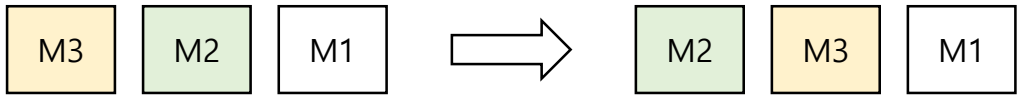
지진경보시스템의 메시지 순서 역전 현상 분석

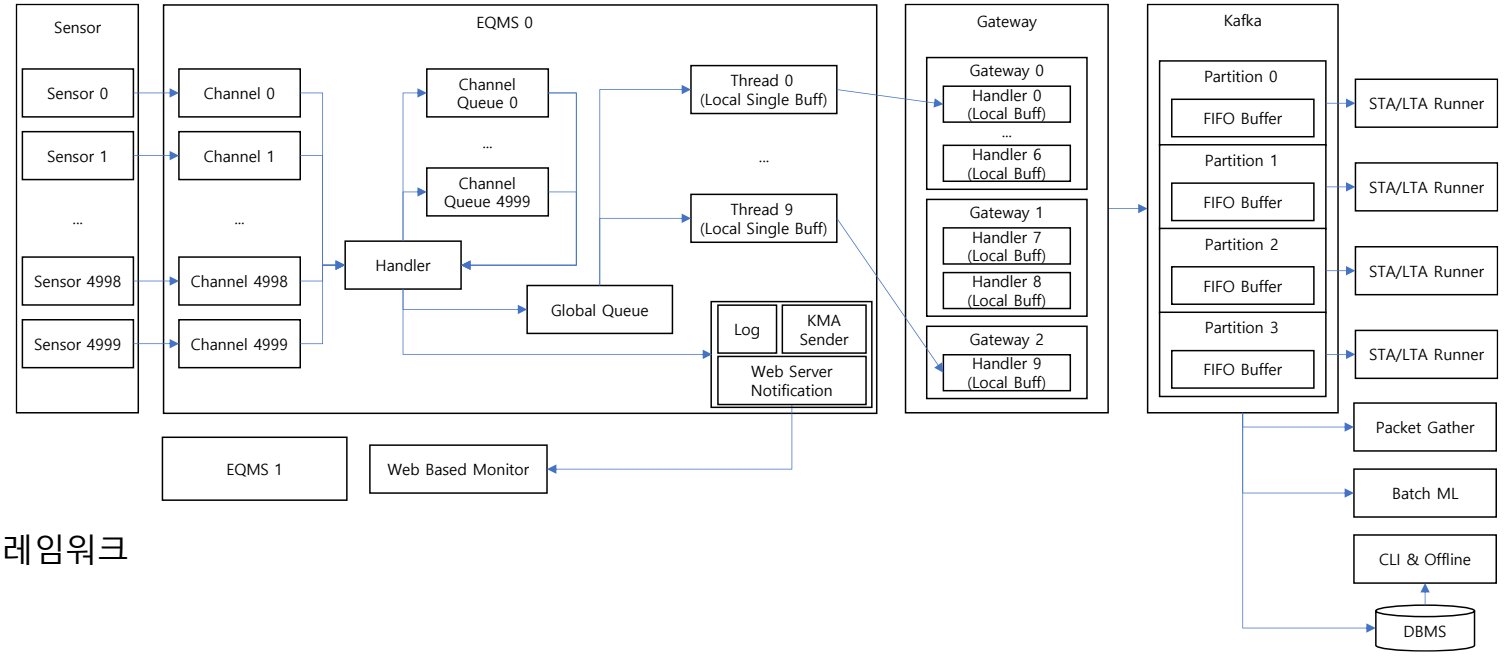
신영술, 최윤자
2023년 07월 05일
소프트웨어재난연구센터

개요: 메시지 순서 역전

- 지진경보시스템에서 메시지 순서 역전 현상 존재
  - 최종 지진 판단의 시간적 지연이나 오판의 야기 가능성



- 원인
  - EQMS가 문제의 시작점
  - EQMS의 구조적 문제
    - 독립적으로 수행되는 멀티 스레드
  - EQMS의 성능적 문제
    - 부족한 메시지 처리 속도

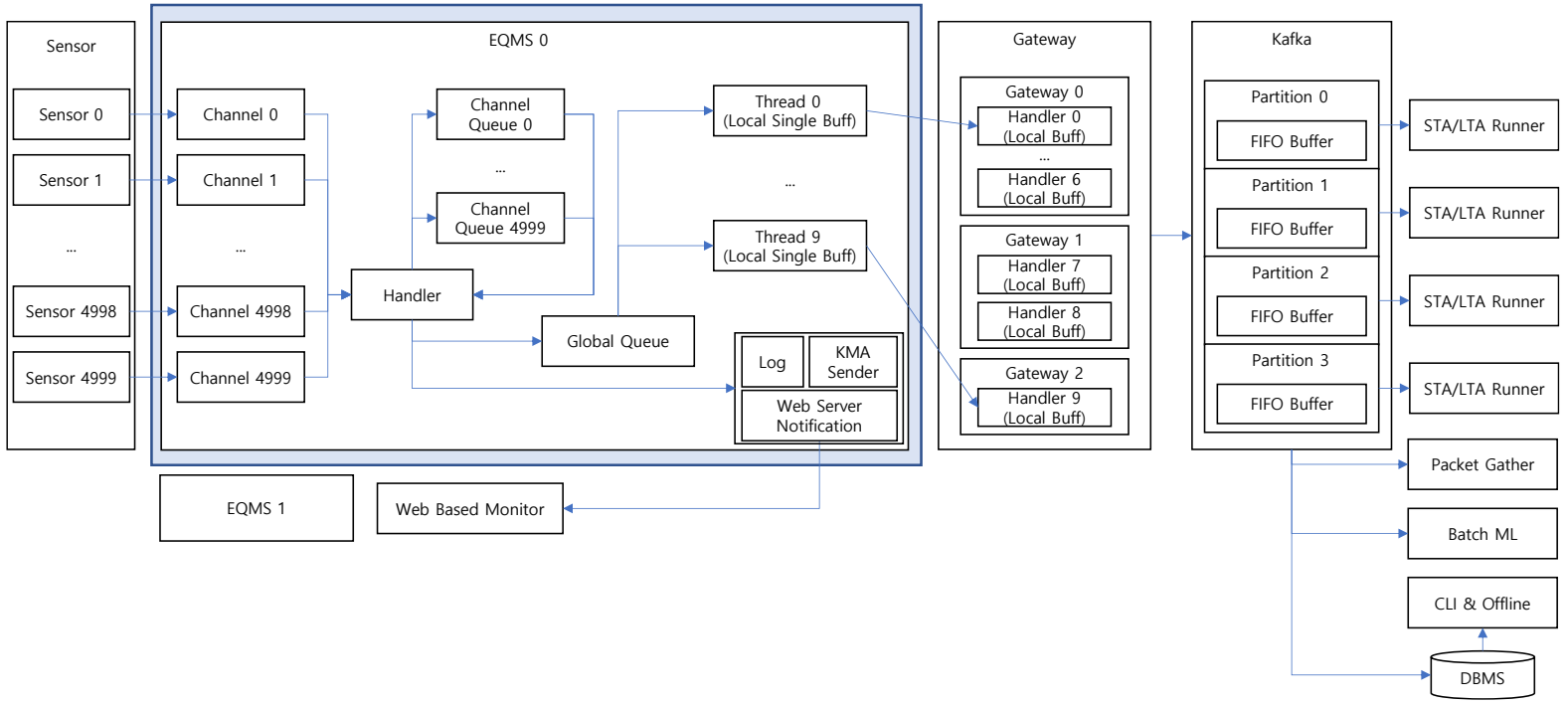


- 해결 방안
  - 구조적 문제 해결 방안
    - 동기화된 멀티 스레드 / 싱글 스레드 / 써드파티 프레임워크
    - 싱글 소켓
  - 성능적 문제
    - 메시지 처리 지연구간 해소: 블로킹 입출력을 넌블로킹 입출력으로 변경

