12. 네트워크 디버깅 및 성능 튜닝

12.1 `tcpdump`, `wireshark` 실습

네트워크 트래픽을 분석하고 디버깅하기 위해 가장 널리 사용되는 도구가 `tcpdump`와 `Wireshark`다. 이 둘은 각각 CLI(터미널 기반)와 GUI(그래픽 기반)의 장점을 가지고 있으며, 리눅스 네트워크 프로그래밍 환경에서는 이 도구들이 거의 필수적으로 활용된다.

✅ `tcpdump`: 커맨드라인 기반 패킷 스니퍼

1. 설치

```
sudo apt install tcpdump
```

2. 기본 명령어 구조

```
sudo tcpdump [옵션] [필터 조건]
```

3. 주요 옵션 설명

옵션	설명
`-i <인터페이스>`	감시할 네트워크 인터페이스 지정 (예: `eth0`, `lo`)
`-n`	IP를 도메인으로 역변환하지 않음 (속도 ↑)
`-v`, `-vv`, `-vvv`	상세 출력 레벨
`-c <개수>`	특정 패킷 수까지만 캡처
`-w <파일>`	패킷을 pcap 파일로 저장
`-r <파일>`	저장된 pcap 파일을 다시 읽음

4. 예제 명령어

- 모든 인터페이스 트래픽 보기:

```
sudo tcpdump -i any
```

- 특정 포트만 캡처:

```
sudo tcpdump -i eth0 port 80
```

- 특정 호스트에서의 패킷만:

```
sudo tcpdump -i eth0 host 192.168.0.10
```

- HTTP 요청만 캡처하고 저장:

```
sudo tcpdump -i eth0 tcp port 80 -w http_traffic.pcap
```

- 저장된 파일 열기:

```
sudo tcpdump -r http_traffic.pcap
```

✅ Wireshark: GUI 기반 네트워크 프로토콜 분석기

1. 설치

```
sudo apt install wireshark
```

설치 시 "비root 사용자도 실행 가능하게 설정"을 허용해야 일반 사용자로 실행 가능하다.

2. 실행

```
wireshark &
```

또는 GUI 메뉴에서 직접 실행.

3. 사용법 요약

- 상단에서 네트워크 인터페이스 선택 → 캡처 시작
- 필터창에 BPF(Berkeley Packet Filter) 문법 입력 가능
  예:
  - `http`
  - `ip.addr == 192.168.0.1`
  - `tcp.port == 22`
- 캡처한 데이터를 `.pcap` 형식으로 저장/불러오기 가능
- 각 패킷의 상세 구조(이더넷 헤더 → IP → TCP 등) 계층별 분석 가능
- Follow TCP Stream, 패킷 재조립, 재전송 감지, RTT 측정도 가능

✅ 실습: Echo 서버 패킷 분석하기

1. `tcpdump`로 Echo 서버가 사용하는 포트를 모니터링:

```
sudo tcpdump -i lo port 12345 -w echo.pcap
```

2. 다른 터미널에서 Echo 클라이언트를 실행하여 메시지 송수신
3. 캡처 종료 후 분석:
   - `tcpdump -r echo.pcap`로 요약 보기

- `wireshark echo.pcap`으로 구조적으로 보기

✅ 실전 활용 팁

- **디버깅 시나리오**: 패킷 손실, 비정상 연결 종료, 잘못된 응답 원인 분석
- **보안 분석**: 암호화되지 않은 민감 정보 확인 (ex. FTP, Telnet)
- **성능 튜닝**: RTT, 세그먼트 크기, 재전송 횟수 등을 분석

12.2 RTT, 패킷 드롭, 리트랜스미션 분석

네트워크 프로그래밍에서 성능 디버깅은 트래픽 흐름의 **지연**, **손실**, **재전송**을 분석하는 데서 시작한다. 이를 통해 병목 현상, 버퍼링 문제, 세션 끊김 등 근본 원인을 찾아낼 수 있다.

