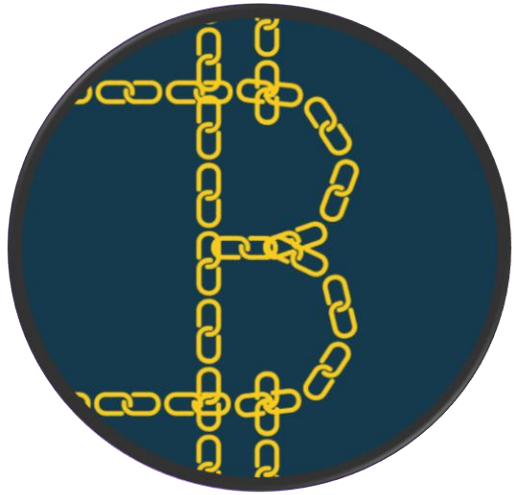


분산컴퓨팅특론

최종 결과 보고

Dept. of IT Convergence Engineering
팀원 | 안지용, 최진서, 문기연

전체 일정 진행 결과

- Project timeline
  - Google spreadsheet link
  - GitHub repository

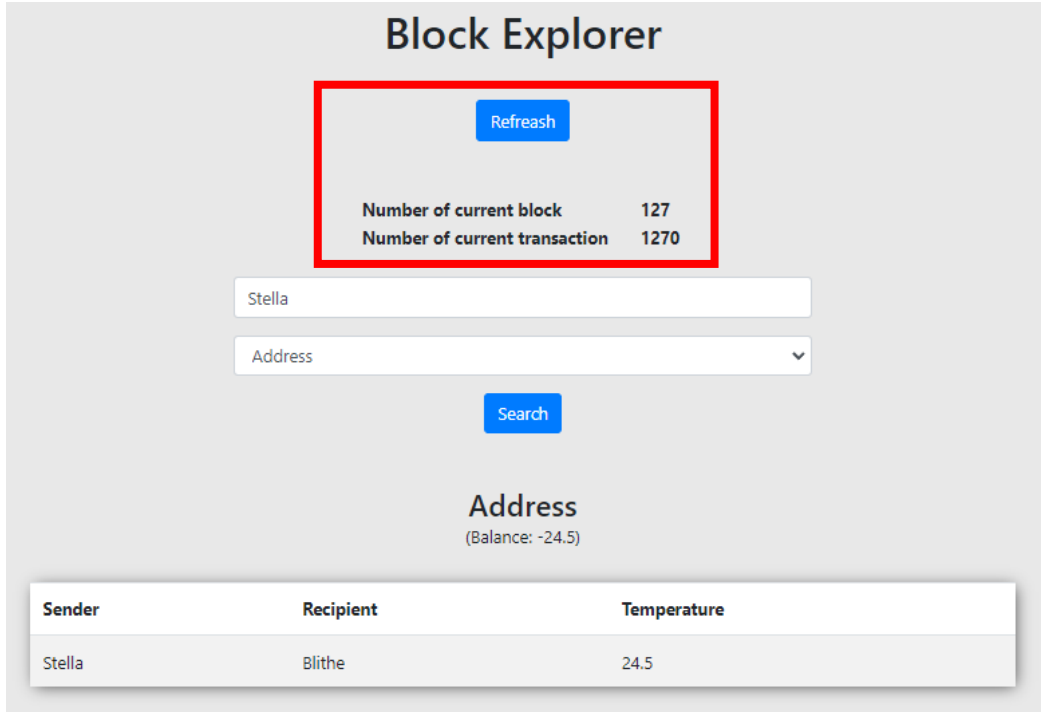
시작일	2023. 4. 1
오늘	2023. 5. 25

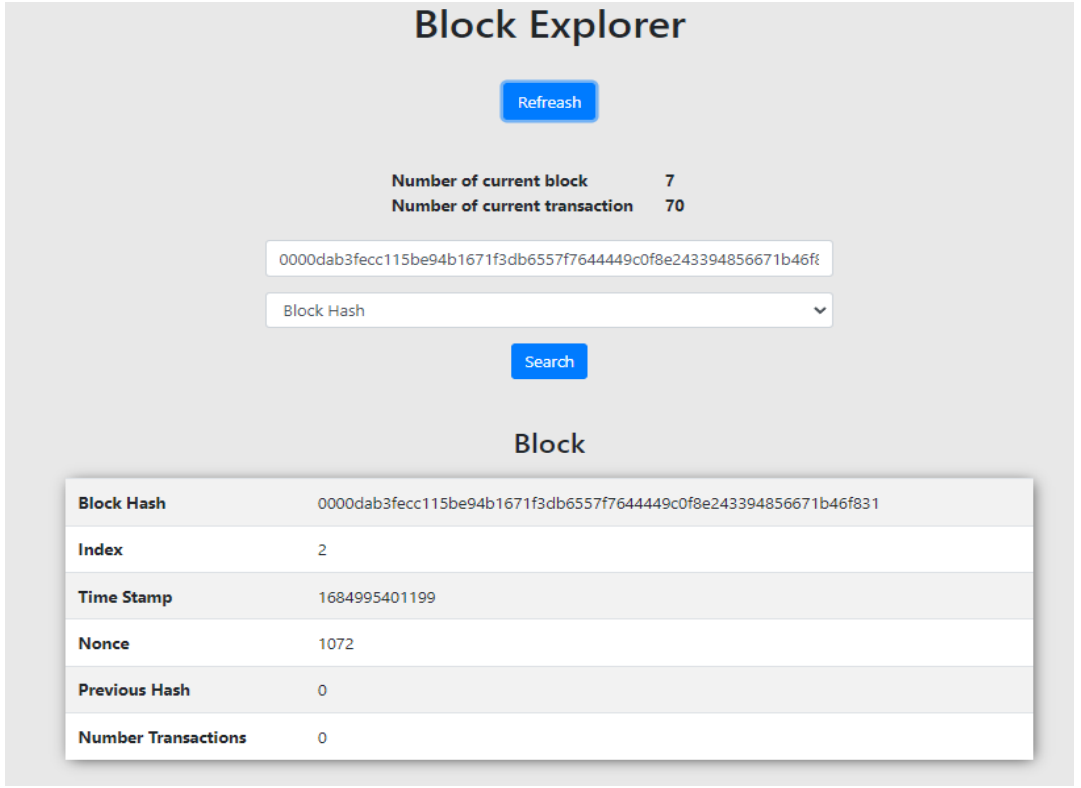
일정		시작일	종료일	남은 일자	5월																				
					28	1	2	3	4	5	8	9	10	11	12	15	16	17	18	19	22	23	24	25	26
Blockchain Network Implementation	Status				금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금	월	화	수	목	금
개발환경 세팅, 모듈 설계	☑	2023. 4. 10	2023. 4. 14	-																					
모듈구현, 모듈 테스트(연장)	☑	2023. 4. 17	2023. 5. 5	-	연장																				
중간점검자료 작성	☑	2023. 4. 24	2023. 4. 28	-	중간점검			구조변경																	
피드백 반영, 추가구현	☑	2023. 5. 1	2023. 5. 5	-				설계		추가모듈 구현															
모듈 통합	☑	2023. 5. 8	2023. 5. 12	-							UI 구현				개발완료						발표자료 작성				
테스트환경 구성, 성능 테스트	☑	2023. 5. 15	2023. 5. 19	-												테스트 시작									
최종 제출자료 작성	☑	2023. 5. 22	2023. 5. 26	D-1																	테스트 완료				제출