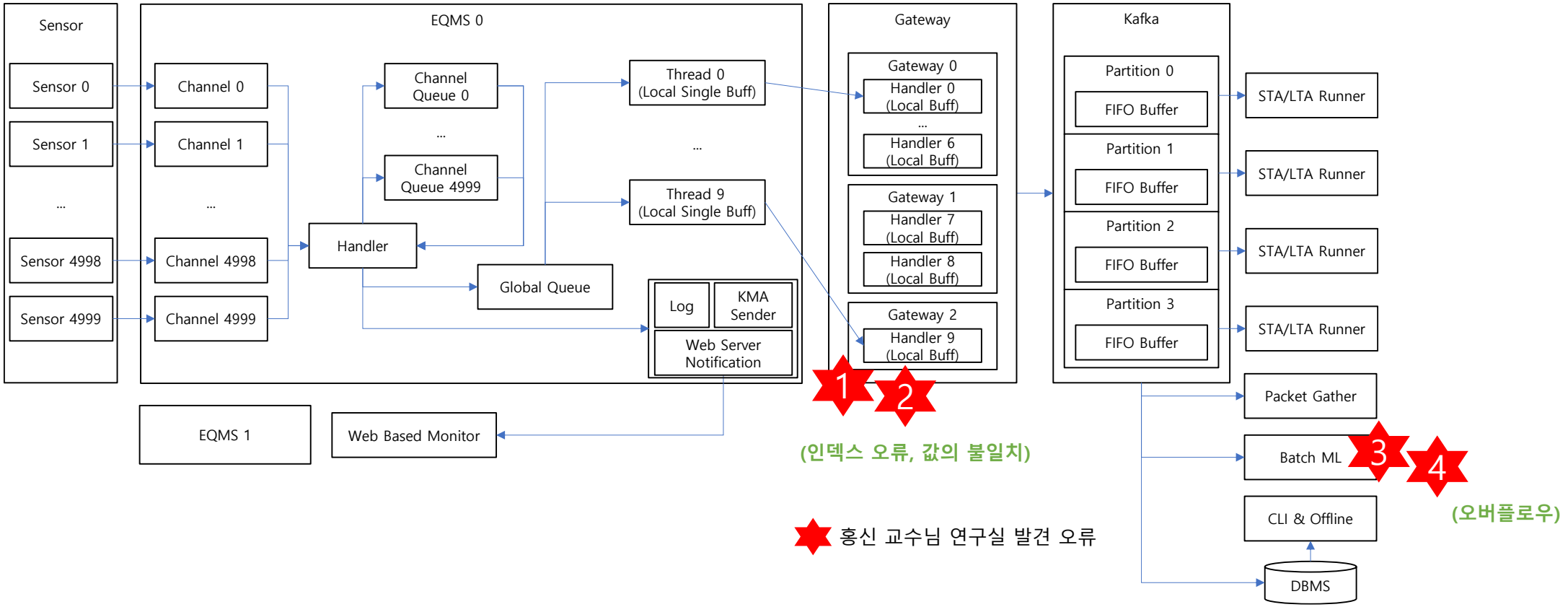
- 논의 사항
  - 소프트웨어 공학적 관점에서의 기술적 대응 방법
  - 다른 해결 방안

메시지 순서 역전의 연구 배경

- 지진경보시스템의 미션
  - 지진을 최대한 정확하고 빨리 감지하여 경보
- 미션 달성에 방해되는 오류를 찾고 수정하는 것이 중요
- 지진경보시스템의 기존 연구 자료 분석
  - 게이트웨이는 이미 연구 중
  - EQMS 연구 필요
- 정적 분석 도구 Coverity와 단위테스트로 오류 분석
  - 지진경보시스템의 미션에 직접적 영향은 발견 못함
- CrowdQuake 개발자 인터뷰
  - 개발 단계의 로그를 통해 메시지 순서 역전 현상 발견
  - 지진 감지에 영향
  - EQMS가 원인의 시작점이라 추측
- EQMS의 메시지 순서 역전 현상의 원인과 영향 분석을 위한 실험 수행



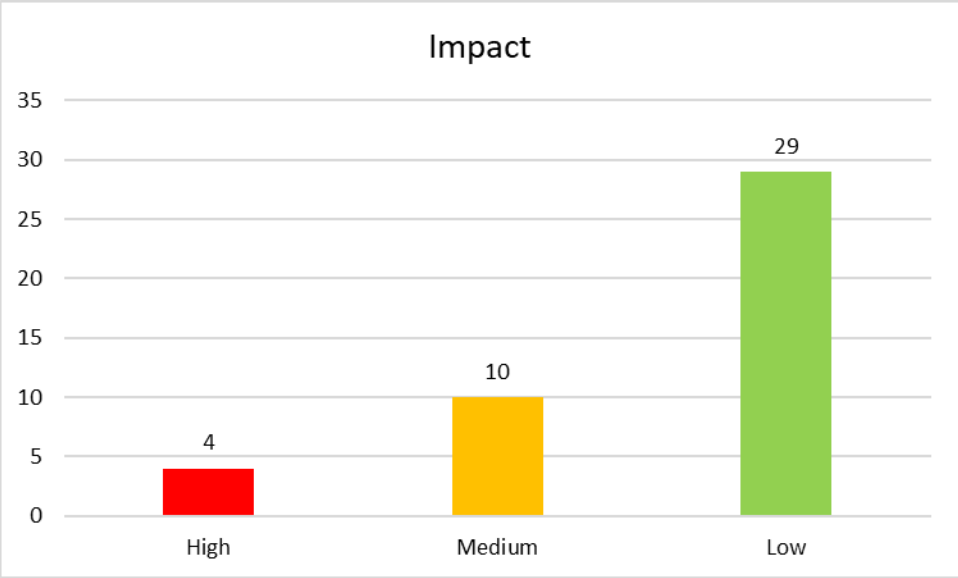
지진경보시스템에서 발견된 기존의 오류



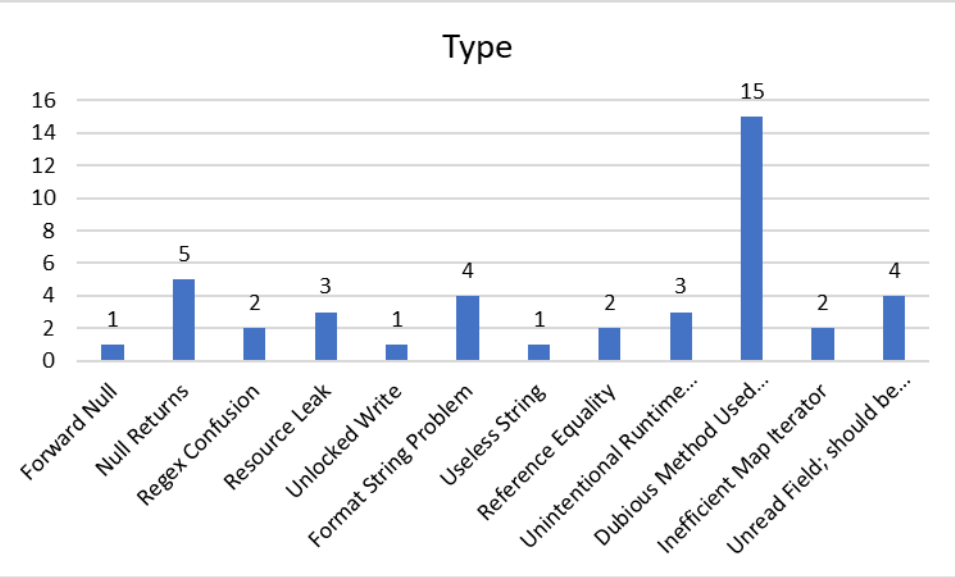
- 오류 #1
  - 대상: Gateway
  - 원인: DATA_LENTH=0 and SENSOR_MESSAGE is empty
  - 증상: 인덱스 오류 Crash
- 오류 #2
  - 대상: Gateway
  - 원인: EQMS_A_MESAGE_TYPE is not one of {0x04, 0x05, 0x0D}
  - 증상: 관련 변수의 값이 불일치
- 오류 #3
  - 대상: Batch ML
  - 원인: BUFF_SIZE < MSG_SIZE, BUF_LENGTH=0
  - 증상: 버퍼 오버플로우 Crash
- 오류 #4
  - 대상: Batch ML
  - 원인: BUF_SIZE - BUF_LENGTH < MSG_SIZE
  - 증상: 버퍼 데이터 손실

