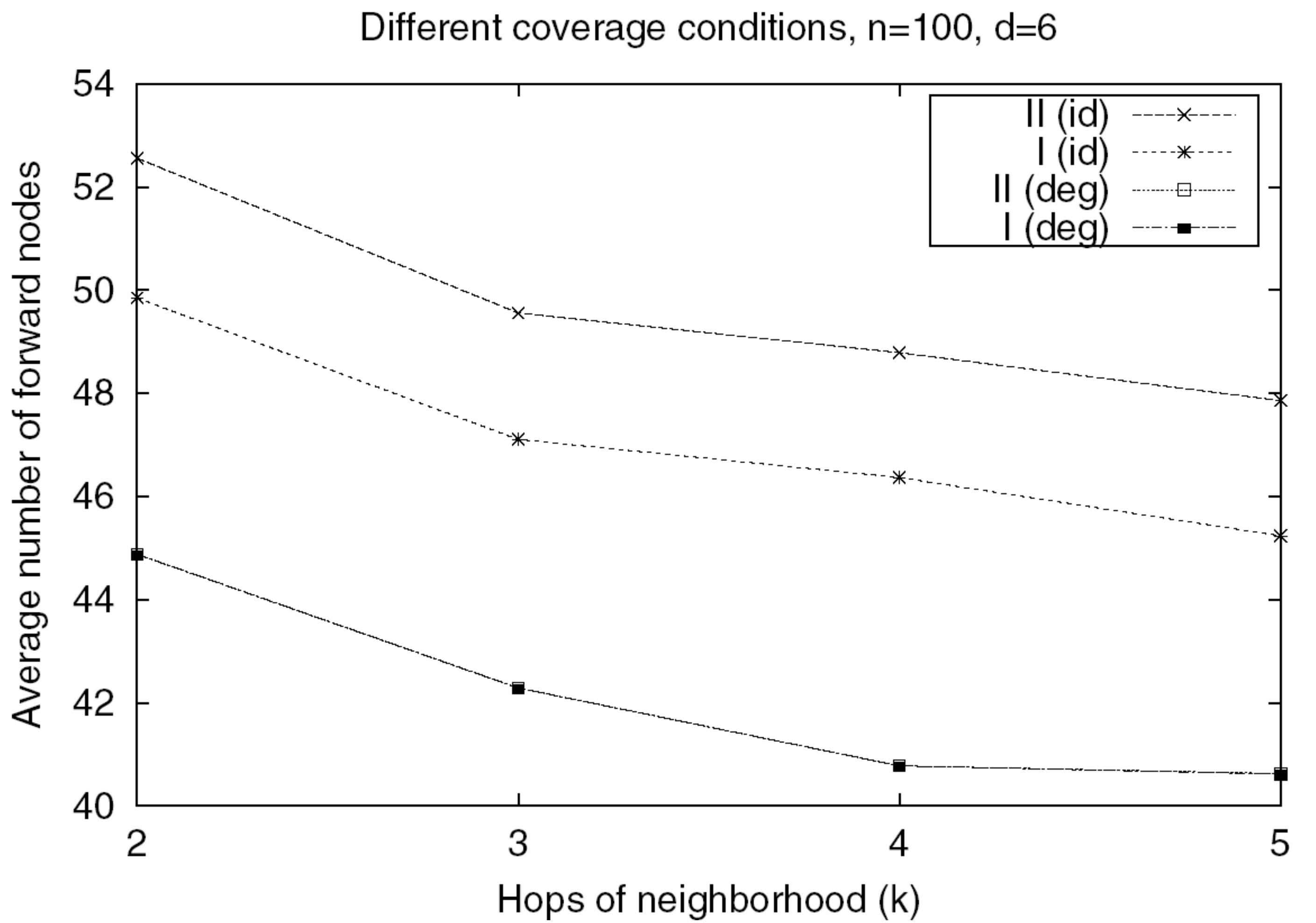
Size k of Neighbor Set (Sparse Network)



