

네트워크 보안 게임 상세 설계 문서 v2.0

문서 개정 이력

- **v1.0:** 초기 계획서 및 기본 설계
- **v2.0:** 세부 로직, 아키텍처 상세 설명 및 모순점 검증 추가

프로젝트 개요

프로젝트명

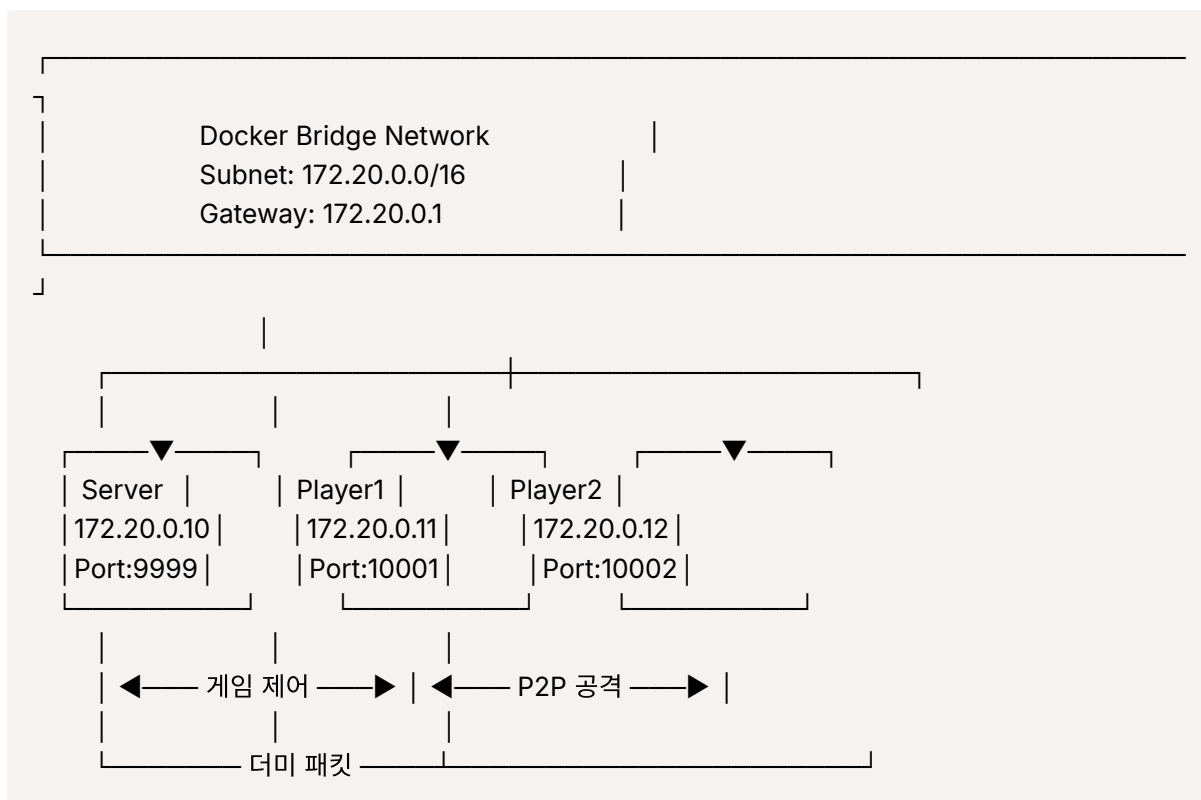
TCP 패킷 분석 기반 네트워크 방어 게임 (Network Security Defense Game)

핵심 목표

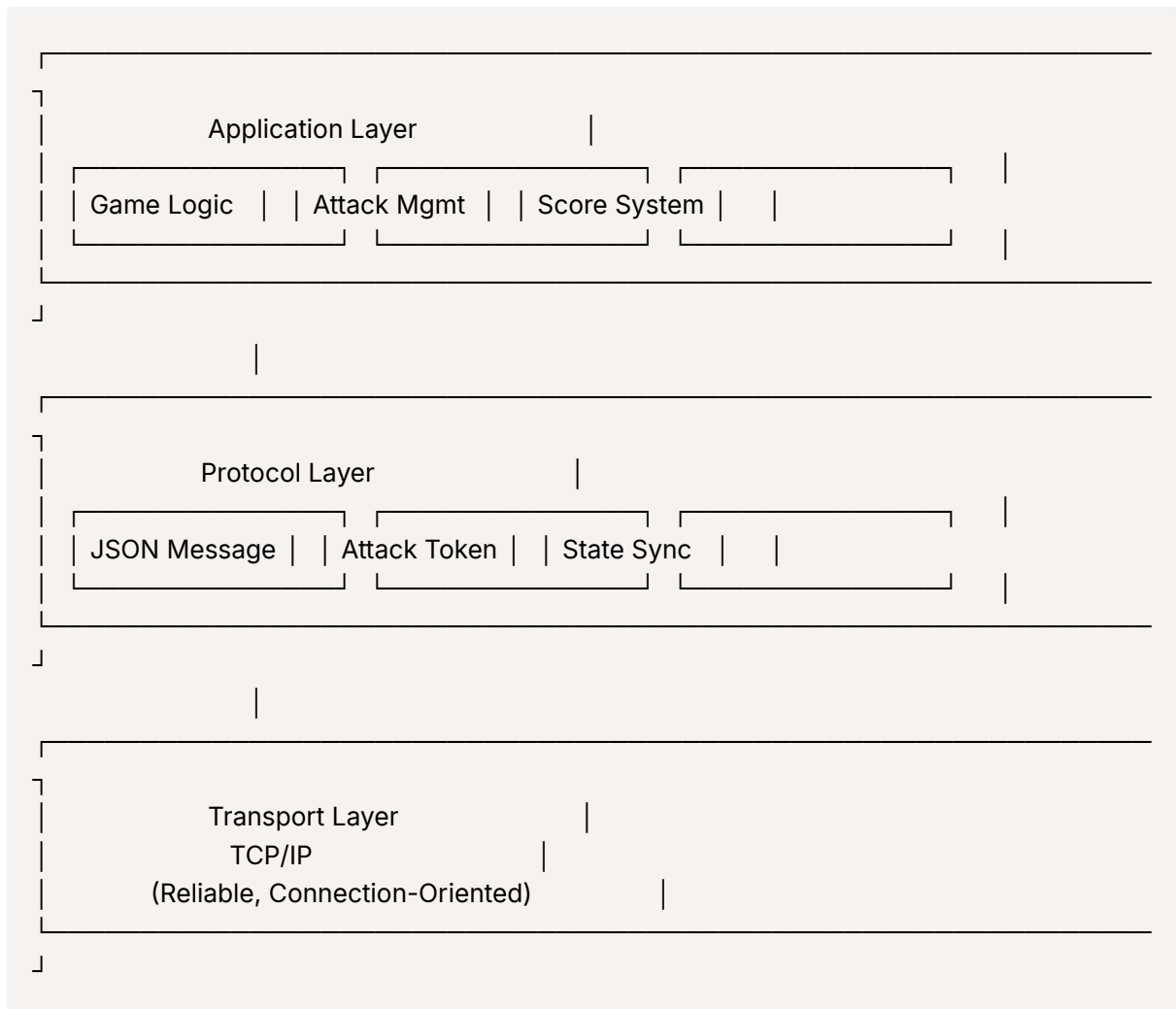
1. **교육적 목표:** TCP/IP 프로토콜과 패킷 분석 실습
2. **기술적 목표:** Docker 기반 격리된 네트워크 환경 구축
3. **게임 목표:** 점진적 난이도 상승을 통한 학습 효과 극대화

