Универзитет у Београду

Електротехнички факултет



Симулациона анализа епидемичних консензус алгоритама

Дипломски рад

|  |  |
| --- | --- |
| Ментор: | Кандидат: |
| проф. др Јелица Протић | Ненад Милошевић  2014/0340 |

Београд, септембар 2018.

Садржај

[Садржај 2](#_Toc525944303)

[1. Увод 3](#_Toc525944304)

[2. Тендерминт консензус и госип алгоритми 5](#_Toc525944305)

[2.1. О тендерминту 5](#_Toc525944306)

[2.2. Тендерминт консензус алгоритам 6](#_Toc525944307)

[2.3. Главне структуре података 8](#_Toc525944308)

[2.4. Тендерминт консензус алгоритам 9](#_Toc525944309)

[2.4.1. ProposalConsensusRoutine 10](#_Toc525944310)

[2.4.2. PrevoteConsensusRoutine 10](#_Toc525944311)

[2.4.3. PrecommitConsensusRoutine 11](#_Toc525944312)

[2.4.4. DecisionConsensusRoutine 12](#_Toc525944313)

[2.5. Тендерминт госип алгоритам 12](#_Toc525944314)

[2.5.1. Госип ReceiveRoutine 13](#_Toc525944315)

[2.5.2. Госип GossipRoutine 19](#_Toc525944316)

[3. Симулатор 24](#_Toc525944317)

[3.1. Мрежа процеса у симулатору 24](#_Toc525944318)

[3.2. Време пристизања поруке 25](#_Toc525944319)

[4. Резултати симулационе анализе 27](#_Toc525944320)

[5. Закључак 31](#_Toc525944321)

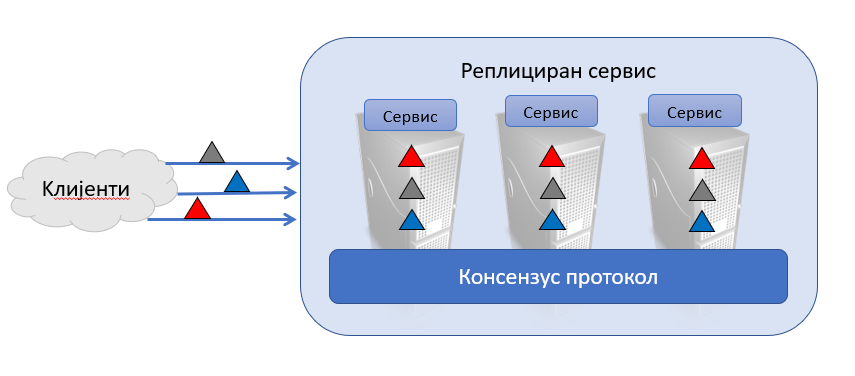
[Литература 32](#_Toc525944322)

[Списак скраћеница 34](#_Toc525944323)

[Списак слика 35](#_Toc525944324)

1. Увод

Консензус је вероватно један од најбазичнијих проблема у контексту дистрибуираних система толерантних на отказе (енглески **fault-tolerant distributed computing**). У академској литератури постоји велики број научних радова који анализирају консензус алгоритме, из теоријског али и практичног угла. Важност конзенсус проблема је повезана са његовом улогом у поступку ”Реплицирања детерминистичких коначних аутомата” (енглески **State Machine Replication**), скраћено **SMR**; генеричкој методи за реплицирање сервиса (програма) који се могу моделовати као детерминистички коначни аутомати [1], [2]. Улога консензуса у **SMR** поступку је да омогући да више реплика истог програма извршава захтеве (од клијената) у истом редоследу. На тај начин дистрибуиране реплике програма остају синхронизоване и пролазе логички кроз иста стања. Овај приступ гарантује исправно и континуирано извршавање програма и услед отказа (енглески **fault**) подскупа реплика.



Слика 1.1. – State Machine Replication

Највећи број теоријских анализа и практичних инсталација система базираних на консензус алгоритмима су за **LAN (Local area network)** мреже и садрже мали број реплика (три до седам), на пример радови [3] и [4]. Реплике су директно повезане и представљају део једног административног домена. Употреба консензус алгоритама у контексту блокчејн система и криптовалута доноси нове изазове у дизајну и анализи консензус алгоритама. Блокчејн апликације као децентрализовани системи не представљају део једног административног домена тако да није могуће обезбедити потпуну мрежну повезаност између процеса. Процеси су део **WAN (Wide area network)** мреже и један процес је најчешће повезан само са подскупом свих процеса у систему. Из тог разлога овакви системи су најчешће базирани на епидемичним (енглески **epidemic** или **gossip**) протоколима комуникација [5]. Такође, број процеса који учествују у оваквим системима је најчешће знатно већи него у традиционалним **LAN** инсталацијама (подразумева стотине или чак хиљаде процеса) [6].

Добре стране епидемичних алгоритама су толеранција отказа подскупа процеса као и могућност адаптирања приликом измене броја процеса и структуре комуникационе мреже између процеса. Са друге стране, време пропагације порука између процеса (енглески **end-to-end communication delay**) као и број дуплицираних порука су знатно већи него код сличних система код којих су процеси директно повезани.

Циљ овог рада је симулациона анализа једног таквог епидемичног система на примеру Тендерминт консензус технологије [7]. Предности симулација у односу на експерименталну анализу су релативно једноставна анализа перформанси система у **LAN** и **WAN** условима, као и за различит број процеса и за различит ниво повезаности комуникационе мреже између процеса.

Наставак овог рада је организован на следећи начин: поглавље (2) говори о Тендерминт консензус и госип алгоритму, њиховим специфичностима и детаљима који су од значаја за овај рад, поглавље (3) говори о симулатору, његовој архитектури и главним апстракцијама, у поглављу (4) се приказују графови са резултатима симулационе анализе и поглавље (5) представља закључак.

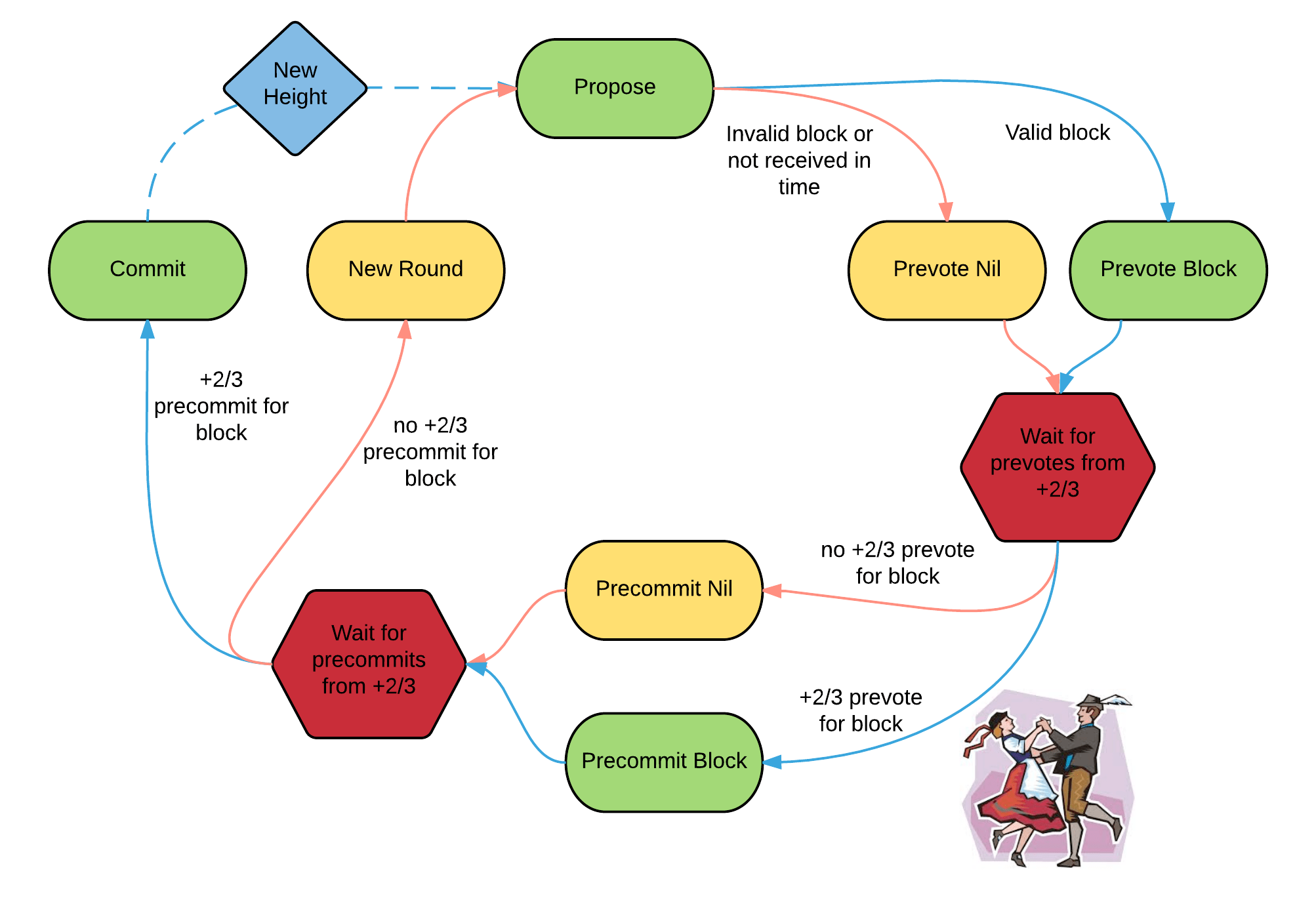
1. Тендерминт консензус и госип алгоритми
   1. О тендерминту

