<주요 데이터>

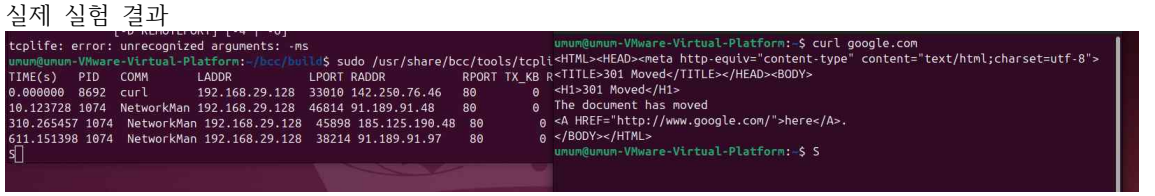
1. RTT

- 원본 필드 명 : MS
  제안하는 필드 명 : tcp.rtt.session_ms (AI 학습 시)

- 필요한 이유 : RTT는 네트워크 Latency를 나타내는 가장 핵심적인 주요 데이터이다. RTT 값이 급격히 증가하는 것은 네트워크 성능 저하의 가장 직접적인 징조이므로 AI 모델의 예측에 필수적이다.

- 수집 방법
  도구 : BCC의 tcplife
  명령어 : sudo /usr/share/bcc/tools/tcplife -t
  선정 이유 : tcplife는 eBPF를 이용해 커널 레벨에서 TCP 연결의 실제 RTT(MS 필드)를 정확하고 효율적으로 수집할 수 있음

- 실제 실험 결과



  실험 내용 1) 위 명령어로 RTT 수집을 시작함
  2) 두 번째 터미널에서 curl google.com을 실행하여 TCP 트래픽 발생
  실험 결과 : tcplife가 curl 트래픽을 성공적으로 감지하고 MS 필드에 RTT값(234.12) 출력

2. 패킷 손실률

수집 방법 (eBPF tracepoint):
dev:dev_queue_xmit (커널이 실제 네트워크 카드로 패킷 전송 시도), tcp:tcp_retransmit_skb (TCP 재전송 발생)
즉, 손실률 = (tcp:tcp_retransmit_skb) / (dev:dev_queue_xmit)
손실률에 따라 장애상황인지 판단 가능, (예 : 10% 이상이면 네트워크 이상 징후)
또한 손실률을 계산할때에 몇초 단위로 할껀지 정해야함 (예 : 5초 동안의 손실률)

자료형 = 0.0 ~ 1.0 사이의 float

필요한 이유:
드롭·재전송 이벤트를 카운트하고, 동일 기간의 송신 시도(또는 송신 바이트)로 나누면 패킷 손실률(또는 재전송률)을 얻음.
재전송은 손실의 직접적 신호이며, 커널의 skb:kfree_skb는 드롭 시점의 상세 원인(메모리 부족, dev queue 등)을 알려줌.

근거 및 예시 : https://phb-crystal-ball.org/monitor-packet-drops-with-ebpf/

3. 연결 실패율

클라이언트.
* 필드명: TCPAttemptFails (연결 시도 실패 횟수), TCPActiveOpens (능동적 연결 성공 횟수)
* 계산식:

$$\frac{\text{TCPAttemptFails}}{\text{TCPAttemptFails}+\text{TCPActiveOpens}} \times 100$$

* 데이터 타입: integer, Float (%)
* 수집 방법: cat /proc/net/snmp

서버.
* 필드명: TCPExtListenDrop (Listen Queue에서 Drop된 횟수), TCPPassiveOpens (수동적 연결 성공 횟수)
* 계산식:

$$\frac{\text{TCPExtListenDrop}}{\text{TCPExtListenDrop}+\text{TCPPassiveOpens}} \times 100$$

* 데이터 타입: integer, Float (%)
* 수집 방법: cat /proc/net/snmp, cat /proc/net/netstat

4. 큐 지연시간(TCP 큐 길이)

- 필드명: Recv-Q (수신 대기 큐), Send-Q (송신 대기 큐)
- 계산식: 큐 크기를 직접 모니터링
- 데이터 타입: Integer
- 수집 방법: ss -lntp (Listen 상태 소켓의 연결 대기열 확인) 및 ss -ntp (Established 상태 소켓의 데이터 큐 확인)