시스템 아키텍처 (상세)

1. 전체 네트워크 구조



2. 통신 프로토콜 계층 구조



🔄 핵심 게임 로직 (상세 분석)

1. 하이브리드 아키텍처: 서버 중심 + P2P

설계 철학

- **서버:** 게임 상태 관리, 공격 승인, 검증
- **P2P:** 실제 공격 패킷 전송 (IP 노출을 통한 Wireshark 분석)

아키텍처 선택 이유

순수 서버 중심 방식의 문제:

- × 서버가 모든 공격을 중계 → IP가 항상 서버 IP로 표시
- × Wireshark에서 출발지 IP 분석 불가능
- × 게임의 핵심 메커니즘(IP 기반 공격자 탐지) 불가능

순수 P2P 방식의 문제:

- × 공격 횟수 제한 우회 가능
- × 부정 행위 방지 불가능
- × 게임 상태 동기화 어려움

하이브리드 방식의 장점:

- ✓ 서버가 게임 규칙 강제 (승인 시스템)
- ✓ P2P로 실제 네트워크 환경 시뮬레이션
- ✓ Wireshark에서 실제 출발지 IP 확인 가능

2. 공격 승인 시스템 (Attack Approval System)

2.1 전체 플로우

플레이어 A가 플레이어 B를 공격하려는 경우:

1단계: 공격 요청

```
Player A → Server: {  
  "type": "ATTACK_REQUEST",  
  "target_id": "player_b",  
  "player_id": "player_a"  
}
```

2단계: 서버 검증

Server 내부 로직:

```
if player_attacks[player_a] >= attack_limit:  
    → 거부 (횟수 초과)  
else:  
    → 승인 프로세스 시작
```

3단계: 공격 토큰 생성 및 배포

```
attack_id = f"{from_id}→{to_id}_{timestamp}"  
  
# pending_attacks에 등록  
pending_attacks[attack_id] = {  
  'from': player_a,  
  'to': player_b,  
  'timestamp': time.time(),  
  'attacker_sent': False,  # 공격자가 실제로 보냈는가?  
  'target_received': False, # 타겟이 실제로 받았는가?  
  'timeout': 5.0          # 5초 타임아웃  
}
```

4단계: 양측에 정보 전달

```
Server → Player A: {  
  "type": "ATTACK_APPROVED",
```

```
"attack_id": "player_a→player_b_1234567890",
"target_ip": "172.20.0.12",
"target_port": 10002
}
```

```
Server → Player B: {
  "type": "INCOMING_ATTACK_WARNING",
  "attack_id": "player_a→player_b_1234567890",
  "attacker_ip": "172.20.0.11"
}
```

5단계: P2P 공격 전송

Player A → Player B (직접):

TCP 연결: 172.20.0.11 → 172.20.0.12:10002

```
Payload: {
  "type": "ATTACK",
  "attack_id": "player_a→player_b_1234567890",
  "from": "player_a"
}
```

 이 순간 Wireshark에서:

Source IP: 172.20.0.11 ← 실제 공격자 IP!

Dest IP: 172.20.0.12

Port: 10002

6단계: 양방향 확인

```
Player A → Server: {
  "type": "ATTACK_SENT_CONFIRM",
  "attack_id": "player_a→player_b_1234567890"
}
```


→ pending_attacks[attack_id]['attacker_sent'] = True

```
Player B → Server: {
  "type": "ATTACK_RECEIVED_CONFIRM",
  "attack_id": "player_a→player_b_1234567890"
}
```

→ pending_attacks[attack_id]['target_received'] = True

7단계: 공격 완료 처리

if attacker_sent AND target_received:

```
#  정상 공격 완료
player_attacks[player_a] += 1
real_attacks.append({
  'from': player_a,
```

```

        'to': player_b,
        'timestamp': timestamp,
        'is_real': True
    })

    # UI 업데이트
    send_to_player(player_a, {
        "type": "ATTACK_COUNT_UPDATE",
        "remaining": attack_limit - player_attacks[player_a]
    })

```

2.2 부정 행위 방지 메커니즘

시나리오 1: 승인 없이 공격 시도

Player A가 승인 없이 Player B에게 패킷 전송

Player B:

```

attack_id = 받은 패킷의 attack_id
if attack_id not in expected_attacks:
    # ⚠️ 예상하지 않은 공격!
    # 서버에 보고하지 않음
    # 무시
    return

