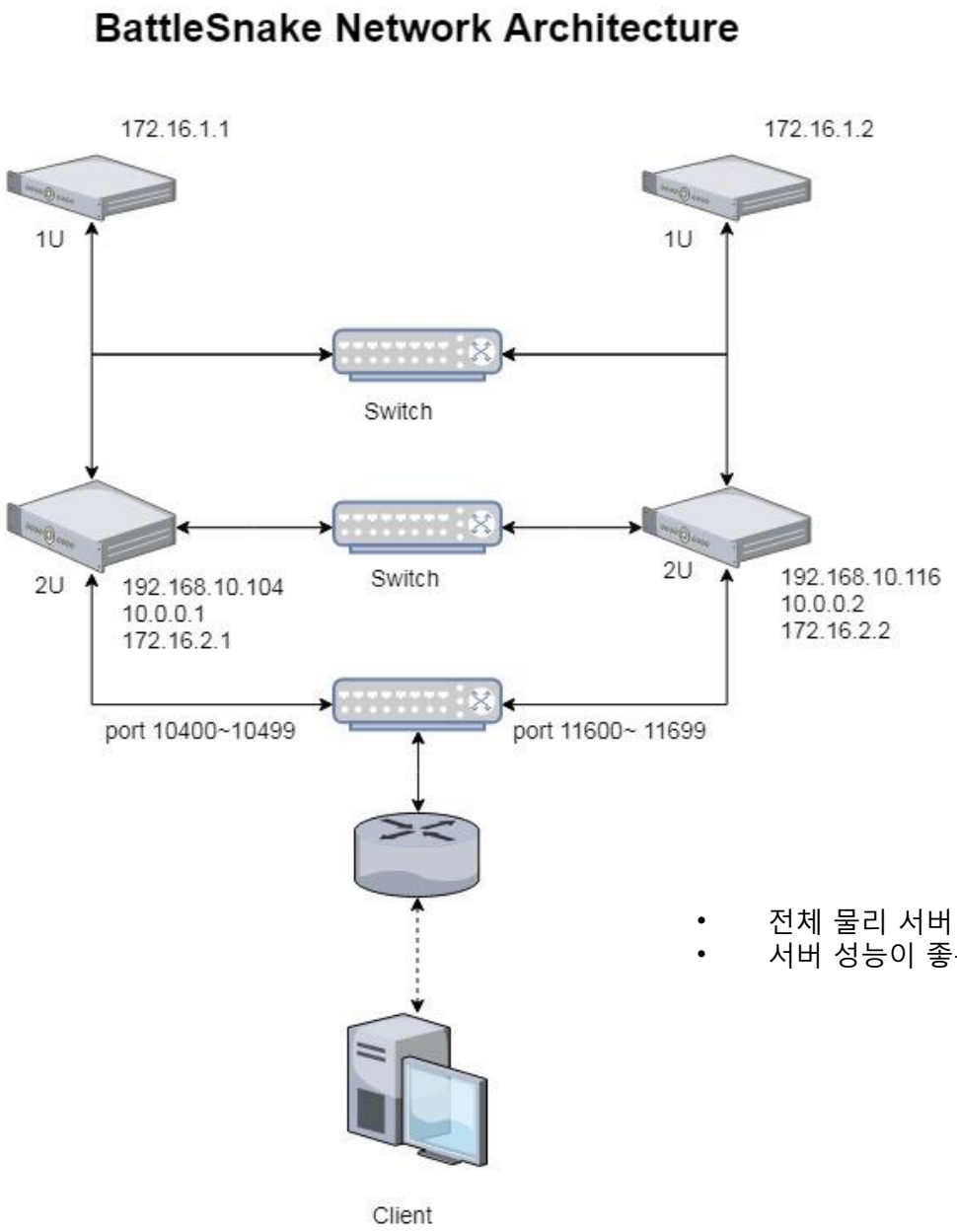
배틀 스네이크 서버

김윤지

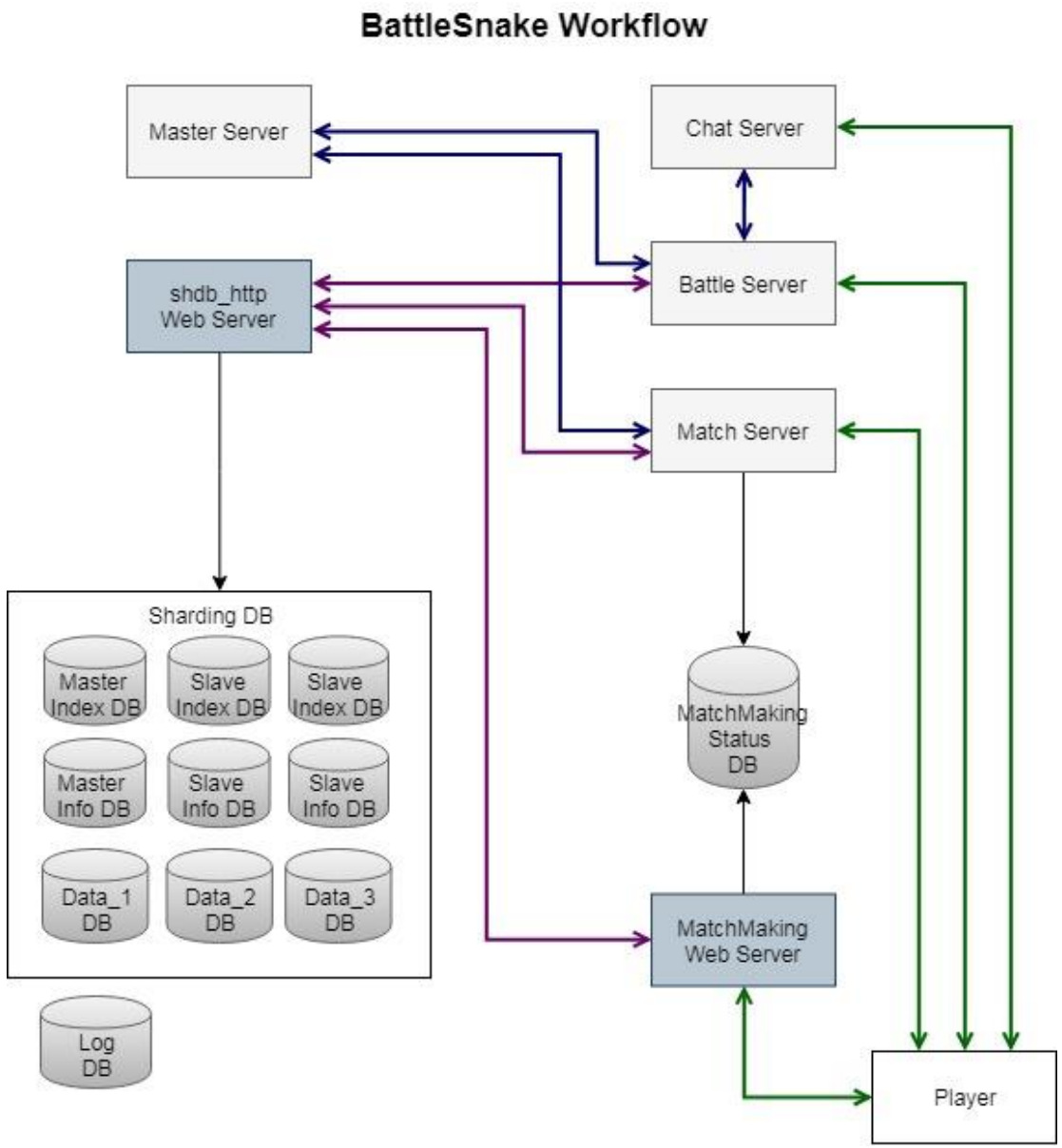
목차

1.1 서버 네트워크 구조

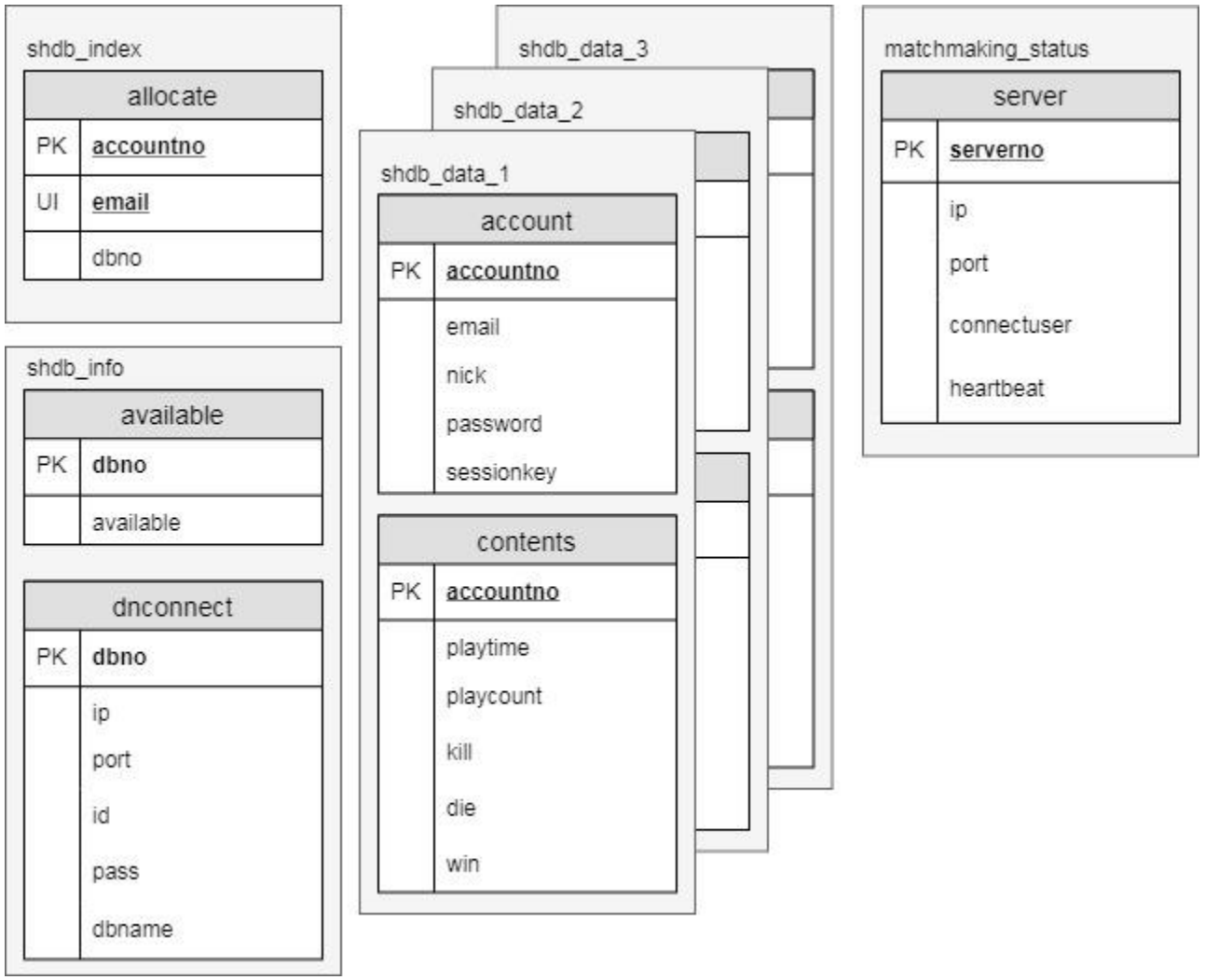


- 전체 물리 서버 총 4대
- 서버 성능이 좋은 2U끼리 다이렉트로 연결되어 있음

1.2 서버 워크플로우


BattleSnake Workflow
Master Server
Chat Server
Battle Server
shdb_http
Web Server
Match Server
Sharding DB
Master
Index DB
Slave
Index DB
Slave
Index DB
Master
Info DB
Slave
Info DB
Slave
Info DB
Data_1
DB
Data_2
DB
Data_3
DB
MatchMaking
Status
DB
MatchMaking
Web Server
Log
DB
Player

1.3 DB 다이어그램



shdb_index

- 사용자별로 어떤 db에 저장되는지 dbno를 보관하는 DB이며 쓰기보다는 읽기가 많으므로 Master 1대(2U), Slave 2대(1U, 1U)로 replication 분산

shdb_info

- dbno별로 연결 정보를 보관하고 여유분을 관리하며, Master 1대(2U), Slave 2대(1U, 1U)로 replication 분산

shdb_data

