1 Bootstrap

Bootstrap server radi na fiksnom portu koji je definisan u konfiguracionoj datoteci. On može da primi dve vrste poruka:

1. Oblik poruke: Hail\n<listener\_port-ipadresa>\n
   1. Oblik odgovora: <listener\_port-ipadresa >\n
2. New\n<listener\_port-ipadresa>\n
   1. Nema odgovora.
3. Exit
   1. Nema odgovora.

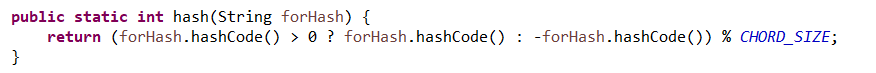
-Prva poruka se koristi kada se čvor prvi put startuje. Poruka sadrži tekst “Hail” u prvom redu, i u drugom redu sadrži jedan int, koji predstavlja port na kojem sluša novi čvor. Odgovor na ovu poruku treba da bude jedan red teksta u kojem se nalazi -1 ako nema aktivnih čvorova, ili port od nekog nasumično izabranog aktivnog čvora. Odgovor se šalje preko istog soketa pre kojeg je stigla “Hail” poruka.

-Druga poruka se koristi kada Chord sistem utvrdi da novi čvor ima dobar ID, tj. da se njegov ID ne poklapa sa nekim drugim aktivnim čvorom. Kao reakciju na drugu poruku Bootstrap server dodaje port čvora u svoju listu portova aktivnih čvorova i ne šalje nikakav odgovor.

- Treća poruka se koristi kada čvor želi da izađe iz sistema. Poruka sadrži tekst “Exit” u prvom redu, a drugom se šalje port-ipadresa čvora koji želi da se ugasi. BootstrapServer briše informacije o čvoru iz svojih listi.

2 Uključivanje čvora u sistem

Chord ID čvora se određuje na osnovu funkcije heširanja koja ima sledeći oblik:



Gde je:

* forHash - string koji se hešira (broj porta i IP adresa na kojem čvor sluša).
* CHORD\_SIZE - maksimalna vrednost za Chord ID.

Kada čvor želi da se uključi u sistem, prvo treba da se javi čvoru kojeg je dobio od Bootstrap servera sa NEW\_NODE porukom.

Čvor koji primi NEW\_NODE poruku prvo treba da proveri da li je novi čvor u koliziji sa nekim već aktivnim čvorom. Ako je to slučaj, novom čvoru se šalje SORRY poruka. Ako ovo nije slučaj, onda postoje dve opcije:

1. Ako je trenutni čvor direktan sledbenik čvora koji je poslao NEW\_NODE, onda šalje WELCOME poruku novom čvoru, u kojoj će se nalaziti i podaci koje taj novi čvor preuzima i lista objekata koji predstavlja sve potrebne informacije o fajlovima koji treba da se upišu u njegovo skladište.
2. Ako trenutni čvor nije direktan sledbenik čvora koji je poslao NEW\_NODE, onda se identična NEW\_NODE poruka šalje čvoru koji je najbliži novom čvoru po Chord pretrazi.

Novi čvor će kad-tad dobiti WELCOME poruku od svog direktnog sledbenika kada sve fajlove upisuje u svoje skladište. Nakon toga ulazi u drugu fazu inicijalizacije, koja se sastoji od osvežavanje tabela sledbenika za ceo sistem. Ovo se obavlja pomoću UPDATE poruke koju novi čvor šalje svom direktnom sledbeniku. Svaki čvor koji primi ovu poruku je šalje dalje svom prvom sledbeniku, i ažurira svoju tabelu sledbenika. Ovo se ponavlja dok UPDATE poruka ne napravi ceo krug oko sistema i vrati se nazad do čvora koji ju je isprva poslao. UPDATE poruka u svom tekstu treba da sadrži brojeve slušalačkih portova od svih čvorova koji su je primili i prosledili dalje, tako da na kraju ciklusa čvor koji ju je isprva poslao može da dobije informaciju koji su slušalački portovi od svih trenutno aktivnih čvorova. Tekst UPDATE poruke je formatiran kao string koji sadrži brojeve slušalačkih portova razdvojene zarezom.

