```python
# 2024 Â© Idan Hazay networking_s.py
# Import required libraries

import struct, traceback, socket  # Struct for data packing, traceback for debugging, socket for networking
from modules import encrypting_s  # Import encryption module
from modules.config_s import *  # Import configuration settings

class Network:
    """
    Handles network communication between server and clients.
    Supports encrypted data transmission, logging, and TCP communication.
    """

    def __init__(self, clients, bytes_recieved, bytes_sent, log=False):
        self.log = log  # Enable or disable logging
        self.clients = clients  # Dictionary of connected clients
        self.encryption = encrypting_s.Encryption()  # Encryption handler
        self.bytes_recieved = bytes_recieved  # Track bytes received per client
        self.bytes_sent = bytes_sent  # Track bytes sent per client

    def logtcp(self, dir, tid, byte_data):
        """
        Logs TCP traffic if logging is enabled.
        """
        if self.log:
            try:
                if str(byte_data[0]) == "0":
                    print("")  # Empty print for readability
            except Exception:
                return  # Ignore exceptions

            if dir == 'sent':
                print(f'{tid} S LOG:Sent     >>> {byte_data}')  # Log sent data
            else:
                print(f'{tid} S LOG:Recieved <<< {byte_data}')  # Log received data

    def send_data(self, sock, tid, bdata):
        """
        Sends data to a client.
        Supports encryption and adds packet length for proper parsing.
        """
        if self.clients[tid].encryption:  # Check if encryption is enabled
            encrypted_data = self.encryption.encrypt(bdata, self.clients[tid].shared_secret)  # Encrypt data
            data_len = struct.pack('!l', len(encrypted_data))  # Pack length as 4-byte integer
            to_send = data_len + encrypted_data  # Combine length header and encrypted data
            to_send_decrypted = str(len(bdata)).encode() + bdata  # Decrypted version for logging
            self.logtcp('sent', tid, to_send)  # Log encrypted data
            self.logtcp('sent', tid, to_send_decrypted)  # Log decrypted data
        else:
            data_len = struct.pack('!l', len(bdata))  # Pack unencrypted data length
            to_send = data_len + bdata  # Combine length and data
            self.logtcp('sent', tid, to_send)  # Log sent data

        try:
            self.bytes_sent[tid] += len(to_send)  # Track bytes sent
            sock.send(to_send)  # Send data
        except ConnectionResetError:
            pass  # Handle client disconnection

    def recv_data(self, sock, tid):
        """
        Receives data from a client.
        Reads packet length first, then retrieves the full message.
        """
        try:
            b_len = b''
            while len(b_len) < LEN_FIELD:  # Ensure full length field is received
                b_len += sock.recv(LEN_FIELD - len(b_len))  # Read remaining bytes

            self.bytes_recieved[tid] += len(b_len)  # Track bytes received
            msg_len = struct.unpack("!l", b_len)[0]  # Extract message length

            if msg_len == b'':
                print('Client seems to have disconnected')  # Detect disconnection

            msg = b''
            while len(msg) < msg_len:  # Keep reading until full message is received
                chunk = sock.recv(msg_len - len(msg))
                self.bytes_recieved[tid] += len(chunk)  # Track bytes received
                if not chunk:
                    print('Server disconnected abnormally.')  # Handle unexpected disconnection
                    break
                msg += chunk

            if tid in self.clients and self.clients[tid].encryption:
                self.logtcp('recv', tid, b_len + msg)  # Log encrypted data
                msg = self.encryption.decrypt(msg, self.clients[tid].shared_secret)  # Decrypt message
```

```
            self.logtcp('recv', tid, str(msg_len).encode() + msg)  # Log decrypted data

        return msg  # Return received message

    except ConnectionResetError:
        return None  # Handle client disconnection
    except Exception as err:
        print(traceback.format_exc())  # Log error

@staticmethod
def dhcp_listen(local_ip, port):
    """
    Listens for DHCP discovery requests and responds with server information.
    Used for automatic client-server connection.
    """
    dhcp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create UDP socket
    dhcp_socket.bind(("", 31026))  # Listen on UDP port 31026

    while True:
        data, addr = dhcp_socket.recvfrom(1024)  # Receive data from clients
        if data.decode() == "SEAR":  # Check if the message is a search request
            response_message = f"SERR|{local_ip}|{port}"  # Construct response with server details
            dhcp_socket.sendto(response_message.encode(), addr)  # Send response to client
```