- 실제 데이터 저장소로 계정정보와 콘텐츠 정보를 저장, 원래는 샤딩되어야하나 물리 저장소가 부족하여 2U서버의 하나의 MySQL서버에 여러 개 만듦

Matchmaking_status

- 현재 연결된 매칭서버의 연결정보와 접속된 유저들을 저장

1.4 네트워크 라이브러리(1)

Network Library

IOCP WorkerThread x N

워커 스레드 하단에서는
해당 세션의 ioCount를 항상 1을 감소시킨다.
만약 0이되면 바로 Release를 하도록한다.

AcceptEx Thread

Accept가 완료되거나, 세션이 Relase 되었을때마다 해당 스레드를 CondtionVariable로 깨워서
세션 indexqueue에서 한개씩봅아 해당 세션의 소켓과 accp_ol을 AcceptEx에 등록한다.
현재 acceptex걸려있는 소켓풀이 설정해놓은 최대 acceptex 소켓풀보다 크다면 다시 잠든다.

Monitor Thread

send/recv TPS, cpu, 메모리 사용량, 이더넷 네트워크 트래픽등 모니터링에 필요한 요소들을
1초마다 추적한다.

st_CLIENT_INFO
clientID
sock
addr
accpt_ol
send_ol
recv_ol
bSend
bDisconnect
bLastPacket
sendPacketSize
ioCount
sendq
recvq
packetBuffer

clientID
각각 클라이언트마다 지니고 있는 고유 ID로 상위 6바이트는 고유 값 , 하위 2바이트는 해당 클라이언트가 저장되어있는
세션 배열의 index를 저장한다.

