ForestNet:
Cluster-Tree Mesh Ad-Hoc Network

Nicholas Palmer, 825059580, EE662

1. Introduction & Assumptions

In this document, we describe a cluster-tree hybrid-mesh ad-hoc network design based on concepts and designs discussed in class.

Preface

- Sequence & Activity Diagrams integrated into Network Architecture (Section 2) & Packet Format (Section 4) Sections for clarity.
- Order of Routing (Section 3) and Packet Format (Section 4) sections swapped for clarity.
- Design Considerations & Trade-offs integrated into Routing (Section 3) for clarity.

Goals

Our motivation for creating such a network is multi-fold, we want to create a resilient network capable of connecting distributed sensor networks effectively. One such application is wildlife monitoring, suppose we needed a network to track primates through a forest, and log their interactions for research purposes. Or emergency networks, suppose we needed a network for first responders that adapts to changing conditions like changing infrastructure (fire, cellular network loss, etc.). These situations call for a network that aligns with the following goals:

- **Low-power**: Our target applications imply battery-constrained operation for the majority of nodes, thus we want to use low-power radios and efficient routing algorithms.
- **Low-memory**: This complements low-power, we want a network that is lightweight and does not require a huge memory footprint. This allows it to run on resource-constrained devices such as microcontrollers.
- **Self-configuring**: Meaning a node can effectively cold-start without prior configuration by automatically joining or creating a network (if capable), and determining its role on the network (node, cluster-head, or router).
- **Self-localizing**: Meaning we can use our spatial location in relation to other nodes, in order to optimize our network configuration (i.e. change node role to match topology optimally).
- **Self-healing**: This is an important requirement since in our applications our nodes have the potential to move spatially and potentially lose all connections. This means we need the ability for the network to automatically make routing decisions when things change, and nodes should dynamically update roles based on our network health as well. We should also consider cases when routers or cluster-heads go offline, our network should be resilient enough to reroute data quickly (if at all possible).
- **Low-complexity**: If possible keep our network simple, to make things easy to understand and speed up development time. But we should make sure we meet our other goals/requirements and enforce this if possible.

Assumptions

We can make several basic assumptions to make building our network simpler. We can assume we will use IEEE 802.15.4 as our physical layer and MAC layer, that we will build our network layer on top of. In terms of the physical layer this means we can make several assumptions and design around these. 802.15.4 dictates we use CSMA/CA but also allows for time managed communication via TDMA (GTS). 802.15.4 supports both peer-to-peer communications, as well as star topologies making it well suited to a hybrid mesh cluster tree network like ours. Security considerations are also delegated to the standard; once a node is associated with a network (paired), the protocol is deemed secure.

Table 1. Summary of Network Assumptions

Item	Assumption	Rationale
Radio Range	~10-100 meters	Depends on TX power and transceiver power budget/antenna sensitivity, higher range possible with lower frequencies.
Frequencies	868 MHz 915 MHz 2.4 GHz	3 Possible frequency bands with multiple channels available per band.
Channel	Multi-channel Operation	Depending on frequency range, we have 10 (915 MHz) or 16 (2.4 GHz) channels available.
Maximum Data Rate	250 kbps (lower supported)	Available on all frequencies, higher schemes require higher modulation schemes (BPSK, O-QPSK, ASK).
Maximum Transfer Unit	127 bytes (802.15.4 frame)	PHY constraint of 802.15.4.
Link Symmetry	Symmetric	Bitrate, power, and encoding scheme is kept constant during transmission, ensuring link symmetry. TX power may be adjusted per transmission.
Clock Drift	Clock Min. Accuracy of +/- 40ppm	IEEE 802.15.4 clock requirement for TDMA synchronization and Guaranteed Time Slot Support (GTS).
Node Mobility	Low to moderate (approx. 2 m/s)	Relatively infrequent routing updates and network reorganization.
Scale	Up to 255 networks, 254 nodes/network	8-bit addressing limit, ~65k total nodes possible. Support for extended 24-bit addressing possible.

2. Network Architecture

Building on the 802.15.4 PHY/MAC, we layer our custom Cluster Tree Hybrid Mesh topology network layer on top to create a resilient and adaptive network fabric. Our network uses “capabilities” and “roles” to describe what a node is currently doing and what it is capable of doing (further discussion below). The network self-organizes into a tree topology with leaf nodes and clusters, clusters have their own “Network ID” and each node within that cluster has a “Node ID”. This effectively means we can address any cluster in our tree, and any node in that cluster. Cluster creation is also an automated process (detailed later) that occurs as-needed, a promotion decision can be made on the edges of the network (root still needs to store new cluster info, but does not handle promotion logic).

Topology

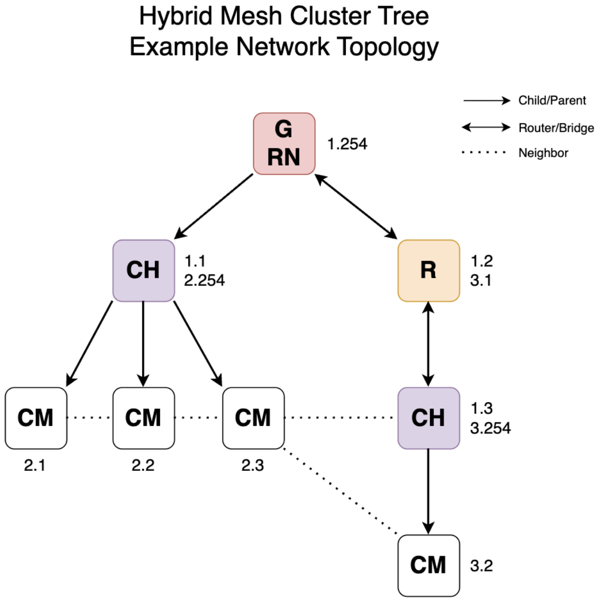

Figure 1. Hybrid Cluster Tree Network Topology

In Fig. 1, we can see an example of a network with examples of each relationship between nodes and roles. The Cluster Head (CH) 3.254 was created when Cluster Member (CM) 3.2 requested to join through that node, and 3.254 promoted itself to facilitate the new node. The Router (R) was created on-demand (as it used to be a CH) when it decided having 2 sequential Cluster Heads (CH) with no nodes to connect was not necessary. We can see the principle of uniqueness vs. aliasing (a property of this topology) working. Addresses 1.1 & 2.254 refer to the same node, but no singular address refers to multiple nodes. Different nodes may store different addresses for the same node depending on their position in the network (neighbor, parent, child).