오류 형태	오류 ID
인덱스 오류	#1
값의 불일치	#2
오버플로우	#3, #4

EQMS의 Coverity 분석 및 단위테스트



Impact	Count
High	4
Medium	10
Low	29
총합	43

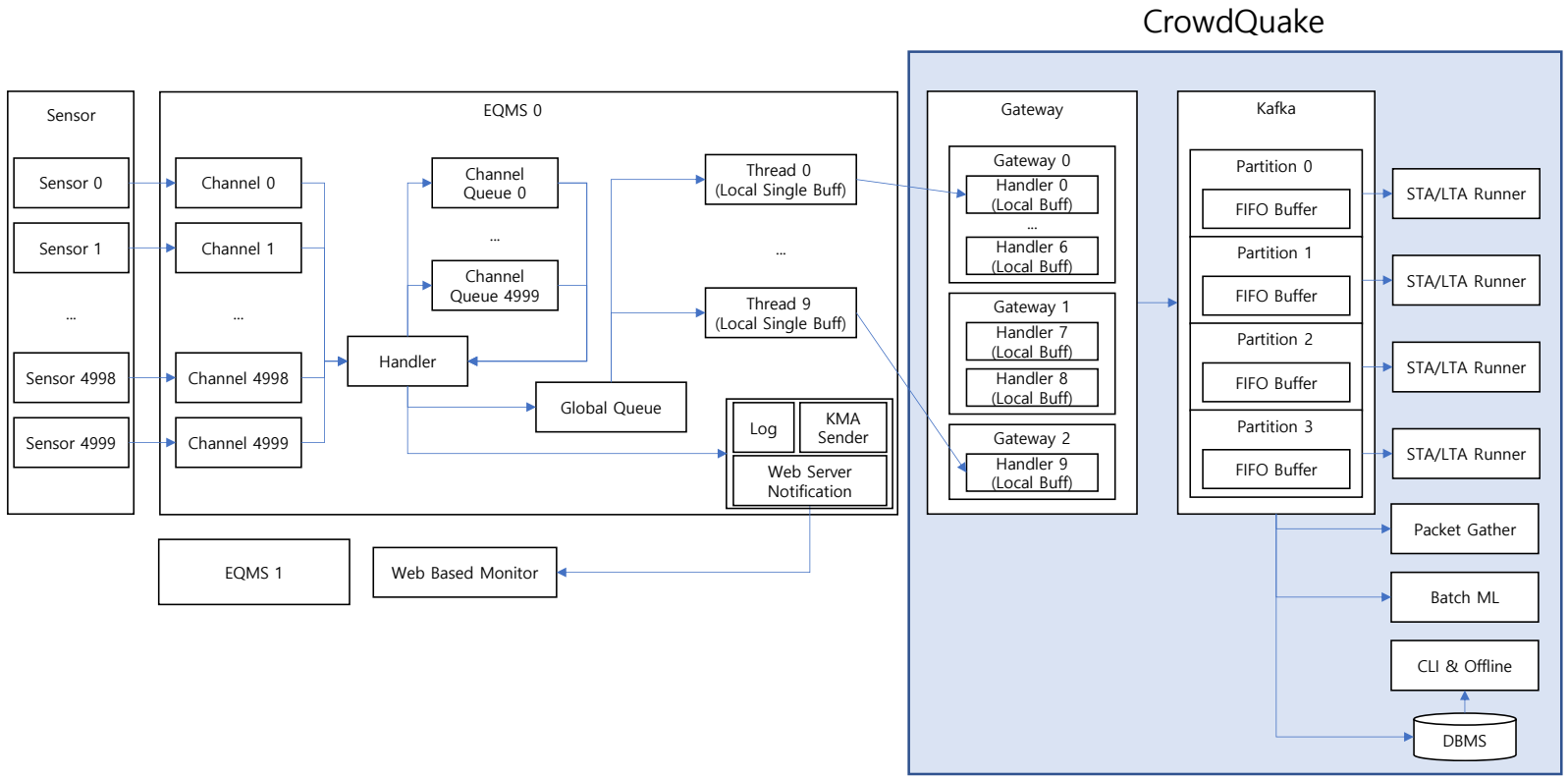


Type	Count
Forward Null	1
Null Returns	5 -> 4
Regex Confusion	2 -> 0
Resource Leak	3
Unlocked Write	1
Format String Problem	4
Useless String	1
Reference Equality	2
Unintentional Runtime Exception Capture	3
Dubious Method Used	15
Inefficient Map Iterator	2
Unread Field: should be static?	4 -> 0

- 지진 감지 미션에 영향을 주는 오류 발견 시도
- Coverity가 찾은 오류는 미션 달성에 영향이 없는 것으로 판단

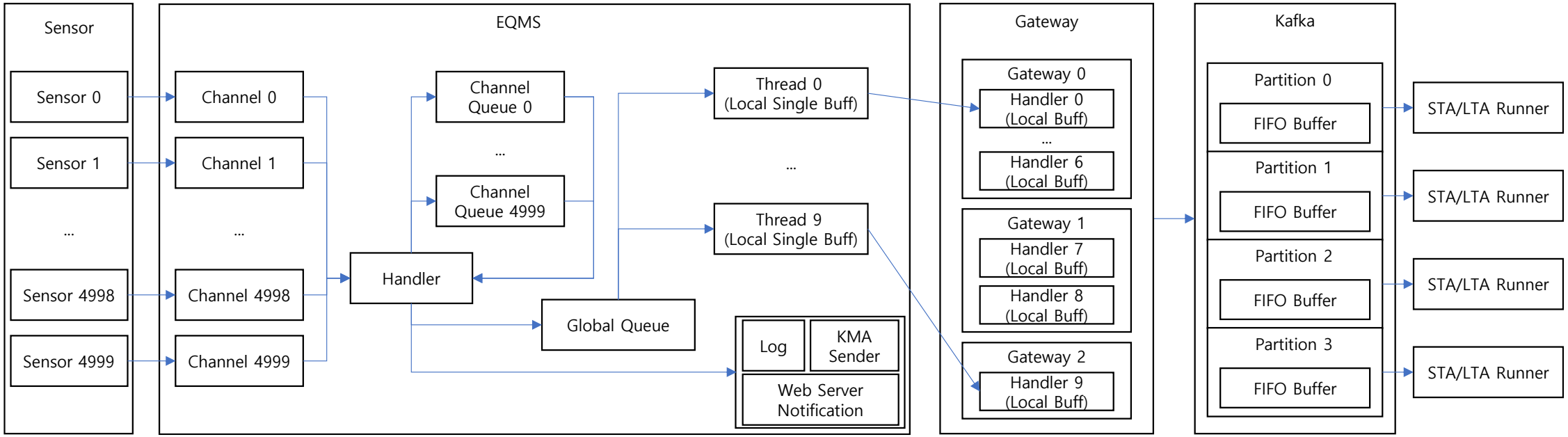
개발자 인터뷰

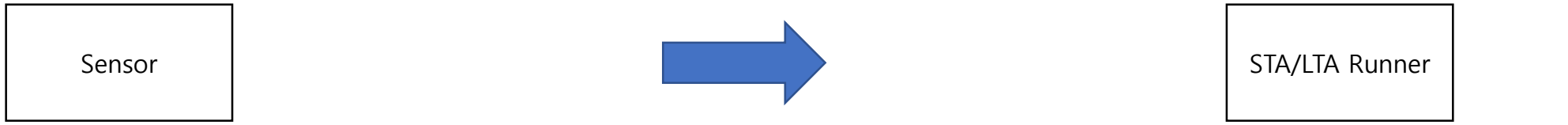
센서가 보낸 메시지 순서와 똑같은 순서로 수신될 것이라 가정하고 개발했는데,
순서가 변경되어서 수신됩니다.
이러면 곤란합니다.

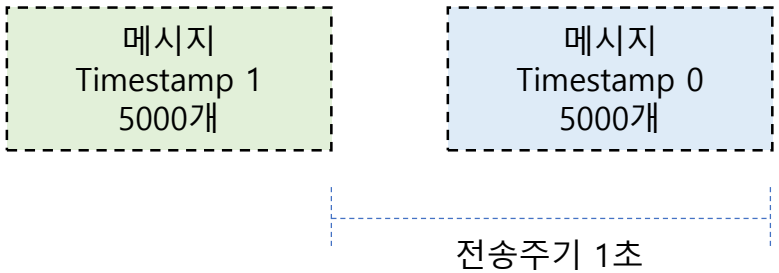


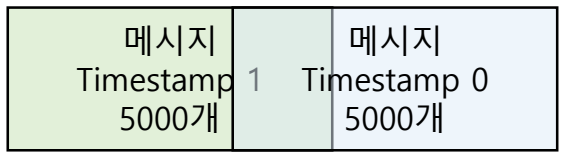
- CrowdQuake 개발자 인터뷰
  - 개발 단계의 로그를 통해 메시지 순서 역전 현상 발견
  - 지진 감지에 영향
  - EQMS가 원인의 시작점이라 추측

메시지 도착 순서 역전



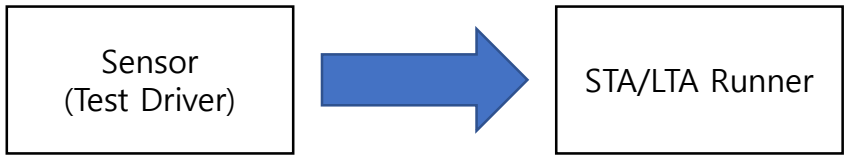






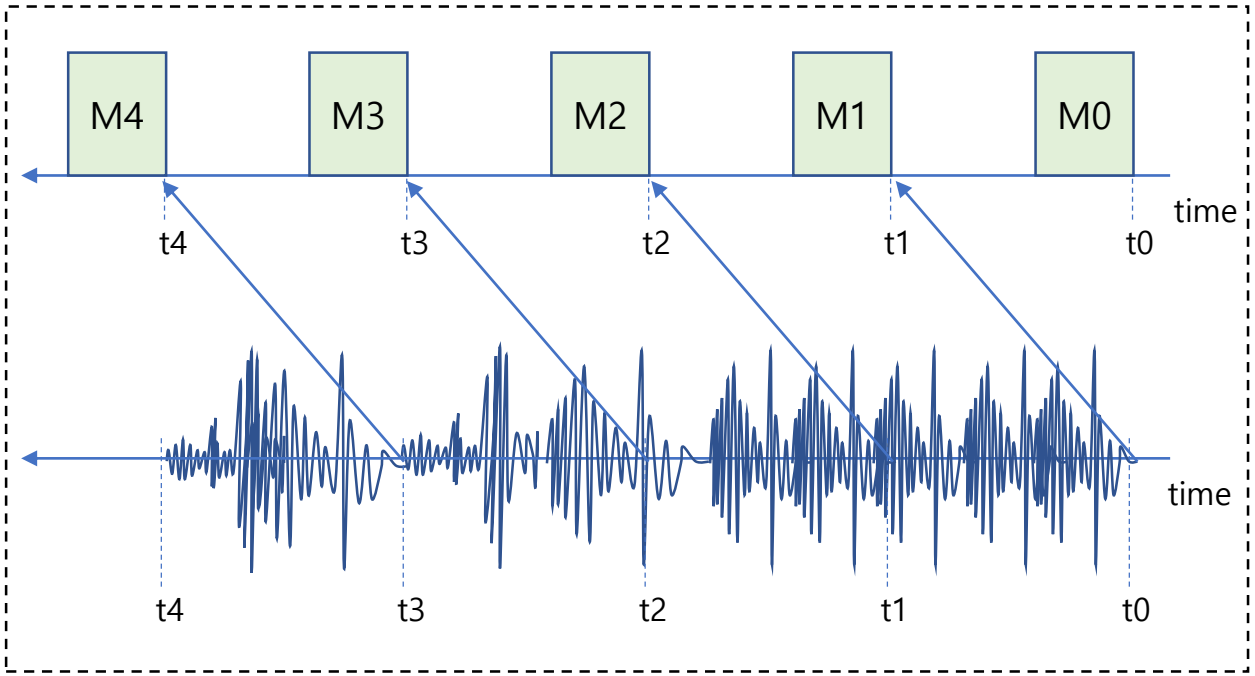
- STA/LTA Runner는 각 센서의 진동을 감지하는 역할
- 타임스탬프 값이 다른 메시지들이 같은 장소에서 섞여 버림