sock
해당 클라이언트 소켓 정보를 저장

addr
해당 클라이언트의 주소 정보를 저장

accpt_ol
해당 클라이언트의 AcceptEx용 Overlap을 저장하며, st_ACCEPT_OVERLAP으로 래핑되어 있다.
구조체 안에는 해당 클라이언트의 세션 배열의 index가 저장되어 있으며, 최초에 1번 설정된다.
이를 저장한 이유는 accept가 완료되었을때 overlap만으로 구별하기 위함.

send_ol / recv_ol
해당 클라이언트의 WSASend/WSaRecv Overlap

bSend
WSASend를 보내기전 설정하고, 완료통지가 올때 물어준다. 이는 패킷 전송을 한번씩만 하기 위함
WSASend를 여러번 호출하게 된다면, 커널 메모리 뿐만아니라 페이지 락이 많이 걸리기 때문에 자원낭비를 막고자 함

bDisconnect
해당 세션이 Release 중이면 켜지는 플래그로, 외부에서 이미 종료중인 세션을 다시한번 종료할 수 있으므로 플래그를 두었음

sendPacketSize
직전에 보낸 패킷의 개수

ioCount
현재 걸려있는 I/O 개수이며, 항상 1이상을 유지하고 있어야하며, ioCount가 0이되는 순간 바로 Relase가 된다.

sendq
리스트형 락프리큐이며, SRWLock이나 CriticalSection을 쓰지 않아도 바로 Enqueue, Dequeue 할 수 있다.
InterlockedComparedExchage를 사용하였음.

recvq
워커스레드 안에서만 사용되기 때문에 일반 원형 큐를 사용하였음.