5. 재전송 횟수

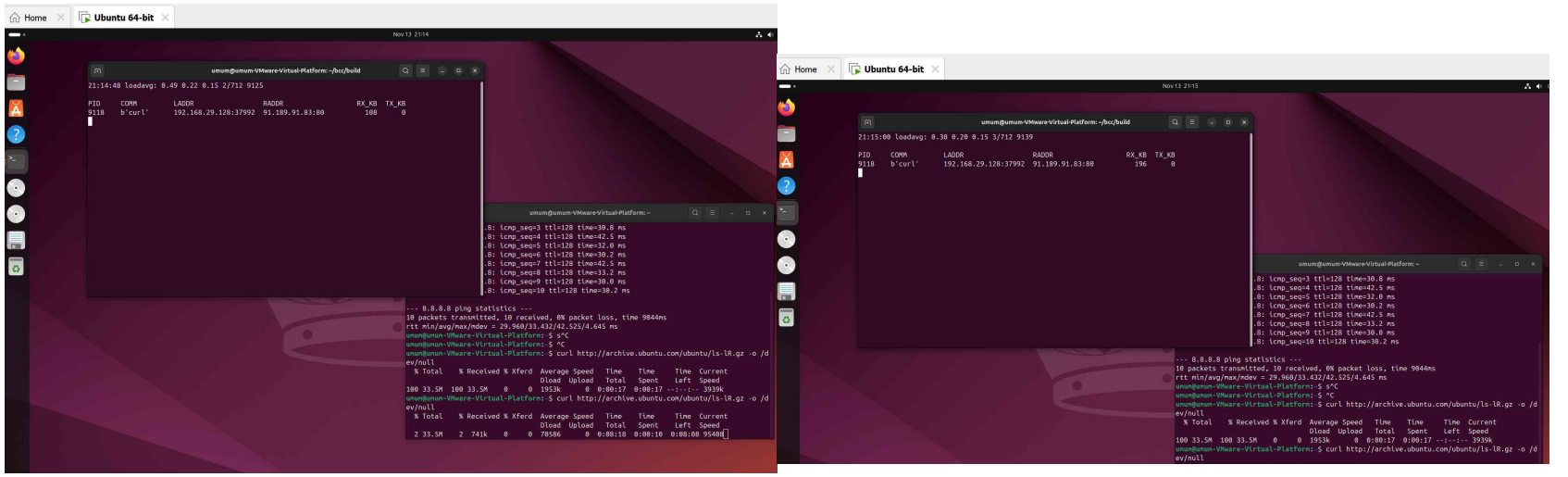
- 필드명: TCPSegRetrans (SNMP), tcp:tcp_retransmit_skb (eBPF)
- 계산식:
  1) 재전송 횟수 (절대량): (T2 시점의 TCPSegRetrans) - (T1 시점의 TCPSegRetrans)
  2) 재전송률 (비율): (단위시간당 tcp:tcp_retransmit_skb 발생 횟수) / (단위시간당 dev:dev_queue_xmit 발생 횟수) * 100
- 데이터 타입: Integer (횟수), Float (비율, %)
- 수집 방법:
cat /proc/net/snmp -> Tcp: 섹션의 Retransmits 값 주기적 수집
eBPF Tracepoint: tcp:tcp_retransmit_skb (재전송 이벤트 발생 시점 추적)
- 필요한 이유:
장애의 직접적 신호: 재전송은 패킷 손실이나 심각한 네트워크 혼잡이 발생했음을 알려주는 가장 명확한 지표

<연관 데이터>

1.1 송신 패킷 용량
- 원본 필드 명 : TX_KB, RX_KB

- 필요한 이유 : 송신/수신 용량(처리량)은 RTT와 밀접한 연관 데이터이다. 특정 프로세스가 비정상적으로 많은 용량(KB)의 트래픽을 주고받으면 네트워크 대역폭을 점유하여 병목 현상을 유발하고, 이는 직접적으로 RTT 증가를 초래한다. AI에게 높은 RTT와 높은 네트워크 용량 사용의 연관성을 학습시키기 위해 필요하다.
- 수집 방법
  도구 : BCC의 tcptop
  명령어 : sudo /usr/sbin/tcptop-bpfcc
  선정 이유 : tcptop은 eBPF를 이용해 커널 레벨에서 TCP 통신을 하는 프로세스별로 송신/ 수신 용량을 실시간으로 추적할 수 있다.