단일 센서 관점에서의 메시지 순서 역전 실험

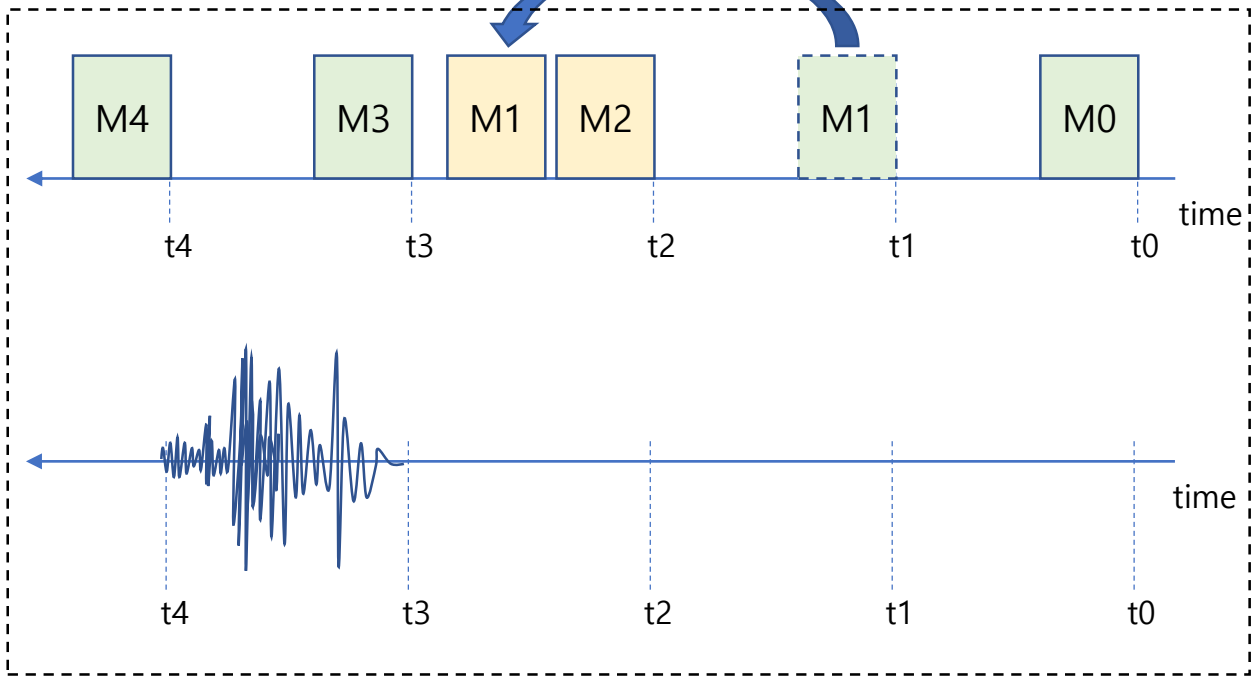


- 진동 시작점의 첫번째 메시지와 두번째 진동 메시지의 위치를 역전시켜 전송
- 진동 감지 시점의 변화를 관찰
- 진동 감지 시점의 지연 및 감지 실패가 관찰됨

- 정상 순서



- 순서 역전

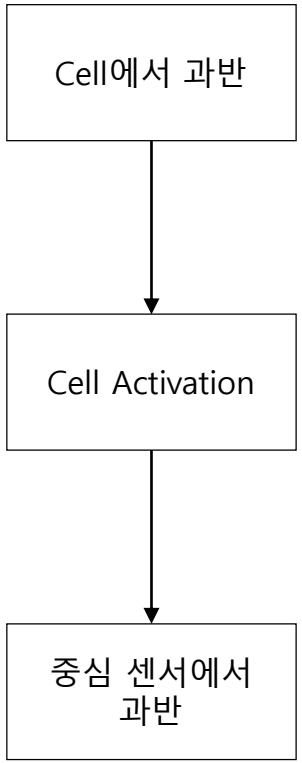


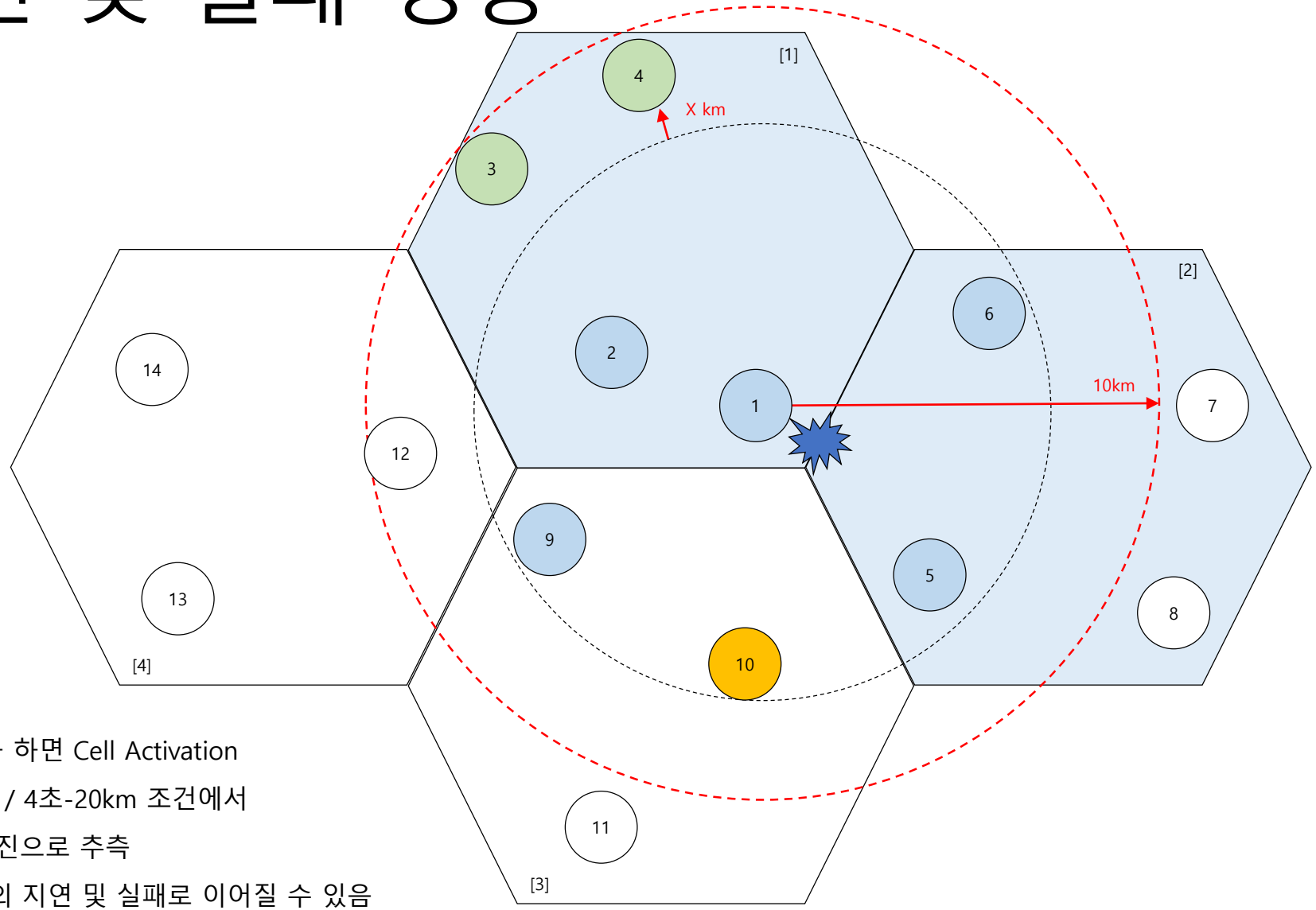
메시지 순서 역전 실험 결과: 감지 실패 및 지연

센서 ID	정상 순서		역전된 순서			진동 감지 시작 지연 크기 (단위: sec)
	진동 감지 시작 시점	진동 감지 종료 시점	역전된 시점	진동 감지 시작 시점	진동 감지 종료 시점	
01231322023	31	41	31, 32	35	41	4
01231321266	38	40	38, 39	감지 실패	감지 실패	감지 실패
01231316295	37	38	37, 38	감지 실패	감지 실패	감지 실패
01232970504	34	45	34, 35	37	45	3
01227509952	48	51	48, 49	50	51	2
01231300010	505	510	505, 506	507	510	2
01231300235	31	45	31, 32	33	45	2
01231315889	496	510	496, 497	498	510	2
01231322799	372	386	372, 373	374	386	2
01232971123	489	496	489, 490	491	496	2
01232972186	499	505	499, 450	501	505	2
(1,161개)	-	-	-	-	-	1