3 Operacije

3.1 **pull name [version]**

Ako se traženi ključ nalazi na trenutnom čvoru, preuzima fajl iz skladišta i prepisuje ga u radni koren. Ukoliko je navedena verzija, ćvor će preuzeti fajl odgovarajuće verzije, a ukoliko nije navedena, preuzima se fajl sa najvećom verzijom. Ukoliko fajl nije pronađen, ispisuje se poruka da fajl ne postoji, ili je izbrisan. Ako to nije slučaj, šalje se PULL poruka ka narednom čvoru, izabranom na osnovu Chord pretrage. PULL poruka u svom telu treba da sadrži broj koji predstavlja traženi ključ i verzija koja će biti -1 ili više u zavisnosti od toga da li je navedena.

Kada neki čvor primi PULL poruku, on ponavlja istu proceduru - ako je on vlasnik tog ključa, onda će vratiti rezultat pomoću PULL\_ANSWER poruke, koja sadrži objekat koji sadrži sve bitne informacije vezane za fajl. Ukoliko fajl nije pronađen unjegovom skladištu, ispisuje se poruka da je fajl izbrisan, ili ne postoji. U tom slučaju šalje se UNLOCK poruka, koja će otpustiti lokalni i distributivni ključ kako bi drugi čvorovi mogli da izvršavaju operacije. Ako nije vlasnik, onda prosleđuje PULL poruku ka narednom čvoru, izabranom na osnovu Chord pretrage.

Kad-tad će čvor koji je originalno poslao PULL poruku dobiti TELL\_GET poruku sa traženim fajl, koji će je onda upisati u svoj radni koren i ispisati poruku da se PULL operacija uspešno izvršila.

3.2 **add name**

Ako se ključ za koji se upisuje vrednost nalazi na trenutnom čvoru, onda prepisujemo fajl i njegov sadržaj u skladište sa početnom verzijom 0. Hash fajla se smešta u naše vrednosti. Ako to nije slučaj, šalje se ADD poruka ga narednom čvoru, izabranom na osnovu Chord pretrage. ADD poruka u svom telu sadrži id čvora koji je zatražio ADD operaciju. Takođe sadrži hash tražene vrednosti i objekat koji sadrži sve informacije bitne za fajl koji želimo da uskladištimo.

Kada neki čvor primi ADD poruku, on ponavlja istu proceduru - ako je on vlasnik tog ključa, onda izvrši upis u svoju mapu i prepisuje fajl i njegov sadržaj u skladište sa početnom verzijom 0. Takođe šalje se UNLOCK poruka, koja će otpustiti lokalni i distributivni ključ kako bi drugi čvorovi mogli da izvršavaju operacije. Ako nije vlasnik, onda prosleđuje ADD poruku ka narednom čvoru, izabranom na osnovu Chord pretrage.

3.3 **commit name**

Ako se ključ za koji se upisuje vrednost nalazi na trenutnom čvoru, onda poredimo verzije fajla (čiji heš odgovara traženom ključu) sa najvećom verzijom i skladišta i fajla koji mi šaljemo. Ukoliko ne dođe do konflikta, upisujemo fajl i povećavamo verziju za jedan. Ukoliko dođe do konflikta, informacije o fajlu sa najvećom verzijom se smeštaju u listu konflikata, i korisniku nudimo opciju da reši konflikte iz svoje liste. Ako to nije slučaj, šalje se COMMIT poruka ga narednom čvoru, izabranom na osnovu Chord pretrage. ADD poruka u svom telu sadrži id čvora koji je zatražio ADD operaciju. Takođe sadrži hash tražene vrednosti i objekat koji sadrži sve informacije bitne za fajl koji želimo da uskladištimo.

COMMIT poruka u svom telu treba da id čvora koji je zatražio COMMIT operaciju. Takođe sadrži hash tražene vrednosti i objekat koji sadrži sve informacije bitne za fajl sa najvećom verzijom koji šaljemo u slučaju da je došlo do konflikta, kao i version polje koji predstavlja trenutno najveću verziju fajla.

Kada neki čvor primi COMMIT poruku, on ponavlja istu proceduru - ako je on vlasnik tog ključa, onda poredimo verzije fajla (čiji heš odgovara traženom ključu) sa najvećom verzijom i skladišta i fajla koji mi šaljemo. Ukoliko ne dođe do konflikta, upisujemo fajl i povećavamo verziju za jedan. Ukoliko dođe do konflikta, smeštamo informacije o fajlu sa najvećom verzijom se smeštaju u odgovarajući objekat. Šalje se poruka COMMIT\_RESPONSE, u kojoj se šalje objekat sa informacijama o fajlu najveće verzija, kao i Boolean vrednost koja kaže da li se desio konflikt. Odgovor na ovu poruku jeste da je fajl ili uspešno komitovan, ili da se razreši konflikt ukoliko je nastao, tako što se korisniku ispisuje lista konflikata i nudi mu se opcija da raszreši neki od konflikata. Ukoliko je čvor vlasnik ključa, ali ključ nije pronašao šalje se UNLOCK poruka, koja će otpustiti lokalni i distributivni ključ kako bi drugi čvorovi mogli da izvršavaju operacije. Ako nije vlasnik, onda prosleđuje ADD poruku ka narednom čvoru, izabranom na osnovu Chord pretrage.