Addressing

Our network allows a packet to specify if we are using standard or extended addressing for Cluster/Node IDs so address size can vary. However, for routing we define a dynamic address composed of 2 identifiers: **NET**.**NODE**, Network ID and Node ID respectively. Nodes also have globally unique addresses (Unique ID) to validate request/join requests, address assignment, and cluster membership.

Table 2. Comparison of Address Identifier Types

Address Identifier Types			
Identifier Type	Size (*bits*)	Relative Uniqueness & Mobility	Scope
Network ID (NET)	8 (std) 24 (ext)	Per Root Dynamic Assignment	Describes the ID of a Cluster Head (CH) relative to a Root Node (RN). Network IDs are assigned by RN on CH promotion via a Network ID assignment response message.
Node ID (NODE)	8 (std) 24 (ext)	Per Cluster Dynamic Assignment	Describes the ID of a Cluster Member (CM) relative to a Network ID. Node IDs are assigned by Cluster Head (CH) to a CM via a Node ID assignment response message.
Unique ID (EUI-64)	64	Globally Unique Statically Defined	Intrinsic to each Cluster Member (CM), used to validate request/response messages for address assignment and network join.

Table 3. Comparison of Addressing Schemes

Address Schemes		
Scheme	Format	Scope
ROLE: Root Node (RN)	1.254	Reserved for Root Node (RN), specific case of Cluster scheme.
ROLE: Cluster Head (CH)	NET.254	Taken by a Cluster Head (CH) on Network ID assignment.
Broadcast	255.255	Network-wide broadcast, when received, is relayed by every Cluster Head (CH) to all nodes.
Unicast	NET.NODE	Cluster Member (CM) to CM address scheme, described by the combination of Network ID and Node ID.

The described addressing schemes imply that addresses always resolve to a unique node, however nodes may take on multiple addresses, for example if they are CHs they will also take on the X.254 address for their assigned Network ID.

Roles

Our network has 5 roles: Cluster Head (CH), Cluster Member (CM), Router (R), Root Node (RN), and Gateway (GW). Each describes how that node interacts with the rest of the network. Roles are not mutually exclusive and a single node may fill multiple roles, for example the RN and GW roles may be the responsibility of a single node in a smaller network.

Cluster Head (CH)

A CH is required to connect a CM to the rest of the network. Each CH carries its own Network ID with a unique local address space, allowing it to be the ‘head’ of its cluster and manage its members. This means managing CMs is a localized responsibility that is not handled by a central point (Dynamic Distributed Address Application). This design has several advantages like: less points of failure, less centralization, and local optimizations (meaning we can make decisions without needing approval from a central node). A CH will be assigned a Network ID and assume the x.254 Node ID for that Network ID.

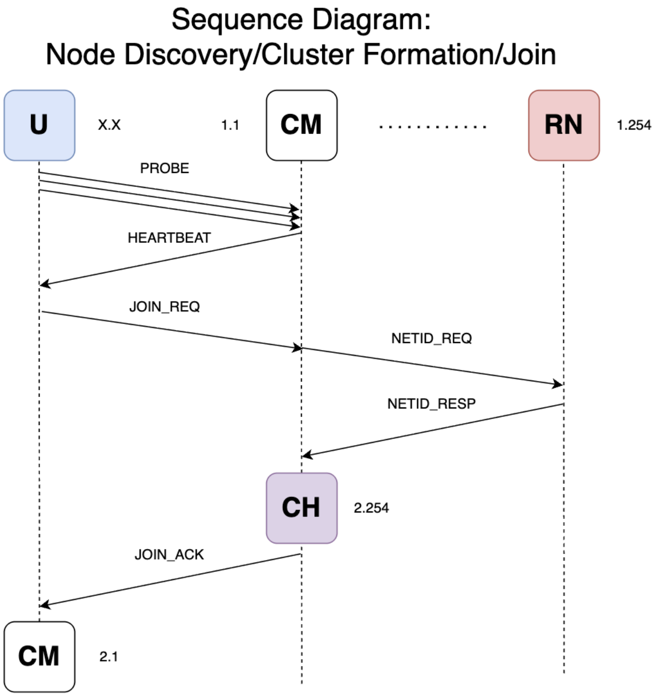

Figure 2. Node Discovery/Network Join (w/Cluster Formation)

The join flow and self-promotion can be seen in Fig. 2, where an unregistered node makes a JOIN_REQ, and the CM is forced to issue a NETID_REQ and self-promote (waiting for a NETID_RESP. Only then can a CM be sent a JOIN_ACK back after the promotion.

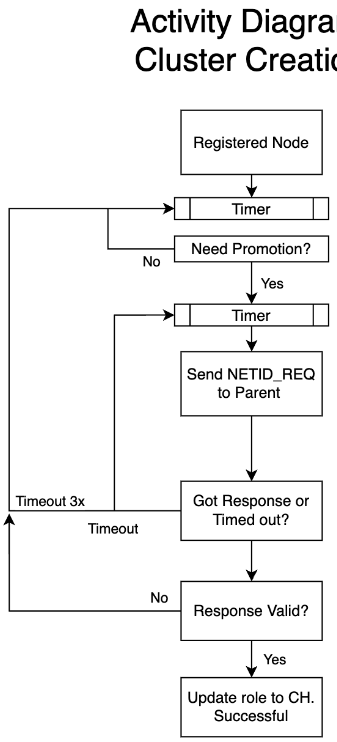

Figure 3. Cluster (Network) Creation

Fig. 3 describes the activity diagram for cluster creation, this is an event driven task that periodically checks if need to be promoted and launches this flow if that is the case.

