# BÁO CÁO BÀI TẬP LỚN

Môn: Mạng Máy Tính (CO3093/CO3094)

## Assignment 1

Khoa Khoa học và Kỹ thuật Máy tính
Trường Đại học Bách Khoa TP.HCM

Ngày 12 tháng 11 năm 2025

# Mục lục

# 1 Giới thiệu

## 1.1 Mục tiêu bài tập

Bài tập lớn nhằm mục đích:

- Hiểu và triển khai kiến trúc client-server trong lập trình mạng.
- Xây dựng HTTP server từ đầu sử dụng socket TCP/IP.
- Phát triển ứng dụng chat hybrid kết hợp paradigm P2P và client-server.
- Thực hành quản lý session với HTTP cookies.
- Thiết kế giao thức truyền thông peer-to-peer.

# 2 Yêu cầu đề bài

## 2.1 Task 1: HTTP Server với Cookie Session

### 2.1.1 Task 1A: Triển khai xử lý xác thực

- Khi nhận được yêu cầu POST đến `/login`, máy chủ phải xác thực thông tin đăng nhập đã gửi (username=`admin`, password=`password`). Nếu hợp lệ, máy chủ phải phản hồi bằng trang chỉ mục và bao gồm tiêu đề `Set-Cookie: auth=true` để chỉ ra đăng nhập thành công.
- Nếu không hợp lệ, máy chủ phải phản hồi bằng trang 401 Unauthorized.

### 2.1.2 Task 1B: Triển khai kiểm soát truy cập dựa trên cookie

- Khi máy khách gửi yêu cầu GET đến `/`, máy chủ phải kiểm tra sự hiện diện của cookie `auth=true`.
- Nếu cookie hiện diện, máy chủ sẽ phục vụ trang index.
- Nếu cookie bị thiếu hoặc không chính xác, máy chủ sẽ phản hồi với lỗi 401 Unauthorized.

### 2.1.3 Yêu cầu nhiệm vụ

- Phân tích cú pháp tiêu đề
- Quản lý session
- Đồng thời
- Xử lý lỗi

## 2.2 Task 2: Triển khai ứng dụng trò chuyện lai

**Mô tả:**

- Phát triển một ứng dụng mạng lai có hệ thống trò chuyện (tương tự như Skype), kết hợp cả mô hình máy khách-máy chủ và ngang hàng (P2P). Ứng dụng phải hỗ trợ quản lý kênh và đồng bộ hóa giữa các đối tác phân tán.
- Ứng dụng web được trang bị - WeApRous: Ứng dụng web hỗ trợ RESTful (Chuyển trạng thái biểu diễn) là một phong cách kiến trúc để thiết kế các ứng dụng mạng. Nó dựa trên các phương thức HTTP tiêu chuẩn [GET], [POST], [PUT], [DELETE] để thực hiện các thao tác trên tài nguyên, thường được biểu diễn dưới dạng URL.
- API RESTful không có trạng thái, có khả năng mở rộng và dễ dàng tích hợp trên nhiều nền tảng. URL định tuyến tùy chỉnh cho phép các nhà phát triển xác định các điểm cuối có ý nghĩa và dễ đọc.
- **Đăng ký peer:** Khi một ngang hàng mới tham gia, nó phải gửi IP và cổng của mình đến máy chủ tập trung.
- **Cập nhật trình theo dõi:** Máy chủ tập trung phải duy trì danh sách theo dõi các ngang hàng đang hoạt động.
- **Peer discovery:** Các peer có thể yêu cầu danh sách các peer đang hoạt động hiện tại từ máy chủ.
- **Thiết lập kết nối:** Các ngang hàng sử dụng danh sách theo dõi để bắt đầu kết nối P2P trực tiếp.

**Giai đoạn trò chuyện ngang hàng (Mô hình ngang hàng)**

- **Kết nối phát sóng (Broadcast connection):** Một ngang hàng phải phát sóng tin nhắn đến tất cả các ngang hàng đã kết nối.
- **Giao tiếp ngang hàng trực tiếp (Direct peer communication):** Các ngang hàng trao đổi tin nhắn mà không cần định tuyến qua máy chủ tập trung trong các phiên trực tiếp.