- 결과물
  - 소스코드
  - 실험 데이터
  - 실험 스크립트
  - 발표자료

피드백 반영

- Blockchain 블록탐색 UI 추가
  - 세미나 교재 소스코드 사용(AngularJS + HTML)
  - **현재 블록, 전체 트랜잭션 개수 확인 기능 추가**
  - 현재 블록, 트랜잭션 개수 획득을 위한 추가 API 구현





추가 변경사항

- UI 데이터 전달을 위한 추가 구현
  - 현재 블록개수 반환 GET API
  - 전체 트랜잭션 개수 반환 GET API
  - Index.html 내 스크립트 추가

```
// get block number
app.get('/blocknumber', function(req, res){
    const block_number = bitcoin.getBlocknumber();
    console.log(block_number);

    res.json({
        block_number: block_number
    });
});

//get overall transaction number
app.get('/transactionnumber', function(req, res){
    const transaction_number = bitcoin.getTransactionnumber();
    console.log(transaction_number);
    res.json({
        transaction_number: transaction_number
    });

});
```

```
<button
    type="button" ng-click="refresh()"
    class="btn btn-primary margin-auto btn-search">
        Refreash
</button>
                    <table style=tb-sd-m  >
                            <tbody>
                                    <tr>
                <td>   </td>
                <td class="col-md-4"><b>Number of current block</b></td>
                    <td class="col-md-2"><b>{{ blocknumber }}</b></td>

                                    </tr>
                                    <tr>
                <td>  </td>
                <td class="col-md-4"><b>Number of current transaction</b></td>
                <td class="col-md-2"><b>{{ transactionnumber }}</b></td>
                                    </tr>
                            </tbody>
                    </table>
```

```
$scope.get_blocknumber = function(){
    $http.get(`/blocknumber`)
    .then(response => {
        $scope.blocknumber = response.data.block_number;
        console.log($scope.blocknumber);
    });
};
 $scope.get_transactionnumber = function(){
    $http.get(`/transactionnumber`)
    .then(response => {
        $scope.transactionnumber = response.data.transaction_number;
        console.log($scope.transactionnumber);
    });
};
```

추가 변경사항

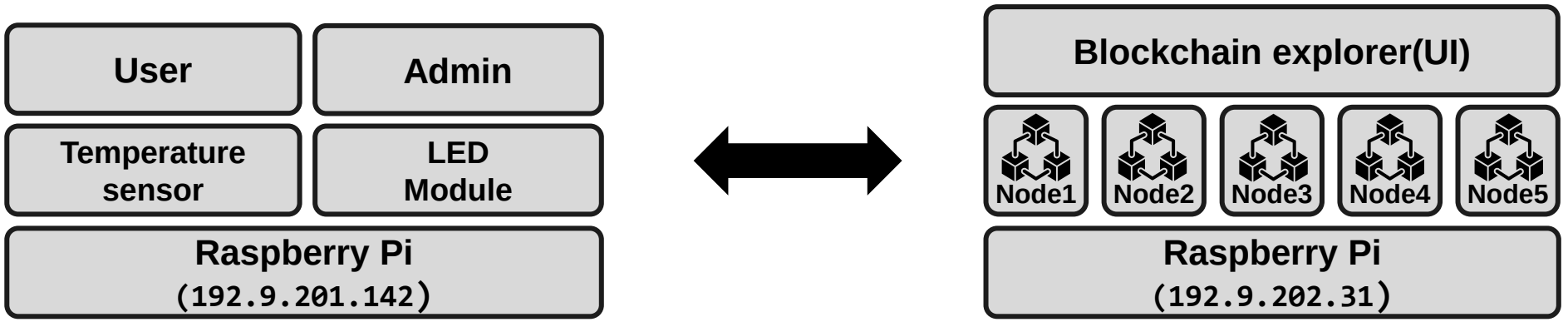
- 트랜잭션 데이터 형식 변경
  - 온도 값 추가
  - 발신자, 수신자 : unique-names-generator 모듈을 사용하여 데이터 추가

```
/*
 * @function 트랜잭션 생성
 * @return GET response 메시지
 * description 무작위로 노드 선택 후 해당 노드에 트랜잭션 요청,
 * 온도센서모듈로 부터 현재온도 획득
 * 트랜잭션 JSON 생성 후 API 요청, 브로드캐스트
 */
function transaction(){
        let rnd_node = Math.floor(Math.random() * port.length);
        while(busyport_idx == rnd_node){
                rnd_node = Math.floor(Math.random() * port.length);
        }
        const rndsender = uniqueNamesGenerator({dictionaries: [names], length: 1});
        const rndrecipient = uniqueNamesGenerator({dictionaries: [names], length: 1});
        let currtemp = temperature.read();
        console.log(url+port[rnd_node]+'/transaction/broadcast');
        const requestOption = {
                        'method': 'POST',
                        'url': url+port[rnd_node]+'/transaction/broadcast',
                        'headers': {'Content-Type': 'application/json'},
                        body: JSON.stringify({
                                "temperature": currtemp,
                                "sender": rndsender,
                                "recipient": rndrecipient
                        })
        };
        return requestOption;
}
```

```
  temperature: '18.9',
  sender: 'Becki',
  recipient: 'Cal',
  transactionId: 'ac031200faf411edbc08e1c5f03af41b'
},
{
  temperature: '18.9',
  sender: 'Cheri',
  recipient: 'Melva',
  transactionId: 'ad3b9200faf411edb06671207910d7e1'
},
{
  temperature: '18.9',
  sender: 'Kania',
  recipient: 'Gennifer',
  transactionId: 'b3e866f0faf411edb06671207910d7e1'
},
... 43 more items
```