3.4 **remove name**

Ako se ključ za traženi fajl nalazi na trenutnom čvoru, onda brišemo fajl iz skladišta. Ukoliko fajl nije pronađen, ispisuje se poruka. Ukoliko se fajl uspešno izbriše, obaveštavamo korisnika. Ako to nije slučaj, šalje se REMOVE poruka ga narednom čvoru, izabranom na osnovu Chord pretrage. REMOVE poruka u svom telu sadrži id čvora koji je zatražio REMOVE operaciju. Takođe sadrži hash tražene vrednosti.

Kada neki čvor primi REMOVE poruku, on ponavlja istu proceduru - ako je on vlasnik tog ključa, briše fajl iz skladišta. Kao odgovor šalje UNLOCK poruku, koja će otpustiti lokalni i distributivni ključ kako bi drugi čvorovi mogli da izvršavaju operacije. Ako nije vlasnik, onda prosleđuje REMOVE poruku ka narednom čvoru, izabranom na osnovu Chord pretrage.

3.5 **quit**

Šalje se Bootstrapu poruka. Poruka sadrži tekst “Quit” u prvom redu, i u drugom redu sadrži jedan int, koji predstavlja port na kojem sluša novi čvor, i string koji predstavlja našu IP adresu. Stopiramo CLIParser, kako ne bismo više čitali komande sa komandne linije. Prolazimo kroz naše vrednosti sadržane u valueMap i pravimo HandOverValuesObject, koji predstavlja sve vrednosti koje naš sledbenik sada treba da upiše kod sebe. HandOverValuesObject sadrži sve bitne informacije o fajlovima. Šalje se HAND\_OVER\_VALUES poruka našem sledbeniku, koji, kada je primi, prepisaće naše fajlove iz skladišta u svoje i upisati nove vrednosti u svoju valueMap. HAND\_OVER\_VALUES sadrži mapu, koja predstavlja za određene heševe listu fajlova, koji predstavlaju različite verzija fajlova sa istim imenom. Naš sledbenik šalje SUCCESSOR\_ANSWER poruku, u kojoj se u telu poruku šalje moj chordId kako bi poruka došla do mene. Ukoliko je poruka došla do odgovarajućeg čvora, on će poslati QUIT poruku sledbeniku.

Ukoliko poruka nije došla do odgovarajučeg čvora, šalje se narednom čvoru, izabranom na osnovu Chord pretrage. EXIT poruka se prolazi kroz sve čovorove. Čvorovi koji nisu inijatori EXIT poruke, izbrisaće čvor koji je uradio quit iz svoje liste nodeInfo i osvežiti listu sledbenika. Takođe prosleđuju svom sledbniku. Na taj način, EXIT poruka će proći kroz svaki čvor i vratiće se inicijatoru. Kada je inicijator primi, on šalje svom sledbeniku UPDATE\_PREDECESSOR poruku, kako bi slednik promenio prethodnika, jer se njegov trenutni prethodnik isključuje iz sistema. Inicijator pravi pauzu od 5 sekundi, kako bi primio poruke, ukoliko su mu se slale, i na kraju gasi SimpleServentListener.

4 Razrešavanje konflikata

4.1 **view name**

Korisniku se za izabrani konflikt ispisuje sadržina fajla sa najvećom verzijom, i nakon toga mu se opet nudi opcija da razreši konflikte. Ukoliko je korisnik uneo ime pogrešnog fajla, ispisuje mu se greška. Pri view komande pristupa se lokalnom i distributivnom ključu, i nakon nje otpuštamo ključeve.

4.2 **pull\_confl name**

Korisniku se za izabrani konflikte upisuje novi sadržaj fajla sa največom verzijom u radni koren. Osvežava se verzija fajla koji imamo u random korenu najvećom verzijom i briše se izabrani konflikt iz verzije. Ukoliko je korisnik uneo pogrešan naziv fajla, ispisuje mu se greška.