Cluster Member (CM)

The CM role indicates that we are a member of some existing cluster with some parent CH. This means that in one of our addresses our Network ID will be the same as that of the CH. When joining a cluster we will be assigned an address by a CH called a Node ID. It should be noted that as a node, we can be a CM of multiple clusters, so we may have more than one address, although only one unique Node ID per Network ID. It is also possible that a single node can be a CM and a CH simultaneously.

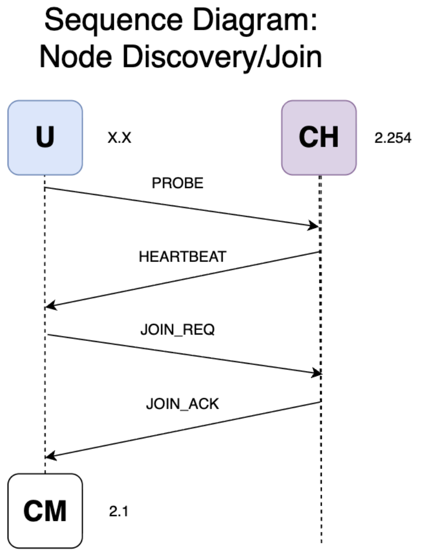

Figure 4. Node Discovery/Network Join

Fig. 4 describes the discovery/join flow in the case that CH promotion is not required. The unregistered node broadcasts a PROBE and waits for responses. Once it determines a network to join we send a JOIN_REQ and get a JOIN_ACK with our CM information.

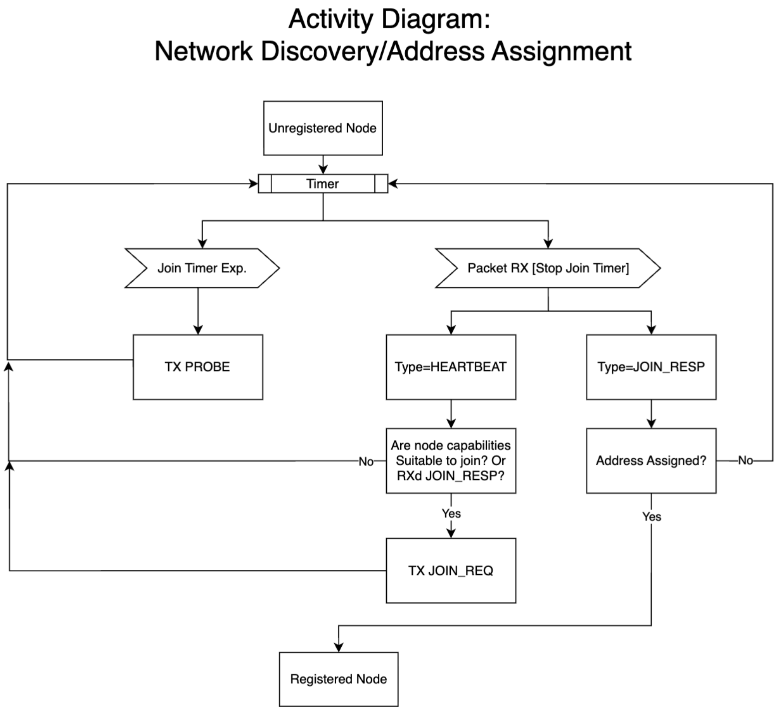

Figure 5. Node Discovery & Address Assignment

Fig. 5 describes the activity diagram for an unregistered node, ie. when we send probe messages and what happens when we receive a packet (event driven).

Router (R)

A R role indicates that we are bridging 2 or more clusters together. If a CH determines we can promote a CM and the network topology can be more optimal, the CH will naturally promote the CM to a R. This process should be distributed and should happen without RN intervention, contributing to the self-configuring and localizing goals of our network. It should be noted we can promote CMs, as well as ‘demote’ CHs if the topology is better suited.

Root Node (RN)

A RN role indicates we are responsible for new Network ID allocation. This could be the same node as the GW, but may also be a different node. Essentially, a RN will handle Network ID requests when a CM asks to be promoted to a CH. The RN will also by definition act as a CH for any child nodes that join its network. The RN will always have an address of 1.254.

Gateway (GW)

A GW role indicates that we have another network interface available that we can bridge. This role is NOT the same as the RN, in that the GW does not handle Network ID allocation, however in many cases a single node may fulfill both roles. The advantage of having a discrete GW role, is the ability to have a single root and multiple gateways to balance traffic to external interfaces. If a node has this role, other nodes will be aware they can send data to reach the external network interface.

3. Routing

To route our packets we need a hybrid protocol that takes advantage of our clustered-tree topology (routes), and one-hop neighbor table (peer to peer). We describe several rules that form an algorithm any CM can use to route packets to a DST.

Mesh vs. Tree Routing Decision

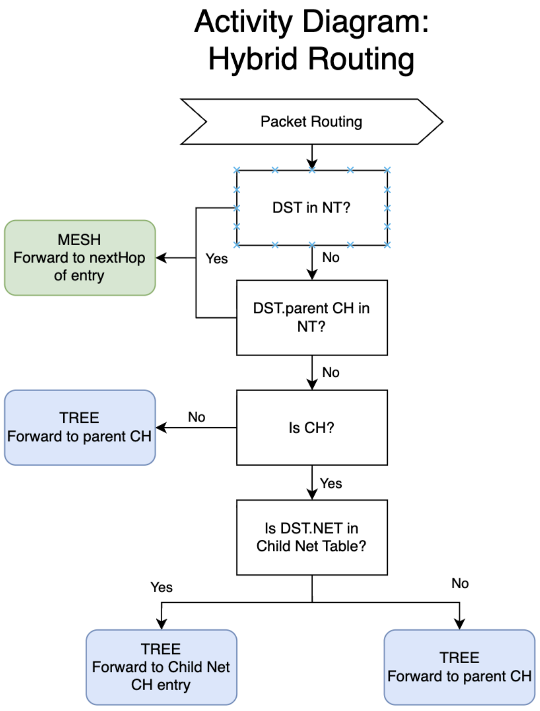

Figure 6. Mesh vs. Tree Routing Decision

We need a set of rules that determine when we use which data structures we have available to us. Our initial set of rules is described below along with justifications for each case and sequence diagrams. Fig. 6 Describes the activity diagram on the routing event, we can decide how we are going to populate the DST/SRC parts of our header accordingly.