- 실제 실험 결과



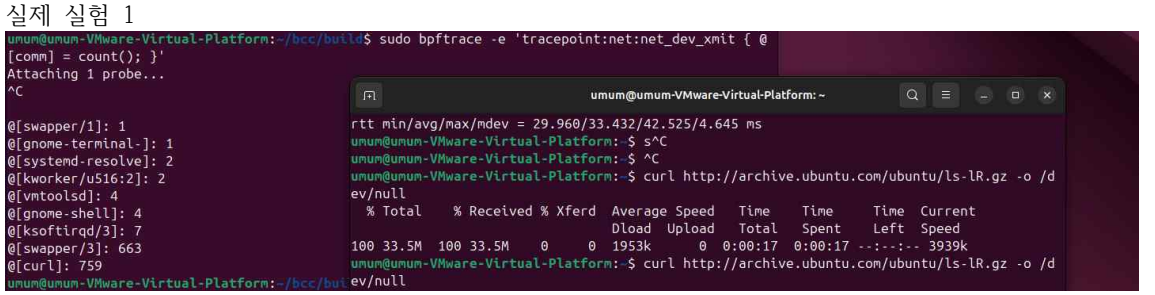
실험 내용 1) 위 명령어로 TCP 트래픽 감시 시작
2) curl을 이용해 파일 다운로드 실행
실험 결과 : tcptop이 curl 트래픽을 성공적으로 감지하고 RX_KB 값이 실시간으로 증가함

1.2 송신 패킷 수
- 원본 필드 명 : bpftrace의 @ 변수에 집계된 count

- 필요한 이유 : 송신/수신 패킷 수는 RTT와 밀접한 연관 데이터이다. 특정 프로세스의 패킷 수가 비정상적으로 급증하면 네트워크 병목 현상을 유발하고, 이는 직접적으로 RTT 증가를 초래한다. AI에게 높은 RTT와 높은 패킷 트래픽의 연관성을 학습 시키기 위해 필요하다.

- 수집 방법
  도구 : bpftrace
  선정 이유 : bpftrace는 커널 이벤트가 발생한 횟수를 집계하는 데 효율적인 도구이기 때문

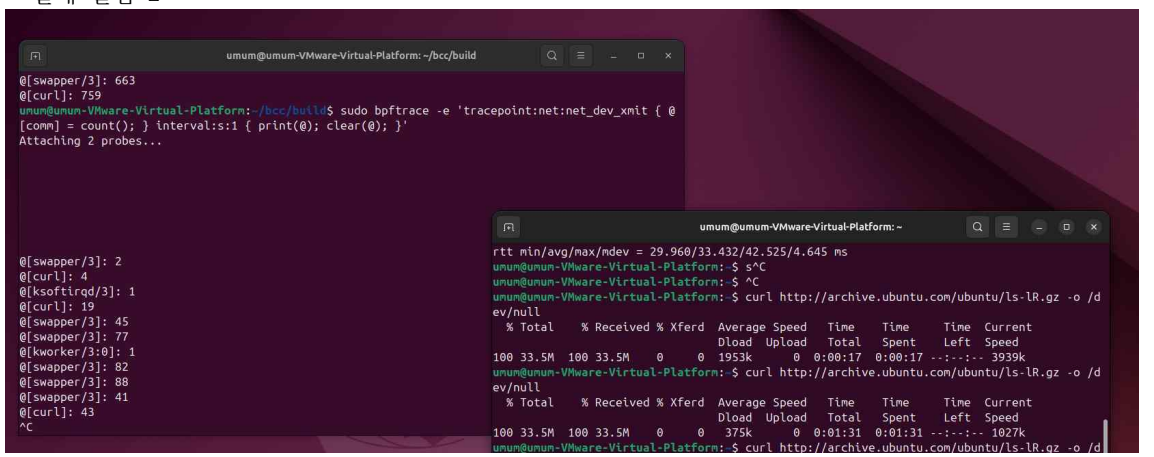
- 실제 실험 1



실험 방법 & 결과

1) 명령어 : sudo bpftrace -e 'tracepoint:net:net_dev_xmit { @[comm] = count(); }'
2) 두 번째 터미널에서 curl 트래픽 발생
3) Ctrl+C 누르면 curl이 실행되는 동안 발생시킨 총 송신 패킷수가 집계됨