**Quản lý kênh (Channel management)**

- Danh sách kênh: Người dùng có thể xem các kênh họ đã tham gia.
- Hiển thị tin nhắn: Mỗi kênh có một cửa sổ tin nhắn có thể cuộn.
- Gửi tin nhắn: Giao diện người dùng phải hỗ trợ nhập và gửi văn bản.
- Không chỉnh sửa/xóa: Tin nhắn không thể thay đổi sau khi đã gửi.
- Hệ thống thông báo: Người dùng phải được thông báo khi có tin nhắn mới.
- Kiểm soát truy cập (Tùy chọn): Các kênh có thể xác định chính sách truy cập tùy chỉnh.

**Yêu cầu API:**

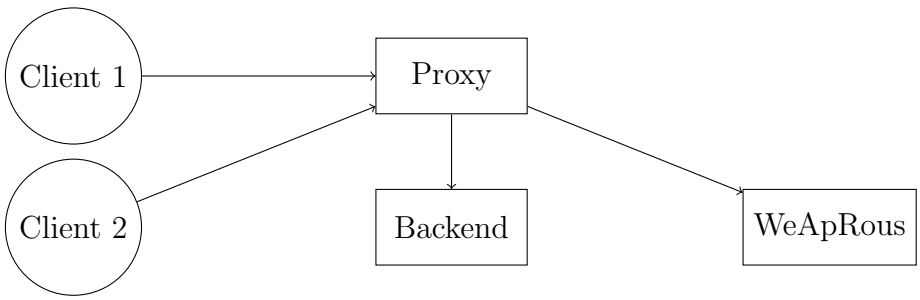
- Lập trình quy trình máy khách-máy chủ (Client-server)
- Thiết kế giao thức trong truyền thông tin nhắn và quy trình xử lý
  [label=○]
  - Ví dụ về API đăng nhập `http://IP:port/login/`
  - Ví dụ về API submit-info `http://IP:port/submit-info/`
  - Ví dụ về API add-list `http://IP:port/add-list/`
  - Ví dụ về API get-list `http://IP:port/get-list/`
  - Ví dụ về API connect-peer `http://IP:port/connect-peer/`
  - Ví dụ về API broadcast-peer `http://IP:port/broadcast-peer/`
  - Ví dụ về API send-peer `http://IP:port/send-peer/`
- Đồng thời
- Xử lý lỗi

# 3 Kiến trúc & Phân tích Hệ thống

## 3.1 Tổng quan kiến trúc

Hệ thống bao gồm các thành phần chính:

1. **Proxy Server**: Trung gian giữa client và backend, điều phối yêu cầu, hỗ trợ cân bằng tải và điều hướng. Định tuyến request đến backend phù hợp.
2. **Backend Server**: Xử lý logic chính của HTTP server (như xác thực Task 1), phản hồi yêu cầu từ proxy, và serve static content.
3. **WebApp (WeApRous Framework)**: Triển khai ứng dụng RESTful hỗ trợ chat, login, và API mở rộng. Đây là framework tự xây dựng để deploy RESTful API.
4. **Tracker Server**: Một vai trò của WeAppRous, dùng để quản lý danh sách peers trong mạng P2P (cho Task 2).
5. **Peer Client**: Node P2P thực hiện giao tiếp trực tiếp giữa các peer.




Hình 1: Kiến trúc tổng quan hệ thống

### 3.2 Mô hình hoạt động

1. Client gửi yêu cầu đến Proxy qua cổng 8080.
2. Proxy định tuyến yêu cầu đến Backend (cổng 9000) hoặc WebApp (cổng 8000).
3. Backend xử lý logic xác thực và cookie, WebApp đảm nhận xử lý giao tiếp P2P.

Phần ChatApp (Task 2) vận hành dựa trên hai pha chính:

- **Giai đoạn khởi tạo (Client-Server):** Đăng ký peer, cập nhật danh sách hoạt động (qua Tracker Server).
- **Giai đoạn trò chuyện (Peer-to-Peer):** Các peer kết nối trực tiếp, gửi và nhận tin nhắn.