Hybrid Routing Rules

1. If DST is found in neighbor table (1 or 2 hop), forward to nextHop entry
2. If parent CH of DST is found in neighbor table, forward to nextHop entry
3. If Network ID found in Child Net Table, forward to nextHop entry
4. Forward to parent CH

Hybrid Routing Rule (Mesh)

If our DST or parent cluster is found in our neighbor table, we can forward packets directly. This rule is akin to a mesh topology network since it connects 1, 2, or n hop neighbors directly across any topology.

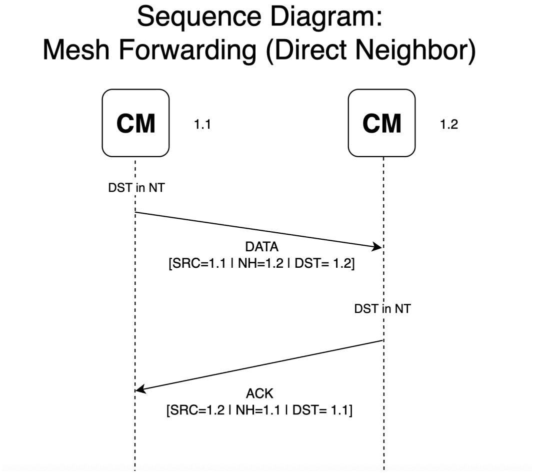
Sequence Diagram:
Mesh Forwarding (Direct Neighbor)
CM
1.1
CM
1.2
DST in NT
DATA
[SRC=1.1 | NH=1.2 | DST= 1.2]
DST in NT
ACK
[SRC=1.2 | NH=1.1 | DST= 1.1]

Figure 7. Mesh Routing Sequence

Fig. 7 showcases the sequence diagram for mesh routing rule between 2 adjacent neighbor nodes. These neighbors will have previous knowledge of each other via their respective neighbor tables. This is the simplest case of transmission in our network and requires minimal routing.

Hybrid Routing Rule (Tree)

However the mesh rules can only get us within 1 or 2 hops of the DST alone, signaling the need for another ruleset that can take our packet up and down our tree topology. The simplest approach is to simply send the packet to the parent. Our parent will know how to handle the packet because of the stored child net table which contains all networks below a parent and corresponding next hops which allow the parent to route the packet accordingly.

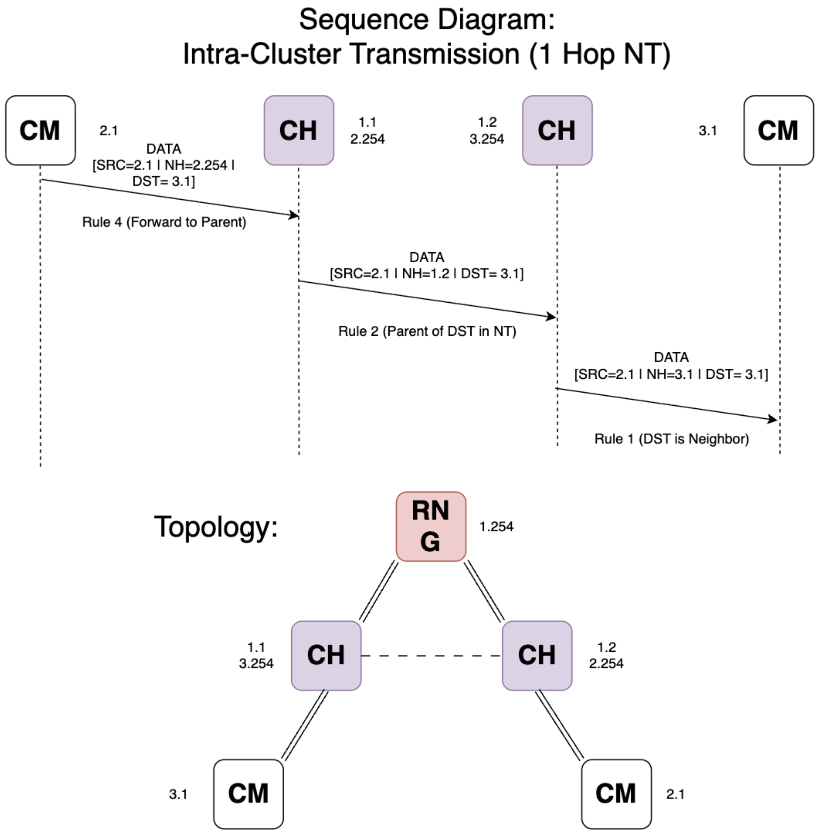
Sequence Diagram:
Intra-Cluster Transmission (1 Hop NT)
CM
2.1
CH
1.1
2.254
1.2
3.254
CH
3.1
CM
DATA
[SRC=2.1 | NH=2.254 | DST= 3.1]
Rule 4 (Forward to Parent)
DATA
[SRC=2.1 | NH=1.2 | DST= 3.1]
Rule 2 (Parent of DST in NT)
DATA
[SRC=2.1 | NH=3.1 | DST= 3.1]
Rule 1 (DST is Neighbor)
Topology:
RNG
1.254
1.1
3.254
CH
CH
1.2
2.254
3.1
CM
CM
2.1

Figure 8. Intra-Cluster Data Transmission

Fig. 8 shows the mixed case with a route that uses the cluster tree rule first to route upward (since no mesh rules apply), then uses a mesh rule to route sideways between neighbors, then the network can either use the mesh rule or child net table to route the packet since it is a direct child.