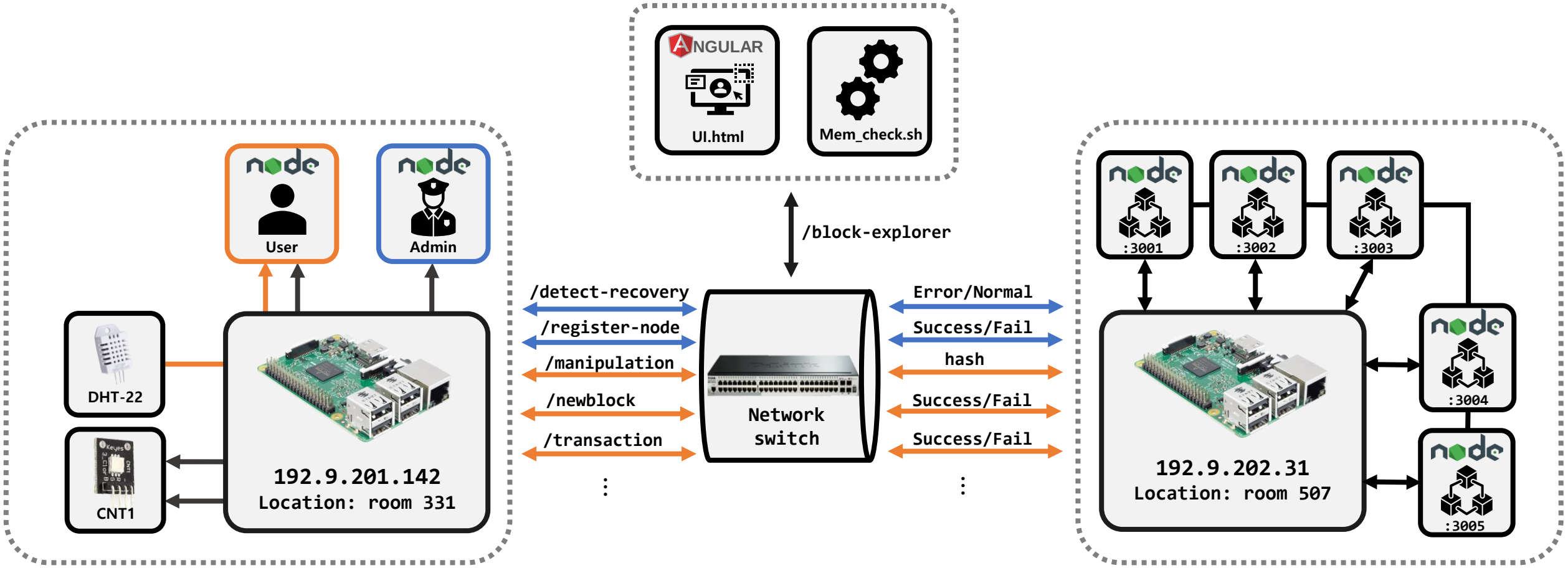
추가 변경사항

- Blockchain 네트워크와 User, Admin 모듈 분리
  - 2개의 Raspberry pi 사용
  - 변경이유: Out of memory 문제 해결



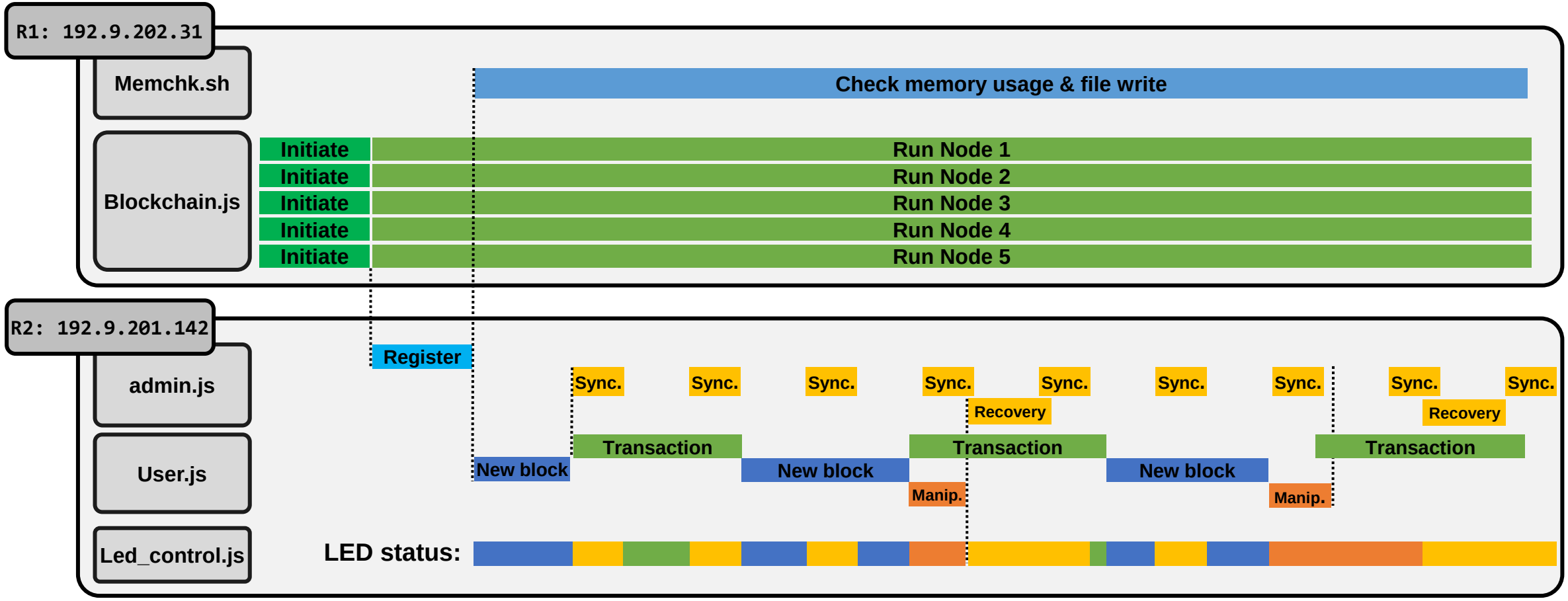
전체 시스템 구조

- 모니터링 도구 추가
  - 메모리 사용량 측정 스크립트 추가



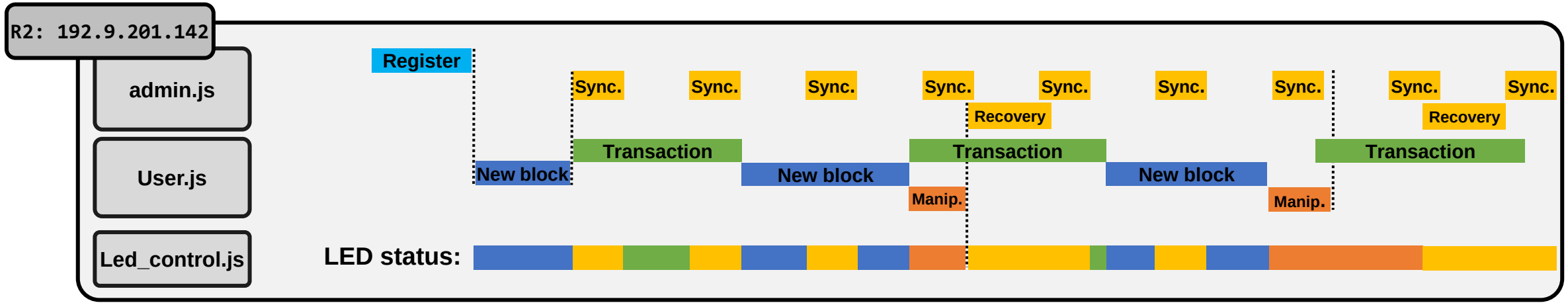
전체 시스템 동작흐름

- 각 구성요소 상호작용
  - R1: 서버, R2: 클라이언트
  - LED 우선순위: 1.데이터 조작발생(■) > 2.동기화, 복구(■) > 3. 블록생성(■) = 4. 트랜잭션(■)