```

→ 서버는 이 공격을 카운트하지 않음

→ 무효 공격

시나리오 2: 승인 1번으로 여러 번 공격 시도

Player A가 동일한 attack_id로 2번 전송

첫 번째 공격:

```

Player B: expected_attacks에서 attack_id 제거 후 서버에 보고
→ 정상 처리

```

두 번째 공격 (같은 attack_id):

```

Player B: expected_attacks에 해당 ID 없음
→ 무시

```

→ 한 번의 승인은 한 번의 공격만 유효

시나리오 3: IP 위조 시도

Player A가 Player C인 척 하고 공격

Server → Player B: "Player A가 공격할 것이다. IP: 172.20.0.11"

Player B가 받은 패킷:

실제 출발지 IP: 172.20.0.13 (Player C의 IP)

expected_attacker_ip: 172.20.0.11 (Player A의 IP)

if 실제_IP != expected_IP:

IP 불일치!

무시

return

→ IP 위조 불가능 (Docker 네트워크에서 IP 스푸핑 차단됨)

시나리오 4: 타임아웃 악용

Player A가 승인 받고 10초 후에 전송 시도

Server (5초 후):

check_attack_timeout(attack_id)

if not (attacker_sent and target_received):

타임아웃!

del pending_attacks[attack_id]

Player B:

expected_attacks에서 이미 삭제됨

→ 무시

→ 타임아웃 후 공격 무효

3. 더미 패킷 시스템 (Dummy Traffic System)

3.1 목적

1. 노이즈 생성: 진짜 공격 패킷을 숨김
2. 네트워크 분석 난이도 증가
3. 실제 네트워크 트래픽 환경 시뮬레이션

3.2 구현 로직

```
def generate_dummy_packets(self):  
    """  
    더미 패킷 생성 스레드  
    - 라운드가 진행되는 동안 지속적으로 실행
```

```

- 난이도에 따라 전송 간격 조정
"""

while self.round_active:
    # 난이도 설정에서 간격 가져오기
    interval = self.difficulty_config['dummy_interval']
    # R1: 2.0초, R2: 1.5초, R3: 1.0초, R4: 0.8초, R5: 0.5초

    # 모든 플레이어에게 더미 패킷 전송
    for player_id, player_info in self.players.items():
        dummy_packet = {
            "type": "DUMMY",
            "timestamp": time.time(),
            "sequence": self.dummy_sequence,
            "data": os.urandom(16).hex() # 랜덤 데이터
        }

        self.send_to_player(player_info, dummy_packet)
        self.dummy_sequence += 1

    time.sleep(interval)

```

3.3 Wireshark에서 보이는 더미 패킷

Time	Source	Destination	Port	Protocol	Info
—					
0.500	172.20.0.10	→ 172.20.0.11	9999	TCP	DUMMY
1.000	172.20.0.10	→ 172.20.0.11	9999	TCP	DUMMY
1.500	172.20.0.10	→ 172.20.0.11	9999	TCP	DUMMY
2.000	172.20.0.10	→ 172.20.0.11	9999	TCP	DUMMY

특징:

- ✓ 출발지 IP: 항상 172.20.0.10 (서버)
- ✓ 목적지 Port: 9999 (서버 제어 포트)
- ✓ 규칙적인 간격
- ✓ 페이로드: {"type": "DUMMY", ...}

4. 노이즈 트래픽 시스템 (R3+)

4.1 목적

라운드 3부터 추가되는 배경 트래픽:

- 플레이어 간 무의미한 통신

- 서버가 아닌 다른 IP에서 오는 패킷
- 진짜 공격과 구분이 더 어려워짐

4.2 구현

```
def generate_noise_traffic(self):
    """
    노이즈 트래픽 생성 (R3 이상에서만)
    - 플레이어 간 무작위 통신 시뮬레이션
    - 실제 공격과 혼동 유발
    """

    if not self.difficulty_config['noise_traffic']:
        return

    while self.round_active:
        # 랜덤하게 두 플레이어 선택
        players = list(self.players.values())
        if len(players) < 2:
            continue

        sender, receiver = random.sample(players, 2)

        # 노이즈 패킷 전송 명령
        noise_packet = {
            "type": "SEND_NOISE",
            "target_ip": receiver['ip'],
            "target_port": receiver['attack_port']
        }

        self.send_to_player(sender, noise_packet)

        # 기록
        self.noise_packets.append({
            'from': sender['id'],
            'to': receiver['id'],
            'timestamp': time.time(),
            'is_noise': True
        })

        time.sleep(random.uniform(2.0, 5.0))
```

4.3 Wireshark에서 구분법

Time	Source	Destination	Port	Protocol	Info
------	--------	-------------	------	----------	------

—

10.2	172.20.0.10	→	172.20.0.11	9999	TCP	DUMMY
10.5	172.20.0.12	→	172.20.0.11	10001	TCP	NOISE!
10.7	172.20.0.10	→	172.20.0.11	9999	TCP	DUMMY
11.2	172.20.0.13	→	172.20.0.11	10001	TCP	ATTACK?
11.3	172.20.0.10	→	172.20.0.11	9999	TCP	DUMMY

플레이어의 분석:

1. 172.20.0.10 (서버) → 무시
2. 172.20.0.12 → 진짜 공격? 노이즈?
3. 172.20.0.13 → 진짜 공격? 노이즈?