Tендерминт је софтвер развијан методологијом отвореног кода (енглески **open source**) [8] koji омогућава писање блокчејн апликација у било ком програмском језику користећи **ABCI (Application Blockchain Interface)** блокчејн интерфејс**.** За разлику од блокчејн система попут Биткоина [6] (енглески **Bitcoin**) и Итиријума [9] (енглески **Ethereum**) где се редослед трансакција и блокова у блокчејну одређује ”рударењем” (енглески **Proof of Work**), Тендерминт је базиран на дистрибуираном консензусу. Консензус алгоритам који се користи у Тендерминту је адаптација класичних консензус алгоритама попут **PBFT (Practical Byzantine Fault Tolerance)** [10] и **DLS (Dwork Lynch Stockmeyer)** [11] за блокчејн контекст [7]. Ради се о **BFT (Byzantine Fault Tolerant)** консензус алгоритму, што значи да Тендерминт толерише отказе подскупа процеса, и то све типове отказа (енглески **arbitrary**). To укључује класичне отказе (на пример диска) као и малициозно понашање (хакерски напади, софтвер заражен вирусом или администраторске грешке). Тендерминт алгоритам наставља нормално функционисање уколико мање од једне трећине процеса откаже; другим речима кажемо да Тендерминт претпоставља да је , где је укупан број процеса а максималан број неипсправних (енглески **fault**) процеса.

Процеси који учествују у Тендерминт консензус алгоритму размењују поруке преко епидемичног или госип (енглески **epidemic** или **gossip**) протокола, и сваки процес успоставља комуникацију само са подскупом свих процеса. Улога Тендерминт госип алгоритма је да омогући комуникацију између свих процеса у систему упркос томе што процеси нису директно повезани једни са другима. У оквиру Тендерминт система може се рећи да консензус алгоритам креира поруке које се онда прослеђују свим процесима преко Тендерминт госип протокола. Ради веће ефикасности Тендерминт госип алгоритам користи ”апликативно” знање консензус алгоритма који ”опслужује”, тако да можемо рећи да се ради о специфичном госип алгоритму (што није случај са класичним госип алгоритмима у литератури).

* 1. Тендерминт консензус алгоритам

Улога консензус алгоритма у Тендерминту је да одреди следећи блок трансакција у блокчејну. Након што процеси донесу одлуку (енглески **decide**) у тренутној консензус инстанци почињу наредну инстанцу. У Тендерминт терминологији број консензус инстанце се јоз назива и висина (енглески **height**). Cвака појединачна консензус инстанца се састоји од рунди (енглески **round**), где свака рунда представља секвенцу размене порука између процеса у покушају да донесу одлуку. У свакој рунди један од процеса има специјалну улогу ***предлагача*** (енглески **proposer**). Предлагач има задатак да предложи вредност око које треба да се постигне консензус у тој рунди. Предлагачи се у Тендерминту одређују детерминистички на основу вредности тренутне консензус инстанце и рунде, ***round robin*** формулом.



Слика 2.2.1 – Тендерминт консензус алгоритам

На слици 2.2.1 приказане су поруке које процеси рамењују као и стања кроз која пролазе у току сваке рунде Тендерминт консензус алгоритма. Процеси размењују следеће поруке у оквиру Тендерминт консензус алгоритма:

* ***Proposal*** – овом поруком се шаље предлог блока трансакција око кога треба да се постигне консензус.
* ***Prevote*** - овом поруком процеси гласају за предлог добијен преко ***Proposal*** поруке.
* ***Precommit*** – овом поруком процеси информишу остале процесе да ли су примили (или не) ***Prevote*** порука за предложену вредност.

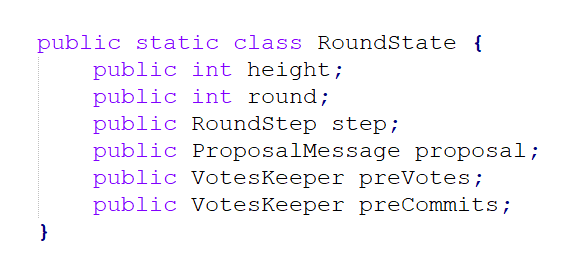
Стања кроз која процес пролази у току једне рунде су:

* ***ProposalWait*** – у овоме стању се налази процес све док не добије ***Proposal*** поруку за тренутну рунду.
* ***PrevoteWait*** – у ово стање процес прелази када добије ***Proposal*** поруку, и он је у овоме стању све док не добије ***Prevote*** порука за предложену вредност за текућу рунду, након тога прелази у ***PrecommitWait*** стање
* ***PrecommitWait*** – у овоме стању процес чека све док не добије ***Precommit*** порука за предложену вредност за текућу рунду, након тога сматрамо да је одлука донесена.

За потребе симулационе анализе у оквиру овог рада имплементиран је само подскуп Тендерминт консензус алгоритма који се односи на понашање у одсуству отказа (енглески **failure-free case**) и када је комуникациона мрежа синхрона и поуздана (нема губитака порука, енглески **message loss)**, што одговара нормалном или уобичајеном раду алгоритма (енглески **normal case**). Фукционалности алгоритма за толерисање отказа процеса и периоде непоуздане мрежне комуникације, као и одговарајућа анализа су ван обима овог рада, и планирани су као будући рад.

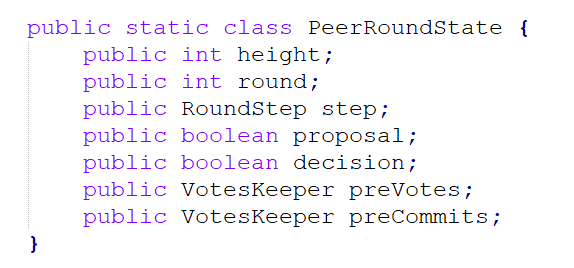
* 1. Главне структуре података

Пре него што опишемо консензус и госип алгоритме коришћене у овом раду дефинисаћемо две структуре података у којима се чувају све информације неопходне за њихово функционисање. За чување информација о тренутној консензус инстанци, сваки процес користи објекат типа ***RoundState*** који је приказан на слици 3.2.1. У овом објекту чувају се информације о тренутној консензус инстанци (висини) и рунди у којој се процес тренутно налази, као и у ком консензус стању за ту рунду је процес. Такође чува се ***Proposal*** порука за текућу инстанцу и рунду, као и све ***Prevote*** и ***Precommit*** које је процес примио у контексту тренутне висине (то укључује и поруке за све рунде те висине у којима је процес био).



Слика 3.2.1 – Класа RoundState

Кључна структура података за функционисање госип алгоритма је ***PeerRoundState*** приказан на слици 3.2.2. Наиме, Тендерминт креира једну инстанцу овог објекта за сваки процес са којим је успостављена директна мрежна комуникација. Те процесе називамо још и суседима (енглески **peer**). Филозофија Тендерминт госип алгоритма је да процес суседу прослеђује оне информације које му омогућавају најбржи напредак. Из тог разлога ***PeerRoundStatе*** структура садржи информације о тренутној висини и рунди у којој се сусед налази, као и његово тренутно консензус стање. Осим ових информација, за сваког суседа се чува и информација о томе да ли је сусед већ примио од неког процеса ***Proposal*** поруку за његову тренутну рунду, као и информације које ***Prevote*** и ***Precommit*** поруке је сусед до сада примио. Уколико се сусед налази у мањој висини, тј. процес већ зна која је одлука донешена у суседовој тренутној инстанци, процес шаље суседу ***Decision*** поруку са одлуком. У овом случају у ***PeerRoundStatе*** структури бележи се информација да смо му послали ***Decision*** поруку за његову тренутну консензус инстанцу, да би се избегло поновно слање ове поруке.