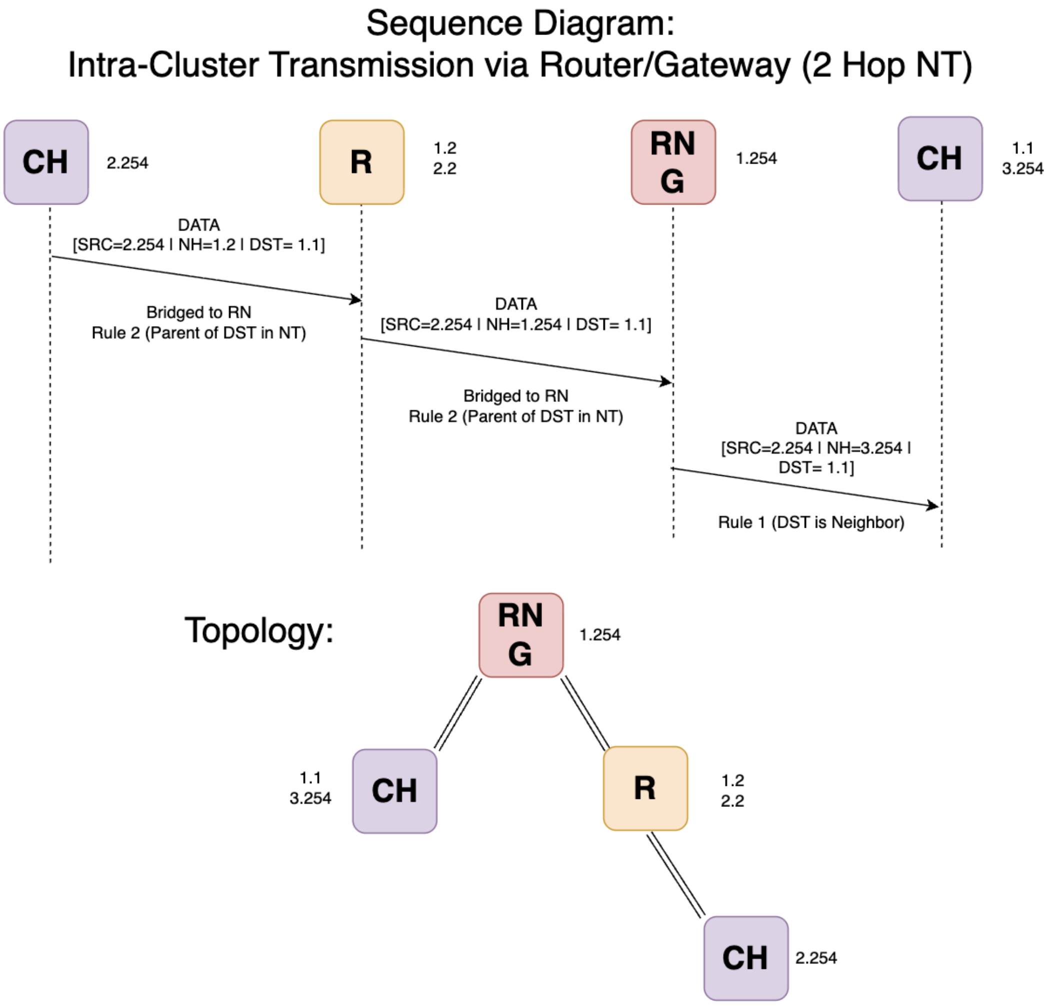
Sequence Diagram:
Intra-Cluster Transmission via Router/Gateway (2 Hop NT)
CH
2.254
R
1.2
2.2
RN G
1.254
CH
1.1
3.254
DATA
[SRC=2.254 | NH=1.2 | DST= 1.1]
Bridged to RN
Rule 2 (Parent of DST in NT)
DATA
[SRC=2.254 | NH=1.254 | DST= 1.1]
Bridged to RN
Rule 2 (Parent of DST in NT)
DATA
[SRC=2.254 | NH=3.254 | DST= 1.1]
Rule 1 (DST is Neighbor)
Topology:
RN G
1.254
1.1
3.254
CH
R
1.2
2.2
CH
2.254

Figure 9. Intra-Cluster Data Transmission (via gateways or routers)

Our networks may also form Routers (R) if CHs are not serving any nodes and their only purpose is bridging 2 networks. Fig. 9 considers the routing case, since routers simply bridge CHs together so the same rules still apply, except our router bridge will inspect which parent network it needs to send the packet to and perform that routing.

Routing Considerations & Trade-Offs

There are 3 main categories of routing algorithms: reactive algorithms, proactive algorithms, and hybrid approaches.

Reactive (on-demand) routing implies the nodes in the network are not aware of the network's topology, meaning they effectively are unaware of any surrounding nodes until they need to be. Routes are only realized when necessary. This means nodes can be lightweight, and do not need to track a large amount of metadata about surrounding nodes/routes at all times in large tables. This type of routing makes sense for burst-type networks, where mobility is often more important than latency. When mobility is not important, reactive routing is not ideal since there is no ‘memory’ of a particular route and the entire route has to be ‘re-generated’ every sequence. Latency is also unpredictable since sequences or even packets may not take the same route every time. In the case when networks are mostly static, a proactive solution makes more sense.

Proactive (table-based) routing is a more ‘traditional’ approach where routes are discovered in advance, stored in tables, then utilized when sending data. Nodes are aware of the network topology (entire network or subset), and use this information to send packets along routes. In a proactive network, nodes must keep routing tables that describe the next hop a packet must take to reach a specific node, and some metric to track the overhead of specific routes. This overhead comes in the form of physical memory per node since they must store (relatively) large amounts of metadata per route, maintain routes. This is not ideal for networks that require high mobility since the overhead of updating our tables becomes relatively large if they change very often. However if a network requires low-latency or high throughput this approach makes more sense.

Hybrid routing combines both of these approaches in some configuration. A typical way this could be done is by not generating routes till they are needed, and filling neighbor tables along the way so this process does not need to happen every time along that route. In a hybrid approach, tables may not be entirely updated, we may only receive a vector for a specific route and we do not generate/regenerate the whole table every time.

Our Algorithm

Our hybrid approach (Fig. X) is a simple rule based algorithm that prioritizes a mesh based route found in our neighbor table, if a route cannot be found we fallback to the tree backbone which uses a more traditional tree-routing approach.