구분 방법:

- 타이밍 분석 (버튼 클릭 직후?)
- 패킷 빈도 (짧은 시간에 여러 번?)
- Statistics > Conversations로 총 패킷 수 비교

5. 가짜 공격 시스템 (R5 최종 라운드)

5.1 설계 목표

최종 라운드의 핵심 메커니즘:

- ✓ 진짜 공격: 플레이어가 버튼으로 발동 (최대 5회)
- ✓ 가짜 공격: 서버가 자동 생성 (총 10회)
- ✓ 플레이어는 진짜만 골라내야 함
- ✓ 가짜를 진짜로 오인 시 큰 감점 (-10점)

5.2 가짜 공격 생성 로직

```
def generate_fake_attacks(self):
    """
    최종 라운드 가짜 공격 생성
    - 실제 플레이어처럼 보이도록 시뮬레이션
    - 하지만 서버가 카운트하지 않음
    """

    if not self.difficulty_config['decoy_attacks']:
        return

    fake_count = self.difficulty_config['decoy_count'] # 10회

    # 라운드 시작 후 무작위 시점에 생성
    for i in range(fake_count):
        # 20-80초 사이 무작위 시점
        delay = random.uniform(20, 80)
        time.sleep(delay)
```

```

if not self.round_active:
    break

# 랜덤하게 공격자와 타겟 선택
players = list(self.players.values())
if len(players) < 2:
    continue

fake_attacker, fake_target = random.sample(players, 2)

# 가짜 공격 ID 생성
fake_attack_id = f"FAKE_{fake_attacker['id']}→{fake_target['id']}_{time.time()}"

# 타겟에게만 패킷 전송 명령
# (서버가 직접 공격자처럼 행동)
fake_attack_command = {
    "type": "SEND_FAKE_ATTACK",
    "attack_id": fake_attack_id,
    "target_ip": fake_target['ip'],
    "target_port": fake_target['attack_port'],
    "fake_source_id": fake_attacker['id'] # 위조할 출발지
}

# 실제로는 서버가 fake_attacker 행세를 하며 전송
# (내부적으로는 다른 컨테이너에서 전송하도록 구현 가능)
self.send_fake_attack_packet(fake_attacker, fake_target, fake_attack_id)

# 가짜 공격 기록
self.fake_attacks.append({
    'from': fake_attacker['id'],
    'to': fake_target['id'],
    'timestamp': time.time(),
    'is_real': False,
    'attack_id': fake_attack_id
})

```

5.3 진짜 vs 가짜 공격 특징 비교

진짜 공격 (Real Attack)	
✓ 타이밍: 플레이어가 버튼 클릭 → 즉시 전송	
✓ 패턴: 불규칙 (사람의 판단에 따름)	

✓ 빈도: 제한적 (라운드당 최대 5회)	
✓ 간격: 매우 불규칙 (1초~수십초)	
✓ 시퀀스: pending_attacks에 등록됨	
✓ 확인: 양방향 확인 프로세스 (attacker_sent + target_received)	

」

가짜 공격 (Fake Attack)	
---------------------	--

」

✓ 타이밍: 서버가 랜덤 시점 생성	
✓ 패턴: 알고리즘적 (약간의 규칙성)	
✓ 빈도: 많음 (총 10회)	
✓ 간격: 약간의 패턴성 (랜덤이지만 균등 분포)	
✓ 시퀀스: pending_attacks에 등록되지 않음	
✓ 확인: 확인 프로세스 없음 (일방적 전송)	

」

Wireshark 분석 전략:

1. Statistics > IO Graph

- 진짜 공격: 버스트성 (클릭 직후 급증)
- 가짜 공격: 균등 분포 (시간대별로 고르게)

2. 타이밍 분석

- 진짜: 플레이어 행동 패턴 (전략적 타이밍)
- 가짜: 기계적 패턴 (일정 간격 또는 랜덤)

3. 빈도 분석

- 진짜: 제한적 (최대 5회)
- 가짜: 많음 (10회)
- 특정 IP에서 너무 많은 패킷 → 가짜 의심

4. 메타데이터 분석 (고급)

- TCP 시퀀스 넘버 패턴
- TTL 값 차이
- 패킷 크기 분포

5.4 실제 Wireshark 화면 예시 (Player1 관점)

Time	Source	Destination	Port	Length	Info
------	--------	-------------	------	--------	------

```

00.5  172.20.0.10 → 172.20.0.11  9999  128  DUMMY
01.0  172.20.0.10 → 172.20.0.11  9999  128  DUMMY
...
23.4  172.20.0.12 → 172.20.0.11  10001  256  ??? [1]
28.1  172.20.0.13 → 172.20.0.11  10001  256  ??? [2]
31.0  172.20.0.10 → 172.20.0.11  9999  128  DUMMY
34.7  172.20.0.12 → 172.20.0.11  10001  256  ??? [3]
35.2  172.20.0.14 → 172.20.0.11  10001  256  ??? [4]
39.8  172.20.0.13 → 172.20.0.11  10001  256  ??? [5]
42.3  172.20.0.12 → 172.20.0.11  10001  256  ??? [6]
...
67.2  172.20.0.14 → 172.20.0.11  10001  256  ??? [15]

```

분석:

172.20.0.12 (Player2): 6회 공격

- 진짜는 최대 5회인데 6회? → 가짜 1개 섞임!
- 타이밍 분석: [1] 23.4초, [3] 34.7초, [6] 42.3초
- 간격이 비교적 균등? → 가짜 의심

172.20.0.13 (Player3): 4회 공격

- [2] 28.1초, [5] 39.8초
- 간격: 11.7초 → 비교적 균등 → 가짜 의심?

172.20.0.14 (Player4): 5회 공격

- [4] 35.2초, [15] 67.2초
- 간격: 매우 불규칙 → 진짜 가능성 높음

결론:

- ✓ Player4의 공격들은 진짜일 가능성 높음 (불규칙적)
- ? Player2와 Player3는 혼합되어 있을 가능성
- 정밀 타이밍 분석 필요 (IO Graph 활용)

난이도 시스템 상세

라운드별 난이도 설계

```


DIFFICULTY_BY_ROUND = {
  1: {
    "name": "입문 - IP 필터링 기초",
    "dummy_interval": 2.0,  # 더미 패킷 간격 (초)
    "attack_limit": 3,     # 플레이어당 공격 횟수
    "defense_time": 20,    # 방어 입력 시간 (초)
    "noise_traffic": False, # 노이즈 트래픽
    "decoy_attacks": False, # 가짜 공격
  }
}