Слика 3.2.2. – Класа PeerRoundState

У наставку рада ће се, у примерима псеудокода, објекту ***RoundState*** приступати преко референце *roundState*. Референца *peerRoundState* ће се односити на објекат суседа, који је типа ***PeerRoundState***, и то на објекат оног суседаод кога смо примили поруку коју обрађујемо у том тренутку.

* 1. Тендерминт консензус алгоритам

Консензус алгоритам који је имплементиран у овом раду ће овде бити приказан у виду четири рутине које се позивају из госип протокола. Свака од рутина се односи на консензус поруку коју је процес примио и која би требало на адекватан начин да се обради.

Рутине које представљају консензус алгоритам су:

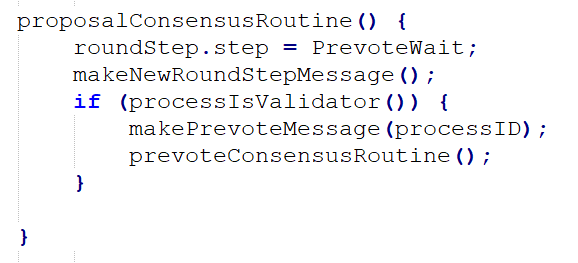
* ***ProposalConsensusRoutine***
* ***PrevoteConsensusRoutine***
* ***PrecommitConsensusRoutine***
* ***DecisionConsensusRoutine***

Свака ће детаљно бити објашњена у наредним одељцима.

* + 1. ProposalConsensusRoutine

Када процес прими ***Proposal*** поруку прелази у ***PrevoteWait*** стање и то је потребно ажурирати и креирати поруку која ће суседима на то указати. Након тога процес креира ***Prevote*** поруку. Све поруке које се креирају током консензуса ће помоћу госипа бити прослеђене суседима. Како је процес могао да прима ***Prevote*** поруке без обзира што још увек није прешао у ***PrevoteWait*** стање потребно је покренути и ***PrevoteConsensusRoutine*** јер можда заједно са његовoм ***Prevote*** поруком он има довољно порука да пређе у ***PrecommitWait*** стање.

Релевентан код дат је на слици 2.4.1.1.

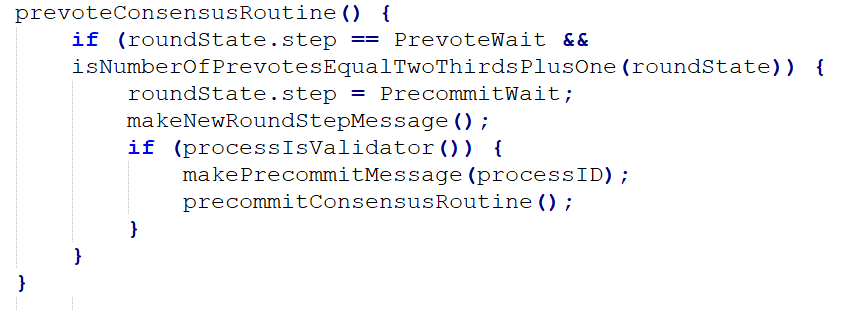


Слика 2.4.1.1. – ProposalConsensusRoutine

* + 1. PrevoteConsensusRoutine

Ова рутина би требало да провери да ли је број примљених ***Prevote*** порука довољан да се пређе у ***PrecommitWait*** стање. У случају да јесте мења се стање и то је потребно забележити и креирати поруку којом ће се та информација пренети суседима. У случају да се ради о процесу валидатору креира се ***Precommit*** порука. Након тога, потребно је покренути и ***PrevoteProposalRoutine*** јер се ***Precommit*** поруке могу примати и у ***PrevoteWait*** стању, тако да је могуће да је довољна још једна ***Precommit*** порука да се заврши текућа консензус итерација.

Псеудокод дат је на слици 2.4.2.1.

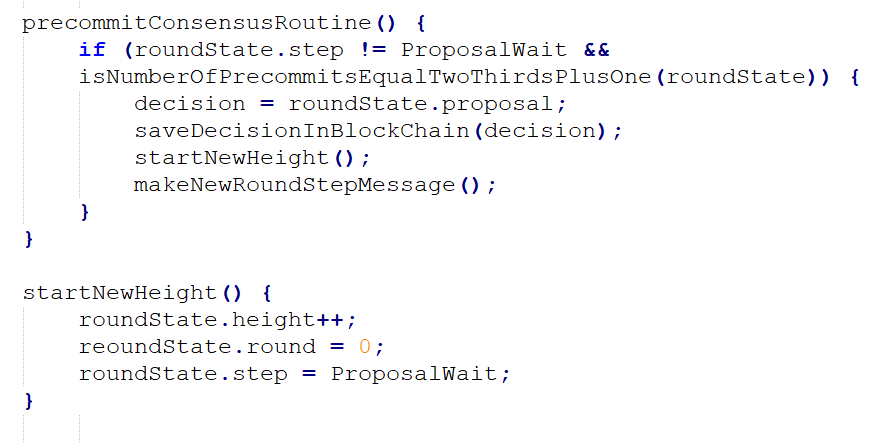


Слика 2.4.2.1. – PrevoteConsensusRoutine

* + 1. PrecommitConsensusRoutine

У овој рутини се проверава да ли су испуњени сви услови да би консензус био донесен. Услови које је потребно испунити јесу да смо примили ***Proposal*** поруку за ту рунду и да смо примили ***Precommit*** порука за ту предложену вредност. У случају да су услови испуњени креће се у нову итерацију консензус алгоритма.

Псеудокод дат је на слици 2.4.3.1.

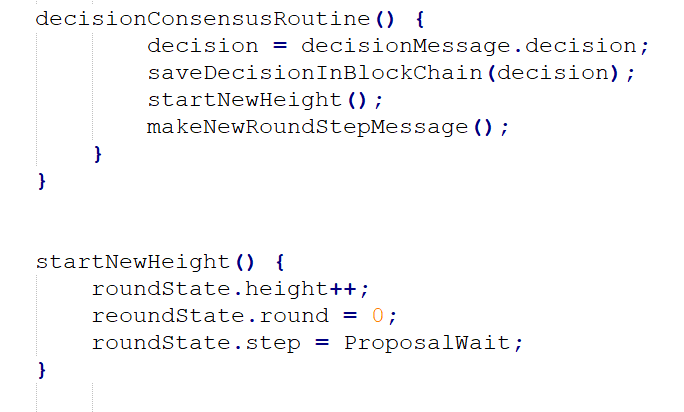


Слика 2.4.3.1. – PrecommitConsensusRoutine

* + 1. DecisionConsensusRoutine

Ова рутина се позива када се добије ***Decision*** порука од суседа за висину и рунду у којој се тренутно налази процес. Процес треба да запамти вредност и да заврши консензус без обзира у ком се стању налазио и да пређе на следећу висину, то јест да крене са извршавањем нове итерације консензус алгоритма.

Релевантан код на слици 2.4.4.1.



Слика 2.4.4.1. – DecisionConsensusRoutine

* 1. Тендерминт госип алгоритам

У оквиру овог рада се анализира мало једноставнија варијанта госип алгоритма која је довољна за анализу перформанси.

Госип алгоритам се може поделити на два кључна дела:

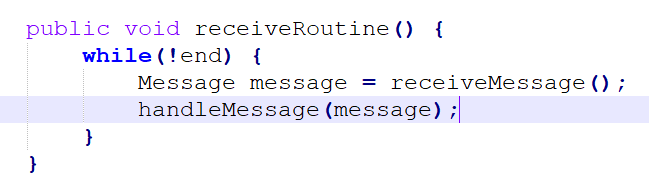
* ***ReceiveRoutine***
* ***GossipRoutine***

Оба дела ће бити детаљно описана у наредним поглављима.

* + 1. Госип ReceiveRoutine

***ReceiveRoutine*** је задужена да прима поруке од суседа, и да их на адекватан начин обрађује.

Имплементација је приказана на слици 2.5.1.1.



Слика 2.5.1.1. – Receive routine

Начин на који се обрађује порука зависи од типа поруке. Госип код Тендерминт-а размењује поруке специфичне за Тендерминт консензус и још неке поруке које олакшавају и оптимизију сам госип протокол, како не би дошло до слања редудантних порука.