✅ 1. RTT(Round-Trip Time) 분석

💡 정의

RTT는 **패킷을 전송하고 응답이 돌아오기까지 걸리는 시간**을 의미하며, 네트워크 지연을 측정하는 핵심 지표다.

🛠️ 측정 방법

🔹 **방법 1: `ping` 사용**

```
ping 8.8.8.8
```

출력 예:

```
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=23.4 ms
```

→ 여기서 `time=23.4 ms`가 RTT

🔹 **방법 2: Wireshark에서 TCP handshake로 측정**

1. Wireshark에서 TCP 세션 필터 적용:

   ```
   tcp.port == 80
   ```

2. SYN 패킷과 그에 대한 SYN+ACK 응답 간 시간 차이를 확인
   - **SYN Timestamp**와 **SYN+ACK Timestamp** 비교
3. 또는 Wireshark 메뉴
   `Statistics → Conversations → TCP`
   → "Min RTT", "Max RTT" 항목 직접 확인 가능

✅ 2. 패킷 드롭(Packet Drop) 분석

💡 정의

패킷 드롭은 네트워크 중간 장비나 커널 버퍼에서 **트래픽이 삭제**되어 목적지에 도달하지 못한 경우다. 원인으로는 버퍼 초과, 라우터 과부하, 전송 오류 등이 있다.

🛠️ 확인 방법

🔹 tcpdump로 실시간 확인

```
sudo tcpdump -i eth0
```

- 특정 요청에 대한 응답이 누락되었는지 **seq 번호** 추적

🔹 Wireshark

- 메뉴: `Analyze → Expert Information`
- 필터:

```
tcp.analysis.lost_segment
```

- 빨간색 경고 항목으로 "Packet loss detected" 등 표시됨
- `TCP ACKed unseen segment` 메시지는 **ACK는 왔는데 해당 세그먼트는 못봤다는 의미**로 드롭 가능성 있음

✅ 3. 리트랜스미션(Retransmission) 분석

💡 정의

TCP는 **패킷 손실**이나 **ACK 누락**이 감지되면 해당 세그먼트를 **재전송**한다. 이 과정은 RTT에 영향을 주고, 혼잡 제어로 속도 저하가 발생한다.

🛠️ 확인 방법

🔹 Wireshark 필터링

```
tcp.analysis.retransmission
```

🔹 유형 구분

유형	설명
`Retransmission`	재전송 (일반)
`Fast Retransmission`	ACK 누락이 감지되어 빠르게 재전송
`Spurious Retransmission`	사실 필요 없었는데 잘못 판단해 재전송
`Out-of-order`	순서가 맞지 않는 세그먼트 도착

🔹 재전송된 패킷 보기

- Info 열에 `TCP Retransmission` 표시
- Sequence Number 중복 확인
- RTT와 함께 비교하면 **왜 재전송되었는지** 유추 가능

✅ 실습 예제: RTT 및 재전송 확인

```
curl http://example.com
```

동시에:

```
sudo tcpdump -i lo port 80 -w curl.pcap
```

이후 Wireshark로 `curl.pcap`을 열고:

- TCP 스트림 필터 설정: `tcp.stream eq 0`
- Packet Details → TCP Layer → Seq/Ack 확인
- 분석 필터:
  - `tcp.analysis.retransmission`
  - `tcp.analysis.ack_rtt`

✅ 실전 팁

- **RTT가 급증**하면 → 네트워크 혼잡, DNS 문제, 라우팅 경로 불안정
- **패킷 드롭이 많고 재전송 증가**하면 → 버퍼 오버플로, MTU 불일치, NIC 설정 확인
- **Wireshark의 "Expert Info"**는 자동으로 이상 패턴 요약 제공하므로 성능 디버깅 시 매우 유용

12.3 `strace`, `lsof`를 통한 시스템 콜 추적

네트워크 프로그램 디버깅에서 트래픽만 분석하는 것으로는 부족할 때가 많다.
이럴 땐 **어떤 시스템 콜이 호출됐는지**, **어떤 파일이나 포트에 접근했는지**까지 추적해야 한다.
`strace`는 **시스템 콜 추적 도구**, `lsof`는 **열린 파일과 포트 추적 도구**로, 네트워크 장애나 프로그램 비정상 동작 시 원인 분석에 핵심적이다.