가상함수 이벤트 핸들러

OnClientJoin	새로운 세션이 Accept되었을때 이벤트 핸들러로 알림
OnClientLeave	세션이 연결이 종료되었을때 이벤트 핸들러로 알림
OnConnectionRequest	Accept 된 소켓에 대해여 새로운 세션으로 등록하기 전에 해당 세션의 IP와 Port를 알림. 유저 필터용
OnRecv	Recv 받은 개별 패킷마다 Recv_Complete 함수에서 호출하며 상속자에게 이벤트 핸들러로 알림
OnSend	보낸 패킷의 길이를 이벤트 핸들러로 알림
OnError	에러 발생시 이벤트 핸들러로 알림

start — 네트워크, 세션 초기화, 워커스레드, AcceptEx 스레드, 모니터 스레드를 실행한다

stop — 리슨 소켓 종료, 스레드 정리, 세션 배열 정리

disconnect — 워커 스레드 외부에서 세션의 ClientID를 기반으로 연결을 끊도록 동기화가 잡혀있으며,
해당 ClientID를 가진 세션이 존재할 시 shutdown을 호출한다.

sendPacket — 워커 스레드 외부에서 세션의 ClientID를 기반으로 패킷 전송가능 하도록
동기화가 잡혀있으며, 해당 ClientID가 존재하지 않거나 이미 Disconnect중이라면 무시한다.
해당 유저의 락프리 send 큐에 인자로 받은 패킷포인터를 Enqueue하고, send_post를 호출한다.
* 유저-서버간 통신에는 이 부분에서 패킷이 인코딩 된다.

accept_post — 해당 세션의 ioCount를 1을 증가시키고 AcceptEx를 호출한다. 실패시
ioCount를 1을 감소시키고, 바로 Release를 하도록 한다.

recv_post — 세션마다 가지고 있는 recvq 버퍼의 가용 사이즈만큼 WSABUF에 포인터와 함께 길이를 저장한다.
원형큐이기 때문에 WSABUF는 최대 2개까지 걸어 놓을 수 있다.
현재 ioCount를 1을 증가시키고 WSARecv 호출하며 실패시 ioCount를 1을 감소시킨다.
만약 ioCount가 0이라면 Release

send_post — 세션마다 가지고 있는 sendq에서 개별 패킷포인터를 한개씩 봅아서
해당 패킷의 포인터와, 패킷의 길이를 WSABUF 배열에 저장한다.
봅은 패킷의 packetBuffer에 저장하여 후에 Send_Complete 통지가 올때
패킷을 해제한다.
최대 100개의 패킷을 보낼 수 있으며, 보내기전 ioCount를 1을 증가시키고
WSASend를 호출하여 실패시 ioCount를 1을 감소시킨다.
만약 ioCount가 0이라면 그대로 Release 한다.

accept_complete — 워커스레드로 AcceptEx 완료가 되면, 인자로 들어온 overlap을 st_ACCEPT_OVERLAP으로
캐스팅하여 해당 구조체에 저장되어 있는 arrayIndex로 세션배열에서 해당 유저를 찾는다.
해당 유저의 ClientID를 부여하고, IOCP에 세션 포인터를 등록, OnClientJoin으로 상속자에게
새로운 세션이 연결되었음을 알린다.

recv_complete — 해당 세션의 Recvq 에서 인자로 받은 byte길이만큼 recvq의 rear의 위치를 옮긴 후,
헤더 사이즈만큼 패킷이 왔는지 확인 후, 패킷이 왔다면 헤더에 포함된 패킷의 데이터 길이만큼
왔는지 확인, 개별 패킷이 온전하게 도착했다면 recvq에서 봅은 후 패킷에 담아 디코딩 후
OnRecv를 호출하여 상속자에게 알린다. recvq가 비거나, 다음패킷이 온전하게 도착하지 않았을때까지
루프를 돌며 모두 봅은 후 다시 Recv_post를 호출한다.

send_complete — 직전에 보냈던 패킷을 저장해놓은 packetbuffer의 개별 패킷들을 Free해주고,
만약 보내고 끊기를 설정해놓은 세션이라면, shutdown을 호출하고, 아니면
Send_Post를 호출한다.

clientIDtoIndex — ClientID에서 하위 2바이트만 마스킹하여 해당 세션의 배열 번호를 반환

createClientID — 1씩 증가하는 고유 번호와 해당 세션의 배열 번호를 조합하여 ClientID를 생성

ReleaseSession — 현재 ioCount가 0인지 확인 후, 아니면 리턴하며 bDisconnect 플래그를 키면서
세션 초기화를 진행한다.