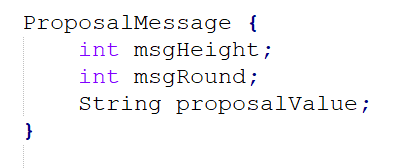
Поруке које се размењују путем госипа су:

* Поруке специфичне за консензус
  + ***Proposal***
  + ***Prevote***
  + ***Precommit***
  + ***Decision***
* Поруке специфичне за госип
  + ***HasVote***
  + ***NewStepRound***

У наредном делу текста биће приказано детаљније које информације носи и како се обрађује свака од ових порука. При томе осим госип дела, показаћемо и у ком тренутку се позивају рутине које би требало да обаве консензус логику.

* + - 1. **Proposal** порука

Релевантни делови Proposal поруке су приказани на слици 2.5.1.2.



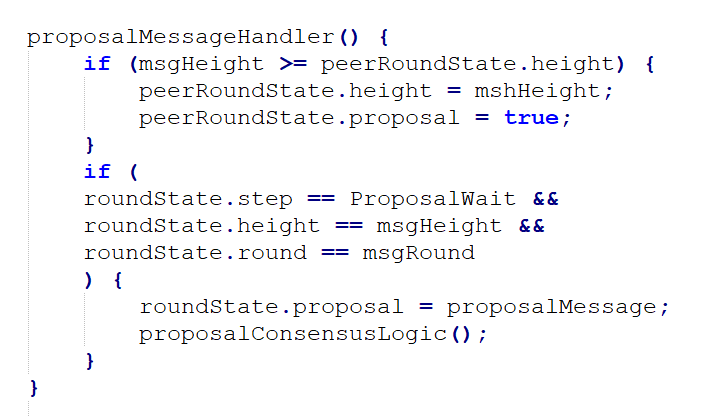
Слика 2.5.1.2. – ProposalMessage

Она садржи висину и рунду на коју се односи порука и предложену вредност.

Приликом пријема ове поруке потребно је проверити да ли је висина на коју се односи порука већа или једнака од висине суседа од кога смо примили поруку, у случају да јесте потребно је ажурирати висину суседа и означити да сусед има ***Proposal*** поруку.

Затим је потребно ажурирати и интерно стање процеса. Прво је потребно проверити да ли процес чека на ***Proposal*** поруку и да ли се порука односи на исту висину и рунду у којој се процес налази, ако је то случај поруку би требало сачувати. Након што смо сачували поруку потребно је покренути консензус рутину која се односи на пријем ***Proposal*** поруке.

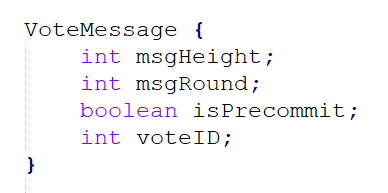
Псеудокод ове обраде дат је на слици 2.5.1.3.



Слика 2.5.1.3. - ProposalMessageHandler

* + - 1. **Prevote** и **Precommit** порука

Ове поруке су у госип алгоритму представљене једном поруком чији главни детаљи су приказани на слици 2.5.1.4.

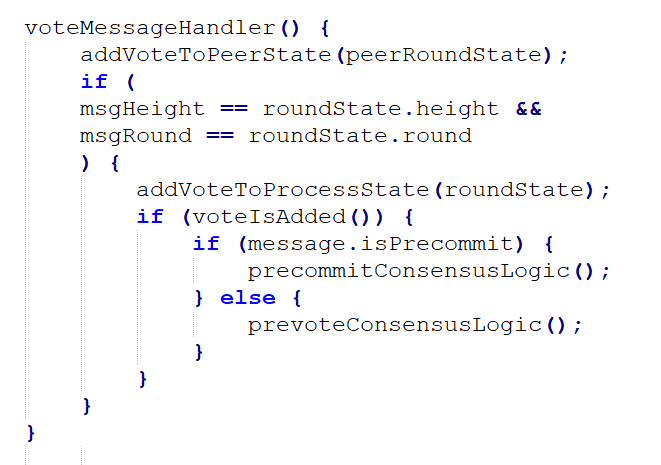


Слика 2.5.1.4. - VoteMessage

Она садржи висину и рунду поруке. Поред тога део ове поруке су идентификатор валидатора који је генерисао ову поруку и ознаку која указује на то да ли се ради о ***Precommit*** или ***Prevote*** поруци.

Са стране госип протокола ове две поруке се обрађују на идентичан начин. Обрада се састоји у томе да се забележи да сусед има ту поруку. Након тога би требало да се провери да ли се порука односи на нашу висину и рунду, у том случају би је требало запамтити. Сваки пут када сачувамо нову ***Prevote*** или ***Precommit*** поруку потребно је креирати нову ***HasVote*** поруку која ће бити послата суседима приликом извршавања ***GossipRoutine*** госип проткола. Исто тако, сваки пут када смо примили до тада непознату ***Prevote*** или ***Precommit*** поруку потребно је покренути консензус логику везану за пријем тог типа поруке.

Псеудокод ове обраде дат је на слици 2.5.1.5.

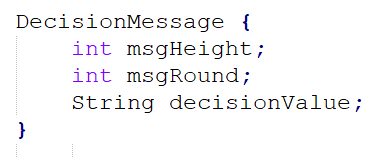


Слика 2.5.1.5. – VoteMessageHandler

* + - 1. **Decision** порука

Поред стандардних поља која се односе на рунду и висину, ова порука садржи и вредност око које је постигнут консензус.

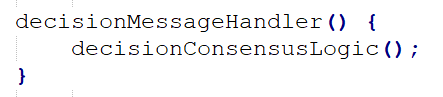
Поља поруке приказана су на слици 2.5.1.6.



Слика 2.5.1.6. - DecisionMessage

Пријем ове поруке је потпуно транспарентан за госип протокол јер њена обрада се само односи на консензус део Тендерминт-а. Потребно је само покренути рутину консензуса која је одговорна за пријем ове поруке.

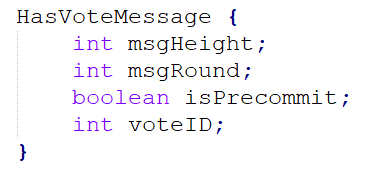
Псеудокод дат је на слици 2.5.1.7.



Слика 2.5.1.7. - DecisionMessageHandler

* + - 1. **HasVote** порука

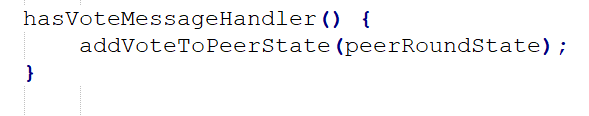
Тело ***HasVote*** поруке садржи иста поља као ***VoteMessage***, приказ њеног тела је на слици 2.5.1.8.



Слика 2.5.1.8. – HasVoteMessage

Ово је једна од порука које су специфичне за госип део. Овом поруком нас сусед обавештава да он има неку ***Prevote*** или ***Precommit*** поруку. На нама је само да то забележимо да му касније не би слали поруку коју већ има. Ова порука се креира и шаље суседима кад год примимо неку нову ***Prevote*** или ***Precommit*** поруку.

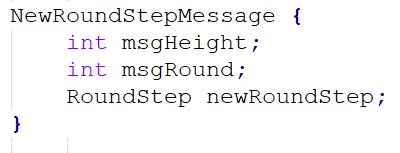
Псеудокод ове обраде дат је на слици 2.5.1.9.



Слика 2.5.1.9. – HasVoteMessageHandler

* + - 1. **NewRoundStep** порука

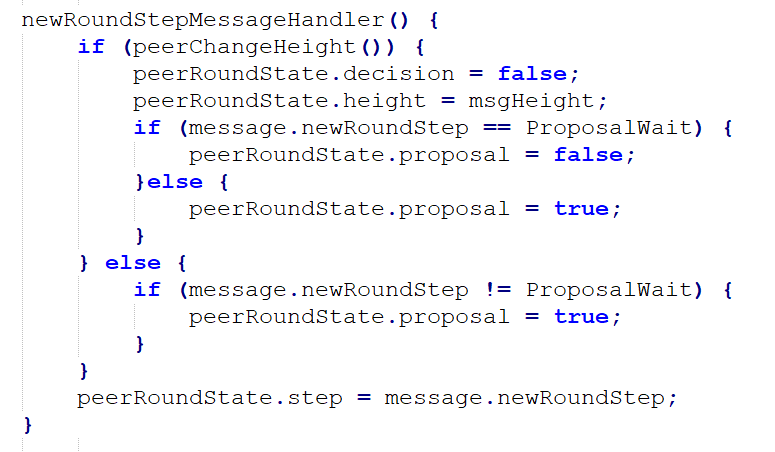
***NewRoundStep*** порука у себи носи информацију о новом стању консензус алгоритмa. Поља поруке се могу видети на слици 2.5.1.10.