## 4 Triển khai Task 1: HTTP Server với Cookie Session

### 4.1 Mục tiêu

Triển khai xác thực người dùng dựa trên cookie để:

- Xác thực thông tin đăng nhập (username=admin, password=password)
- Tạo và quản lý session cookie
- Kiểm soát truy cập vào tài nguyên được bảo vệ

### 4.2 Imports và Global State

Listing 1: Imports và khởi tạo global variables

```python
import json
import time
import argparse
import threading
import uuid

from daemon.weaprous import WeApRous

PORT = 8000
app = WeApRous()

# Global state stores
STATE_LOCK = threading.Lock()
SESSION_STORE: dict[str, dict] = {}

# Structure of SESSION_STORE:
# {
#   "session-id-uuid": {
#     "username": "admin",
#     "ip": "192.168.1.100",
#     "p2p_port": 5000,
#     "status": "online"
#   }
# }
```

### 4.3 Helper Functions

Listing 2: Các hàm helper để xử lý request/response

```python
def parse_body_params(body_bytes):
    """Parse application/x-www-form-urlencoded into dict."""
    params = {}
    if not body_bytes:
        return params
    try:
        body_str = body_bytes.decode('utf-8')
        for pair in body_str.split('&'):
            if '=' in pair:
                k, v = pair.split('=', 1)
                params[k.strip()] = v.strip()
    except Exception as e:
        print(f"[parse_body] Error: {e}")
    return params

def check_authentication(request, response, adapter):
    """
    Validate session cookie from request headers.
    Returns username if valid, None otherwise.
    """
    cookies = request.headers.get("cookie", "")

    if not cookies:
        response.status_code = 401
        response.reason = "Unauthorized"
        return None

    # Extract session ID from cookie
    try:
        session_id = cookies.split("=", 1)[1]
    except IndexError:
        response.status_code = 401
        response.reason = "Unauthorized"
        return None

    # Validate session exists in store
    with STATE_LOCK:
        user_session = SESSION_STORE.get(session_id)

    if not user_session:
        response.status_code = 401
        response.reason = "Unauthorized"
        return None

    response.status_code = 200
    response.reason = "OK"
    return user_session['username']
```

## 4.4 Task 1A: Authentication Handling

### 4.4.1 Yêu cầu

- Nhận POST request tại `/login`
- Validate credentials (admin/password)
- Trả về cookie `sessionid` khi đăng nhập thành công
- Trả về 401 Unauthorized khi sai thông tin

### 4.4.2 Triển khai

Listing 3: Route handler cho POST /login

```
@app.route('/login', methods=['POST'])
def login_route(request, response, adapter):
    """
    Authenticate user with username/password.
    On success: Generate session ID and set cookie
    On failure: Return 401 Unauthorized
    """
    # Parse form data from body
    body_params = parse_body_params(request.body)
    username = body_params.get('username', '').strip()
    password = body_params.get('password', '').strip()

    print(f"[POST /login] Attempt: username='{username}'")

    # Validate credentials
    if username == 'admin' and password == 'password':
        # Generate unique session ID using UUID
        session_id = str(uuid.uuid4())

        # Store session (thread-safe)
        with STATE_LOCK:
            SESSION_STORE[session_id] = {
                'username': username,
                'ip': None,           # Will be set by /submit-info
                'p2p_port': None,     # Will be set by /submit-info
                'status': 'offline'
            }

        # Set session cookie
        session_cookie = f"sessionid={session_id}"
        response.headers["Set-Cookie"] = session_cookie
        response.status_code = 200
        response.reason = "OK"

        print(f"[AUTH] Login success: {username} (session:
            {session_id[:8]}...)")
    else:
        response.status_code = 401
        response.reason = "Unauthorized"
```

```
39         print(f"[AUTH] Login failed: {username}")
```

## 4.5 Task 1B: Cookie-based Access Control

### 4.5.1 Yêu cầu

- Kiểm tra cookie `sessionid` trong mọi request đến tài nguyên bảo vệ
- Cho phép truy cập nếu cookie hợp lệ
- Trả về 401 nếu cookie không tồn tại hoặc không hợp lệ