- 1172 진동 정보 파일
  - 각 파일 약 850 ~ 960 초 길이
- 첫 번째 파형의 감지 시점 변화를 관찰
  - 감지 실패 및 감지 지연 발생 확인

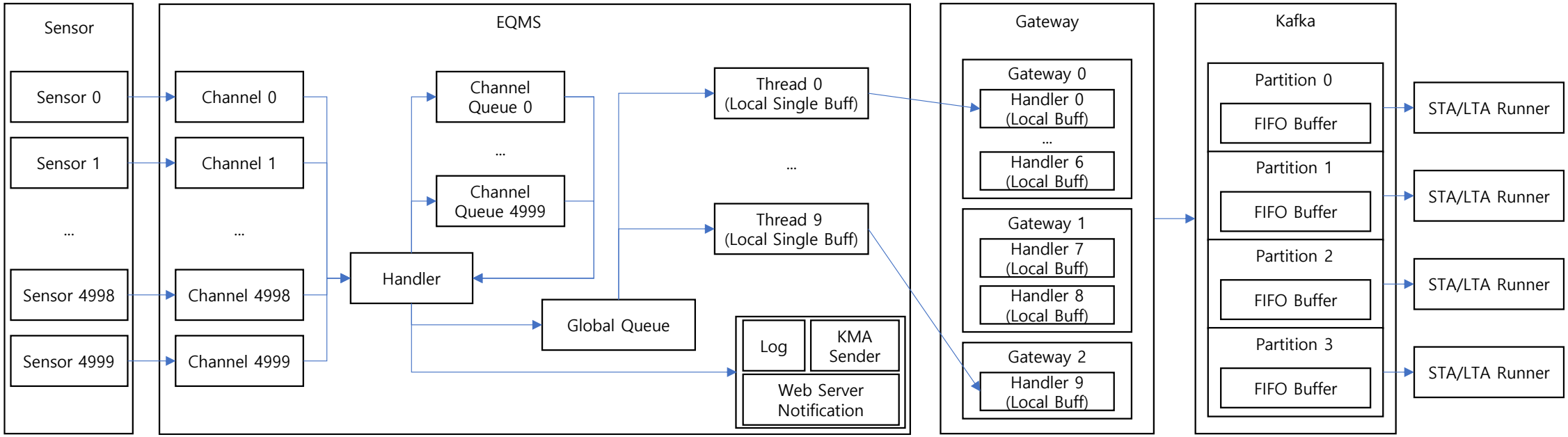
센서의 감지 지연 및 실패 영향

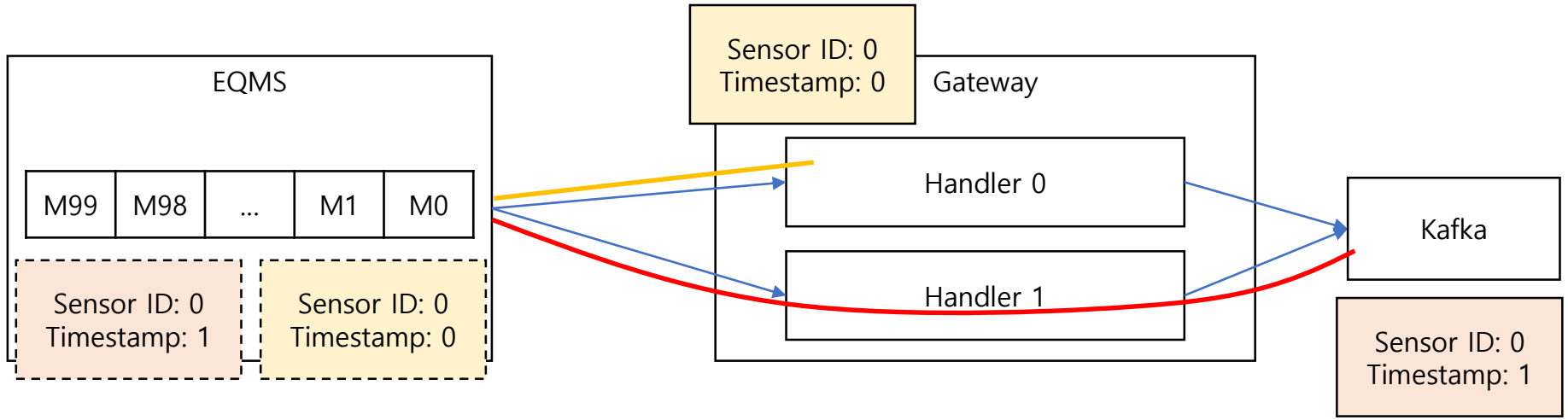




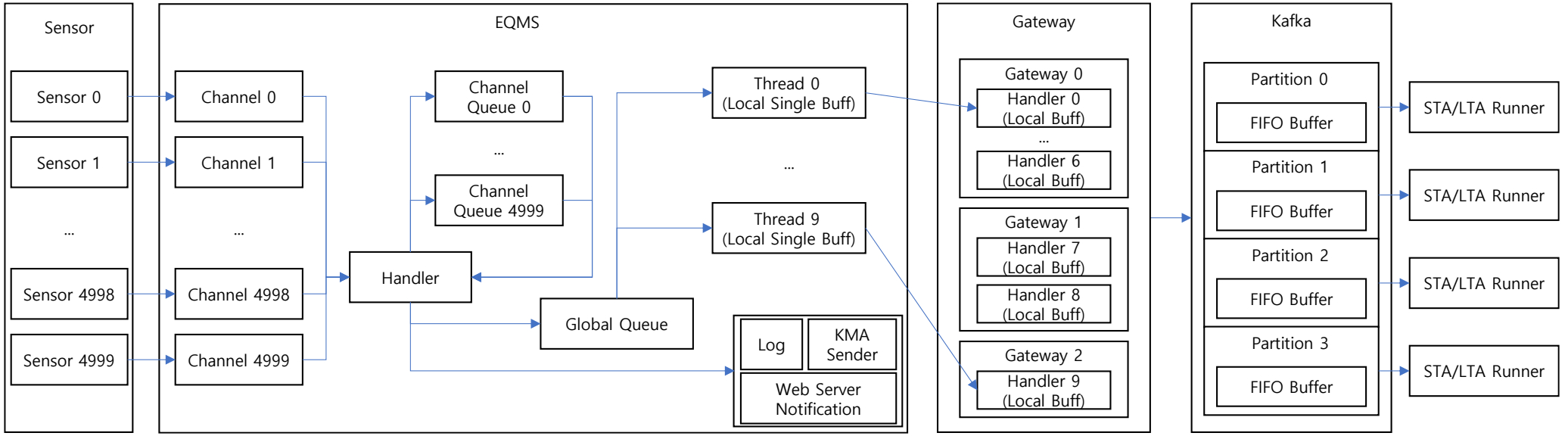
- Cell에서 과반의 센서가 분석 대상 파형 감지를 하면 Cell Activation
- 최초 감지한 센서(중심 센서)로부터 2초-10km / 4초-20km 조건에서
  과반의 센서가 분석 대상 파형 감지를 하면 지진으로 추측
- 센서의 진동 감지 지연이나 실패는 최종 판단의 지연 및 실패로 이어질 수 있음
  - 센서: 감지 딜레이 1초, 감지 실패
  - 최종판단: (1초 딜레이) or (X km / 6 ~ 8km/h 딜레이) or (감지 실패)