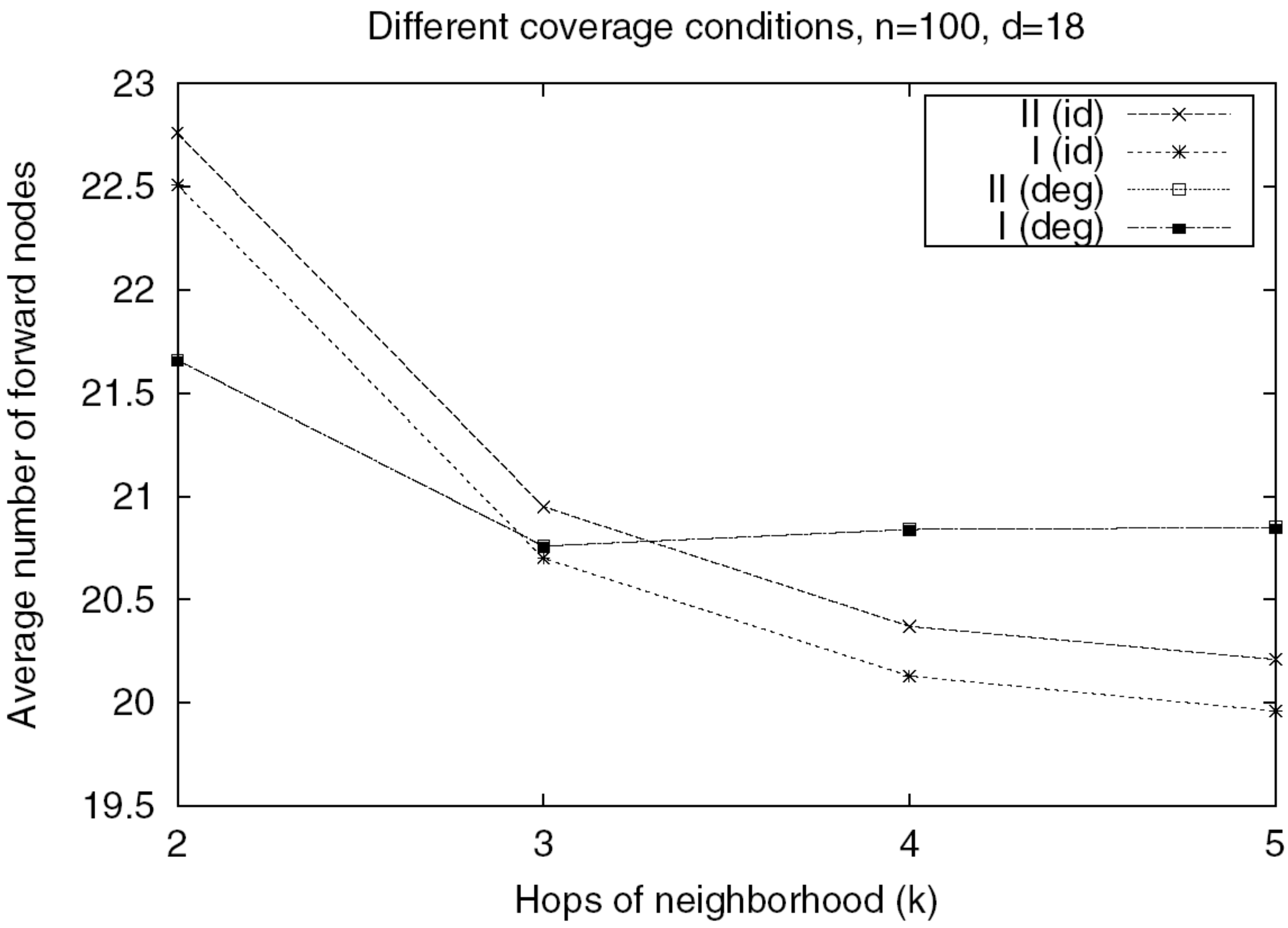
Size k of Neighbor Set (Dense Network)



Type of Coverage Condition (Sparse Network)



Type of Coverage Condition (Dense Network)



Summary

What we have learned today:

- Basics of Mobile Ad Hoc Networks (MANETs)
- The Broadcast Storm Problem:
  - Redundancy
  - Contention
  - Collision
- How to avoid these problems:
  - Generic approach based on Self-Pruning
    - coverage conditions as approximation of a MCDS
  - ➥ Through simulation results we obtain a suitable configuration.

