UNIT II

Broadcasting Storm

A **Broadcast Storm** refers to a situation where excessive broadcast packets flood a network, resulting in severe performance degradation. It typically occurs in **Mobile Ad-Hoc Networks (MANETs)** or **Wireless Sensor Networks (WSNs)**, where nodes are constantly mobile or network conditions change rapidly.

Causes of Broadcast Storm:

1. **Excessive Broadcast**:
   - When a node continuously broadcasts data without adequate checks, the network becomes flooded with unnecessary broadcast packets. This leads to network congestion and can severely slow down data transmission.
2. **Lack of Proper Coordination**:
   - In the absence of proper coordination among network nodes, multiple nodes may send broadcast packets without checking whether other nodes have already broadcasted the same information.
3. **Flooding**:
   - Flooding is a mechanism used in many routing protocols, where a packet is broadcasted to all the nodes within range, which then repeat the process. If this process is not controlled, it can quickly lead to a broadcast storm, particularly in dense networks.

Effects of Broadcast Storm:

1. **Network Congestion**:
   - As a result of continuous broadcast, the **network becomes congested** with broadcast packets, reducing the bandwidth available for other traffic. This can cause delays, reduced throughput, and ultimately, failure to deliver packets.
2. **High Latency**:
   - Broadcast storms increase the time it takes for packets to reach their destinations, leading to **high latency** in the network. Routing protocols that rely on fast and efficient data forwarding can suffer greatly.
3. **Packet Collision**:
   - With many nodes transmitting data simultaneously, packet collisions are likely to occur. Collisions require retransmissions, which further increases the network load and delays.
4. **Energy Drain**:
   - Nodes in wireless networks often rely on battery power. **Excessive broadcasting** leads to unnecessary use of the node's power, draining the batteries more quickly. This is especially problematic in **mobile networks** where nodes may not have a reliable power source.

Mitigation Strategies for Broadcast Storm:

1. **TTL (Time to Live) and Hop Count**:
   - **TTL** limits the number of hops that a broadcast packet can make in the network. This helps in controlling how far the broadcast can propagate.
   - By reducing the **hop count**, broadcast storms can be limited to a smaller area, avoiding network-wide flooding and reducing congestion.
2. **Probabilistic Flooding**:
   - Instead of broadcasting to all nodes, **probabilistic flooding** involves a node sending the broadcast packet with a certain probability. This reduces the likelihood of multiple nodes sending the same broadcast simultaneously and lowers the chances of network congestion.
3. **Efficient Routing Protocols**:
   - **Efficient routing protocols** can help reduce the need for excessive broadcasting. For instance, **Location-Based Routing** or **Quorum-Based Routing** can limit broadcast traffic by making rout-

ing decisions based on location or predefined node groupings, avoiding the need for network-wide broadcasts.

4. **Channel Reservation (TDMA)**:
   - **TDMA (Time Division Multiple Access)** allows nodes to transmit during different time slots, thus preventing collisions and **mitigating the effects of broadcast storms**.
   - By reserving time slots for transmission, the network is less likely to experience congestion due to simultaneous broadcasts from multiple nodes.
5. **Selective Flooding**:
   - **Selective flooding** involves broadcasting only to specific nodes rather than all nodes in the network. This selective process can be based on certain factors, such as proximity to the destination or node priority, which reduces unnecessary overhead.

Rebroadcasting schemes in wireless networks are strategies to propagate messages or data through the network, often in scenarios like routing protocols or wireless sensor networks. These schemes are designed to balance network efficiency, reliability, and resource usage. Here are some of the main types:

1. **Simple Flooding:**
   - **Concept**: Simple flooding is one of the most straightforward methods where each node, upon receiving a message, rebroadcasts it to all of its neighbors.
   - **Advantages**:
     - Very simple to implement.
     - Ensures that the message is disseminated to all reachable nodes.
   - **Disadvantages**:
     - Can lead to high network traffic and congestion due to excessive rebroadcasting.
     - Not energy-efficient as nodes might rebroadcast messages multiple times.
2. **Probability-Based Methods:**
   - **Concept**: In this method, nodes use a probabilistic approach to decide whether to rebroadcast a received message. Instead of always rebroadcasting, each node may rebroadcast the message with a certain probability, reducing the number of rebroadcasts.
   - **Advantages**:
     - Reduces unnecessary rebroadcasting, decreasing network congestion and power consumption.
     - Balances between flooding and energy usage.
   - **Disadvantages**:
     - It may result in some nodes missing the message if the probability is too low.
     - It might not guarantee complete message delivery, especially in networks with high mobility or large scale.
3. **Area-Based Methods:**
   - **Concept**: This approach involves dividing the network area into smaller regions or zones. Nodes in these regions determine whether to rebroadcast based on their geographic location and the area coverage. For instance, if a node is located in an area where the message has already been broadcasted, it may not rebroadcast it.
   - **Advantages**:
     - Helps control congestion by ensuring that messages are not unnecessarily rebroadcasted within the same area.
     - More efficient than simple flooding, as it reduces unnecessary retransmissions.