In our case single-hop neighbors can be found trivially by simply listening for packets, even if they are not meant for us we can still store neighbor information. But an N hop table requires exchanging information about neighbors of neighbors either piggybacking on network data or performing some sort of route discovery. This is possible with our network rules however we need another scheme (ie. route discovery through flooding) to build a greater than 1-hop table.

Routing Metric

Our current approach does not use any metric to pick routes, our algorithm does prioritize peer-to-peer neighbors if possible. Currently we do not track any data on link quality or transmit power for particular routes. However, in future network versions we could include a weighted metric that uses hops (minimize the number of hops along route), expected transmission count (similar to number of hops but takes into account a weighted expected transmission count per node ie. 1 node is unreliable), or energy use along a route.

4. Packet Format & Data Structures

Our Network utilizes a common frame header, and several discrete packet types to fulfill discovery, joining, promotion, addressing, maintenance, and data transfer. They are each described below.

Table 4. List of Network Packet Types

Type	Type ID	Description
PROBE	0x0	Requesting node or network state info
HEARTBEAT	0x1	Responding/advertising node or network info
JOIN_REQ	0x2	Requesting a CH to join network and assign address
JOIN_ACK	0x3	Response to join request
LEAVE	0x4	Remove self from network, frees the node info and address
ADDR_RENEW	0x5	Renew address from CH
NETID_REQ	0x6	Request to be promoted to CH, anticipate Network ID assigned
NETID_RESP	0x7	Response to network ID request
DATA	0x8	Data Payload
ACK/NAK	0x9	Packet Receive Acknowledgement

All subsequent packets fit into the common frame header, and populate the payload with additional field data as described below.

Table 5. Common Packet Header Description

Field	Size (bits) & Range	Description	Example
Frame Control 1. Frame Type 2. Address Type 3. ACK Req. 4. Reserved	4	Corresponding packet type (see table)	0000 (PROBE)
	2	Standard(00) / Extended(11) Addressing	00 (std)
	1	Request an Acknowledgement? (0|1)	1 (ack)
	9	TBD For Future Use	0_0000_0000
seqNo	16	Sequential wrapping value, for ACK back and fragmentation support (future)	0x000F (15)
Dest. Address (DST)	16/48	Destination Address	0x01FE (1.254)

Next Hop (NH)	16/48	Next Hop Address, overridden by forwarding nodes while routing	0x0102 (1.2)
Src. Address (SRC)	16/48	Source Address	0x0201 (2.1)
Payload (Packet)	0 - 920	Packet data payload, variable length per packet type	
CRC	16	16-bit Packet Checksum	0x7E8A

A PROBE packet is used to request information from a node or set of nodes (via broadcast), with the information varying based on the probe type. One use case is during the discovery phase. First we search for a CH to join, and if none are available we search for a CM (which triggers promotion). An internal probe can be destined for any node and requests network information from that node.

Table 6. PROBE Packet Description

PROBE (Fields encapsulated within Common Header) [0x0]			
Field	Size (bits) & Range	Description	Example
probeType	8	0x00: CH Join Probe (Query CH for Info) 0x01: CM Join Probe (Query CM for Info) 0x02: Internal Probe (Query network status info)	0x01 (CM Join Probe)
reserved	X	TBD for Future Use	N/A

A HEARTBEAT packet is used to respond to a PROBE message with information useful based on the probeType. For example during a CH discovery PROBE a CH will respond with one of these detailing its roles and capabilities, and distance from the root. This allows a prospective unregistered node to pick an optimal CH if multiple candidates are found. A CM would also send one of these to describe its capabilities, which is crucial since there may be an instance when a CM is not capable of promoting to CH and thus cannot be joined.

Table 7. HEARTBEAT Packet Description

HEARTBEAT (Fields encapsulated within Common Header) [0x1]			
Field	Size (bits) & Range	Description	Example
roleAttributes	8	**Flag Bits:** 0: isClusterHead 1: isReg (Registered on Network?)	0b11111

		2: isRouter 3: isRootNode 4: isGateway	
capabilities	8	**Flag Bits:** 0: capableClusterHead 1: Reserved 2: capableRouter 3: capableRootNode 4: capableGateway	0b1X111
distToRoot	8	Distance (hops) to RN 0x00: Currently RN 0xXX: XX hops to RN 0xFF: Unknown hops to RN	0x01
reserved	X	TBD for Future Use	N/A

A JOIN_REQ packet is used by an unregistered node to request to join a network through a specific node. It is a unicast packet to the DST node we want to join through, that sends its UID and capabilities to it.

Table 8. JOIN_REQ Packet Description

JOIN_REQ (Fields encapsulated within Common Header) [0x2]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure response will be assigned correctly	0x001A2B3C4D5E
capabilities	8	Send a copy of our capabilities along with the JOIN_REQ **Flag Bits:** 0: capableClusterHead 1: Reserved 2: capableRouter 3: capableRootNode 4: capableGateway	0b1X111
reserved	X	TBD for Future Use	N/A

A JOIN_ACK packet is the response to a JOIN_REQ sent by the node requested from, it is a broadcast packet since the node does not know its assigned address yet. We send back the UID to ensure the assignment is for the correct node. We also tell it its new address, how long it is valid, and of course if the request succeeded.

Table 9. JOIN_ACK Packet Description

JOIN_ACK (Fields encapsulated within Common Header) [0x3]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure response will be assigned correctly	0x001A2B3C4D5E
promotedToReg	1	0: Join Request Denied 1: Join Successful, node registered	0b1
clusterAddr	16/48	Newly Assigned Cluster Address	0x0101 (1.1)
timeAddrValid	16	Number of minutes the Addr is valid from packet send	0x5A0 (24hr)
reserved	X	TBD for Future Use	N/A

A LEAVE packet is sent from a node to its parent, which tells it we are leaving the network for some reason. We attach a UID to ensure the request matches the correct node, and also tell the parent our cluster address to verify everything again. While we do not use it yet, we also specify a leave code that tells our parents why we left, which may inform future decisions. The parent will also update its routing/neighbor tables accordingly.