전체 네트워크 동작흐름

- 세부설명
  - Admin.js 노드가 자동으로 블록체인 네트워크 등록
  - 네트워크 등록 후 User.js 실행 -> 제네시스 블록 생성
  - 제네시스 블록 생성이 종료된 후 네트워크 동기화(+감시) 수행
  - **Sync.**: 동기화 + 네트워크 무결성 감시, **2초마다** 수행, **User.js와 비동기적으로 실행됨,** 무결성 침해 탐지 시 복구
  - **Transaction**: 블록 당 **10번 요청**, 각 트랜잭션은 무작위로 노드에 온도데이터를 포함하여 **2초 마다 요청**
  - **New block**: 트랜잭션 10번 수행 후 요청, **무작위로 노드에 요청**, **블록 생성이 완료된 이후 트랜잭션 요청**
  - **Manipulate**: 블록 생성 이후 블록 **해시 값 조작**, 현재 **무작위로 노드를 선택**하고 **블록 중 무작위로 선택**하여 해시 값 변경



실험

- 블록생성
  - 블록 당 트랜잭션: 10
- 측정요소
  - 트랜잭션, 블록생성 성공률
  - 평균 트랜잭션 수행시간
  - 평균 블록생성 시간
  - 전체 수행시간
  - 메모리 사용량

실험결과

- 트랜잭션 성공률
  - Jmeter 사용
  - 사용자 수: 10
  - 전체 트랜잭션수행횟수:
  - 100(10*10), 1000(10*100), 10000(10*1000)
- 결과
  - 트랜잭션 성공률: 100%
  - 트랜잭션 평균 처리시간: 3ms , 11ms, 6ms

<Transaction: 100>

라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초	전송 KB/초
HTTP 요청	100	3	3	15	1.46	0.00%	102.6/sec	29.35	35.16
총계	100	3	3	15	1.46	0.00%	102.6/sec	29.35	35.16

<Transaction: 1000>

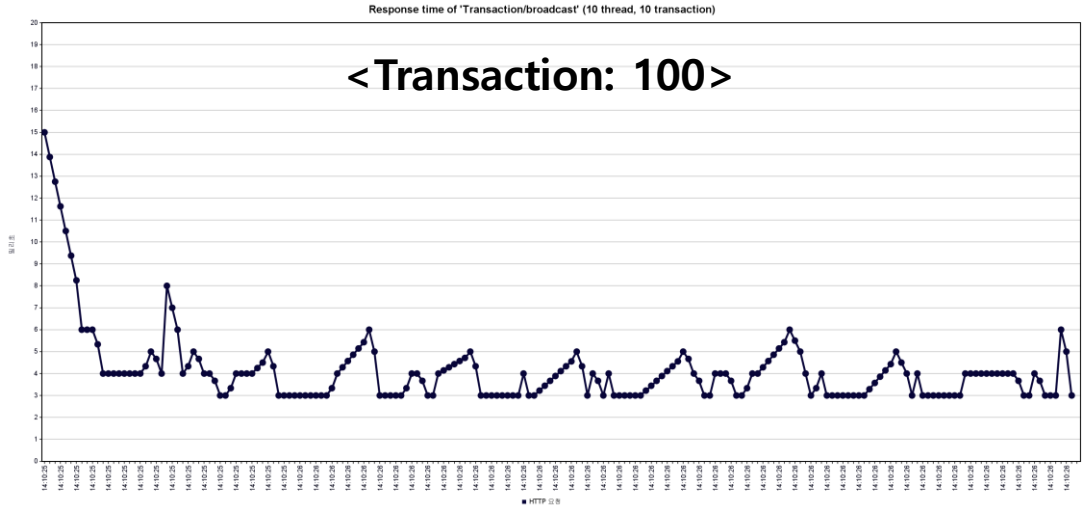
라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초	전송 KB/초
HTTP 요청	1000	11	2	216	13.42	0.00%	256.5/sec	73.39	87.91
총계	1000	11	2	216	13.42	0.00%	256.5/sec	73.39	87.91