- **Disadvantages**:
    - Requires knowledge of the network's geographical structure, which may not always be available or may incur additional overhead.
    - Depending on the size and density of the area, it could still result in redundant broadcasts.

4. **Neighbor Knowledge-Based Methods:**

- **Concept**: In this method, nodes make decisions based on their knowledge of neighbors (such as how many neighbors have received a message). Nodes may choose to rebroadcast based on their knowledge of which neighbors have already been covered.
- **Advantages**:
    - Reduces the probability of redundant rebroadcasts since nodes can make informed decisions.
    - Improves efficiency compared to simple flooding.
- **Disadvantages**:
    - Requires maintaining some form of neighbor table or knowledge, which can incur overhead.
    - Nodes might not always have accurate or up-to-date information about their neighbors, leading to incomplete coverage.

Method	Key Concept	Advantages	Disadvantages
Simple Flooding	Rebroadcasts to all neighbors	Simple, guarantees message delivery	High network traffic, congestion, energy-inefficient
Probability-Based	Rebroadcasts with a probability	Reduces redundancy, less traffic	May miss nodes if probability is too low
Area-Based	Rebroadcasts based on geographical zones	Efficient, avoids redundant broadcasts	Requires area knowledge, less flexible
Neighbor Knowledge	Rebroadcasts based on neighbor information	Reduces unnecessary rebroadcasts	Requires neighbor knowledge, may not be accurate

Simple Backoff Algorithm

A simple backoff algorithm helps manage access to a shared communication medium (like a wireless channel) by delaying retransmissions after a failure or collision, aiming to reduce the chance of further collisions. Below is an example of a basic random backoff algorithm, which is commonly used in ad hoc networks:

Steps of a Simple Backoff Algorithm:

Start with a Backoff Timer:

When a node needs to transmit, it checks if the channel is idle or busy.

If the channel is idle, the node can proceed with transmission.

If the channel is busy (another node is transmitting), the node waits for the channel to become free.

Random Back off: After detecting the channel is busy, the node generates a random backoff time. This time is usually measured in time slots or units of time.

The range for the random backoff time is typically between 0 and a maximum limit (let's call this CW_max for Contention Window).

Wait for the Backoff Time:

The node then waits for its backoff timer to expire. Each time the node detects that the channel is still busy, it may increase its backoff time slightly by randomly selecting a new value within the backoff range.

Transmit When the Channel is Idle:

Once the backoff timer expires, the node checks again if the channel is idle. If it is, the node will transmit its data.

Repeat if Collisions Occur:

If the transmission fails (because of a collision), the node will increase its backoff range and repeat the process.

Example of a Simple Random Backoff Algorithm:

Initialization:

Each node has a contention window (CW) from which it selects a random backoff time. The window size could be from 0 to CW_max. For simplicity, assume CW_max = 3.

Collision Occurrence:

After a failed transmission (collision), the node will pick a random backoff time again, but the window size doubles after each failure to reduce congestion.

Algorithm Steps in Detail:

Node detects channel status:

If idle, transmit.

If busy, wait.

Backoff Timer Selection:

Node picks a random value between 0 and CW_max (initially, CW_max = 3).

Example: Node selects random(0, 3) → result could be 2.

Node waits for backoff period:

Wait for the backoff time (2 time slots).

Transmit when Channel is Idle:

If the channel is idle after the backoff period, the node transmits.

Collision Detected:

If the transmission fails (collision), the backoff window size (CW_max) is doubled. So, the new CW_max becomes 6.

The node picks a new random backoff time between 0 and 6 (e.g., random(0, 6) → 3).

Repeat:

The process continues until a successful transmission occurs.

Pseudocode:

python

```
def backoff_algorithm(CW_max):

# Node waits for channel to be idle

if channel_is_idle():

# Node transmits data

transmit_data()

else:

# Channel is busy, node picks random backoff time

backoff_time = random(0, CW_max)

wait_for(backoff_time)

# After waiting, check channel again

if channel_is_idle():

transmit_data()

else:

# If transmission failed, double the CW_max and retry

CW_max *= 2

backoff_algorithm(CW_max)
```