✌ Thank you for your attention.

Applications

- scientific use
  - sensor networks
  - archaeological or ecological expeditions
- civilian use
  - disaster recovery
  - search and rescue
- military use
  - battlefield

Why Broadcasting in a MANET?

- Broadcasts are common operations in MANETs
- Necessary for solving particular tasks in a MANET
  - sending alarm signals
  - paging particular hosts
  - possible last resort realisation of uni- and multicast messages in networks with a rapidly changing topology
  - many routing protocols use broadcasts to exchange routing information
  - ➥ Due to the dynamic topology in MANETs, we expect broadcasts to occur more frequently.

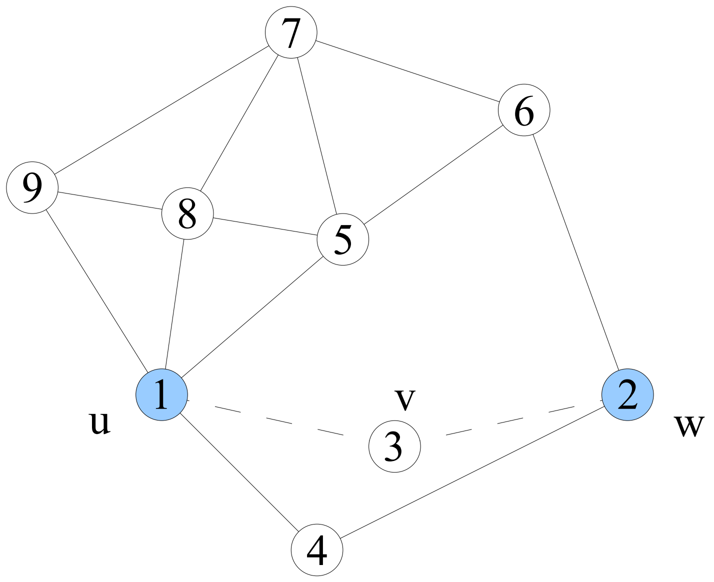
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, ..., P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, ..., P_n$.

MaxMin(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).



➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

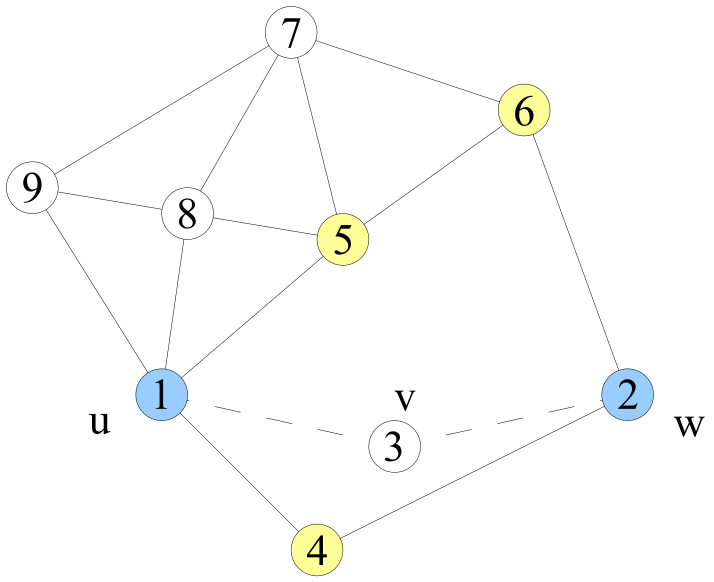
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, \ldots, P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, \ldots, P_n$.

MAXMIN(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MAXMIN$(u, x, v), x,$ MAXMIN(x, w, v)).