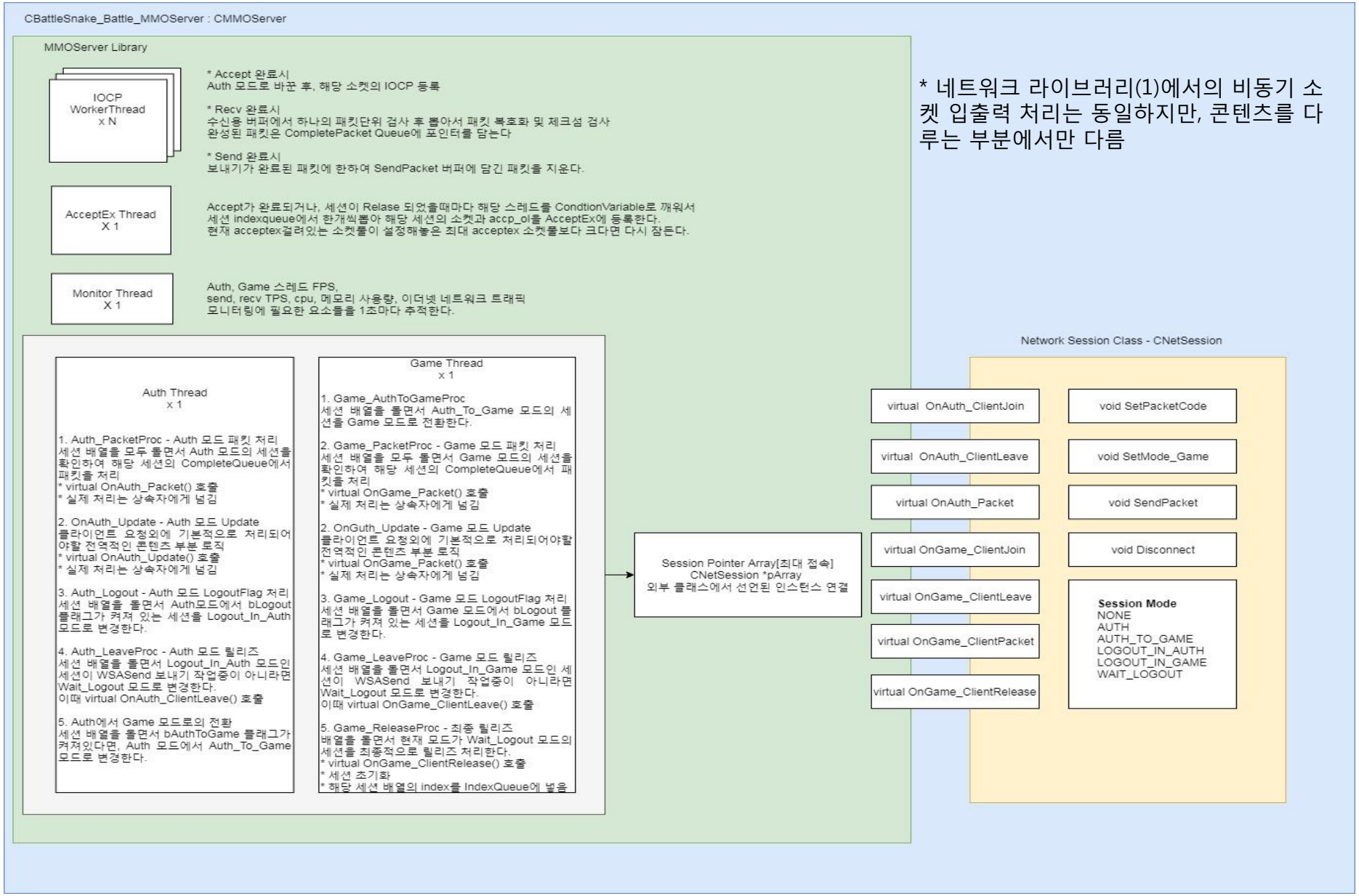
AcquireSessionLock — 외부에서 보내는 패킷의 경우에는 ClientID만 알고있기 때문에, 해당 클라이언트의 ID의 유저가
존재하는지, ioCount가 0인지, bDisconnect 플래그가 켜져있는지 확인이 필요하며,
ioCount를 1을 올려놓는다. 이는 보내는 중에 세션이 종료됨을 막기 위함

ReleaseSessionLock — ioCount를 1을 감소시키고, 만약 ioCount가 0이라면 Relase를 진행한다.

InitializeSession — 해당 세션의 ClientID, SendQ, RecvQ, packetBuffer등을 초기화한다.