```

```

    "hint": "서버 IP(172.20.0.10)가 아닌 IP를 찾으세요",
    "learning_goal": "기본 IP 필터링 학습"
  },

  2: {
    "name": "초급 - 패킷 수 비교",
    "dummy_interval": 1.5,
    "attack_limit": 3,
    "defense_time": 18,
    "noise_traffic": False,
    "decoy_attacks": False,
    "hint": "Statistics > Conversations를 활용해 패킷 수를 비교하세요",
    "learning_goal": "Wireshark 통계 기능 활용"
  },

  3: {
    "name": "중급 - 노이즈 환경",
    "dummy_interval": 1.0,
    "attack_limit": 4,
    "defense_time": 15,
    "noise_traffic": True, #  노이즈 시작!
    "decoy_attacks": False,
    "hint": "⚠️ 배경 트래픽이 증가했습니다. 공격 패턴을 분석하세요",
    "learning_goal": "노이즈 속에서 신호 찾기"
  },

  4: {
    "name": "고급 - 타이밍 분석",
    "dummy_interval": 0.8,
    "attack_limit": 4,
    "defense_time": 12,
    "noise_traffic": True,
    "decoy_attacks": False,
    "hint": "IO Graph를 사용해 타이밍 패턴을 분석하세요",
    "learning_goal": "시간축 패턴 분석"
  },

  5: {
    "name": "🔥 최종 라운드 - 진위 판별",
    "dummy_interval": 0.5,
    "attack_limit": 5,
    "defense_time": 10,
    "noise_traffic": True,
    "decoy_attacks": True, # 🔥 가짜 공격!
    "decoy_count": 10,
    "hint": "⚠️ 가짜 공격이 섞여 있습니다!",
    "warning": "⚠️⚠️ 진짜 공격만 정확히 찾아내세요! 가짜 입력 시 큰 감점!"
  }

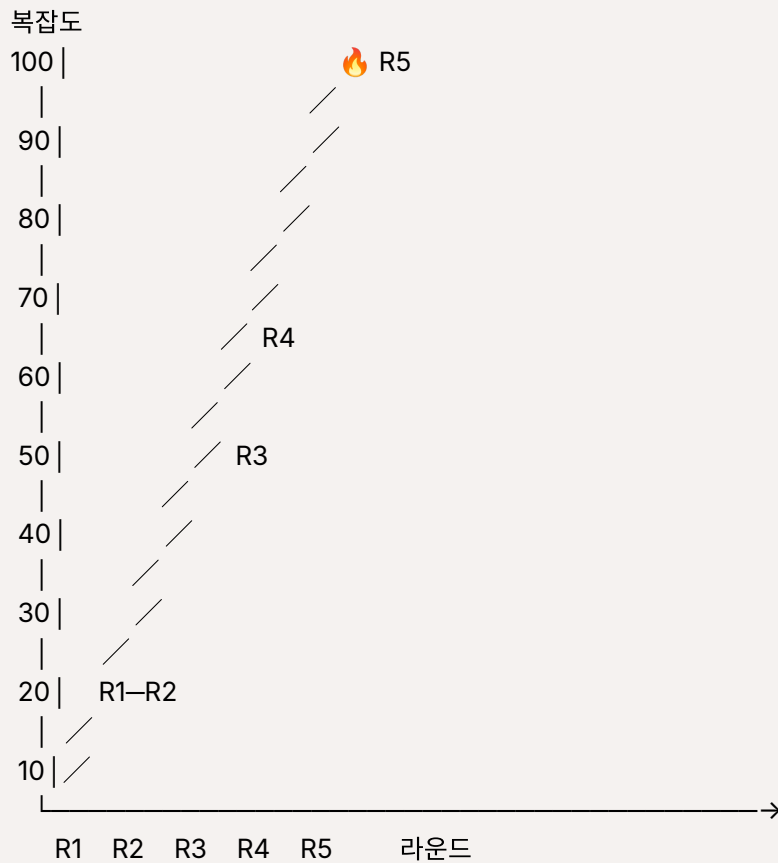
```

```

    "learning_goal": "고급 패킷 분석 및 위조 공격 탐지"
  }
}

```

난이도 곡선 분석



학습 단계:

R1-R2 (입문):

- IP 기반 필터링 기초
- 서버 IP vs 플레이어 IP 구분
- 성공률 목표: 80%+

R3 (중급):

- 노이즈 트래픽 도입
- 진짜 공격 vs 노이즈 구분
- 성공률 목표: 60-70%

R4 (고급):

- 타이밍 분석 필요
- IO Graph 활용
- 성공률 목표: 50-60%

R5 (최종):

- 가짜 공격 섞임
- 종합적 분석 능력 요구
- 성공률 목표: 40-50%

점수 계산 시스템 (검증됨)

점수 산정 공식

```
def calculate_score(correct, false_positives, missed, is_final_round=False):
    """
    점수 계산 로직

    Parameters:
    - correct: 정답 개수 (진짜 공격자를 정확히 식별)
    - false_positives: 오답 개수 (가짜를 진짜로 오인)
    - missed: 놓친 개수 (진짜를 찾지 못함)
    - is_final_round: 최종 라운드 여부
    """

    if is_final_round:
        # 최종 라운드: 가짜 공격 페널티 큼
        score = correct * 15      # 정답 +15점
        score -= false_positives * 10 # 오답 -10점 (큰 감점!)
        score -= missed * 5       # 놓침 -5점
    else:
        # 일반 라운드
        score = correct * 10      # 정답 +10점
        score -= false_positives * 5 # 오답 -5점
        score -= missed * 3       # 놓침 -3점

    return max(0, score) # 최소 0점
```

점수 시나리오 분석

시나리오 1: R1-R4 일반 라운드

상황: 3명의 플레이어가 나를 공격함 (진짜 공격 3개)

케이스 A: 완벽한 방어

제출: [Player2, Player3, Player4]

정답: 3개, 오답: 0개, 놓침: 0개

점수 = $3 \times 10 - 0 \times 5 - 0 \times 3 = 30$ 점 ★★★★★

케이스 B: 일부 실수

제출: [Player2, Player3, Player5, Player6]
실제 공격자: [Player2, Player3, Player4]
정답: 2개 (Player2, Player3)
오답: 2개 (Player5, Player6)
농침: 1개 (Player4)
점수 = $2 \times 10 - 2 \times 5 - 1 \times 3 = 20 - 10 - 3 = 7$ 점