- 실제 실험 2



실험 방법 & 결과
1) 명령어 : sudo bpftrace -e 'tracepoint:net:net_dev_xmit { @[comm] = count(); } interval:s:1 { print(@); clear(@); }' -> 실시간 수집 가능
2) 두 번째 터미널에서 curl 트래픽 발생
3) curl 트래픽이 발생하는 동안, 1초마다 curl이 보낸 초당 패킷 수(PPS)가 @[curl]: 처럼 실시간으로 갱신되며 출력됨

2. 소켓 버퍼 길이
- 수집방법:
eBPF 로도 수집 가능하지만 /proc 으로도 수집 가능 - 커널이 소켓 버퍼 상태를 노출하고 있기 때문
/proc/net/tcp 명령어로 모든 TCP 소켓의 상태 및 큐 길이를 얻을 수 있고, 이를 단위시간마다 파싱해서 평균을 내면 됨

- 자료형 : 0 보다 큰 float (머신러닝 전 feature scaling 필요)

- 필요한 이유:
소켓 버퍼는 TCP 소켓마다 존재하는 전송 대기열.
송신 버퍼: 아직 네트워크로 보내지 못한 데이터
수신 버퍼: 애플리케이션이 아직 읽지 않은 데이터
소켓 송신 버퍼의 지속 증가 또는 평균값 상승은 전송 지연, 상대 호스트 반응성 저하, 또는 NIC/qdisc 병목의 전조증상임.
소켓 버퍼가 가득 차면 send() 호출이 블로킹되거나 커널이 패킷을 드롭하게 되고, 결국 서비스 응답 저하 또는 연결 타임아웃으로 이어짐.
소켓 버퍼 지표는 네트워크 스택 내부의 backpressure를 직접 보여주는 중요한 지표라고 볼수 있음.

3. NIC 대역폭 사용률
- 필드명: rx_bytes (수신 바이트 수), tx_bytes (송신 바이트 수), rx_dropped (수신 드롭 패킷 수), tx_dropped (송신 드롭 패킷 수)
- 계산식:

$$\left(\frac{(\Delta rx\_bytes + \Delta tx\_bytes)/\Delta time}{\text{NIC Max Speed}} \times 100 \right)$$

- 데이터 타입: integer, Float (%)
- 수집 방법: cat /proc/net/dev, ip -s link (주기적으로 수집하여 특정 시점 간의 변화량 구해야 함.)

4. TCP 큐 길이
- 필드명: tx_queue_len (송신 큐 바이트), rx_queue_len (수신 큐 바이트)
- 계산식: 개별 소켓의 현재 값 또는 단위시간당 평균값을 사용
- 데이터 타입: Integer (Bytes)
- 수집 방법:
  1) cat /proc/net/tcp (또는 ss 명령어)
  2) 출력된 16진수 tx_queue 및 rx_queue 컬럼 값을 파싱하여 10진수 바이트 값으로 변환
- 필요한 이유:
큐 길이가 길어지는 것 (= 네트워크의 처리 속도보다 데이터가 더 빨리 도착하는 경우) 은 패킷 손실로 이어짐 -> 데이터들이 강제로 버려짐
큐 길이 증가 = 처리 속도가 도착 속도를 따라가지 못한다 = 네트워크 스택 어딘가에 병목이 있다.