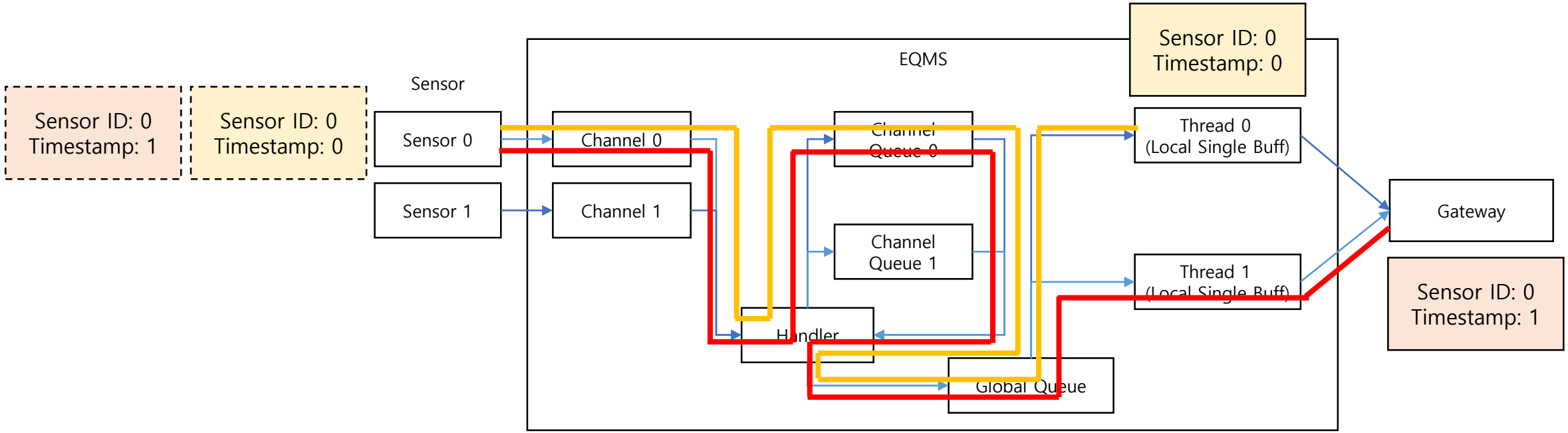
구조 문제: Gateway 모델 체킹



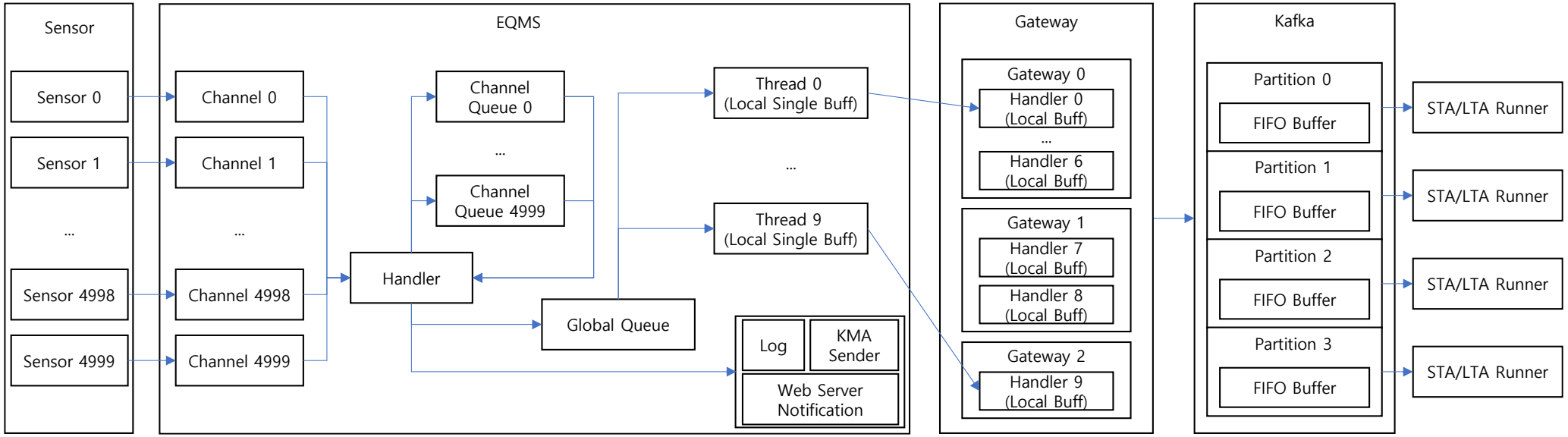


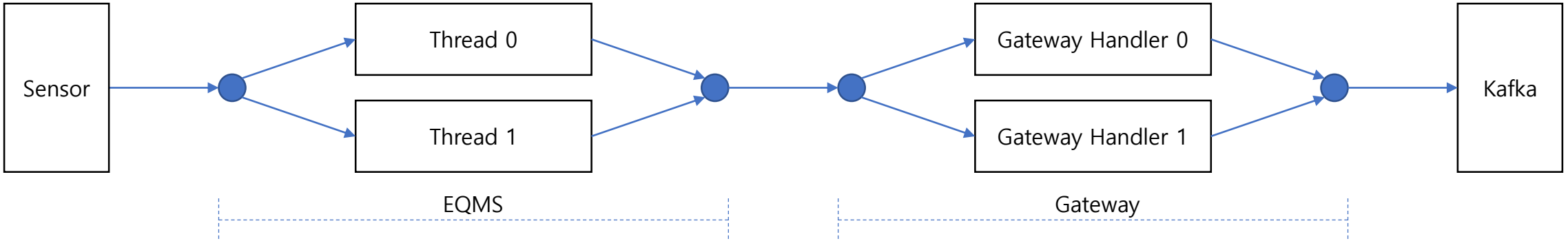
구조 문제: EQMS 모델 체킹





구조 문제: 모델 체킹 결과 리뷰

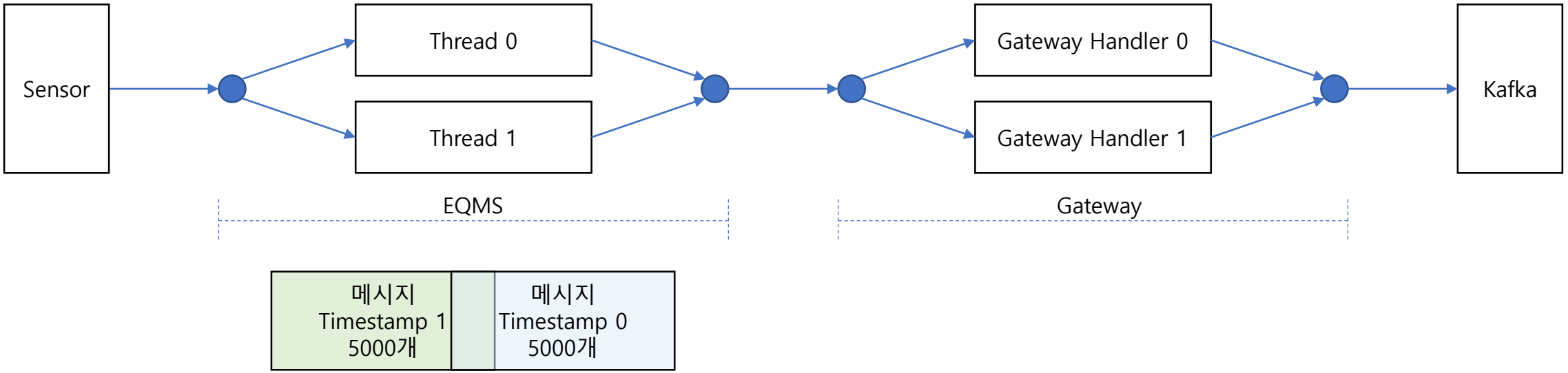




- 메시지 추월 방법
- 같은 장소(컴포넌트)에서 만난다
  - 뒤에 출발한 메시지가 앞선 메시지의 뒤꽁무니를 바짝 추격한다
- 추월 차선을 통해서 추월 한다

- 구조적으로 메시지 순서 역전의 가능성 존재
- 그런데, 센서가 1시간마다 한 번 메시지를 보낸다면 메시지 순서 역전이 발생할까?

성능 문제: EQMS의 메시지 처리 지연 모델 체킹



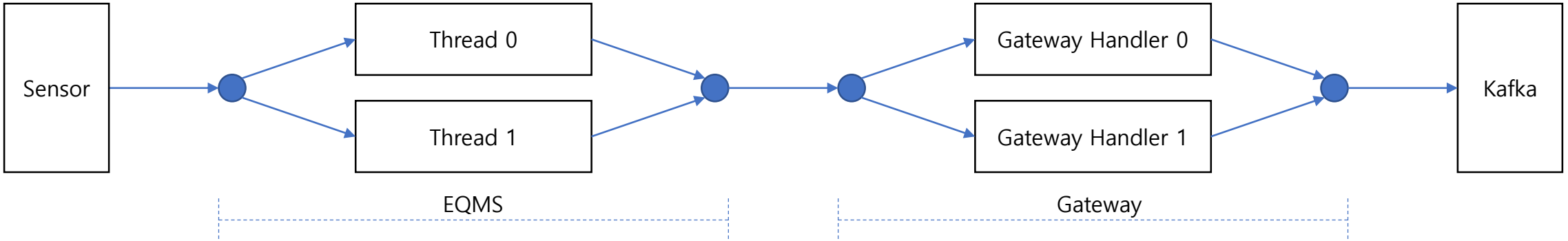
- 모든 스레드의 총 수행 횟수 < 모든 센서의 전송 주기별 총 메시지 수
  - 메시지 역전 발생 가능