➥ Maximal replacement path: $(u,$MAXMIN$(u, w, v), w)$

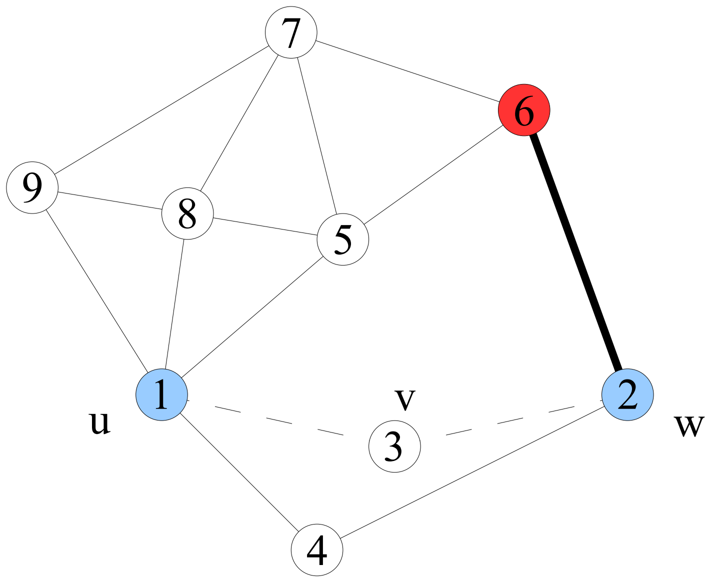
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, ..., P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, ..., P_n$.

MaxMin(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).



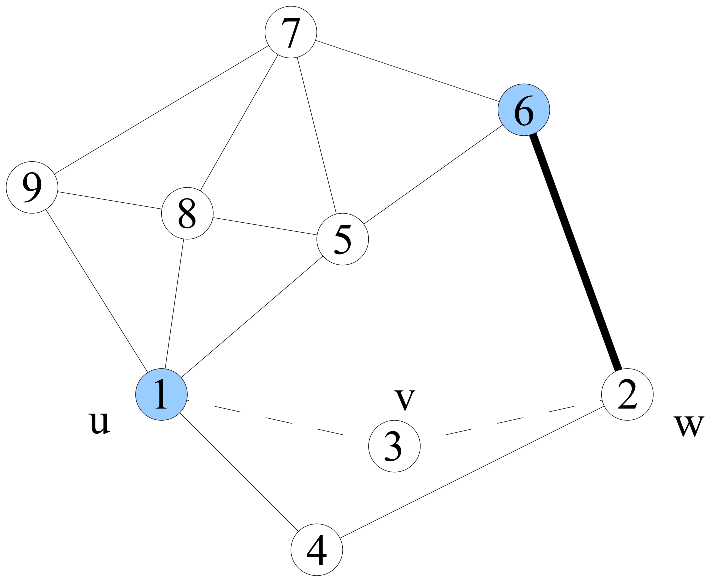
➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, ..., P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, ..., P_n$.

MaxMin(u, w, v)	
1:	**if** u and w are directly connected **then return** $\emptyset$.
2:	Find the max-min node x for (u, w, v).
3:	**return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).



➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

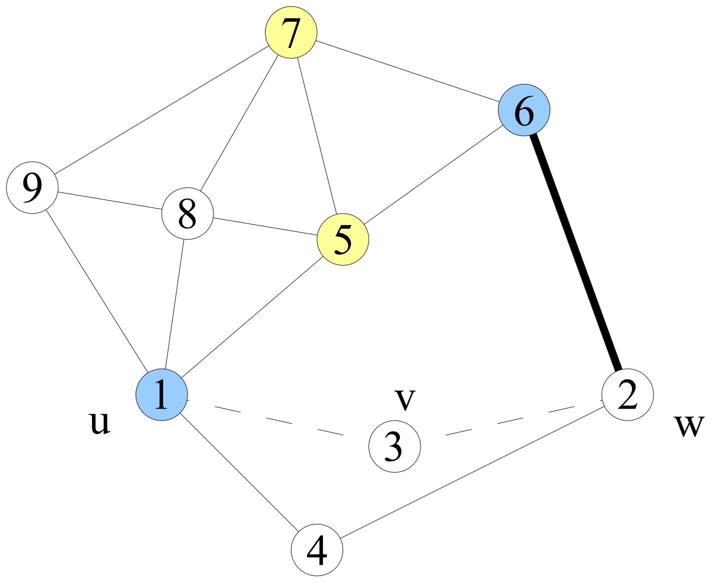
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, \ldots, P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, \ldots, P_n$.

MAXMIN(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MAXMIN$(u, x, v), x,$ MAXMIN(x, w, v)).