4.3 **push name**

Ako se ključ za koji se upisuje vrednost nalazi na trenutnom čvoru, onda prepisujemo sadržaj fajl i njegov sadržaj u skladište i osvežavamo verziju. Konflikt se briše iz liste. Korisniku se ispisuje poruka o uspešnoj operaciji. Ako to nije slučaj, šalje se PUSH poruka ga narednom čvoru, izabranom na osnovu Chord pretrage. PUSH poruka u svom telu sadrži id čvora koji je zatražio PUSH operaciju. Takođe sadrži hash tražene vrednosti i objekat koji sadrži sve informacije bitne za fajl koji želimo da uskladištimo kao i najveću verziju.

Kada neki čvor primi PUSH poruku, on ponavlja istu proceduru - ako je on vlasnik tog ključa, onda izvrši upis u svoju mapu i prepisuje fajl i njegov sadržaj u skladište sa najnovijom verzijom. Kao odgovor šalje poruku UPDATE\_VERSION\_AFTER\_CONFLICT kako bi se na čvoru sa konfliktom osvežila najnovija verzija fajla i kako bi otpustio distributivni i lokalni ključ. Ako nije vlasnik, onda prosleđuje PUSH poruku ka narednom čvoru, izabranom na osnovu Chord pretrage.

4 Chord pretraga

Chord pretraga se obavlja pomoću tabele sledbenika. Ako je tražen ključ k (0 <= k <= N), obavljaju se sledeće provere:

1. Da li ja posedujem ključ k? Ako da, onda vrati svoj ID.
2. Da li moj prvi sledbenik poseduje ključ k? Ako da, onda vrati njegov ID.
3. Tražimo prvu stavku u tabeli sledbenika gde je ID veći od ključa, i onda vratimo prethodnu stavku u tabeli.

Treći korak se izvršava na ovaj način zato što nismo sigurni da je čvor u našoj tabeli sledbenika baš onaj koji poseduje ključ, čak i ako je on prvi koji ima ID veći od ključa. To je zbog toga što može da postoji čvor pre njega koji takođe ima ID veći od ključa, ali koji sasvim slučajno nije u našoj tabeli. Zato prosleđujemo upit prvom čvoru u našoj tabeli koji je pre pronađenog, i puštamo njega da utvrdi kome dalje treba proslediti poruku. Kad-tad će poruka stići do čvora čiji direktan sledbenik poseduje ključ, i onda ćemo sigurno znati tačnog vlasnika.

5 Poruke

Koristimo Java serijalizaciju za slanje poruka. Sledi spisak klasa koje opisuju poruke u našem sistemu.

**public enum MessageType {**

***NEW\_NODE*, *WELCOME*, *SORRY*, *UPDATE*, *ADD*, *HANDLE\_LOCK*, *UNLOCK*, *COMMIT*, *REMOVE*, *QUIT*, *HAND\_OVER\_VALUES*, *UPDATE\_PREDECESSOR*, *PUSH*, *PULL\_CONFLICT*, *PULL*,**

***PULL\_ANSWER*,*UPDATE\_VERSION\_AFTER\_CONFLICT*, *POISON*, *SUCCESSOR\_ANSWER*,**

***COMMIT\_RESPONSE***

**}**

public interface Message extends Serializable {

int getSenderPort();

int getReceiverPort();

String getReceiverIpAddress();

MessageType getMessageType();

String getMessageText();

int getMessageId();

}

public class BasicMessage implements Message {

private static final long serialVersionUID = -9075856313609777945L;

private final MessageType type;

private final int senderPort;

private final int receiverPort;

private final String messageText;

private final int messageId;

}

**public** **class** AddMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1448253009574569714L;

**private** **final** **int** hash;

**private** **final** FileObject addObject;

**public** AddMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** hash, FileObject addObject, **int** chordId) {

**super**(MessageType.***ADD***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.hash = hash;

**this**.addObject = addObject;

}

**public** **class** CommitMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = -2452874439664150424L;

**private** **final** FileObject commitObject;

**private** **final** **int** version, hash;

**public** CommitMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** hash,

FileObject commitObject, **int** version, **int** chordId) {

**super**(MessageType.***COMMIT***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.commitObject = commitObject;

**this**.hash = hash;

**this**.version = version;

}

**public** **class** CommitResponseMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** **int** hash;

**private** CommitResponseObject conflictObject;

**private** **boolean** conflictHappened;

**public** CommitResponseMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** chordId, **int** hash,

CommitResponseObject conflictObject, **boolean** conflictHappened) {

**super**(MessageType.***COMMIT\_RESPONSE***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.hash = hash;

**this**.conflictObject = conflictObject;

**this**.conflictHappened = conflictHappened;

}

**public** **class** ExitMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** ExitMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp,