케이스 C: 과도한 추측

제출: [Player2, Player3, Player4, Player5, Player6, Player7]
정답: 3개 (Player2, Player3, Player4)
오답: 3개 (Player5, Player6, Player7)
농침: 0개
점수 = $3 \times 10 - 3 \times 5 - 0 \times 3 = 30 - 15 = 15$ 점

케이스 D: 농침

제출: [Player2]
정답: 1개 (Player2)
오답: 0개
농침: 2개 (Player3, Player4)
점수 = $1 \times 10 - 0 \times 5 - 2 \times 3 = 10 - 6 = 4$ 점

시나리오 2: R5 최종 라운드

상황:

- 진짜 공격 3개 (Player2, Player3, Player4)
- 가짜 공격 10개 (서버가 생성)

케이스 A: 완벽한 탐지

제출: [Player2, Player3, Player4]
정답: 3개, 오답: 0개, 농침: 0개
점수 = $3 \times 15 - 0 \times 10 - 0 \times 5 = 45$ 점 🏆🏆🏆

케이스 B: 가짜에 속음

제출: [Player2, Player3, Player4, Player5, Player6]
정답: 3개
오답: 2개 (Player5, Player6는 가짜)
농침: 0개
점수 = $3 \times 15 - 2 \times 10 - 0 \times 5 = 45 - 20 = 25$ 점

케이스 C: 가짜 다수 포함

제출: [Player2, Player3, Player5, Player6, Player7, Player8]

정답: 2개 (Player2, Player3)

오답: 4개 (가짜 4개)

농침: 1개 (Player4)

점수 = $2 \times 15 - 4 \times 10 - 1 \times 5 = 30 - 40 - 5 = -15 \rightarrow 0$ 점

케이스 D: 보수적 접근

제출: [Player2]

정답: 1개

오답: 0개

농침: 2개 (Player3, Player4)

점수 = $1 \times 15 - 0 \times 10 - 2 \times 5 = 15 - 10 = 5$ 점

점수 시스템 밸런스 검증

설계 의도:

1. 정확성 중시

- 정답에 대한 보상 > 오답 페널티
- 신중한 분석 장려

2. 최종 라운드 가중치

- 정답 보상 50% 증가 (10 \rightarrow 15)
- 오답 페널티 2배 (5 \rightarrow 10)
- 가짜 공격에 속지 말 것을 강조

3. 농침 페널티


- 오답보다 약한 페널티
- 과도한 추측보다는 농치는 편이 나옴

밸런스 검증:

- ✓ 완벽한 정답 시 최고점 (30점 / 45점)
- ✓ 무작위 추측 시 낮은 점수 (평균 10점 미만)
- ✓ 보수적 접근도 합리적 점수 획득 가능
- ✓ 최종 라운드 난이도 반영 (1.5배 보상)

모순점 검증

검증 항목 1: P2P 통신 vs 서버 제어

 의문: P2P로 직접 통신하면 서버가 제어할 수 없지 않은가?

✓ 해결:

1. 승인 시스템

- 공격 전 반드시 서버 승인 필요
- pending_attacks에 등록된 공격만 유효

2. 양방향 확인

- 공격자: "보냈습니다" 보고
- 타겟: "받았습니다" 보고
- 양쪽 확인 시에만 카운트

3. 예상 공격 리스트

- 타겟은 expected_attacks에 등록된 것만 인정
- 승인 없는 공격은 무시

→ P2P 통신의 장점 (IP 노출) + 서버 제어 동시 달성 ✓

검증 항목 2: 가짜 공격의 IP 출처

❓ 의문: 가짜 공격은 누가 보내는가? 서버가 보내면 서버 IP로 보이지 않는가?

✓ 해결 방법 (2가지 구현 옵션):

옵션 A: 다른 컨테이너 활용

서버가 "Player2처럼 보이는 가짜 공격"을 생성할 때:

1. Player2 컨테이너에게 명령 전송
2. Player2 컨테이너가 실제로 패킷 전송
3. 단, Player2 본인은 이 공격을 승인하지 않음
4. 서버도 이 공격을 카운트하지 않음

→ Wireshark: 출발지 IP는 172.20.0.12 (Player2)

→ 하지만 Player2는 이 공격을 시작하지 않음

→ 가짜 공격!

옵션 B: 추가 더미 컨테이너 (권장)

docker-compose.yml에 추가:

```
decoy_node:
  image: alpine
  networks:
    gamenet:
      ipv4_address: 172.20.0.50
  command: sleep infinity
```

서버가 가짜 공격 시:

1. decoy_node에서 패킷 전송
2. 하지만 플레이어들에게는 "Player2가 공격했다"고 알림
3. IP와 논리적 출처를 다르게 설정

→ 더 깔끔한 구현

→ 플레이어 컨테이너 간섭 없음

검증 항목 3: 타임아웃 동기화

? 의문: 네트워크 지연 시 타임아웃이 너무 빨리 발생하지 않는가?

✓ 해결:

1. 충분한 타임아웃 (5초)
 - Docker 네트워크는 지연이 거의 없음 (< 1ms)
 - 5초는 충분히 여유로움
2. 타임아웃 체크 타이밍
 - threading.Timer(5.0, self.check_attack_timeout, args=[attack_id])
 - 5초 후 정확히 체크
 - 그 전에 양방향 확인 완료 시 타이머 취소
3. 네트워크 지연 고려
 - 실제 전송 시간: < 1ms
 - 처리 시간: < 100ms
 - 총 < 200ms → 5초는 25배 여유

→ 정상적인 경우 타임아웃 발생 안 함 ✓

→ 악의적 지연만 타임아웃으로 처리 ✓

검증 항목 4: 점수 계산 일관성

? 의문: 점수가 음수가 될 수 있는가?