➥ Maximal replacement path: $(u,$MAXMIN$(u, w, v), w)$

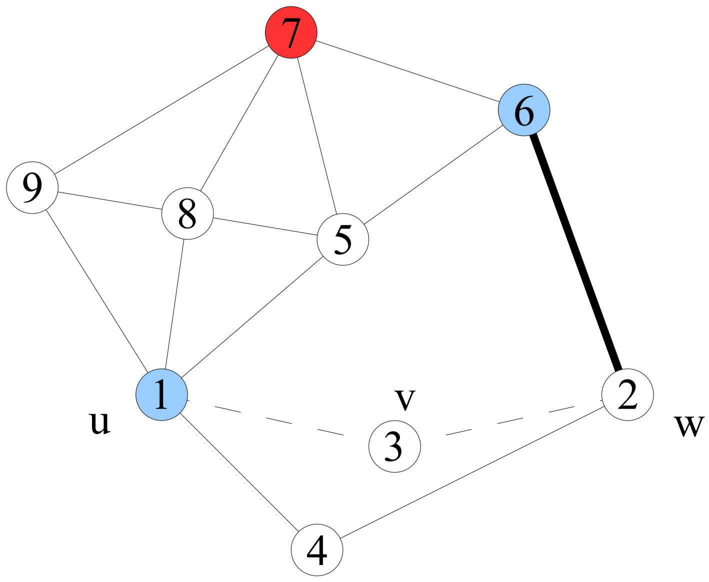
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, \ldots, P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, \ldots, P_n$.

MaxMin(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).



➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

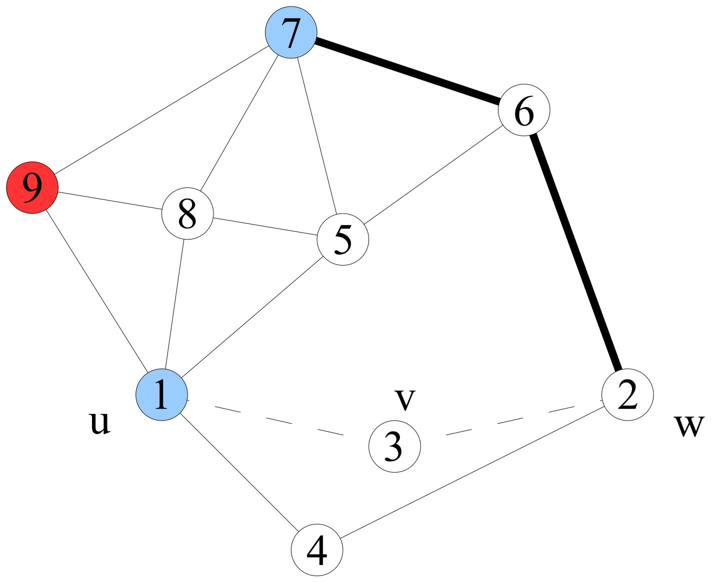
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, ..., P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, ..., P_n$.

MaxMin(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).



➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

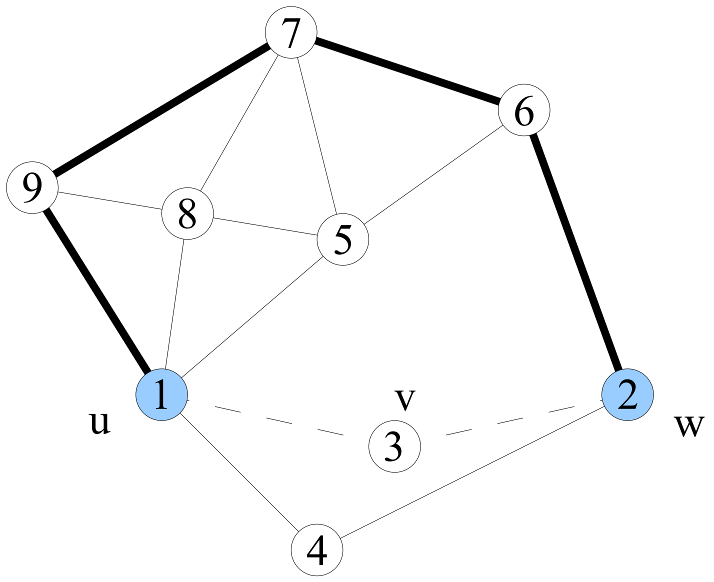
Maximal Replacement Path

minimum node: In a path $P = (u, v_1, ..., v_n, w)$ a *minimum node* is the intermediate node v_i with lowest priority value.