✅ 1. `strace`: 시스템 콜 추적 도구

🔹 개요

- `strace`는 프로세스가 호출하는 **모든 시스템 콜과 반환값**을 실시간으로 추적해 보여준다.
- 주로 `socket()`, `connect()`, `read()`, `write()` 등이 어떤 인자로 호출되었고, 어떤 오류가 발생했는지 확인할 수 있다.

🔹 설치

```
sudo apt install strace
```

🔹 기본 사용법

```
sudo strace ./my_program
```

- 또는 이미 실행 중인 프로세스에 attach:

```
sudo strace -p <PID>
```

🔹 네트워크 관련 시스템 콜 필터링

```
sudo strace -e trace=network ./my_program
```

출력 예:

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(80), sin_addr=...}, 16) = 0
send(3, "GET / HTTP/1.1\r\n...", 128, 0) = 128
recv(3, ..., 1024, 0) = 512
```

🔹 유용한 옵션

옵션	설명
`-e trace=network`	네트워크 관련 시스템 콜만 추적
`-T`	각 시스템 콜 수행 시간 표시
`-tt`	타임스탬프 포함
`-o <file>`	출력 파일로 저장

✅ 2. `lsof`: 열린 파일과 포트 보기

🔹 개요

- `lsof (List Open Files)`는 현재 **열려 있는 파일, 소켓, 포트, 파이프** 등을 보여주는 도구다.
- 파일뿐 아니라 TCP/UDP 포트도 `파일`로 간주되므로 네트워크 상태 추적에도 유용하다.

🔹 설치

```
sudo apt install lsof
```

🔹 기본 사용법

```
sudo lsof
```

🔹 네트워크 소켓 보기

```
sudo lsof -i
```

예시 출력:

```
COMMAND   PID  USER   FD   TYPE  DEVICE SIZE/OFF NODE NAME
server    1324 user   3u  IPv4 1234567      0t0  TCP *:8080 (LISTEN)
curl      2456 user   4u  IPv4 9876543      0t0  TCP localhost:53212->example.com:http
(ESTABLISHED)
```

🔹 포트별 프로세스 확인

```
sudo lsof -i :8080
```

🔹 특정 PID가 열고 있는 소켓 보기

```
sudo lsof -p <PID>
```

🔹 네트워크만 보기

```
sudo lsof -i -nP
```

- `-nP`: 호스트/포트 역변환 안 함 (속도 향상)

✅ 실전 예제: Echo 서버 디버깅

1. `strace`로 시스템 콜 추적

```
sudo strace -e trace=network -T ./echo_server
```

→ `socket()`, `bind()`, `accept()` 등이 호출되는지, 어떤 인자였는지 확인

2. `lsof`로 포트 점유 확인