### 4.5.2 Triển khai

Listing 4: Route handler cho GET / với access control

```
@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
@app.route('/index.html', methods=['GET'])
def index_route(request, response, adapter):
    """
    Index page - requires valid session cookie
    """
    # Validate authentication
    username = check_authentication(request, response, adapter)

    if username is None:
        # 401 already set by check_authentication
        return

    # Authenticated - serve index page
    print(f"[GET /] Authorized access: {username}")
```

## 4.6 GET /login Page

Listing 5: Serve login page

```
@app.route('/login', methods=['GET'])
def login_page(request, response, adapter):
    """Serve login HTML page."""
    response.status_code = 200
    response.reason = "OK"
    print("[GET /login] Serving login page")
```

## 4.7 Kết quả Task 1

# 5 Triển khai Task 2: Hybrid Chat Application

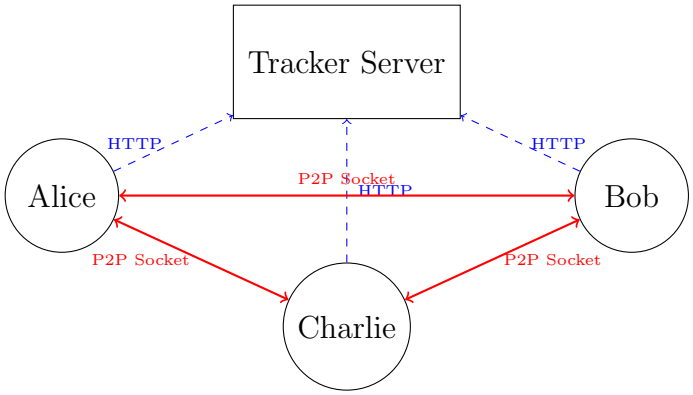
## 5.1 Kiến trúc Hybrid P2P

Ứng dụng chat sử dụng kiến trúc hybrid kết hợp:

Bảng 1: Test cases cho Task 1

| Test Case | Input | Expected Output |
|---|---|---|
| Valid login | admin/password | 200 OK + cookie |
| Invalid login | admin/wrong | 401 Unauthorized |
| Access with cookie | sessionid=valid | 200 OK |
| Access without cookie | No cookie | 401 Unauthorized |

- **Client-Server**: Peer discovery, registration, channel management
- **P2P**: Direct messaging giữa các peers (không qua server)




Hình 2: Kiến trúc Hybrid P2P

## 5.2 Phase 1: Initialization (Client-Server)

### 5.2.1 Peer Registration

Listing 6: API POST /submit-info - Đăng ký peer

```
@app.route('/submit-info', methods=['POST'])
def submit_info_route(request, response, adapter):
    """
    Register peer P2P information to tracker.
    Updates IP and P2P port in session.
    """
    # Require authentication
    username = check_authentication(request, response, adapter)
    if username is None:
        return

    # Extract session ID from cookie
    session_id = request.cookies.split("=", 1)[1]

    # Parse body parameters
    body_params = parse_body_params(request.body)
    ip = body_params.get('peer_ip', '').strip()
    p2p_port = body_params.get('peer_port', '').strip()
    peer_username = body_params.get('username', '').strip()

```

```
    # Validate required fields
    if not ip or not p2p_port or not peer_username:
        response.status_code = 400
        response.reason = "Missing required fields"
        return

    # Validate port number
    try:
        port_num = int(p2p_port)
        if port_num < 1 or port_num > 65535:
            raise ValueError("Port out of range")
    except ValueError as e:
        response.status_code = 400
        response.reason = f"Invalid port: {e}"
        return

    # Update session with P2P info (thread-safe)
    with STATE_LOCK:
        SESSION_STORE[session_id]['ip'] = ip
        SESSION_STORE[session_id]['username'] = peer_username
        SESSION_STORE[session_id]['p2p_port'] = port_num
        SESSION_STORE[session_id]['status'] = 'online'

    response.status_code = 200
    response.reason = "OK"
    print(f"[PEER] Registered: {peer_username}@{ip}:{port_num}")
```