max-min node: Assume $\{P_1, \ldots, P_n\}$ is the set of all replacement paths for node v that connect u and w. Then a *max-min node for* (u, w, v) is the node with the highest priority value of all *minimum nodes* in $P_1, \ldots, P_n$.

MaxMin(u, w, v)

1: **if** u and w are directly connected **then return** $\emptyset$.
2: Find the max-min node x for (u, w, v).
3: **return** path (MaxMin$(u, x, v), x,$ MaxMin(x, w, v)).

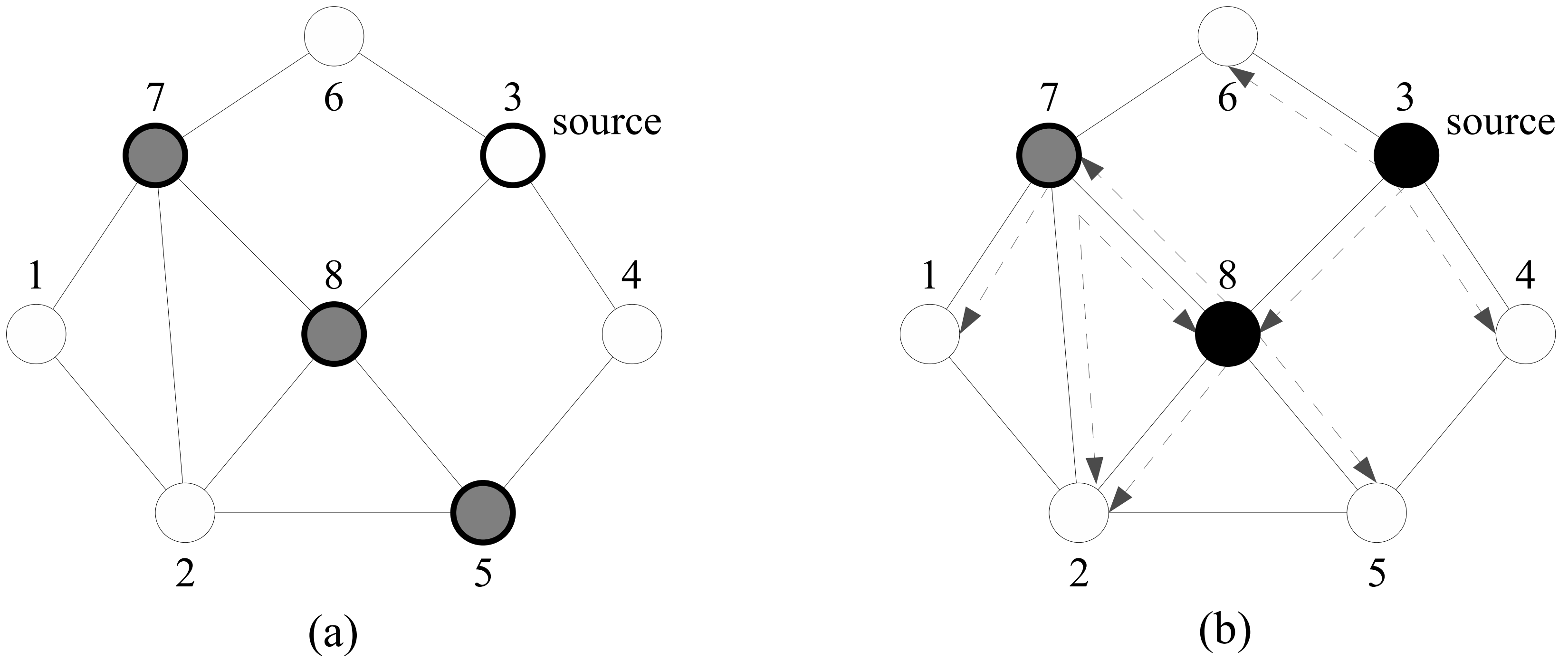


➥ Maximal replacement path: $(u,$MaxMin$(u, w, v), w)$

Routing History

- Our approach does not consider the source of a broadcast.
- No need to transmit a broadcast to nodes where it comes from.