Слика 2.5.1.10. – NewRoundStepMessage

Овом поруком сусед нас обавештава да је променио консензус стање. И на нама је само да то забележимо како би му слали оне поруке које му могу помоћи да даље напредује. Треба обратити пажњу на то да се при промени стања суседа може сазнати и то да је он прешао у нову рунду или дошао до консензуса, тако да би и то требало да буде забележено. За креирање ове поруке је задужен консензус део Тендерминт-а.

Псеудокод ове обраде дат је на слици 2.5.1.11.

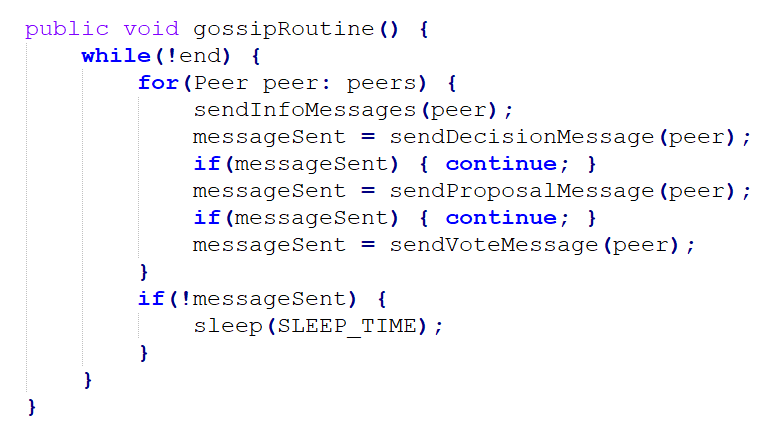


Слика 2.5.1.11. – NewRoundStepMessageHandler

* + 1. Госип GossipRoutine

Ова рутина госип протокла је задужена да на основу интерног стања процеса и информација које процес има о суседима одлучи које поруке треба да пошаље којем суседу и да то и уради.

Псеудокод ове рутине дај на слици 2.5.2.1.



Слика 2.5.2.1. – GossipRoutine

***GossipRoutine*** се састоји од четири дела:

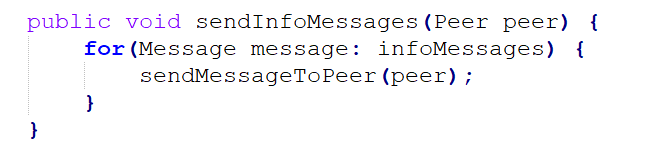
* ***InfoMessagesSender***
* ***DecisionMessagesSender***
* ***ProposalMessagesSender***
* ***VoteMessagesSender***

У наредним пододељцима биће детаљно описан сваки од њих. Такође, важно је напоменути да редослед ових рутина мора бити овакав како би се постигле најбоље перформансе и најмања редудантност порука.

* + - 1. ***InfoMessagesSender***

Овај део ***GossipRoutine*** има задатак да свим суседима пошаље све поруке које су настале при обради порука у ***ReceiveRoutine***. Поруке које се овде шаљу су поруке које нису везане за консензус већ имају задатак да убрзају рад госип протокола, тако што ће обезбедити да сваки процес има што бољи увид у стање свог суседа и да самим тим му шаље само оне поруке које су му заиста потребне. Овде се говори о ***HasVote*** и ***NewRoundStep*** порукама. Такође, треба напоменути да сваку од порука шаљемо само једном сваком од суседа.

Псеудокод дат је на слици 2.5.2.2.



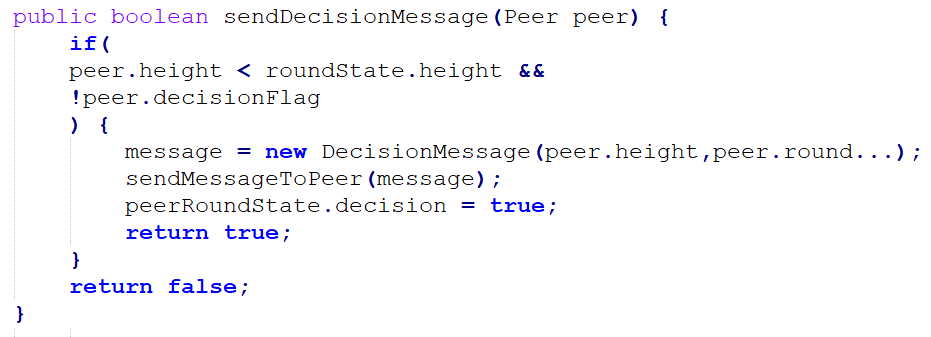
Слика 2.5.2.2. – InfoMessagesSender

* + - 1. **DecisionMessagesSender**

У случају да је процес напредовао више од свог суседа и да је висина на којој се он налази већа него висина његовог суседа, он свом суседу шаље ***Decision*** поруку. Овиме он жели да убрза суседов напредак, јер је у интересу консензус алгоритма да сви процеси напредују. Такође, ово је валидан поступак зато што ако смо ми већ донели одлуку за висину у којој се налази сусед то значи да је коректних процеса послало ***Precommit*** поруку и само је питање времена када би и наш сусед донео одлуку, ми овиме само убрзавамо његов напредак.

Још једна ствар коју је потребно урадити јесте да ажурирамо у стању суседа да смо му послали ***DecisionMessage*** како му не би у следећој итерацији опет слали.

Псеудокод дат је на слици 2.5.2.3.



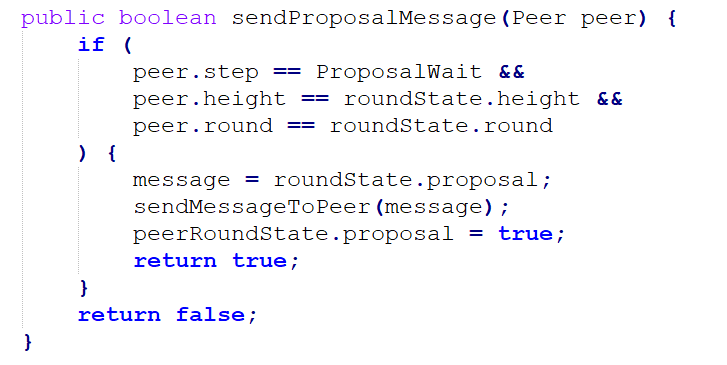
Слика 2.5.2.3. – DecisionMessagesSender

* + - 1. **ProposalMessagesSender**

У оквиру овог дела потребно је да оном суседу који се према информацијама које ми имамо о њему налази у ***ProposalWait*** стању и који се налази на истој висини и рунди као и ми пошаљемо Proposal поруку.

Након слања поруке потребно је ажурирати да смо му послали поруку како му је више не бисмо слали.

Псеудокод дат је на слици 2.5.2.4.



Слика 2.5.2.4. – ProposalMessagesSender

* + - 1. **VoteMessagesSender**

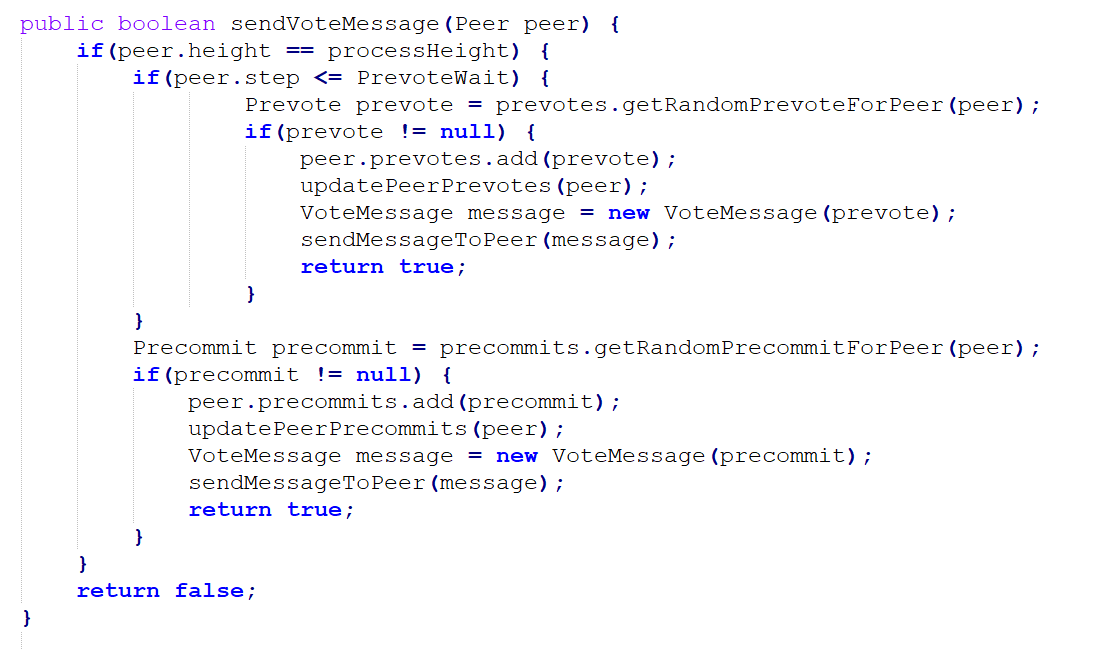
Ако смо дошли до овог дела ***GossipRoutine*** то значи да ни ***Decision*** ни ***Proposal*** порука није послата и да сусед од нас очекује да му пошаљемо ***Prevote*** или ***Precommit*** поруку.