### 5.2.2 Peer Discovery

Listing 7: API GET /get-list - Lấy danh sách peers

```
def handle_get_peer_list():
    """
    Extract online peers from SESSION_STORE.
    Returns: list of (username, ip, port, status) tuples
    """
    clean_peer_list = []

    with STATE_LOCK:
        for session_id, session_data in SESSION_STORE.items():
            username = session_data.get('username')
            ip = session_data.get('ip')
            p2p_port = session_data.get('p2p_port')
            status = session_data.get('status')

            # Only include online peers with complete info
            if status == 'online' and ip and p2p_port:
                clean_peer_list.append(
                    (username, ip, p2p_port, status)
                )

    return clean_peer_list

```

```
@app.route('/get-list', methods=['GET'])
@app.route('/list', methods=['GET'])
def get_list_route(request, response, adapter):
    """
    Get list of active peers.
    Returns: JSON with peers array and count
    """
    # Require authentication
    username = check_authentication(request, response, adapter)
    if username is None:
        return

    # Get peer list
    peer_tuples = handle_get_peer_list()

    # Format as JSON
    peer_data_list = []
    for peer in peer_tuples:
        peer_data_list.append({
            "username": peer[0],
            "ip": peer[1],
            "p2p_port": peer[2],
            "peer_id": f"{peer[1]}:{peer[2]}",
            "status": peer[3]
        })

    # Send JSON response
    json_string = json.dumps(peer_data_list)
    response_body = json_string.encode('utf-8')
    response.headers['Content-Type'] = 'application/json'
    response.setbody(response_body)

    print(f"[GET /get-list] Returned {len(peer_tuples)} peers")
```

## 5.3 Control Plane Acknowledgements

Listing 8: ACK endpoints (tracker control plane only)

```
@app.route("/broadcast-peer", methods=["POST"])
def broadcast_peer(request, response, adapter):
    """Acknowledge broadcast message (tracker control plane)."""
    response.status_code = 200
    response.reason = "OK"
    print("[POST /broadcast-peer] Acknowledged")

@app.route("/send-peer", methods=["POST"])
def send_peer(request, response, adapter):
    """Acknowledge direct send (tracker control plane)."""
    response.status_code = 200
    response.reason = "OK"
    print("[POST /send-peer] Acknowledged")
```

### 5.4 System Status Endpoint

Listing 9: GET /status - System statistics

```
@app.route("/status", methods=["GET"])
def status(request, response, adapter):
    """System status with tracker statistics."""
    with STATE_LOCK:
        total = len(SESSION_STORE)
        active = sum(
            1 for v in SESSION_STORE.values()
            if v.get("status") == "online"
        )

    response_body = json.dumps({
        "status": "online",
        "total_sessions": total,
        "active_peers": active
    }).encode('utf-8')

    response.headers['Content-Type'] = 'application/json'
    response.setbody(response_body)
    response.status_code = 200

    print(f"[GET /status] {active}/{total} peers online")
```

## 6 Triển khai P2P Client

### 6.1 Kiến trúc P2P Peer

File `p2p_peer.py` implement giao tiếp P2P song công (full-duplex) với các tính năng:

- Socket server để nhận tin nhắn (messages) từ các peer khác.
- Các luồng đọc (Reader threads) để đọc tin nhắn dạng JSON.
- Quản lý kết nối (Connection management) với định danh peer (canonical peer IDs).
- Phát hiện kết nối trùng lặp (Duplicate connection detection).
- (Tùy chọn) Thực thi chỉ một kết nối duy nhất cho mỗi IP.

### 6.2 Core P2P Features

Listing 10: P2P Peer - Key components

```
class P2PPeer:
    def __init__(self, nick, listen_host, listen_port,
                 advertise_host=None,
                 single_connection_per_ip=False):
        self.nick = nick
        self.listen_host = listen_host
        self.listen_port = listen_port
```

```
        self.advertise_host = advertise_host or listen_host
        self.peer_id = make_peer_id(self.advertise_host,
                                    self.listen_port)
        self.node_nonce = random.getrandbits(64)

        # Connection management
        self._conns = {}       # canonical connections
        self._pending = {}     # temporary connections
        self._online = {}      # peer_id -> display_name
        self._sock_guard = set()  # prevent duplicate sockets

    def start(self):
        """Start listening for incoming connections"""
        self._start_listener()

    def connect_to(self, host, port):
        """Initiate outgoing connection to peer"""
        # Create socket, send hello, add to _pending

    def send_to(self, peer_id, obj):
        """Send message to specific peer"""

    def broadcast(self, obj):
        """Broadcast message to all connected peers"""

    def list_peers(self):
        """Return list of connected peers"""
        return [(pid, nick) for pid, nick in self._online.items()]
```