➥ Consider the *routing history* or *visited node set* $D_h(v)$, which contains the last h recent nodes.

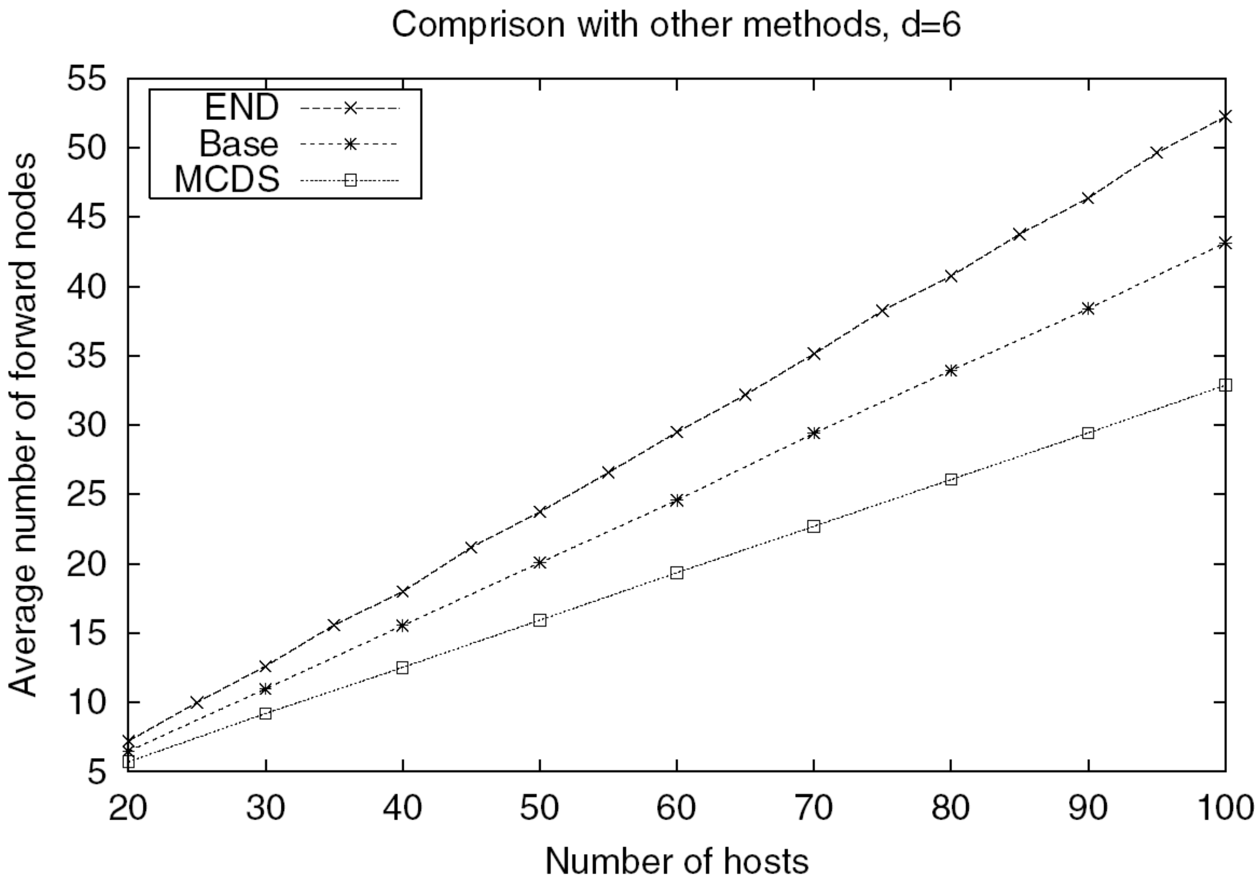


Priority Function

- Different priority function are possible:
  - unique node id
  - node degree
  - neighborhood connectivity