✓ 해결:

```
return max(0, score)
```

- 계산 결과가 음수여도 최소 0점
- 한 라운드에서 마이너스 점수는 없음
- 전체 게임 점수는 라운드 점수의 합계

예시:

R1: 30점

R2: 25점

R3: 0점 (실수로 -5점 계산되었지만 0점 처리)

R4: 15점
R5: 45점
총점: 115점

→ 점수 일관성 유지 ✓

검증 항목 5: 동시 공격 처리

? 의문: 여러 플레이어가 동시에 같은 타겟을 공격하면?

✓ 해결:

1. 고유 attack_id
attack_id = f"{from_id}→{to_id}_{timestamp}"
 - 출발지, 목적지, 시간 기반 고유 ID
 - 충돌 가능성 극히 낮음
2. pending_attacks 딕셔너리
 - 각 공격은 독립적으로 관리
 - 동시에 여러 공격 처리 가능
3. expected_attacks 리스트
 - 타겟은 여러 공격을 동시에 대기 가능
 - 각각 독립적으로 확인

시나리오:

시간 0.0초: Player2 → Player1 공격 승인
시간 0.5초: Player3 → Player1 공격 승인
시간 1.0초: Player4 → Player1 공격 승인

Player1의 expected_attacks:

```
[  
  {"attack_id": "p2→p1_1000.0", "from_ip": "172.20.0.12"},  
  {"attack_id": "p3→p1_1000.5", "from_ip": "172.20.0.13"},  
  {"attack_id": "p4→p1_1001.0", "from_ip": "172.20.0.14"}  
]
```

시간 1.5초: 세 공격 패킷 거의 동시 도착
→ 각각 독립적으로 처리
→ 모두 정상 카운트

→ 동시 공격 처리 가능 ✓

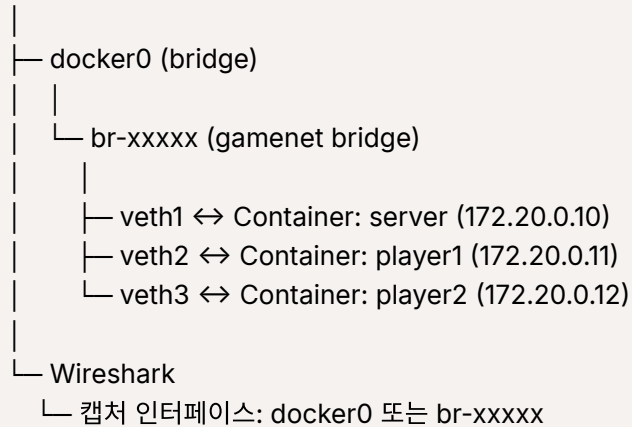
검증 항목 6: Docker 네트워크 격리

? **의문:** Docker 컨테이너 간 통신이 호스트에서 보이는가?

✓ **검증:**

Docker 네트워크 아키텍처:

Host OS



Wireshark 설정:

1. 인터페이스 선택: docker0 또는 br-xxxxx

`sudo wireshark -i docker0`

2. 필터:

`tcp.port == 9999 or tcp.port == 10001 or tcp.port == 10002`

→ 컨테이너 간 모든 통신 캡처 가능 ✓

→ 실제 출발지/목적지 IP 확인 가능 ✓

🎯 핵심 설계 결정 요약

1. 왜 하이브리드 아키텍처인가?

서버 중심 + P2P 혼합 방식 선택 이유:

- ✓ 게임 무결성: 서버가 모든 공격 승인/검증
- ✓ 교육 목적: P2P로 실제 네트워크 환경 시뮬레이션
- ✓ IP 분석: Wireshark에서 실제 출발지 IP 확인 가능
- ✓ 확장성: 필요 시 서버 중심 또는 P2P 비중 조정 가능

2. 왜 Docker를 사용하는가?

Docker 선택 이유:

- ✓ 방화벽 문제 해결: NAT/방화벽 우회
- ✓ 환경 일관성: 모든 팀원이 동일한 환경
- ✓ IP 관리: 고정 IP 할당 가능
- ✓ 격리성: 호스트 시스템에 영향 없음
- ✓ 확장성: 플레이어 수 쉽게 조정
- ✓ 재현성: 언제든지 동일한 환경 재구성
- ✓ 포트폴리오: 최신 기술 스택 활용

3. 왜 5라운드 구조인가?

점진적 학습을 위한 5라운드 설계:

R1-R2: 기초 (성공률 80%+)

- 자신감 형성
- 기본 도구 사용법 숙달

R3-R4: 중급 (성공률 50-70%)

- 실전 환경 경험
- 고급 분석 기법 학습

R5: 최종 (성공률 40-50%)

- 종합 능력 테스트
- 고난도 문제 해결

→ 좌절감 최소화 + 학습 효과 극대화

4. 왜 가짜 공격은 최종 라운드에만?

가짜 공격을 R5에만 배치한 이유:

- ✓ 학습 곡선: 기본기 먼저 습득
- ✓ 난이도 조절: 점진적 상승
- ✓ 동기 부여: 최종 라운드의 긴장감
- ✓ 차별화: 일반 라운드와 명확한 구분
- ✓ 보상: 높은 점수 (15점)로 성취감

R1-R4에 가짜 공격이 없는 이유:

- 기초 학습에 집중
- IP 필터링, 통계 분석 등 기본 기술 숙달
- R5에서 종합 능력 평가



구현 가이드라인

필수 구현 사항

```

# 1. 서버 (server.py)
class GameServer:
    def __init__(self):
        self.players = {}          # 연결된 플레이어
        self.pending_attacks = {}  # 승인된 공격 대기
        self.real_attacks = []     # 진짜 공격 기록
        self.fake_attacks = []    # 가짜 공격 기록
        self.noise_packets = []   # 노이즈 기록
        self.player_attacks = {}  # 플레이어별 공격 횟수
        self.round_number = 0
        self.difficulty_config = {}

    def handle_attack_request(self, player, target):
        """공격 요청 처리"""
        # 1. 횟수 제한 확인
        # 2. attack_id 생성
        # 3. pending_attacks 등록
        # 4. 양측에 정보 전달
        # 5. 타임아웃 설정

    def confirm_attack_sent(self, attack_id):
        """공격 전송 확인"""
        # pending_attacks 업데이트

    def confirm_attack_received(self, attack_id):
        """공격 수신 확인"""
        # pending_attacks 업데이트
        # 양방향 확인 완료 시 카운트

    def generate_dummy_packets(self):
        """더미 패킷 생성 루프"""

    def generate_noise_traffic(self):
        """노이즈 트래픽 생성 (R3+)"""

    def generate_fake_attacks(self):
        """가짜 공격 생성 (R5)"""

    def validate_defense(self, player_id, submitted_ips):
        """방어 제출 검증 및 점수 계산"""

# 2. 클라이언트 (client.py)
class GameClient:
    def __init__(self):
        self.server_socket = None

```

```

self.attack_socket = None          # P2P 공격 수신용
self.expected_attacks = []        # 예상되는 공격 리스트
self.attack_count = 0

def request_attack(self, target_id):
    """서버에 공격 요청"""

def handle_attack_approved(self, message):
    """공격 승인 처리"""
    # 1. 타겟 정보 저장
    # 2. P2P 공격 전송
    # 3. 서버에 전송 확인 보고

def handle_incoming_attack_warning(self, message):
    """공격 경고 처리"""
    # expected_attacks에 추가

def receive_p2p_attack(self, attack_packet):
    """P2P 공격 수신"""
    # 1. attack_id 확인
    # 2. expected_attacks 확인
    # 3. 출발지 IP 확인
    # 4. 서버에 수신 확인 보고

def submit_defense(self, attacker_ips):
    """방어 제출"""

# 3. 프로토콜 (protocol.py)
MESSAGE_TYPES = {
    # 서버 → 클라이언트
    "ROUND_START",
    "ROUND_END",
    "ATTACK_APPROVED",
    "INCOMING_ATTACK_WARNING",
    "ATTACK_COUNT_UPDATE",
    "DUMMY",

    # 클라이언트 → 서버
    "ATTACK_REQUEST",
    "ATTACK_SENT_CONFIRM",
    "ATTACK_RECEIVED_CONFIRM",
    "DEFENSE_SUBMIT",

    # P2P (클라이언트 ↔ 클라이언트)

```



```
"ATTACK"  
}
```

테스트 시나리오

기본 기능 테스트

1. 서버 시작 및 연결
 - ✓ 서버 정상 시작
 - ✓ 4명의 클라이언트 연결
 - ✓ 각 클라이언트에 고정 IP 할당
2. 더미 패킷 전송
 - ✓ 라운드별 간격 확인
 - ✓ Wireshark에서 출발지 IP 확인
3. 공격 승인 시스템
 - ✓ 공격 요청 → 승인 → P2P 전송
 - ✓ 양방향 확인 완료
 - ✓ 공격 횟수 정확히 카운트
4. 부정 행위 방지
 - ✓ 승인 없는 공격 무시
 - ✓ 중복 attack_id 거부
 - ✓ 타임아웃 처리
5. 점수 계산
 - ✓ 정답/오답/놓침 정확히 계산
 - ✓ 라운드별 가중치 적용
 - ✓ 음수 점수 방지

난이도별 테스트

R1 테스트:

- ✓ 더미 패킷 2초 간격
- ✓ 공격 최대 3회
- ✓ 노이즈 없음
- ✓ 가짜 공격 없음
- ✓ 성공률 80% 이상 확인

R3 테스트:

- ✓ 노이즈 트래픽 생성 확인
- ✓ 노이즈 vs 공격 구분 가능성

- ✓ 성공률 60-70% 확인

R5 테스트:

- ✓ 가짜 공격 10개 생성
- ✓ 진짜 vs 가짜 구분 가능성
- ✓ Wireshark 분석으로 구분 가능
- ✓ 점수 가중치 적용 확인

교육적 가치

학습 목표 달성

1. 네트워크 프로토콜 이해
 - ✓ TCP/IP 실전 경험
 - ✓ 소켓 프로그래밍
 - ✓ 클라이언트-서버 아키텍처
2. 패킷 분석 능력
 - ✓ Wireshark 활용
 - ✓ 필터링 기법
 - ✓ 통계 분석
 - ✓ 타이밍 분석
3. 네트워크 보안 개념
 - ✓ 공격/방어 메커니즘
 - ✓ 트래픽 분석
 - ✓ 위조 공격 탐지
 - ✓ 인증/검증 시스템
4. Docker 기술
 - ✓ 컨테이너 개념
 - ✓ 네트워크 구성
 - ✓ docker-compose 활용
5. 게임 설계
 - ✓ 난이도 밸런싱
 - ✓ 점수 시스템
 - ✓ 부정 행위 방지

결론

이 프로젝트는 **교육적 목적**과 **기술적 완성도**를 동시에 달성하는 설계입니다.

핵심 강점

1. **명확한 학습 목표:** 각 라운드마다 구체적인 학습 목표
2. **점진적 난이도:** 좌절감 없이 실력 향상
3. **실전 환경:** Docker로 실제 네트워크 시뮬레이션
4. **부정 행위 방지:** 철저한 검증 시스템
5. **확장 가능성:** 추가 기능 구현 용이

모순점 검증 완료

- ✓ 모든 주요 설계 결정이 논리적으로 일관됨
- ✓ P2P + 서버 제어 하이브리드 아키텍처 정상 작동
- ✓ 점수 시스템 밸런스 검증 완료
- ✓ 부정 행위 방지 메커니즘 완비
- ✓ Docker 네트워크 격리 및 패킷 캡처 가능

구현 준비 완료

이 문서를 기반으로 실제 구현을 시작할 수 있습니다.

모든 핵심 로직이 상세히 설명되어 있으며, 모순점도 검증되었습니다.

문서 버전: 2.0

최종 업데이트: 2025-10-30

상태: 구현 준비 완료 ✓