- 모든 Application 서버에 사용
- 서버간 통신, 서버와 유저 통신용으로 구분됨
- 받은 패킷들은 가상함수 이벤트 핸들러에 ClientID를 함께 넘겨서 처리

1.4 네트워크 라이브러리(2)



* 네트워크 라이브러리(1)에서의 비동기 소켓 입출력 처리는 동일하지만, 콘텐츠를 다루는 부분에서만 다름

- 배틀서버 전용 네트워크 라이브러리
- 네트워크 세션과 컨텐츠 세션의 구분이 하나로 되어있음
- Auth 스레드와 Game 스레드를 분리하여 MMO 게임 구조에 적합
- 각 모드간에는 동기화 로직 필요하지 않음

1.5 shdb_http

create.php
- 이메일을 기준으로 새로운 accountno와 dbno를 할당.
- 할당 받은 db의 accountno/contents 테이블에 해당 accountno로 insert하여 공간을 확보
- Request {"email": "test@abcd.com"}
- Response {"result":1, "accountno" :1, "email" :"test@abcd.com", "dbno" : 1}

select_account.php
- accountno 또는 email을 기준으로 유저의 계정 데이터를 얻음
- Request {"email": "test@abcd.com" 또는 "accountno" :1 }
- Response {"result":1, "accountno" :1, "email" :"test@abcd.com", "password" : "abcd", "sessionkey" : "abcd", "nick": "abcd"}

select_contents.php
- accountno 또는 email을 기준으로 유저의 콘텐츠 데이터를 얻음
- Request {"email": "test@abcd.com" 또는 "accountno" :1 }
- Response {"result":1, "accountno" :1, "playtime" : 1, "playcount" : 1, "kill" : 3, "die" : 1, "win" : 1}

update_account.php
- accountno 또는 email을 기준으로 입력된 account를 저장
- Request {"email": "test@abcd.com" 또는 "accountno" :1, "password" : "abcd", "sessionkey" : "avdc", "nick": "abcd" }
- Response{"result" : 1}

update_contents.php
- accountno 또는 email을 기준으로 입력된 account를 저장
- Request {"email": "test@abcd.com" 또는 "accountno" :1, "playtime" : 1, "playcount" : 1, "kill" : 3, "die" : 1, "win" : 1}
- Response{"result" : 1}

1.6 MatchMaking

get_matchmaking.php

- 유저의 accountno를 기반으로 shdb_http를 호출하여 해당 유저의 계정 정보를 가져온다 (accountno, email, password, sessionkey, nick)
- Matchmaking_status Slave DB에 연결하여 현재 연결 중인 매칭 서버 중 가장 연결 인원 수가 적은 컬럼을 가져온다.
- Request {"email": "test@abcd.com" 또는 "accountno" : 1}
- Response {"result":1, "serverno" :1, "ip" :"111.111.111.111", "port" : 10000}

check_session.php

- 로그인 후에는 로그인 정보를 클라이언트에 저장하고 accountno와 sessionkey를 보관하여 자동 로그인이 되도록 함
- 같은 세션키를 사용하면 불안하기 때문에 재인증시마다 새로운 세션키를 발급받아 사용하도록 한다.
- 유저의 accountno를 기반으로 shdb_http를 호출하여 해당 유저의 계정 정보를 가져온다 (accountno, email, password, sessionkey, nick)
- 새로운 세션키를 발급 후 shdb_http를 호출하여 해당 계정 정보를 update
- shdb_http에서 콘텐츠 정보를 요청
- Request {"email": "test@abcd.com" 또는 "accountno" : 1, "sessionkey" : "abdc"}
- Response {"result":1, "sessionkey" :"abcd", "nick" : "abdc" , "playtime" : 1, "playcount" : 1, "kill" : 3, "die" : 1, "win" : 1}

2.3 서버간 통신

마스터 서버 프로세스

- 마스터 서버는 배틀/매칭 서버 목록을 관리
- 마스터 서버는 각 유저를 관리하기 위해 고유값으로 매칭서버에서 생성한 ClientKey와 AccountNo를 사용한다.
- 매칭서버로부터 유저의 입장 가능한 대기방 요청이 오면 입장 가능한 방을 찾아 해당 방이 속해있는 배틀, 채팅 서버 정보를 즉각 결과를 알려준다.
- 매칭서버로부터 유저의 방입장 성공 또는 실패 결과를 알려주며 이에 따라 관리하고 있는 대기방의 참여인원수를 조절한다.
- 방이 생성/파괴 될때마다 요청한 배틀 서버의 ServerNo와 함께 받은 RoomNo로 방을 관리
- 배틀서버로부터 대기방에서 유저가 나갔다는 패킷을 받으면, 대기방의 참여인원수를 되돌린다. 이는 사용자가 들어가는 부분은 마스터가 카운팅이 가능하지 만, 배틀서버에 들어갔다가 나온 유저는 마스터 서버에서 관리하기 어렵기 때문

매칭 서버 프로세스

- 매칭서버는 마스터 서버 로그인 시 매칭서버의 연결정보를 알림
- 매칭서버는 유저가 SessionKey를 기반으로 로그인 요청하면, shdb_http 에서 조회하여 인증 한 후 클라이언트의 고유 키를 생성(ClientKey)
  - ClientKey의 용도는 마스터 서버에서 클라이언트를 구분하는 값으로 매치메이킹 서버가 마스터서버에게 방생성 요청, 입장 확인시 사용됨
- 유저로부터 방 정보 요청을 받으면 마스터 서버에게서 배틀서버 , 채팅서버, 방정보를 받아 클라이언트에게 전달한다.
- 클라이언트로부터 방입장 성공 패킷을 받으면 마스터에게 전달한다. 만약 성공패킷을 받지 못한상태에서 클라이언트가 연결을 끊게 되면 마스터 서버에게 입장 실패 패킷을 마스터 서버에게 전달
- 매칭서버는 일정 시간 및 현재 세션 수 변화량에 따라 MatchMakingStatus DB에 업데이트

