```
// src/mesh/meshApi.ts

import { WnspFrame } from "../protocol/frameTypes";
import { encodeTextToFrames } from "../codec/
frameEncoder";
import { decodeFramesToText } from "../codec/
frameDecoder";

/**
 * High-level message shape visible to applications.
 */
export type OpticalMeshMessage = {
  fromId: string;
  toId?: string;
  text: string;
  frames: WnspFrame[];
};

/**
 * Optical mesh node interface.
 */
export interface OpticalMeshNode {
  id: string;
  sendMessageText(text: string, toId?: string): Promise<void>;
  onMessage(
    callback: (msg: { fromId: string; text: string; frames:
WnspFrame[] }) => void
  ): void;
}
```

```
/**
 * Simple in-memory bus for simulating an optical mesh
network.
 * In a real implementation, frames would be encoded into
light
 * and received via camera/photodiode.
 */
class InMemoryMeshBus {
  private static instance: InMemoryMeshBus;
  private subscribers: Map<
    string,
    (msg: OpticalMeshMessage) => void
  > = new Map();

  static getInstance(): InMemoryMeshBus {
    if (!InMemoryMeshBus.instance) {
      InMemoryMeshBus.instance = new InMemoryMeshBus();
    }
    return InMemoryMeshBus.instance;
  }

  subscribe(
    nodeId: string,
    handler: (msg: OpticalMeshMessage) => void
  ): void {
    this.subscribers.set(nodeId, handler);
  }

  publish(msg: OpticalMeshMessage): void {
    // If toId is specified, only deliver to that node; otherwise
```

```
broadcast.
    if (msg.toId) {
      const handler = this.subscribers.get(msg.toId);
      if (handler) handler(msg);
      return;
    }

    for (const [nodeId, handler] of this.subscribers.entries()) {
      if (nodeId === msg.fromId) continue;
      handler(msg);
    }
  }
}

/**
 * Concrete implementation of OpticalMeshNode using the in-
memory bus.
 */
export class InMemoryOpticalMeshNode implements
OpticalMeshNode {
  public id: string;
  private bus: InMemoryMeshBus;
  private messageHandlers: Array<
    (msg: { fromId: string; text: string; frames: WnspFrame[] })
=> void
  > = [];

  constructor(id: string) {
    this.id = id;
    this.bus = InMemoryMeshBus.getInstance();
    this.bus.subscribe(this.id, (msg) =>
this.handleIncoming(msg));
  }

  async sendMessageText(text: string, toId?: string):
Promise<void> {
    const frames = encodeTextToFrames(text);
    const meshMessage: OpticalMeshMessage = {
      fromId: this.id,
      toId,
      text, // human readable
      frames,
    };
    this.bus.publish(meshMessage);
  }

  onMessage(
    callback: (msg: { fromId: string; text: string; frames:
WnspFrame[] }) => void
  ): void {
    this.messageHandlers.push(callback);
  }

  private handleIncoming(msg: OpticalMeshMessage): void {
    const decodedText = decodeFramesToText(msg.frames);
    for (const handler of this.messageHandlers) {
      handler({
        fromId: msg.fromId,
        text: decodedText,
        frames: msg.frames,
      });
    }
  }
}
```