Прво што треба да урадимо јесте да проверимо да ли се сусед налази на истој висини као и ми јер једино у том случају ми му шаљемо поруку. Ако се налази на мањој висини онда ћемо му слати ***Decision***, а ако се налази на већој висини онда му ми не можемо никако помоћи.

Као што је горе напоменуто, задатак овог дела рутине јесте да пошаље неку од ***Prevote*** или ***Precommit*** порука. ***Prevote*** порука се шаље ако је његово консензус стање ***ProposalWait*** или ***PrevoteWait*** и ако ми имамо неку ***Prevote*** поруку коју наш сусед нема. ***Precommit*** порука се шаље у случају да је стање суседа ***PrecommitWait*** или у случају да сусед има све ***Prevote*** поруке као и ми и да немамо ниједну ***Prevote*** поруку коју бисмо могли да му пошаљемо. Наравно у оба случаја је потребно да имамо ***Precommit*** поруку коју наш сусед нема.

Такође, потребно је да када смо послали суседу неку од порука да то забележимо како му не би слали исту поруку два пута.

Релевантан код дат је на слици 2.5.2.5.



Слика 2.5.2.5. – VoteMessageSender

1. Симулатор

Симулатор симулира један дистрибуирани систем који се састоји од великог броја процеса који су насумично повезани. Сваки процес се састоји од две нити, једна нит представља ***ReceiveRoutine*** госип протокола, а друга нит представља ***GossipRoutine*** госип протокола. Оне се понашају у складу са описом који је дат у претходним поглављима.

Главне две апстракције које су симулиране у оквиру симулатора јесу мрежа и време пристизања порука. Обе ће бити детаљно описане у наредним потпоглављима.

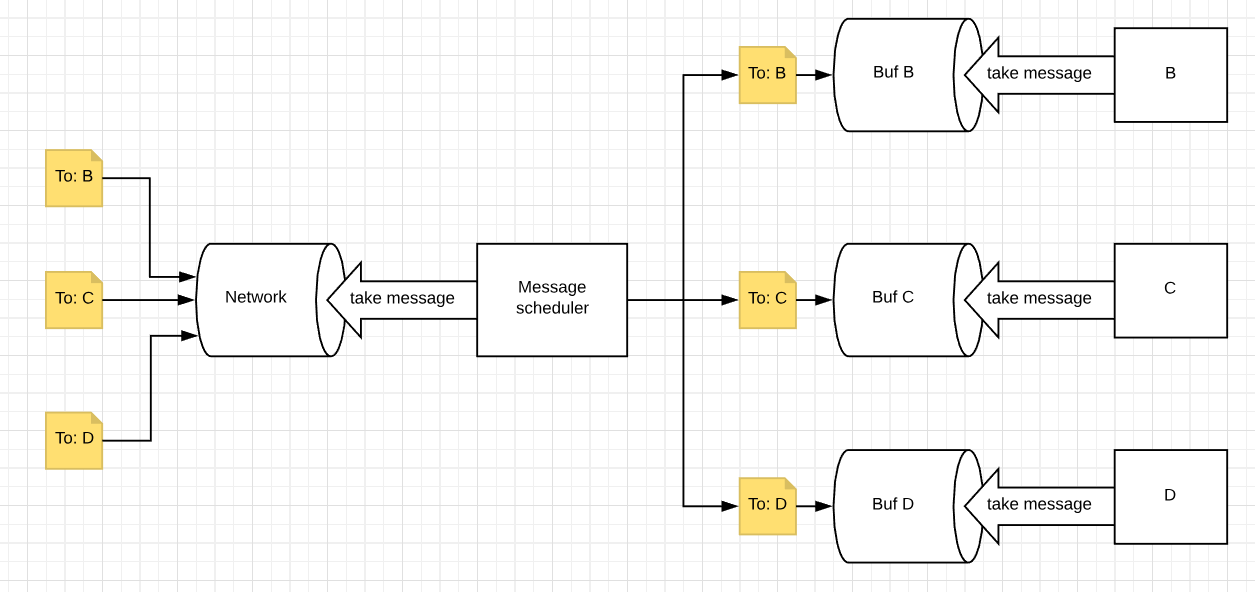
* 1. Мрежа процеса у симулатору

Као параметри симулације уносе се број процеса и број излазних конекција сваког процеса. Када се каже излазних конекција мисли се на број конекција које ће сваки процес покушати да успостави, а сами линкови између процеса суседа су бидирекциони. Везе између суседа могу бити **LAN** и **WAN** типа, за потребе симулација узето је да се између два суседа у 80% случајева успоставља веза **WAN** типа, а у 20% случајева веза **LAN** типа.

Сама размена порука између процеса је симулирана тако што када процес A жели да пошаље поруку процесу Б, он ће у оквиру поруке назначити коме шаље поруку и назначиће у ком тренутку би та порука требало да стигне процесу Б. О начину израчунавања времена пристизања поруке биће речи у наредном потпоглављу. Када је уписао у поруку коме је порука намењена и време испоручења он поруку шаље у мрежу, која је овде имплементирана као један бафер које је неопадајуће уређен по времену пристизања порука. Порука ће бити у том баферу све док јој не истекне време након тога ће бити послата процесу коме је намењена.

О прослеђивању порука одговарајућим процесима задужена је засебна нит. Она има задатак да узима прву поруку из бафера са порукама и у случају да јој је истекло време, требало би да је проследи процесу којем је намењена. Како је сваки процес у мрежи јединствено означен, у симулацији постоји једна хеш мапа која као кључ има јединствени идентификатор процеса, а као вредност има бафер са порукама. Из тога следи, да се прослеђивање поруке одређеном процесу Б са идентификатором састоји у томе да из хеш мапе узмемо бафер сачуван под кључем и да у њега ставимо одговарајућу поруку.

Приказ кретања порука кроз мрежу дат је на слици 4.1.1.



Слика 4.1.1. – Ток порука кроз мрежу

* 1. Време пристизања поруке

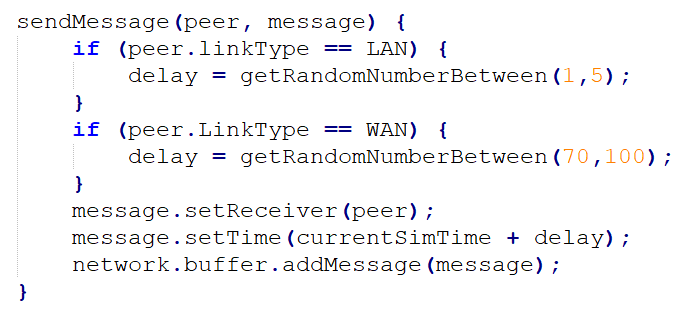
Како је један од главних циљева овог симулатора мерење времена које протекне док велики број процеса у дистрибуираном систему донесе консензус, било је потребно на неки начин симулирати време које протекне од тренутка када неки процес пошаље поруку до тенутка када та порука стигне до процеса примаоца.

За потребе симулације узета је апроксимација да време које прође између тренутка слања и тренутка пристизања поруке зависи од типа линк-а који се налази између два суседа.

Како је раније напоменуто да линкови могу бити **LAN** и **WAN** типа, узето је да то време у случају **LAN** линка износи од 1 до 5 ms, a у случају **WAN** линка оно износи од 70 до 100 ms.