Table 10. LEAVE Packet Description

LEAVE (Fields encapsulated within Common Header) [0x4]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure Leave request is legitimate	0x001A2B3C4D5E
clusterAddr	16/48	Unique address in the cluster, to remove ambiguity about which cluster we are leaving	0x0101 (1.1)
leaveReason	8	Why we left the cluster: 0x0: Re-optimizing CH configuration 0x1: Battery low 0xX: TBD	0x00
reserved	X	TBD for Future Use	N/A

An ADDR_RENEW packet is sent by a node to its CH to request a new address when it is about to expire (or for some other reason). We attach a UID to ensure the address is assigned correctly, and also list our current cluster address for verification. The node also sends its capabilities again so the parent has the latest update if they have changed from the previous state.

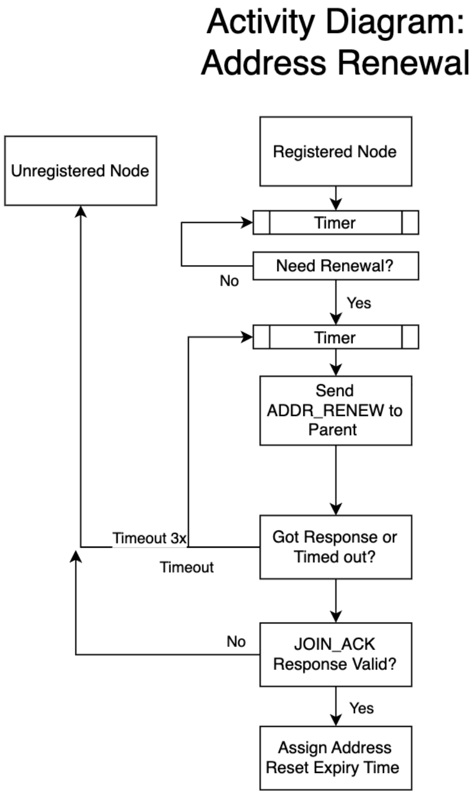

Figure 10. Address Renewal Activity Diagram

The address renewal activity diagram in Fig. 10 describes the flow, if we need a renewal we request it using the ADDR_RENEW packet, and parse the response. If it is invalid or we never get it, we become an unregistered node.

Table 11. ADDR_RENEW Packet Description

ADDR_RENEW (Fields encapsulated within Common Header) [0x5]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure response will be assigned correctly	0x001A2B3C4D5E
clusterAddr	16/48	Cluster Addr before renew	0x0101 (1.1)
capabilities	8	Send our capabilities along with the renew encase they have changed	0b1X111

		Flag Bits: 0: capableClusterHead 1: Reserved 2: capableRouter 3: capableRootNode 4: capableGateway	
reserved	X	TBD for Future Use	N/A

A NETID_REQ packet is sent by a CM to request promotion to a CH from the RN. This is triggered on an as-needed basis when a new node requests to join through that CM. We attach our UID to confirm promotion for the right node. We also attach our address in the cluster as well as the hop count which gets incremented along the way. This allows nodes along the routes to store the hop count and update their child net table accordingly.

Table 12. NETID_REQ Packet Description

NETID_REQ (Fields encapsulated within Common Header) [0x6]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure new assignment is legitimate	0x001A2B3C4D5E
clusterAddr	16/48	Dynamic address in the network	0x0101 (1.1)
hops	8/24	Incremented on each hop by sequential nodes till the RN, allows hop count in Child Network Table	0x05 (5 hops)
reserved	X	TBD for Future Use	N/A

A NETID_RESP packet is the response to the above NETID_REQ packet, where the Network ID is returned along with the UID to the prospective CH. We tell the node if it was promoted or not, and its new network ID.

Table 13. NETID_RESP Packet Description

NETID_RESP (Fields encapsulated within Common Header) [0x7]			
Field	Size (bits) & Range	Description	Example
uid	64	Unique Hardware ID, sent to ensure response will be assigned correctly	0x001A2B3C4D5E
promotedToCH	1	If we were promoted to CH? (0/1)	0x1

networkID	8/24	New Network ID, valid if we were promoted in the previous flag. (our address will be networkID.254 in our new cluster)	0x03
reserved	X	TBD for Future Use	N/A

A DATA packet is the data transfer unit packet that actually fulfills the purpose of the network, sending data to a DST or broadcasting it. It is relatively simple, in that it just fields a length and the data.

Table 14. DATA Packet Description

DATA (Fields encapsulated within Common Header) [0x8]			
Field	Size (bits) & Range	Description	Example
len	8	Data Length (0 to 116 bytes)	0x20
data	0 to 912	Data Payload	…

The ACK packet is sent as a response to another packet that has the ack field in its header enabled. The packet implements an ack/nack however only ack is supported for this network version. The sequence number is also referenced to acknowledge a specific packet.

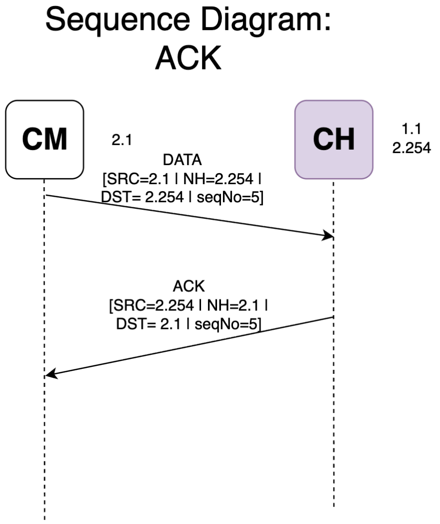

Figure 11. ACK Sequence Diagram

Fig 11. Showcases an ACK back sequence diagram, showcasing that the sequence number of the packet must match the ACK of the original packet. This sequence increases sequentially and wraps the 16-bit value when enough packets are sent.