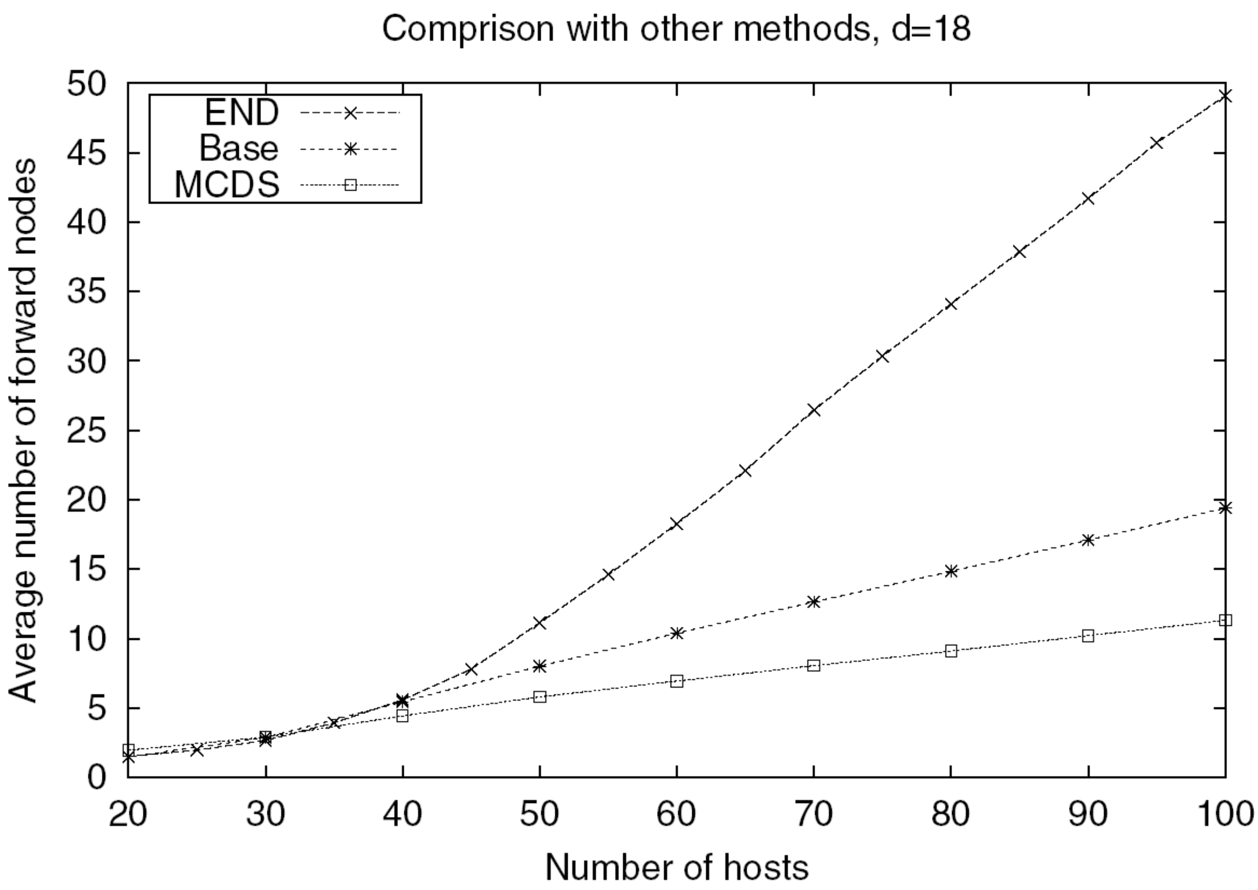
$$= \frac{|\text{pairs of not directly connected neighbors}|}{|\text{pairs of any neighbors}|}$$

Approximation of the MCDS (Sparse Network)



- **Base** – Base Configuration:
  Coverage condition I with 2-hop neighbor set information
- **END** – Enhanced neighbor-designating algorithm

Approximation of the MCDS (Dense Network)



- **Base** – Base Configuration:
  Coverage condition I with 2-hop neighbor set information
- **END** – Enhanced neighbor-designating algorithm