Сваки процес пре него што пошаље поруку свом суседу он треба да на основу типа везе између суседа и њега изгенерише случајан број у опсезима од 1 до 5 или од 70 до 100 и да на тренутно време симулације дода тај број. Нит која прослеђује поруке ће тестирати време симулације и када оно буде веће од времена пристизања поруке, она ће проследити поруку оном процесу коме је порука намењена.

Релевантан код дат је на слици 4.2.1.

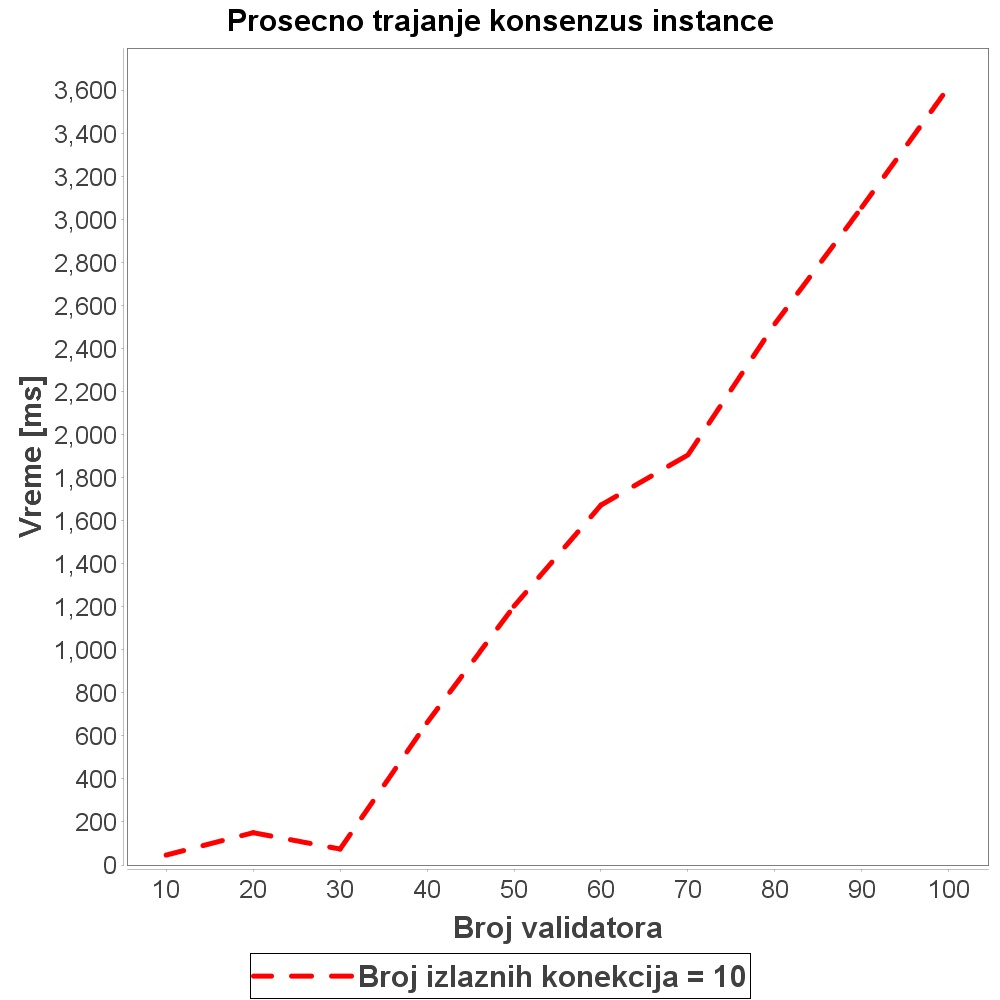


Слика 4.2.1. – Рачунање времена пристизања поруке

1. Резултати симулационе анализе

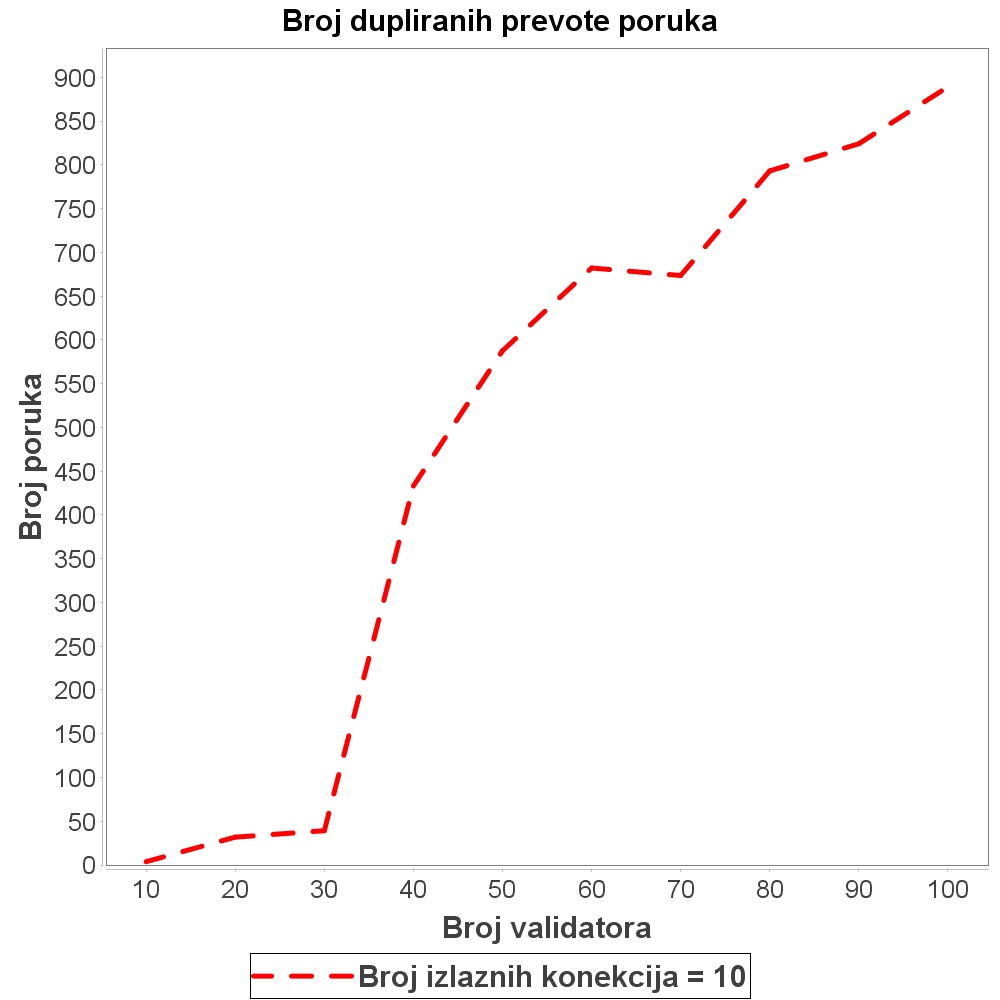
У наставку рада биће приказани резултати симулационе анализе. Они ће показивати просечно време које протекне да процес постигне консензус, просечан број дуплираних ***Prevote*** порука које процес добије и просечан број дуплираних ***Precommit*** порука. Параметар који ће се мењати приликом симулација је број процеса. Сви резултати су настали као просечни резултат резултата добијених из 100 итерација консензус-а.

На графицима који се налазе на сликама 5.1.,5.2. и 5.3. приказани су резултати за симулације где се број процеса мењао у опсегу од 10 до 100, а број излазних конекција је 10.