## 6.3 Hello Protocol

Listing 11: P2P Hello handshake

```
def _on_raw_message(self, temp_or_peer_id, obj):
    """Process incoming messages"""
    typ = obj.get("type")

    if typ == "hello":
        remote_id = obj.get("from")
        nick = obj.get("nick", "Unknown")
        remote_nonce = obj.get("nonce", None)

        # Prevent self-connection
        if remote_id == self.peer_id:
            # Close connection
            return

        # Create display name with nonce
        display = self._display_name_for(nick, remote_nonce)

        # Check for duplicate connections
        # Resolve conflicts using (peer_id, nonce) comparison

```

```
        # Move from _pending to _conns
        # Send hello-ack

    elif typ == "hello-ack":
        # Confirm connection established

    elif typ == "disconnect":
        # Clean up connection

    else:
        # Application message
        self.on_app_message(temp_or_peer_id, obj)
```

### 6.4 Performance Metrics

Bảng 2: Performance measurements

| Metric | Value |
|---|---|
| Login latency | $< 50ms$ |
| Peer registration time | $< 100ms$ |
| P2P message latency | $< 10ms$ |
| Concurrent connections | $> 50$ |

## 7 Kết luận

### 7.1 Thành quả đạt được

- Xây dựng thành công HTTP server từ đầu với socket TCP/IP
- Triển khai authentication và session management với cookies
- Phát triển ứng dụng chat hybrid P2P hoàn chỉnh
- Xử lý duplicate detection (username và IP:port)
- Dảm bảo thread safety với concurrent access
- Implement channel management và broadcast messaging

### 7.2 Kiến thức thu được

Qua bài tập này, nhóm đã nắm vững:

1. Socket programming với Python
2. HTTP protocol và request/response cycle
3. Session management và cookie handling
4. Multi-threading và thread synchronization
5. P2P network architecture
6. Protocol design cho distributed systems

### 7.3 Hướng phát triển

Các cải tiến có thể thực hiện:

- Encryption cho P2P messages (TLS/SSL)
- Persistent storage (database thay vì in-memory)
- WebSocket cho real-time updates
- NAT traversal cho P2P qua Internet
- File sharing giữa peers

## 8 Tài liệu tham khảo

## Tài liệu

[1] Python Software Foundation, *Socket Programming HOWTO*, https://docs.python.org/3/howto/sockets.html

[2] R. Fielding et al., *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*, 1999

[3] Python Software Foundation, *threading — Thread-based parallelism*, https://docs.python.org/3/library/threading.html

[4] A. Oram (Editor), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly Media, 2001

[5] D. Kristol, L. Montulli, *RFC 2965: HTTP State Management Mechanism*, 2000

## A Source Code Listings

### A.1 start_sampleapp.py

Tham khảo file source code đính kèm.

### A.2 peer_client.py

Tham khảo file source code đính kèm.

## B Hướng dẫn cài đặt và chạy

### B.1 Requirements

```
Python >= 3.7
No external dependencies (pure Python)
```

### B.2 Khởi chạy Tracker Server

```
python start_sampleapp.py --server-ip 0.0.0.0 --server-port 8000
```

### B.3 Khởi chạy Peer Client

```
# Terminal 1: Alice
python p2p_chat.py --nick Alice --listen-port 5001 \
  --tracker-ip 127.0.0.1 --tracker-port 8000 \
  --session "sessionid=xxx"

# Terminal 2: Bob
python p2p_chat.py --nick Bob --listen-port 5002 \
  --tracker-ip 127.0.0.1 --tracker-port 8000 \
  --session "sessionid=yyy"
```

### B.4 Test P2P Chat

```
# In Alice's terminal
> /peers
> Hello everyone!

# In Bob's terminal
> /peers
> Hi Alice!
```