```
sudo lsof -i :12345
```

→ 프로그램이 정상적으로 포트를 열고 대기 상태인지 확인 가능
→ `SO_REUSEADDR` 미설정으로 포트 재사용 불가 시 확인

✅ 활용 포인트 요약

도구	주요 용도
`strace`	프로그램이 내부적으로 호출하는 시스템 콜 확인
`lsof`	열린 포트 및 네트워크 소켓 상태 확인

- 네트워크 오류가 발생할 경우, `tcpdump + strace + lsof` 조합으로 **트래픽 + 시스템 콜 + 포트 점유 현황**을 모두 분석하는 것이 이상적

12.4 커널 파라미터 (`/proc/sys/net/`)

리눅스 커널은 네트워크 동작 방식의 많은 부분을 **동적으로 조절 가능한 파라미터**를 통해 제어할 수 있다. 이 파라미터들은 `/proc/sys/net/` 디렉터리에 파일 형태로 존재하며, `sysctl` 명령어로도 접근 가능하다. 네트워크 성능 튜닝, 보안 설정, TCP 동작 방식 조정 등에 매우 유용하다.

✅ 1. 기본 개념

- `/proc/sys/net/` 디렉터리는 커널 네트워크 서브시스템의 실시간 설정 정보를 노출
- 모든 항목은 텍스트 기반이며, **읽기/쓰기** 가능
- 설정 변경 즉시 적용됨 (재부팅 시 초기화되므로 `/etc/sysctl.conf` 또는 `/etc/sysctl.d/*.conf`로 영구 설정)

✅ 2. 주요 하위 디렉터리

디렉터리	설명
`/proc/sys/net/ipv4/`	IPv4 관련 파라미터
`/proc/sys/net/ipv6/`	IPv6 관련 파라미터
`/proc/sys/net/core/`	네트워크 커널 코어 설정 (버퍼, 큐 등)
`/proc/sys/net/unix/`	유닉스 도메인 소켓 관련 설정
`/proc/sys/net/netfilter/`	netfilter/iptables 관련 커널 설정

✅ 3. 주요 파라미터와 의미

🔹 `net.ipv4.tcp_syncookies`

- SYN Flood 방어용

```
cat /proc/sys/net/ipv4/tcp_syncookies
# 1이면 활성화 (권장)
```

🔹 `net.ipv4.ip_forward`

- 라우터 역할 허용 여부 (1이면 패킷 포워딩 허용)

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

🔹 `net.core.somaxconn`

- `listen()` 큐의 최대 대기 연결 수

```
cat /proc/sys/net/core/somaxconn
# default: 128
```

🔹 `net.ipv4.tcp_max_syn_backlog`

- SYN 수신 후 ACK 전까지 대기열 크기

```
cat /proc/sys/net/ipv4/tcp_max_syn_backlog
# SYN Flood 대응 시 증가시킴 (예: 4096)
```

🔹 `net.ipv4.tcp_fin_timeout`

- 소켓이 `FIN_WAIT2` 상태에 머무는 시간

```
cat /proc/sys/net/ipv4/tcp_fin_timeout
# 기본: 60 (초), 짧게 줄이면 리소스 빨리 회수 가능
```

🔹 `net.core.rmem_default`, `rmem_max`

- 수신 버퍼의 기본값 및 최대값

```
cat /proc/sys/net/core/rmem_default
cat /proc/sys/net/core/rmem_max
```

🔹 `net.core.wmem_default`, `wmem_max`

- 송신 버퍼의 기본값 및 최대값

✅ 4. 실습: 성능 개선을 위한 조정

```
sudo sysctl -w net.core.somaxconn=1024
sudo sysctl -w net.ipv4.tcp_max_syn_backlog=2048
sudo sysctl -w net.ipv4.tcp_fin_timeout=30
```

→ 즉시 적용됨.
→ 영구 적용하려면 `/etc/sysctl.conf` 또는 `/etc/sysctl.d/99-custom.conf`에 다음 추가:

```
net.core.somaxconn=1024
net.ipv4.tcp_max_syn_backlog=2048
net.ipv4.tcp_fin_timeout=30
```

적용:

```
sudo sysctl -p
```

✅ 5. 파라미터 목록 전체 보기

```
sysctl -a | grep tcp
sysctl -a | grep net
```

✅ 6. 주의사항

- 설정 변경 시 실시간 트래픽에 즉각적인 영향을 미침
- 잘못 조정할 경우 오히려 연결 실패, 대기열 초과, 속도 저하 등 부작용 발생
- 특히 `rmem_max`, `wmem_max`, `backlog` 관련 설정은 트래픽 패턴에 맞게 신중히 조정

12.5 `iperf`, `netperf`를 이용한 대역폭 측정

네트워크 성능을 정량적으로 평가하려면 단순한 `ping`이나 `traceroute`만으로는 부족하다.
`iperf`와 `netperf`는 대역폭(Bandwidth), 지연(Latency), 손실률(Packet Loss) 등 다양한 지표를 통해 TCP/UDP 성능을 정확히 측정할 수 있는 **전문 테스트 도구**다.

✅ 1. `iperf` - 네트워크 대역폭 측정 도구

🔹 특징

- TCP, UDP 지원
- 클라이언트/서버 모델 기반
- 단방향 또는 양방향 테스트
- 윈도우 크기, 포트, 시간 등 상세 설정 가능

🔹 설치

```
sudo apt install iperf3
```

🔹 기본 구조

```
[클라이언트] ↔ [서버]

서버:
iperf3 -s

클라이언트:
iperf3 -c <서버 IP>
```

🔹 예제

TCP 대역폭 측정 (기본)

```
iperf3 -s           # 서버에서 실행
iperf3 -c 192.168.0.10   # 클라이언트에서 실행
```

출력 예:

```
[ ID] Interval       Transfer     Bandwidth
[  5] 0.00-10.00 sec 1.10 GBytes  942 Mbits/sec
```

UDP 테스트 (패킷 손실/지연 분석)

```
iperf3 -s -u
iperf3 -c 192.168.0.10 -u -b 10M
```

양방향 테스트

```
iperf3 -c 192.168.0.10 -d
```

윈도우 크기 조정

```
iperf3 -c 192.168.0.10 -w 256K
```

시간 및 포트 변경

```
iperf3 -c 192.168.0.10 -t 30 -p 5202
```

✅ 2. `netperf` – 고급 네트워크 성능 측정 도구

🔹 특징

- TCP_STREAM, TCP_RR, UDP_STREAM, UDP_RR 등 다양한 테스트 모드
- 지연, 처리량, 요청-응답 시간 측정에 강력
- 실시간 벤치마킹 자동화에 적합

설치

```
sudo apt install netperf
```

기본 구조

```
[클라이언트] ↔ [서버]

서버:
netserver

클라이언트:
netperf -H <서버 IP>
```

주요 테스트 모드

테스트 모드	목적
TCP_STREAM	TCP 처리량 측정
UDP_STREAM	UDP 처리량 측정
TCP_RR	요청-응답 지연 측정
UDP_RR	UDP 요청-응답 지연 측정

예제

TCP 스트리밍

```
netperf -H 192.168.0.10 -t TCP_STREAM
```

UDP 스트리밍

```
netperf -H 192.168.0.10 -t UDP_STREAM
```

TCP 요청-응답

```
netperf -H 192.168.0.10 -t TCP_RR
```

세션 시간 조정

```
netperf -H 192.168.0.10 -t TCP_STREAM -- -l 30
```

✅ 3. `iperf` vs `netperf` 비교 요약

항목	`iperf3`	`netperf`
설치 용이성	매우 간단	비교적 복잡
사용 방식	직관적	상세 제어 가능
스트리밍 테스트	TCP/UDP 모두 지원	TCP/UDP 모두 지원
요청-응답(RR)	제한적	강력 지원
커스터마이징	일부 제한	매우 다양

✅ 실전 활용 팁

- **서버는 항상 외부 연결 가능한 공인 IP 또는 같은 서브넷 내 IP 필요**
- **UDP 전송 시 대역폭을 `-b` 옵션으로 지정하지 않으면 1Mbps로 설정됨**
- 테스트 전후에 `top`, `nload`, `ifstat`, `htop` 같은 모니터링 도구 병행하면 병목 구간 파악 가능