5. TCP 혼잡 상태
- 필드명:
  .infra.tcpwin..snd_cwnd.* (송신 혼잡 윈도우 크기)

.infra.tcpwin..snd_ssthresh.* (느린 시작 임계값)
- 계산식: eBPF가 측정한 원본 값을 직접 모니터링
- 데이터 타입: Integer (Bytes)
- 수집 방법:
  ss -ti 후 cwnd값(혼잡 윈도우 크기), ssthresh(느린 시작 임계값) 가져오기 또는
  eBPF (커널의 TCP 혼잡 제어 함수에 tcpwin과 같은 트레이서를 연결하여 수집)

<간접 데이터>

1.1 메모리 사용률 - 증상
- 원본 필드 명 : Mem, total, used

- 필요한 이유 : 메모리 사용률은 네트워크 성능에 간접적인 영향을 준다. 메모리 부족 증상은 시스템 전반에 지연을 일으키고 이는 네트워크 RTT를 간접적으로 급증할 수 있다.

- 수집 방법
  명령어 : free -m (전통적인 방법)
  선정 이유 : 시스템의 현재 메모리 상태를 가장 쉽고 정확하게 파악할 수 있다.

- 실험 결과

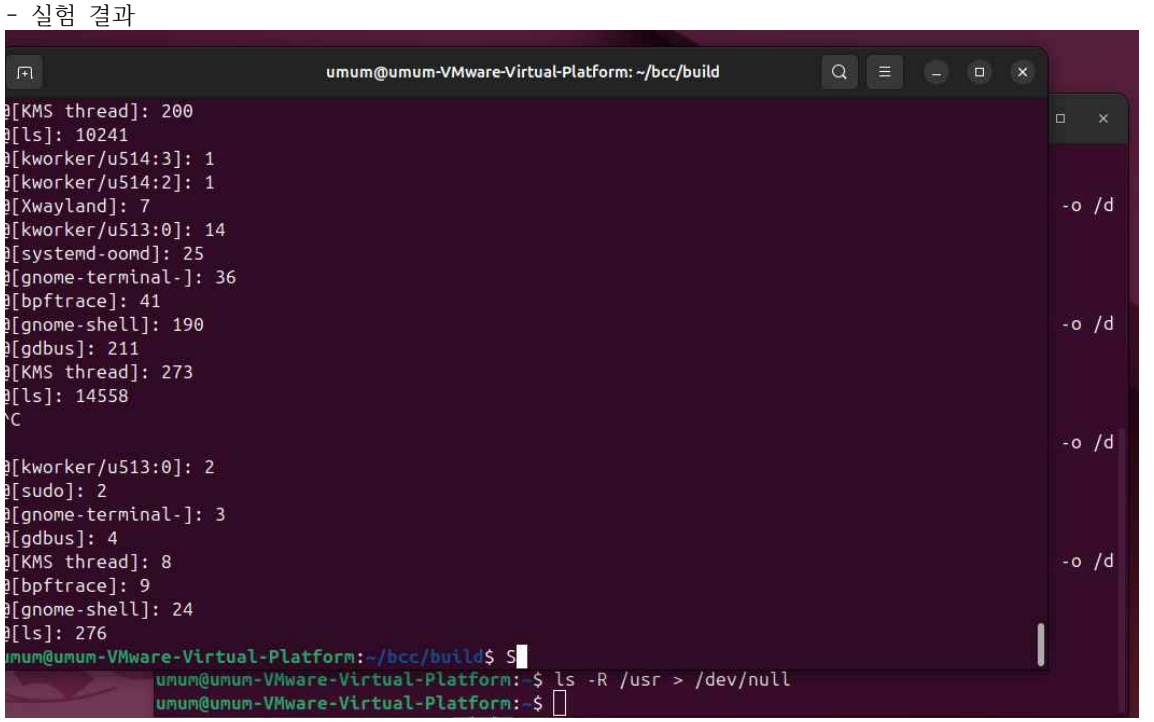
```
umum@umum-VMware-Virtual-Platform:~/bcc/build$ free -m
               total        used        free      shared  buff/cache   available
Mem:            7893        1482        2021          44        4733        6411
Swap:              0           0           0
```

활용 방법 1) (used / total) * 100으로 사용률(%)을 계산할 수 있다. (위 실험 : 약 18.8%)
2) 주기적으로 위 명령어를 실행하여 사용률을 수집한다.

1.2 메모리 사용률 - 원인
- 원본 필드 명 : bpftrace의 @ 변수에 집계된 count

- 수집 방법
  도구 : bpftrace
  선정 이유 : 메모리 부하의 원인/징조가 되는 초당 커널 메모리 요청 횟수를 수집하는 데 좋다. 커널이 공식적으로 열어둔 톨게이트(tracepoint:kmem:kmalloc)를 직접 추적하여, 프로세스별로 요청 횟수를 정확하게 집계할 수 있다.