배틀 서버 프로세스

- 배틀 서버는 마스터 서버 로그인 시 연결되어 있는 채팅서버와, 서버 정보를 알림
- 배틀 서버는 유저 로그인/로그아웃 때마다 shdb_http 서버에 세션 조회 및 콘텐츠 업데이트를 요청
- 배틀 서버는 항상 일정한 대기방 수를 유지하며 생성될때마다 배틀서버의 ServerNo, RoomNo, MaxUser등을 마스터서버에게 알림
- 방에 유저가 MaxUser만큼 다 차게 된다면 마스터에게 방이 닫힘을 알림.
- 배틀 서버는 마스터 서버와 채팅 서버에게 항상 대기방 생성/파괴를 알림
- 배틀 서버는 유저가 대기방에서 나갈때마다 해당유저의 AccountNo와 RoomNo로 현재 방 인원감소를 마스터 서버에게 알림
  - 클라이언트는 배틀서버의 방에 입장 확인 패킷을 받기 전까지 매칭, 배틀서버와의 연결을 유지하기 때문에 사용자가 방 배정을 받은 후 매칭/배틀에 연결된 상태에서 배틀에 방입장 요청 한뒤에 바로 연결을 끊어버리면 매칭->마스터 방입장 실패, 배틀->마스터 방나감 패킷을 둘다 보낼 수 있음 그래서 AccountNo를 포함

채팅 서버 프로세스

- 채팅서버는 배틀서버에 로그인 시 채팅서버의 연결정보를 알림
- 채팅서버는 배틀서버로부터 방 생성/파괴 요청에 따라 채팅방 관리

2.3 서버와 유저 통신

Type	Detail
Match	• 매칭서버에 연결 후 입장 가능한 방을 요청, 응답받은 배틀/채팅/방정보를 기반으로 배틀/채팅 서버에 연결 • 배틀 서버 연결에 성공하면 매칭서버와의 연결 종료
Battle	• 대기방의 인원수가 차면 준비방 전환 알림 • 준비방에서 카운트 다운 후 플레이방으로 전환 알림 • 일정 시간마다 레드존 활성화 알림/경고 방인원 전체에게 알림 • 살아 있는 유저가 1명이 될때 방인원 전체에게 결과 알림 • 레드존에 의해 데미지를 받은 유저 정보를 방인원 전체에게 알림 • 매칭 서버로부터 받은 방정보를 기반으로 배틀/채팅서버 연결 및 입장 • 방 입장 성공 시 내 정보를 방인원 전체에게 알림 • 방 입장 성공 시 방 인원 전체 정보를 나에게 알림 • 방 퇴장 시 방 인원 전체에게 나갔음을 알림 • 이동/타격 패킷 요청시마다 처리 결과를 해당 유저를 제외한 방인원 전체에게 알림 • 발사/타점 패킷 요청시마다 해당 유저에게 그대로 응답
Chat	• 채팅 메시지를 보내면 속해있는 방 전체에게 그대로 브로드 캐스팅

3.1 테스트 환경

Dummy
- Session : 3000
- 방 최대 입장 인원 : 10
- Login Delay : 1000ms (db 반영시간)
- Request Delay : 5ms
- 테스트 기간 2일