**int** chordId) {

**super**(MessageType.***QUIT***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

}

}

**public** **class** HandleLockMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** HandleLockMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp) {

**super**(MessageType.***HANDLE\_LOCK***, senderPort, senderIp, receiverPort, receiverIp);

}

}

**public** **class** HandOverValuesToSuccessor **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** Map<Integer, ArrayList<HandOverValuesObject>> toSend;

**public** HandOverValuesToSuccessor(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp,

Map<Integer, ArrayList<HandOverValuesObject>> toSend, **int** chorId) {

**super**(MessageType.***HAND\_OVER\_VALUES***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chorId));

**this**.toSend = **new** HashMap<>(toSend);

}

**public** **class** NewNodeMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** NewNodeMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp) {

**super**(MessageType.***NEW\_NODE***, senderPort, senderIp, receiverPort, receiverIp);

}

}

**public** **class** PoisonMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = -5625132784318034900L;

**public** PoisonMessage() {

**super**(MessageType.***POISON***, 0, "", 0, "");

}

}

**public** **class** PullAnswerMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** FileObject addObject;

**private** **final** **int** newVersion;

**public** PullAnswerMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, FileObject addObject, **int** chordId, **int** newVersion) {

**super**(MessageType.***PULL\_ANSWER***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.addObject = addObject;

**this**.newVersion = newVersion;

}

**public** **class** PullMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = -4536785660919652205L;

**private** **final** **int** hash, version;

**public** PullMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** hash, **int** version, **int** chordId) {

**super**(MessageType.***PULL***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.hash = hash;

**this**.version = version;

}

**public** **class** PushMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** CommitResponseObject conflictObject;

**private** **final** **int** hash, version;

**public** PushMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** hash,

CommitResponseObject conflictObject, **int** chordId, **int** version) {

**super**(MessageType.***PUSH***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.conflictObject = conflictObject;

**this**.hash = hash;

**this**.version = version;

}

**public** **class** RemoveMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** Integer hash;

**public** RemoveMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** hash,**int** chordId) {

**super**(MessageType.***REMOVE***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.hash = hash;

}

**public** **class** UnlockMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String print;

**public** UnlockMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, **int** chordId,

String print) {

**super**(MessageType.***UNLOCK***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.print = print;

}

**public** **class** SuccessorAnswerMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** SuccessorAnswerMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp,

**int** chordId) {

**super**(MessageType.***SUCCESSOR\_ANSWER***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

}

}

**public** **class** SorryMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 8866336621366084210L;

**public** SorryMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp) {

**super**(MessageType.***SORRY***, senderPort, senderIp, receiverPort, receiverIp);

}

}

**public** **class** UpdateMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 3586102505319194978L;

**public** UpdateMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, String text) {

**super**(MessageType.***UPDATE***, senderPort, senderIp, receiverPort, receiverIp, text);

}

}

**public** **class** UpdatePredecessorMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String predecessorIp;

**private** **int** predecessorPort;

**public** UpdatePredecessorMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp,

String predecessorIp, **int** predecessorPort) {

**super**(MessageType.***UPDATE\_PREDECESSOR***, senderPort, senderIp, receiverPort, receiverIp, "");

**this**.predecessorIp =predecessorIp;

**this**.predecessorPort = predecessorPort;

}

**public** **class** UpdateVersionAfterConflict **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** **int** version;

**private** **final** String file;

**public** UpdateVersionAfterConflict(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp, String file, **int** version, **int** chordId) {

**super**(MessageType.***UPDATE\_VERSION\_AFTER\_CONFLICT***, senderPort, senderIp, receiverPort, receiverIp, String.*valueOf*(chordId));

**this**.version = version;

**this**.file = file;

}

**public** **class** WelcomeMessage **extends** BasicMessage {

**private** **static** **final** **long** ***serialVersionUID*** = -8981406250652693908L;

**private** Map<Integer, ArrayList<File>> values;

**private** Map<Integer, ArrayList<HandOverValuesObject>> files;

**public** WelcomeMessage(**int** senderPort, String senderIp, **int** receiverPort, String receiverIp,

Map<Integer, ArrayList<File>> values, Map<Integer, ArrayList<HandOverValuesObject>> files) {

**super**(MessageType.***WELCOME***, senderPort, senderIp, receiverPort, receiverIp);

**this**.values = **new** HashMap<>(values);

**this**.files = **new** HashMap<>(files);

}