- 모든 스레드의 총 수행 횟수 == 모든 센서의 전송 주기별 총 메시지 수
  - 메시지 역전 없음

- 메시지 순서 역전 방지를 위해서 전송 주기 기준으로 스레드의 수행 횟수가 전송 주기별 메시지 수만큼 보장 필요

- Gateway Handler도 스레드 형태로 실행되므로 동일한 문제가 적용됨

EQMS 상에서 메시지 역전 원인 요약

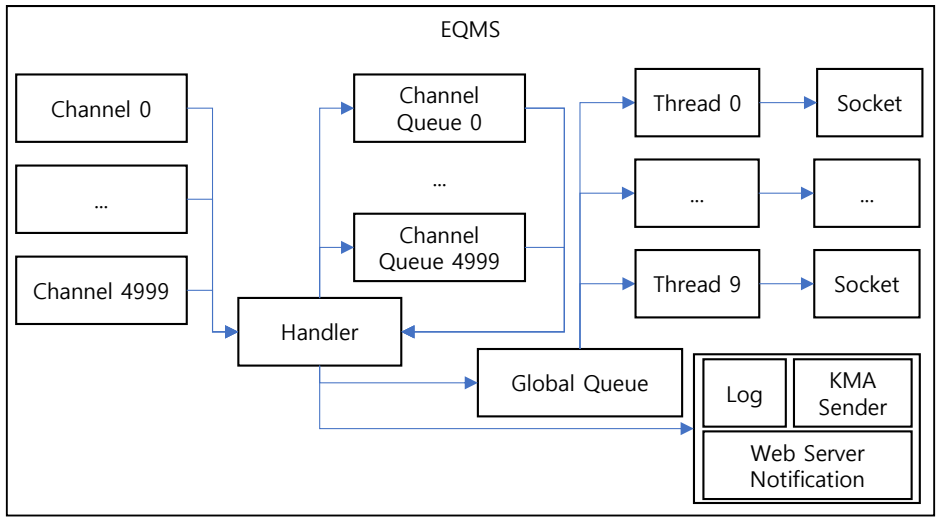


- 구조 문제와 성능 문제의 합작품
  - 구조 문제
    - 독립적으로 수행되는 두 개 이상의 스레드
  - 성능 문제
    - 메시지 전달 지연 문제
    - 센서의 메시지 전송 주기보다 느린 메시지 처리

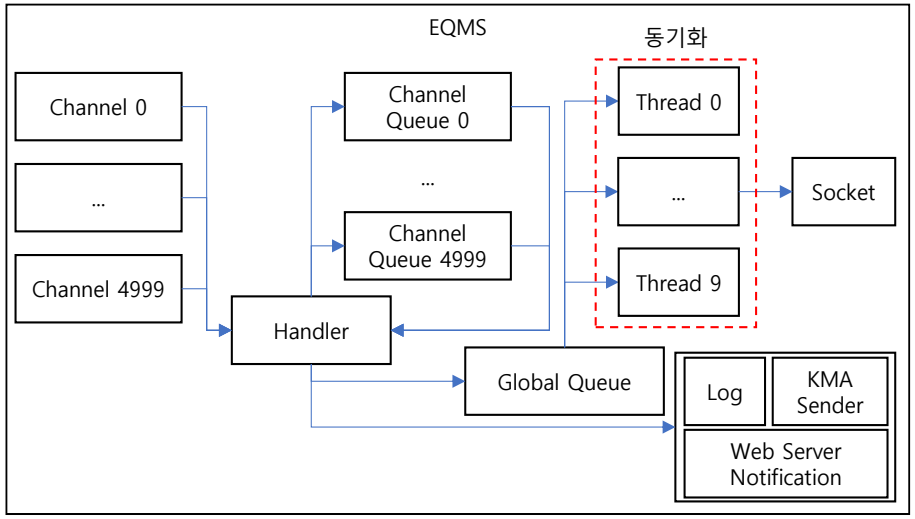
- 테스트 베드 상에서 확인 실험 예정
  - 실제 코드의 실행을 통해 모델 체킹 결과를 확인

구조 문제의 해결방안

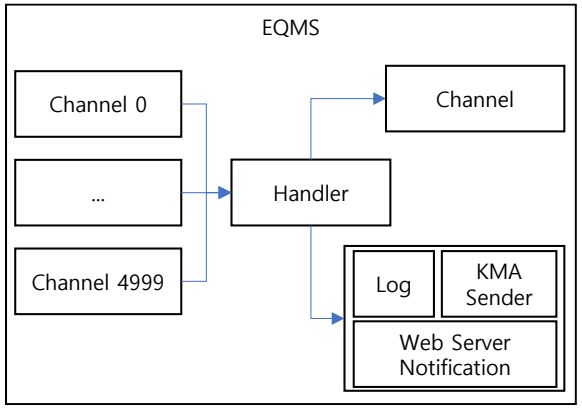
- 현재 EQMS 구조



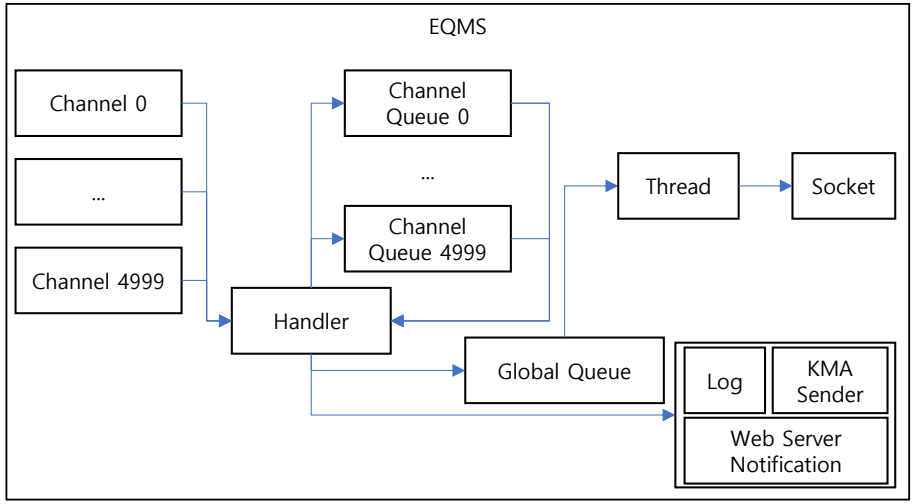
- 동기 멀티 스레드 해결 방안



- Netty 프레임워크 해결 방안



- 싱글 스레드 해결 방안



- 소켓에 차례대로 데이터를 쓰기
  - 동기화된 멀티 스레드
  - 싱글 스레드
  - Netty 프레임워크

- 네트워크 상에서 차례대로 데이터를 전달하기
  - 싱글 소켓: 멀티 스레드 경우에만 해당

구조 문제 해결방안에 대한 코드 수정: 스레드

- 원본 코드

```
public class DataToKnuSender {
    //...

    Runnable sendDataTask = () -> {
        //...
        Socket sock = null;
        InputStream inStrm = null;
        OutputStream outStrm = null;
        //...
        // 게이트웨이로 데이터 송신
        ByteBuffer buffer = bqSendDatas.take();
        outStrm.write(buffer.array());
        outStrm.flush();
        //...
    };

    no usages
    public void start() {
        //...
        // 스레드 생성 루프
        Thread thread = new Thread(sendDataTask, "sendClient-" + idx);
        //...
    }
}
```

- 동기화된 멀티 스레드

```
synchronized (lock) {
    ByteBuffer buffer = bqSendDatas.take();
    outStrm.write(buffer.array());
    outStrm.flush();
}
```

- 싱글 스레드

```
public void start() {
    senderRunning = true;
    IntStream.range( i: 0, SERVER_PORT_COUNT).forEach(idx -> {
        BlockingQueue<ByteBuffer> sendQueue = new LinkedBlockingQueue<>();
        Thread thread = new Thread(sendDataTask, s: "sendClient-" + idx);
        thread.start();
        threads.add(thread);
    });

    ...
}
```

10 -> 1

구조 문제 해결방안에 대한 코드 수정: 싱글 소켓

- 원본 코드

```
public class DataToKnuSender {
    //...

    Runnable sendDataTask = () -> {
        //...
        Socket sock = null;
        InputStream inStrm = null;
        OutputStream outStrm = null;
        //...
        // 게이트웨이로 데이터 송신
        ByteBuffer buffer = bqSendDatas.take();
        outStrm.write(buffer.array());
        outStrm.flush();
        //...
    };

    no usages
    public void start() {
        //...
        // 스레드 생성 루프
        Thread thread = new Thread(sendDataTask, "sendClient-" + idx);
        //...
    }
}
```

- 싱글 소켓

```
Socket sock = null;
InputStream inStrm = null;
OutputStream outStrm = null;
```

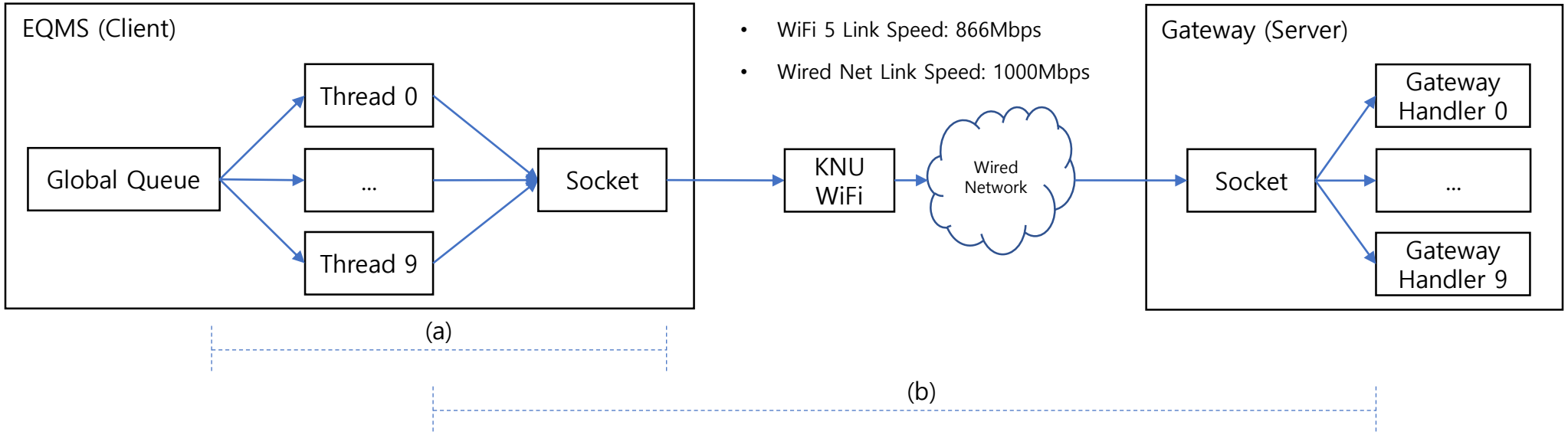
함수의 지역변수 -> 클래스 멤버변수

```
synchronized (lock) {
    if(socket != null) { continue; }
    try {
        sock = new Socket(serverIp, serverPort);
        sock.setTcpNoDelay(true);
        if (sock.isConnected()) {
            inStrm = sock.getInputStream();
            outStrm = sock.getOutputStream();
        } else { threadSleep( ms: 200); }
    } catch (Exception e) { e.printStackTrace(); }
}
```