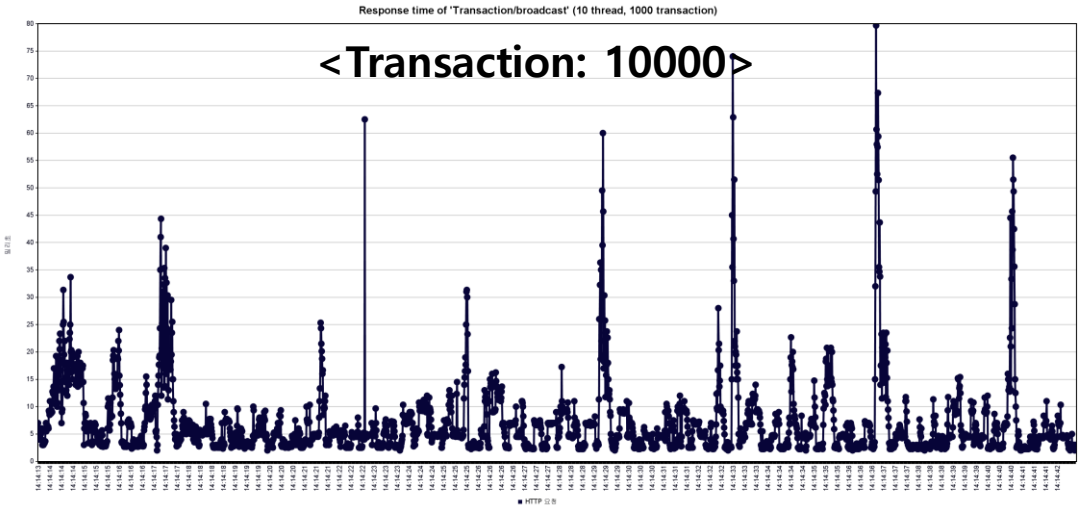
<Transaction: 10000>

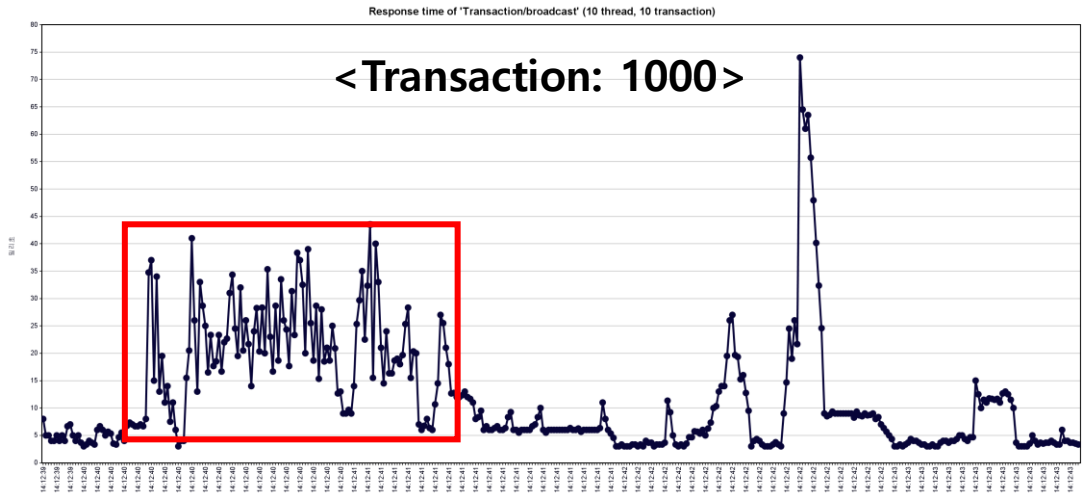
라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초	전송 KB/초
HTTP 요청	10000	6	2	235	6.66	0.00%	348.7/sec	99.76	119.51
총계	10000	6	2	235	6.66	0.00%	348.7/sec	99.76	119.51

실험결과

- 트랜잭션 응답시간 분포



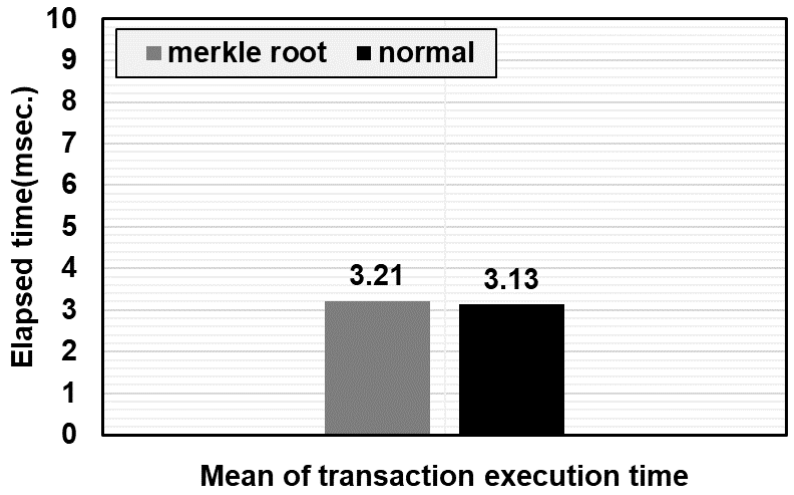




실험결과

- 트랜잭션 수행시간 비교
  - 100회 평균값 측정
  - 비교대상: 기존 코드(merkle tree 적용 전)

- 결과
  - **Normal** 이 더 빠른 것으로 측정됨(약 0.1ms)
  - 트랜잭션 수행에서는 merkle root 적용 시 코드 상 성능이득이 없음(거의 동일한 결과)



실험결과

- 블록생성 성공률
  - 사용자 수: 5
  - 3001, 3002, 3003, 3004, 3005 노드 당 1사용자(쓰레드) 할당
  - 트랜잭션 없이 연속적으로 블록생성 요청
  - 10, 50, 100
- 결과
  - 블록생성 성공률: 100%
  - 블록생성 평균 처리시간: 2.5sec , 2.3sec, 2.4sec

실험결과

- 로그
  - 분당 처리량이 약 5개로 비슷한 결과를 보임

<Block: 10>

라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초
genblock 3001	10	2154	118	6455	1952.92	0.00%	5.2/min	0.02
genblock 3002	10	3076	140	7592	2532.48	0.00%	5.0/min	0.02
genblock 3003	10	1732	248	6724	1825.76	0.00%	5.2/min	0.02
genblock 3004	10	2863	114	8939	2901.60	0.00%	5.3/min	0.02
genblock 3005	10	2989	343	6382	1828.43	0.00%	5.5/min	0.03
총계	50	2563	114	8939	2311.55	0.00%	23.4/min	0.11

<Block: 50>

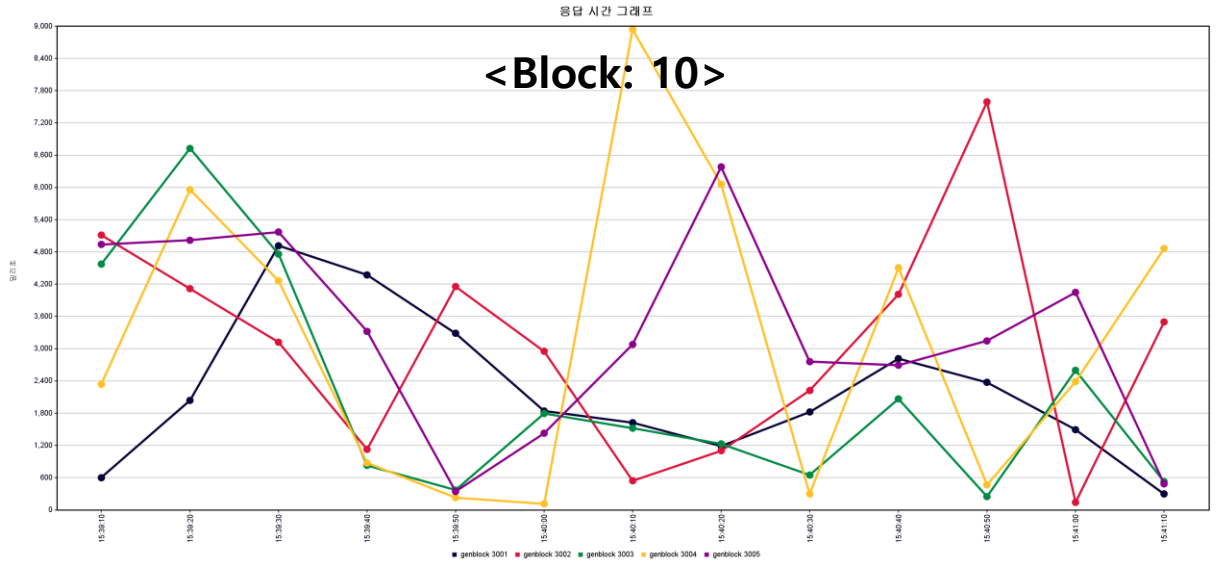
라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초
genblock 3001	50	2314	156	10193	2366.94	0.00%	5.0/min	0.02
genblock 3002	50	2593	211	7951	2208.82	0.00%	5.0/min	0.02
genblock 3003	50	2618	208	10327	2316.62	0.00%	5.1/min	0.02
genblock 3004	50	2494	86	8521	2095.45	0.00%	5.1/min	0.02
genblock 3005	50	2053	138	7331	1938.25	0.00%	5.1/min	0.02
총계	250	2414	86	10327	2200.72	0.00%	24.8/min	0.12