- 실험 결과



실험 방법
1) 명령어 : sudo bpftrace -e 'tracepoint:kmem:kmalloc { @[comm] = count(); } interval:s:1 { print(@); clear(@); }' -> 메모리 요청 톨게이트 감시 시작 (실시간)
2) 명령어 : ls -R /usr > /dev/null -> 메모리 부하를 인위적으로 발생시킴

실험 결과
: 다른 기본 프로세스들(gnome-shell 등)과 달리, @[ls] 필드의 '초당 요청 횟수'가 폭발함

2. 프로세스 수 - 정확히는 대기열 (run queue) 에 쌓인 프로세스 수

즉, 지금 실행 대기 중인 프로세스가 얼마나 많은가를 의미, 이 수치가 높으면 CPU 가 처리해야 할 작업이 밀리고 있다는 뜻

- 수집 방법:
CPU run queue 길이 -> /proc/schedstat 또는 task_struct.nr_running (굳이 eBPF 사용할 필요 없고 직접 읽기 가능)
단위시간동안 읽은 후, (예 : 5초) 5로 나누면 평균 대기열 크기가 나옴
자료형 : 0 보다 큰 float (머신러닝 전 feature scaling 필요)

- 필요한 이유: nr_running 또는 run queue의 증가 → CPU 스케줄링 대기 → 사용자/네트워크 요청 처리 지연 가능.

- 핵심:
네트워크 패킷 처리 지연은 종종 CPU 스케줄링 병목(즉, run queue가 길어짐)에서 기인.
즉 네트워크 지표가 나빠지기 전에 시스템 레벨에서 병목이 쌓이는 경우가 많음.
특히 패킷 처리량이 큰 시점에 CPU가 포화되면 네트워크 스택에서 처리 지연/큐 쌓임/드롭이 발생할수 있으므로 이를 미리 포착하면 사전 대응이 가능.
근거: https://eunomia.dev/en/tutorials/9-runqlat/

3. I/O Latency

- 필드명: await(평균 I/O 대기 시간), avgqu-sz(평균 요청 큐 길이), IO_latency_us (커널 I/O 지연 시간)

비교적 간단한 iostat으로 먼저 해보고 eBPF 데이터로 교체해도 될 듯

- 계산식:N/A
- 데이터 타입: Float (ms), Integer.
- 수집 방법: iostat -x 1 -> (await, avgqu-sz) /// eBPF Tracing -> IO_latency_u

4. CPU Throttling
- 필드명: nr_throttled (Throttling 발생 횟수), throttled_time (Throttling된 총 시간)
- 계산식: 단위시간당 nr_throttled 증가량
- 데이터 타입: Integer (횟수), Integer (nanoseconds), Float (비율, %)
- 수집 방법:
  1) (컨테이너/Cgroup 환경): cat /sys/fs/cgroup/cpu/cpu.stat (또는 v2의 cpu.stat) 파일의
     nr_throttled, throttled_time 필드 값 수집
  2) (시스템 전체): iostat 등은 이 정보를 직접 보여주지 않으므로 Cgroup 통계를 확인하는 것이 핵심

- 필요한 이유: CPU Throttling은 해당 프로세스를 강제로 멈춘다 -> 네트워크 장애를 직접 유발시킴
프로세스 멈춤 -> 패킷 처리 못함 -> 패킷이 큐에 쌓이게 되고, tcp 큐 길이가 증가 -> 패킷 손실이 발생 -> 주요데이터를 악화시키는 근본적인 원인이 된다.
네트워크 지표가 나빠지기 전에 미리 예측할 수 있는 핵심 신호가 될 수 있다.

5. SoftIRQ 횟수
- 필드명: NET_RX (네트워크 수신 처리), NET_TX (네트워크 송신 처리)
- 계산식: NET_RX_per_sec = (NET_RX_t2 - NET_RX_t1) / (t2 - t1)
- 데이터 타입: Integer (누적 횟수), Float (초당 횟수)
- 수집 방법: cat /proc/softirqs