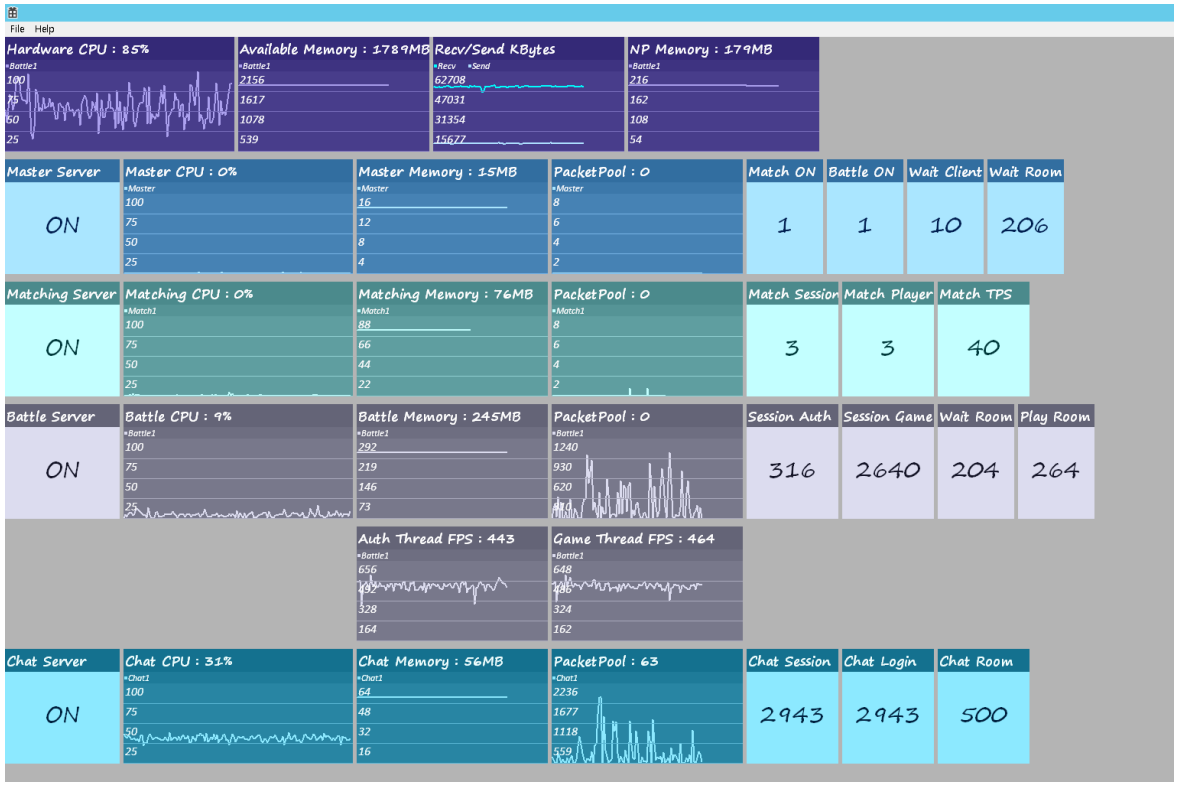
Server
- 2U 배틀/shDB/MatchMakingStatusDB/Dummy
- 2U 채팅/DataDB/로그DB
- 1U 마스터/매칭
- 1U 모니터링서버/모니터링 클라이언트

3.2 테스트 결과

```
StartTime : 2018.9.20 01:46:17
-----------------------------------------
Test Mode : Lobby & Matchmaking & Battle & Chat Contents Test
Play ( S Key Stop )
-----------------------------------------
    Client Number : 100 thread x 30 client = 3000
------------------------------------------------ Lobby Part
   AI Bot CheckIn : 0
------------------------------------------------ Lobby Part
         in Lobby : 6 Clients
     Request TPS  : 30
   Response Wait  : 6
          Delay   : Max 2532 ms / Avr 104 ms
   Session Error  : 0
      HTTP Error  : 1318
 MatchServer None : 0
------------------------------------------------ Matchmaking Part
    Connect Total : 4256282
   in Matchmaking : 2 Clients
     Login TPS    : 22
     Login Delay  : Max 16437 ms / Avr 44 ms
     Login Fail   : 0
  Room Req TPS    : 22
  Room Req Delay  : Max 7562 ms / Avr 4 ms
  Room Req Fail   : 0
   Response Wait  : 0
  Timeout Player  : 0
 Error Disconnect : 0
------------------------------------------------ Battle Part
    Connect Total : 4256282
       in Battle  : 2962 Clients
     Login TPS    : 24
     Login Delay  : Max 145031 ms / Avr 71 ms
     Login Fail   : 0
 Room Enter TPS   : 28
 Room Enter Delay : Max 3032 ms / Avr 17 ms
 Room Enter Fail  : 0
    Ready Client  : 0
 Countdown Client : 250
     Play Client  : 2710
   Response Wait  : 1
  Timeout Player  : 0
 Error Disconnect : 0
------------------------------------------------ Chat Part
    Connect Total : 4256282
         in Chat  : 2962 Clients
     Login TPS    : 28
     Login Delay  : Max 5609 ms / Avr 41 ms
     Login Fail   : 0
 Room Enter TPS   : 30
 Room Enter Delay : Max 2890 ms / Avr 42 ms
 Room Enter Fail  : 0
     Play Client  : 2960
     Message TPS  : 26098
   Message Delay  : Max 9313 ms / Avr 97 ms
   Response Wait  : 2751
  Timeout Player  : 0
 Error Disconnect : 0

CPU usage [T:57.0% U:27.3% K:29.7%] [Stress:30.9% U:16.5% K:14.5%]
```

더미 클라이언트



모니터 클라이언트

3.2 시연 영상

배틀 스네이크 시연 URL

3.3 시행착오

- shdb_http, matchmaking 웹서버 스트레스 테스트를 했을 때 time_wait이 너무 많이 남아 포트 고갈 문제가 생겼었음. 이는 서버의 레지스트리 tcptimedwaitdelay 수치를 낮춤으로써 해결

- 배틀/채팅/매칭 모든 서버를 하나의 컴퓨터에서 실행 시켰을 때, 서버 자체의 트래픽, io 과부하로 인해 더미 클라이언트 또는 서버가 응답을 받지 못하는 상황이 발생, 이는 배틀 서버와 채팅 서버를 서로 다른 2U에 분산시키고, 1U에 마스터와 매칭을 둠으로써 해결