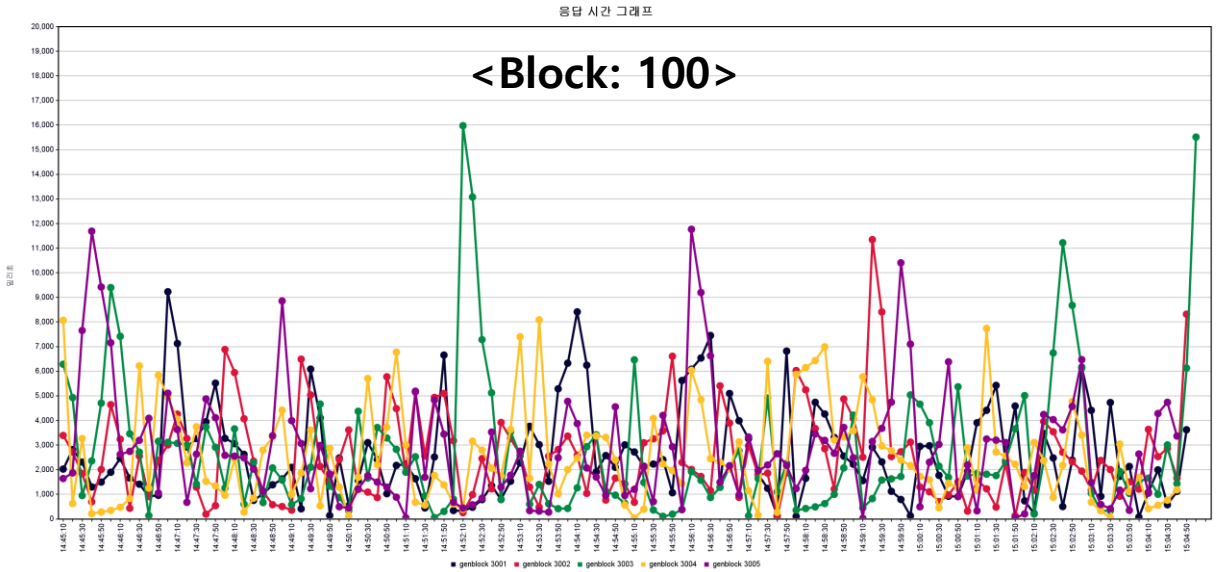
<Block: 1000>

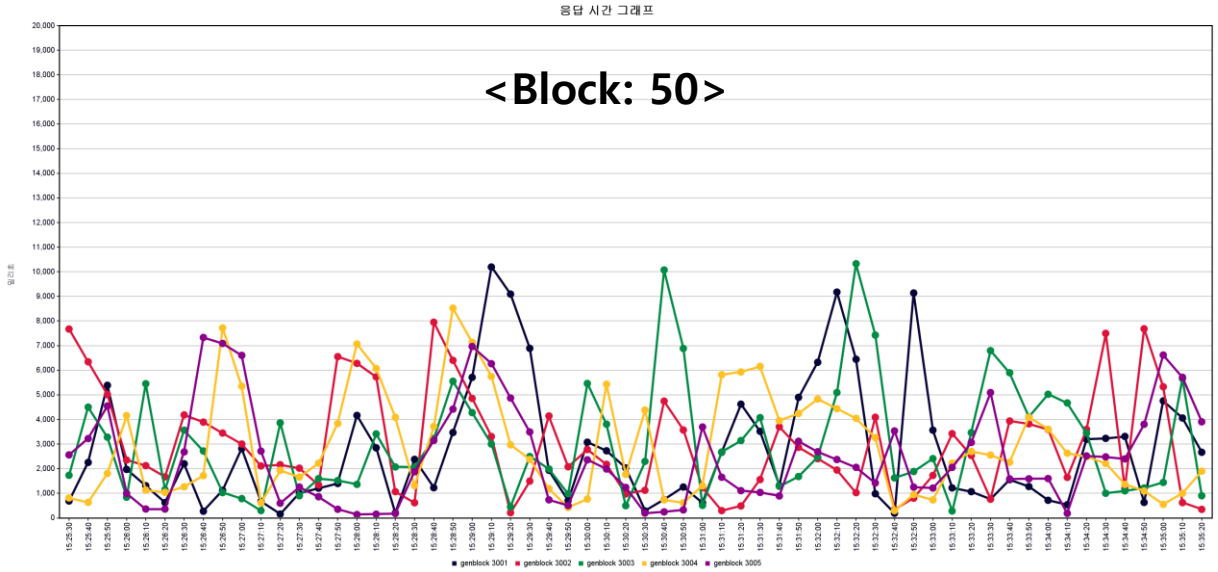
라벨	표본 수	평균	최소값	최대값	표준편차	오류 %	처리량	수신 KB/초
genblock 3001	100	2358	51	9228	1965.03	0.00%	5.1/min	0.02
genblock 3002	100	2453	50	11346	2042.01	0.00%	5.0/min	0.02
genblock 3003	100	2404	54	15974	2744.05	0.00%	5.0/min	0.02
genblock 3004	100	2375	38	11184	2158.86	0.00%	5.0/min	0.02
genblock 3005	100	2582	22	11766	2409.23	0.00%	5.0/min	0.02
총계	500	2435	22	15974	2282.91	0.00%	24.6/min	0.11

실험결과

- 블록생성 개수별 응답시간 분포

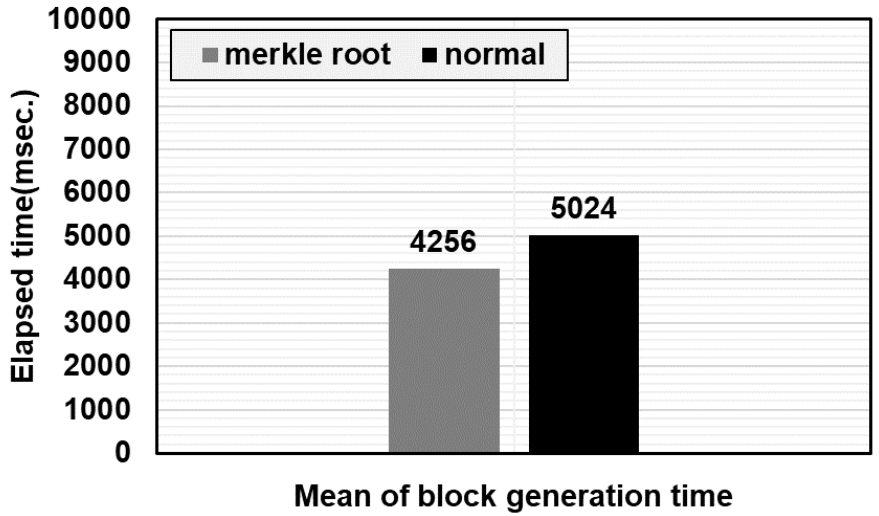






실험결과

- 평균 블록생성 시간비교
  - 트랜잭션 10회 요청 후 블록생성
  - 10회 평균값 측정
- 결과
  - 2초 ~ 8초 까지 다양한 시간분포를 보임(작업 증명과정)
  - **merkle root** 가 더 **15.2% 빠른** 것으로 측정됨
  - 블록생성 시 블록 해시 값 생성방식에서 **merkle root**가 더 빠른 것으로 보임