구조 문제 해결방안의 실험

- CPU: Intel 11th i5-1135G7 2.4GHz, 8 Cores
- Mem: 8GB SSD 2667MHz
- NIC: Intel WiFi 6 AX201 160MHz

- CPU: Intel i5-10400 2.9GHz, 12 Cores
- Mem: 8GB SSD 2667MHz
- NIC: Realtek RTL8168 1Gigabit



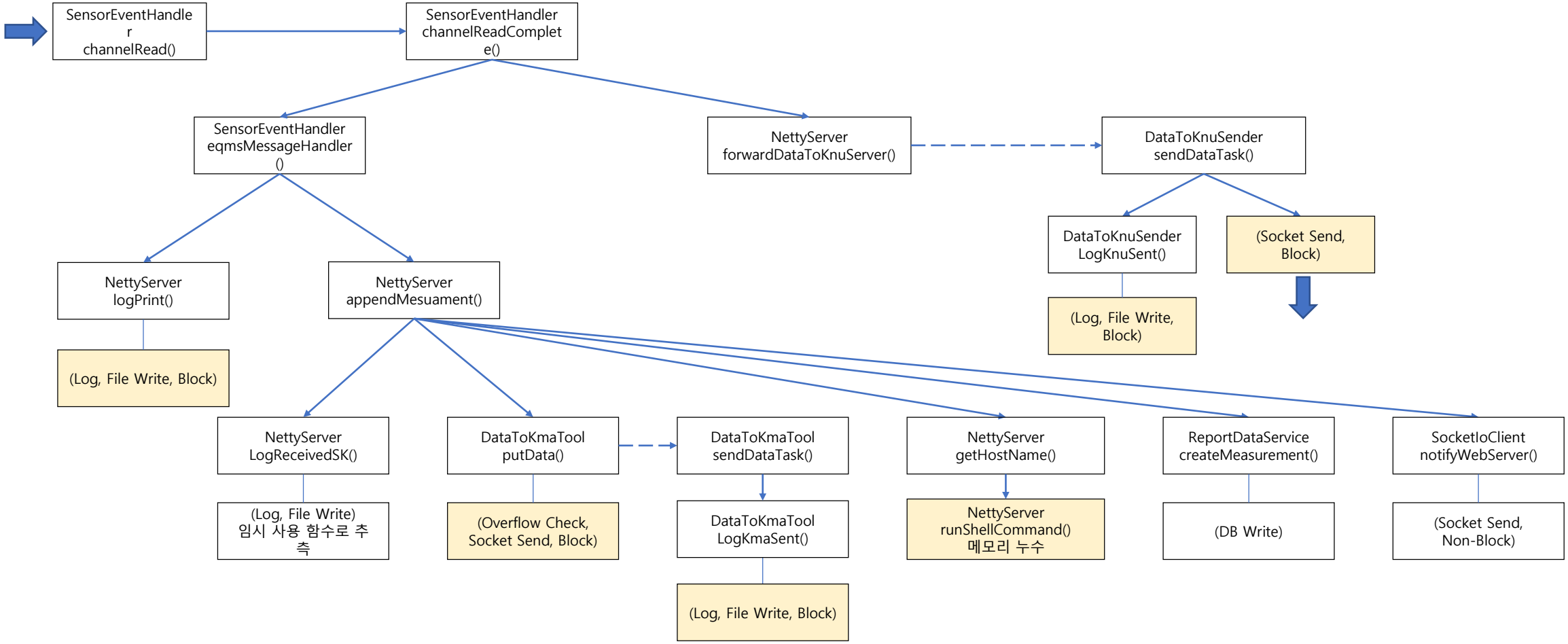
- WiFi 5 Link Speed: 866Mbps
- Wired Net Link Speed: 1000Mbps

(a)	현재 구조: 비동기 스레드 10개	해결방안: 동기 스레드 10개	해결방안: 스레드 1개	해결방안: Netty 프레임워크
(a) 평균값 (millisec)	0.057	0.063	0.065	(실험 예정)
(b) 평균값 (millisec)	197.375	220.351	295.120	(실험 예정)

- (a) : EQMS가 (721Byte * 5000개)를 소켓의 송신 버퍼에 전달하는데 소요되는 시간
- (b) : 메시지가 EQMS 응용 계층과 Gateway 응용 계층 사이를 횡단하는데 소요되는 시간

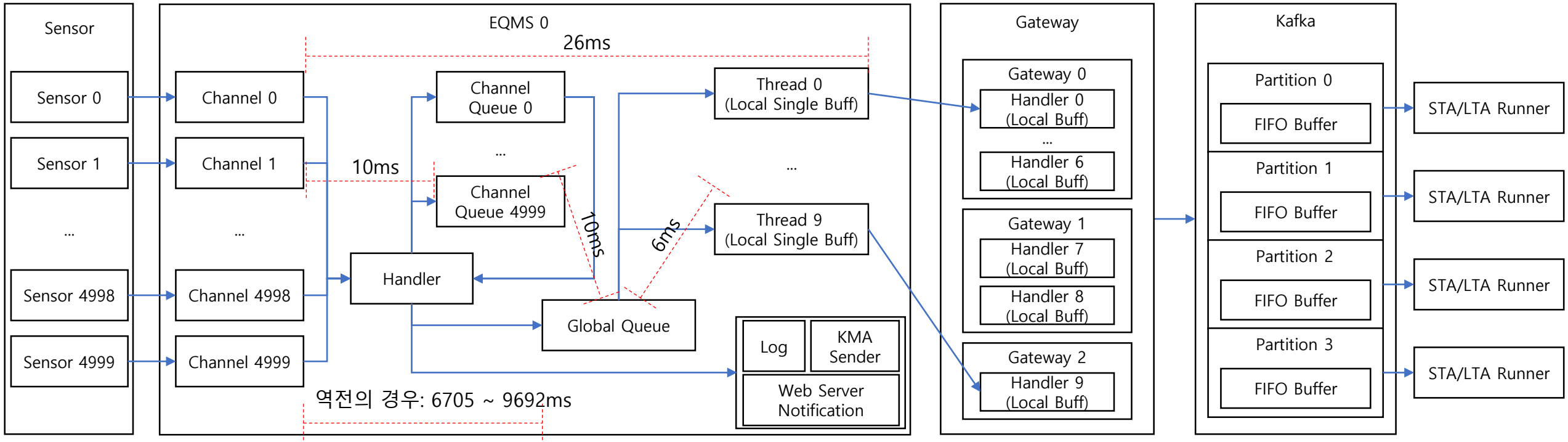
- 응용 계층에서 메시지 캡처: 12시간 동안 메시지 순서 역전 없음
- 해결방안의 성능이 현재 구조의 성능보다 다소 부족한 것으로 판단됨
  - 적어도 현재의 성능보다는 좋거나 같아야 함
  - 다른 부분에서 성능 보상을 받아야 함

성능 문제의 원인 추측



- 원인 추측
  - 파일 Open, Write, Close 함수
  - 소켓을 통한 Blocking 송신 함수
  - BufferedReader 타입 객체의 메모리 누수
    - Coverity 분석으로 발견
- 해결 방안
  - 블로킹 입출력을 논블로킹 입출력으로 변경
  - 사용을 마친 객체는 close() 호출
  - 구조적 해결방안에서 손해본 성능까지 보상 가능

EQMS 내부 지연 시간 (단편적 실험 데이터)



- 센서 5000개
  - 메시지 크기: 721Byte
  - 5000 * 721Byte = 3.605MByte

- 메시지 전달에만 필요한 순수 시간은 센서 전송 주기인 1초 대비 3%에 불과

- 97%의 시간을 소비하는 지연 요소가 있을 것이라 추측함

논의 사항

- 메시지 순서 역전 문제가 흥미 있는 문제인가?

- 지진 미탐 원인(메시지 순서 역전)을 찾을 수 있는 소프트웨어공학 기술은 무엇인가?

- 해결 방안에 대한 논의
  - 지진 미탐 해결방안(구조 및 성능 문제)을 소프트웨어공학 기술로 대응 가능한가?
  - 메시지 순서 역전 현상에 대한 다른 해결 방안은 무엇인가?

- 향후 필요 연구 방향은 무엇인가?

- 포스터 발표 시간에도 많은 피드백 부탁드립니다.

감사합니다