Table 15. ACK Packet Description

ACK/NAK (Fields encapsulated within Common Header) [0x9]			
Field	Size (bits) & Range	Description	Example
ackNak	8	0: ACK ~~1: NAK~~ (**NOT IMPLEMENTED**)	0x00 (ACK)
seqNo	16	Sequence number for ACK/NACK reference	0x0005
reserved	X	TBD for Future Use	N/A

Network Data Tables

Our network describes 3 tables to store corresponding network information. A neighbor table, member table, and child network table. The neighbor table stores info about n-hop neighbors, the member table (CH only) stores info about cluster members, and the child network table (CH only) stores info about all the networks underneath a CH. They are described in further detail below.

Neighbor Table (Single Hop, All Cluster Members)

The neighbor table is present in every node, it allows a node to store a registry of its neighbors to avoid having to route packets through multiple hops and instead send the data directly. The table is updated when a node learns something about its neighbor. Basically on any RX, even if the message is not meant for our node we can fill in data like the dynamic address and lastHeard. Knowing roles and capabilities may require querying our neighbors with a heartbeat occasionally, however such info will be beneficial if we ever need to alter the network and we need to know what our neighbors are capable of.

Table 16. Neighbor Table Description

Neighbor Table (N=1 default; N>1 optional, All Cluster Members)		
Field	Size (bits)	Description
nodeAddr	16/48	Dynamic address of neighbor entry (NET.NODE)
nextHopAddr	16/48	Dynamic address of next hop en route to nodeAddr (NET.NODE)
hops	8	The hop count to reach this nodeAddr (to support N>1)
capabilities	8	**Flag Bits:** 0: capableClusterHead 1: Reserved 2: capableRouter 3: capableRootNode

		4: capableGateway
knownRoles	8	**Flag Bits:** 0: isClusterHead 1: isReg (Registered on Network?) 2: isRouter 3: isRootNode 4: isGateway
lastHeard	32	Number of timesteps we last heard the node active (could also be stored as unix time). Serves as stale/active, if too old purge from table.

Neighbor Table Maintenance Rules

- On any packet, store/update: srcAddress, and timestamp
- On HEARTBEAT, store: capabilities and roles
- Remove entries older than a few minutes (to maintain network mobility)

Member Table (Cluster Head Only)

Only CHs contain member tables, which simply store dynamic cluster addresses and Unique IDs, as well as expiryTime and address renewalState. On an Address Assignment request, we log the UID. And if we are able to assign an address, when we send the address response we set the address and set the renewState flag high, until we get an ACK back then we can confirm that the node has received its address and we can set the flag back low. The expiryTime is also logged when this occurs so we know how long the Address is valid for.

Table 17. Member Table Description

Member Table (Cluster Head Only)		
Field	Size (bits)	Description
memberAddr	16/48	Dynamic address of member (NET.NODE)
uid	64	Unique Hardware ID, to track associations between a HW node and dynamically assigned address.
renewState	8	A Flag that goes high when we receive an address renewal request, will not clear till we successfully receive the address assign response ACK back.
expiryTime	32	Time at which the dynamic address expires unless the node renews it, may be stored in delta timesteps or future unix timestamp. This also serves as the address “valid” flag. If expired remove address entry from table

Member Table Maintenance Rules

- On JOIN_REQ, create new entry and store: memberAddr, UID, renewState (0x01, waiting for ACK), expiryTime (future time at which entry is invalid)
- On the ACK to JOIN_ACK, set our renewState (0x00, address ACK'd)
- If we never receive an ACK to JOIN_ACK, remove entry
- On ADDR_RENEW, set a new expiryTime timestamp.
- On LEAVE, check if the UID matches, then remove the entry.
- On expiryTime (checked by timer task), remove entry.

Child Network Table (Cluster Head Only)

Similarly, only CHs contain Child Network Tables. These track which networks our node knows to be available on the network. Thus we store the address of the head, the networkID accessible via that address, the nextHop that takes us to the CH described in the entry, the # of hops (for routing decisions), and the last time we heard from the address on this route.

A Child Network Table is populated from NETID_REQ & on accepted NETID_RESP messages that pass through a node. Meaning that when we forward these, we inspect them to see if a new CH gets created. This lets us fill in the address (NETID_RESP DST), Network ID (NETID_RESP assignment), nextHop (from NETID_REQ packet), and hops (from NETID_REQ packet) fields.

Table 18. Child Net Table Description

Child Net Table (Cluster Head Only)		
Field	Size (bits)	Description
networkID	8/24	The child Network ID the CH exposes.
childAddr	16/48	Dynamic address of the child Cluster Head (NET.NODE)
nextHop	16/48	The subsequent nextHop that will get us to that child network.
hops	8/24	The number of hops it takes us to reach the network's Cluster Head.
netState	8	The validity of the entry in the table (0=VALID, 1=PENDING, 2=STALE)
lastHeard	32	Number of timesteps we last knew the route was active, may also be a unix timestamp

Child Net Table Maintenance Rules

- On NETID_REQ, create entry and store SRC, hops (before incrementing), netState (0x01, PENDING), nextHop (address RX'd from), and lastHeard.
- On corresponding NETID_RESP, if promotion is successful store networkID and netState (0x00, VALID), otherwise remove entry.
- On any packet from matching networkID, update lastHeard (and hops if possible).

- Mark entries older than a few minutes with netState (0x02, STALE), and purge much older STALE networks.

5. Conclusion

To summarize, we have implemented a hybrid cluster-tree mesh network layer that includes reactive and proactive (hybrid) routing principles, built on top of the IEEE 802.15.4 PHY/MAC, we can support up to ~64k nodes and send packets to any address from any address on the network.

Our algorithm uses several simple rules to scalably route packets to neighbors, and if that fails fall back to the tree based approach. Simple 1-hop neighbor tables are populated by simply listening to received transmissions. While child net tables describe networks below a specific CH, and member tables detail which CMs are part of a CHs cluster (network). This approach is robust because we have 2 routing schemes that we can fall back on if mesh routing fails. It is also efficient because we try to send packets directly to our neighbors if possible, reducing routing overhead and power consumption.

Future improvements could involve a larger neighbor table, and packets and methods like flooding used to populate it, giving us an AODV-type hybrid protocol. We can also improve our backbone by optimizing routing along the tree, and re-organizing nodes periodically.