실험결과

- 평균 블록생성 코드비교

<merkle root>

```
// mine a block
app.get('/mine', function(req, res) {
    const lastBlock = bitcoin.getLastBlock();
    const previousBlockHash = lastBlock['hash'];
    const mktree = new MerkleTree();
    console.log(bitcoin.pendingTransactions);
    mktree.nodes = mktree.leaves = bitcoin.pendingTransactions.map(s => new MerkleNode(s));
    mktree.buildTree();
    const nonce = bitcoin.proofOfWork(previousBlockHash, mktree.getRoot());
    const blockHash = bitcoin.hashBlock(previousBlockHash, mktree.getRoot(), nonce);
    const newBlock = bitcoin.createNewBlock(nonce, previousBlockHash, blockHash, mktree);
    const requestPromises = [];
    bitcoin.networkNodes.forEach(networkNodeUrl => {
        const requestOptions = {
                    uri: networkNodeUrl + '/receive-new-block',
                    method: 'POST',
                    body: { newBlock: newBlock },
                    json: true
            };
            requestPromises.push(rp(requestOptions));
        });
```

Merkle tree 생성

루트 해시 값만 자격증명,
해시 값 생성에 사용됨

<normal>

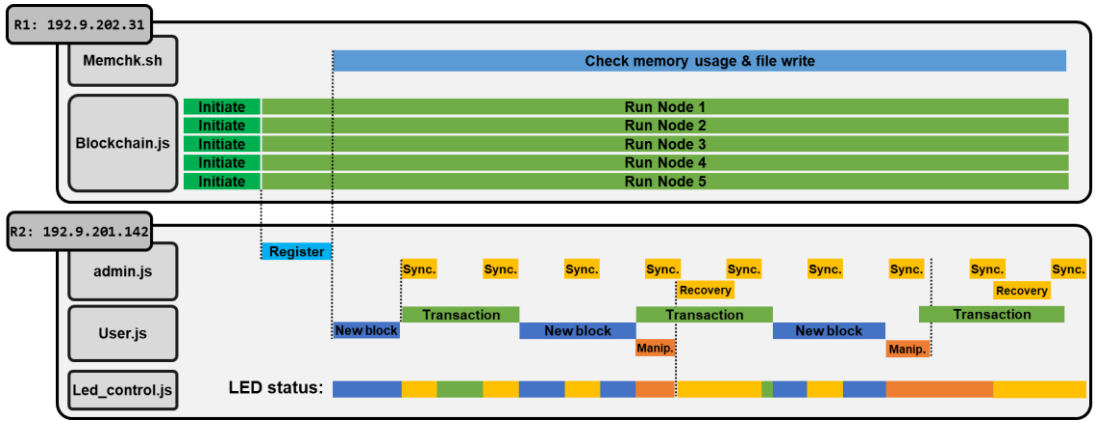
```
// mine a block
app.get('/mine', function(req, res) {
        const lastBlock = bitcoin.getLastBlock();
        const previousBlockHash = lastBlock['hash'];
        const currentBlockData = {
                transactions: bitcoin.pendingTransactions,
                index: lastBlock['index'] + 1
        };
        const nonce = bitcoin.proofOfWork(previousBlockHash, currentBlockData);
        const blockHash = bitcoin.hashBlock(previousBlockHash, currentBlockData, nonce);
        const newBlock = bitcoin.createNewBlock(nonce, previousBlockHash, blockHash);
        const requestPromises = [];
        bitcoin.networkNodes.forEach(networkNodeUrl => {
                const requestOptions = {
                        uri: networkNodeUrl + '/receive-new-block',
                        method: 'POST',
                        body: { newBlock: newBlock },
                        json: true
                };

                requestPromises.push(rp(requestOptions));
        });
```

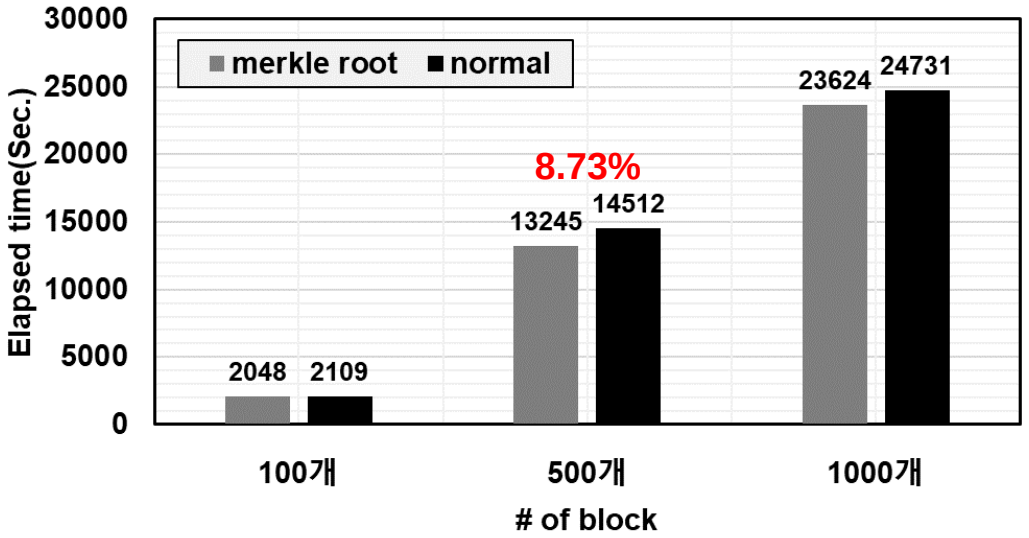
Transaction 리스트 생성

트랜잭션 리스트 전체를 사용하여 자격증명,
해시 값 생성

실험결과

- 시나리오 수행 시 실행시간 비교
  - 블록개수 100, 500, 1000개 생성
  - `console.time(),console.timeEnd()`를 사용하여 실행시간 측정
  - **normal:** 2109 초(35.1 분), 14512 초(241.8 분), 24731 초(412.1 분)
  - **merkle root:** 2048 초(34.1 분), 14512 초(220.7 분), 23624 초(393.7 분)
  - **merkle root**가 **최대 8.73%** 수행시간이 빠름
  - 블록 당 transaction 개수를 늘릴 경우 더 크게 차이가 날 것으로 예상됨





실험결과

- 메모리 사용량 측정
  - free 커맨드 사용, 쉘 스크립트 작성
  - 5초마다 한번씩 메모리 사용량을 파일에 기록

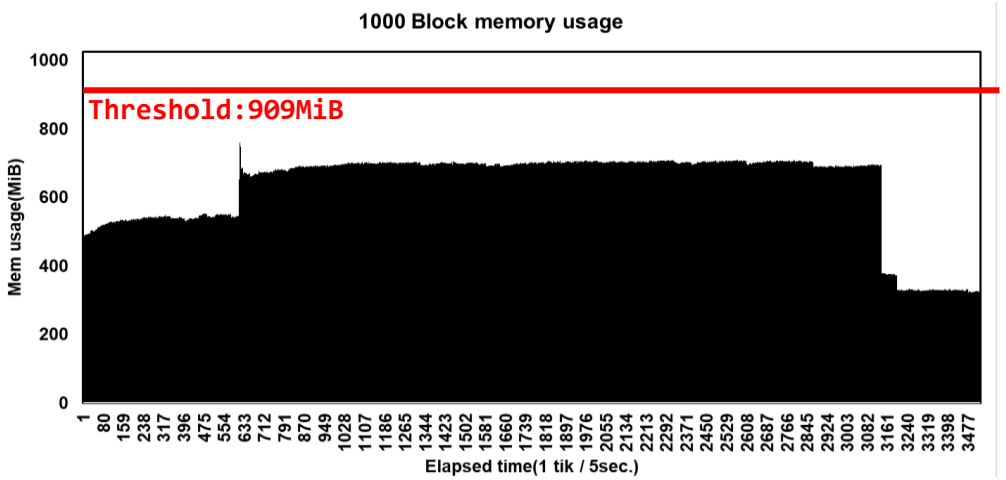
```
#!/bin/bash

while :
do
    free -m  | grep -n 'Mem' |&tee -a memlog.txt
    sleep 5
done

```

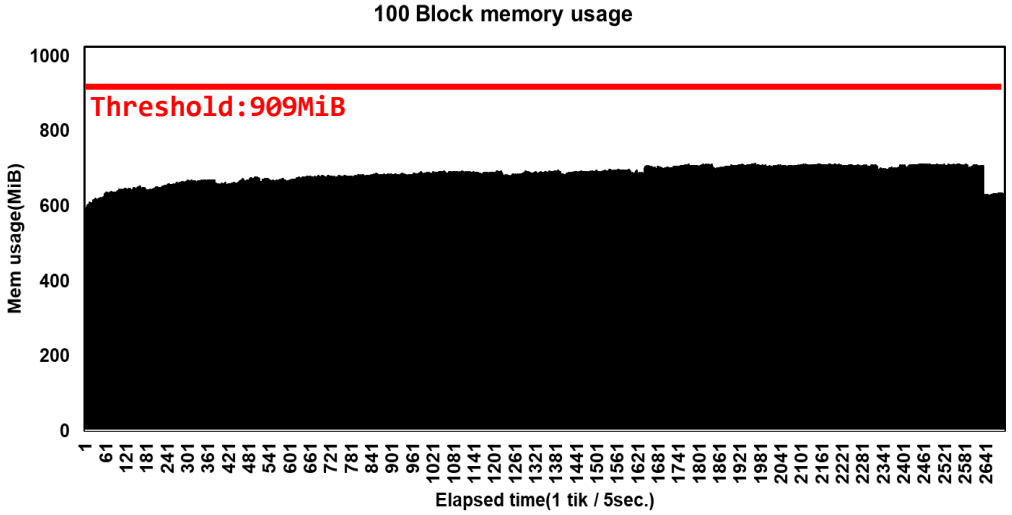
  - memlog.txt 일부

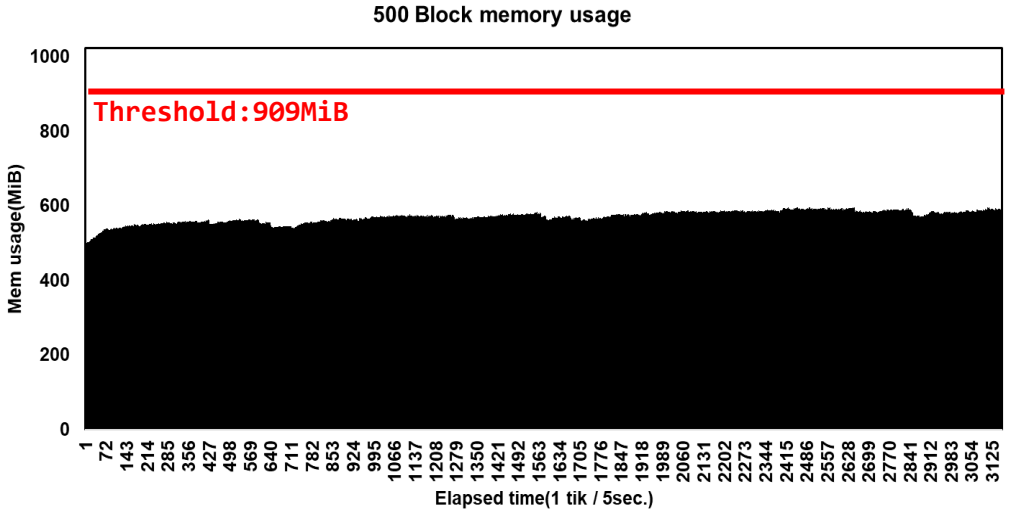
		Memory size	Mem. usage	Free mem.	Shared mem.	Buffer	
1	2:Mem:	909	482	193	0	234	371
2	2:Mem:	909	475	199	0	234	378
3	2:Mem:	909	476	198	0	234	377
4	2:Mem:	909	479	195	0	234	374
5	2:Mem:	909	482	192	0	234	371
6	2:Mem:	909	478	196	0	234	375
7	2:Mem:	909	480	195	0	234	373
8	2:Mem:	909	481	193	0	234	372
9	2:Mem:	909	480	194	0	234	373
10	2:Mem:	909	479	195	0	234	374
11	2:Mem:	909	478	196	0	234	375
12	2:Mem:	909	480	194	0	234	373
13	2:Mem:	909	482	192	0	234	371
14	2:Mem:	909	482	192	0	234	371
15	2:Mem:	909	484	190	0	234	369
16	2:Mem:	909	484	190	0	234	369
17	2:Mem:	909	483	191	0	234	370
18	2:Mem:	909	489	185	0	234	364

실험결과

- 메모리 사용량 측정
  - Block 개수
  - 100
  - 500
  - 1000
- 결과
  - 1000개 블록 생성 중 메모리 사용량이 줄어드는 것을 확인







QnA

- 감사합니다.