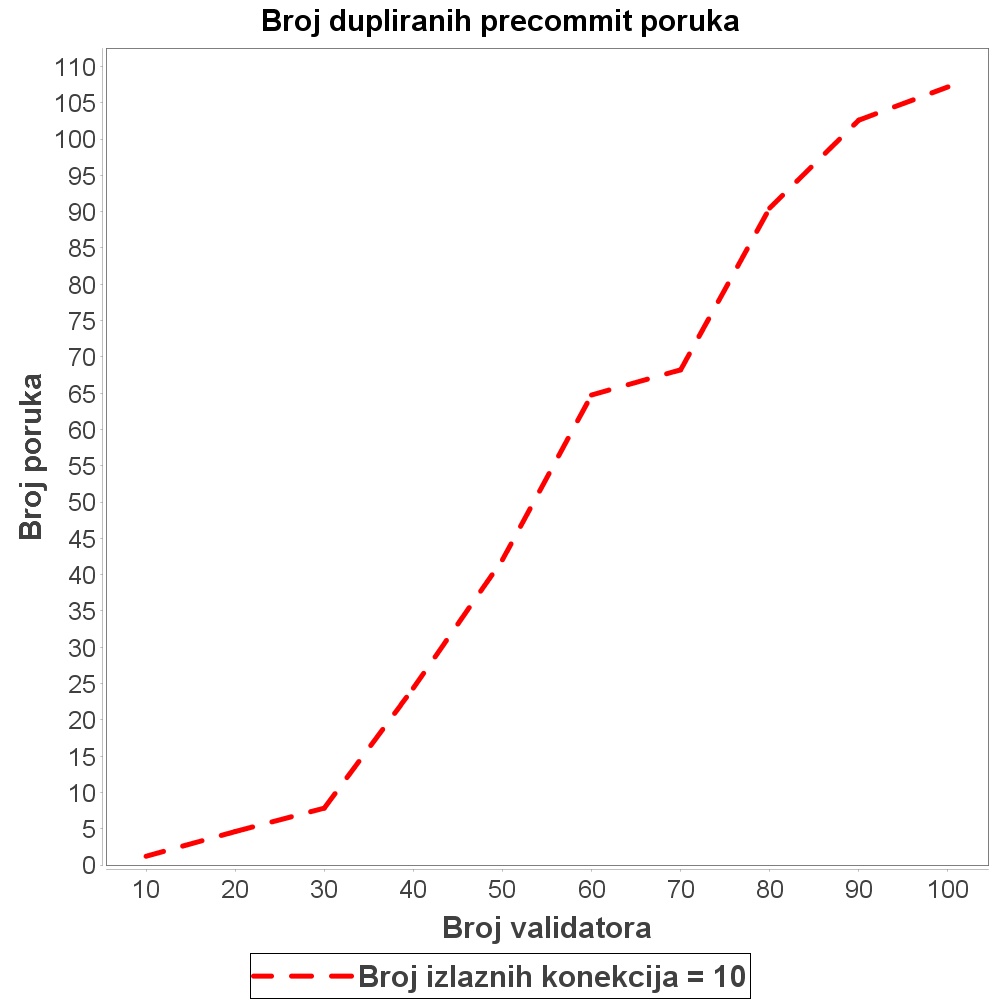
Слика 5.1. – Просечно трајање консензус инстанце

На графику на слици 5.1. можемо приметити да се време консензус инстанце повећава са повећањем броја процеса. Ово су резултати које смо и очекивали али занимљиво је што овом симулацијом добијамо време у ms које протекне док се не дође до консензуса и видимо да оно не прелази 3,4 секунде. Такође, можемо приметити да је највећи скок настао између 30 и 40 процеса, једно од објашњења зашто је до овога дошло је то што у просеку сваки процес има око 20 суседа и у случају када имамо 30 процеса довољно је да процес добије поруке само од својих суседа и да постигне консензус јер они чине две трећине свих процеса.



Слика 5.2. – Приказ односа дуплираних Prevote порука

На графику приказаном на слици 5.2. можемо видети да се број дуплираних **Prevote** порука процеса нагло повећава до границе од 40 процеса. Након тога и даље расте али мањим интензитетом. Једно од објашњења за нагли скок се наслања на то да је и пораст у времену постизања консензуса био у том пределу највећи па је самим тим и време за које је процес могао да прими дупликате највеће.



Слика 5.3. – Приказ односа дуплираних Precommit порука 1

На графику са слике 5.3. видимо да је и број дуплираних ***Precommit*** порука порастао са порастом броја процеса. Може се приметити да је тај број далеко мањи од бројa ***Prevote*** порука. Разлог за то лежи у самој имплементацији госипа који даје предност ***Prevote*** порукама у односу на ***Precommit*** поруке када се процес налази у неком од ***ProposalWait*** или ***PrevoteWait*** стању.

Још једна ствар коју нам симулација омогућава да тестирамо и упоредимо јесте колико се разликују претходни параметри у зависности од тога колико је процеса повезано **LAN** линком, а колико је процеса повезано **WAN** линком. У претходним графицима однос је био 80-20 у корист **WAN** линкова. Ови проценти се могу лако мењати, али су они узети јер је то најреалнији случај.

1. Закључак

Рад представља симулациону анализу епидемичног консензус алгоритма који се користи у Тендерминт блокчејн систему. Анализиран је утицај броја процеса на просечно трајање консензус инстанци као и просечан број дуплицираних ***Prevote*** и ***Precommit*** порука. Резултати симулација показују да се просечно трајање консензус инстанци линеарно повећава са повећањем броја процеса. У смислу апсолутних бројева, просечно трајање консензус инстанце за 100 процеса је 3,4 секунде, што омогућава коришћење Тендерминта и у апликацијама које захтевају обраду захтева у реалном времену. С друге стране број дуплицираних порука које сваки процес прими је доста велики и повећава се скоро експоненцијално са бројем процеса. Ово је аспект Тендерминт консензус алгоритма који је далеко од оптималног и где постоји доста простора за побољшање.

Као следећи корак у истраживању планирана је упоредна експериментална и симулациона анализа која ће послужити за валидацију и додатне корекције симулационог модела. Осим тога идеја будућег истраживања је проширење анализе на сценарија која укључују различите типове отказа и непоузданости мреже.

Литература

1. F. B. Schneider, "Implementing fault-tolerant services using the state machine approach - a tutorial" 1990. [Online]. Available: <https://www.cs.cornell.edu/fbs/publications/SMSurvey.pdf> (28.09.2018.)
2. L. Lamport, “Time, clocks, and the ordering of events in a distributed system,“ Commun. ACM, 1978. [Online]. Available: <https://amturing.acm.org/p558-lamport.pdf> (28.09.2018.)
3. M. Burrows, “The chubby lock service for loosely-coupled distributed systems,“ in OSDI, 2006, pp. 335-350. [Online]. Available: <https://research.google.com/archive/chubby-osdi06.pdf> (28.09.2018.)
4. M. Castro, B. Liskov, "Practical Byzantine fault tolerance and proactive recovery" 2002. [Online]. Available: <http://www.pmg.csail.mit.edu/papers/bft-tocs.pdf> (28.09.2018.)
5. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, “Epidemic algorithms for replicated database maintenance,” in Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, ser. PODC ’87. New York, NY, USA: ACM, 1987, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/41840.41841
6. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system" 2009. [Online]. Available: <http://www.bitcoin.org/bitcoin.pdf> (28.09.2018.)
7. E. Buchman, J. Kwon, Z. Milosevic, "The latest gossip on BFT consensus", 2018. [Online]. Available: <https://github.com/tendermint/spec/releases/download/v0.5/paper.pdf> (28.09.2018.)
8. Тендерминт - <https://github.com/tendermint/tendermint> (28.09.2018.)
9. V. Buterin, “Ethereum: A next-generation smart contract and decentralized application platform“ [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper> (28.09.2018.)
10. ——, “Practical byzantine fault tolerance and proactive recovery,” in Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, Feb. 1999.
11. C. Dwork, N. Lynch, and L. Stockmeyer, “Consensus in the presence of partial synchrony,” JACM, 1988.
12. Код симулатора - https://github.com/NenadEtf/ConsensusSimulator (28.09.2018.)

Списак скраћеница

SMR – *State Machine Replication*

LAN – *Local area network*

WAN – *Wide area network*

ABCI **-** *Application Blockchain Interface*

PBFT - *Practical Byzantine Fault Tolerance*

*DLS - Dwork Lynch Stockmeyer*

BFT **-** *Byzantine Fault Tolerant*

Списак слика

[Слика 1.1. – State Machine Replication 3](#_Toc525944055)

[Слика 2.2.1 – Тендерминт консензус алгоритам 6](#_Toc525944056)

[Слика 3.2.1 – Класа RoundState 8](#_Toc525944057)

[Слика 3.2.2. – Класа PeerRoundState 9](#_Toc525944058)

[Слика 2.4.1.1. – ProposalConsensusRoutine 10](#_Toc525944059)

[Слика 2.4.2.1. – PrevoteConsensusRoutine 11](#_Toc525944060)

[Слика 2.4.3.1. – PrecommitConsensusRoutine 11](#_Toc525944061)

[Слика 2.4.4.1. – DecisionConsensusRoutine 12](#_Toc525944062)

[Слика 2.5.1.1. – Receive routine 13](#_Toc525944063)

[Слика 2.5.1.2. – ProposalMessage 14](#_Toc525944064)

[Слика 2.5.1.3. - ProposalMessageHandler 14](#_Toc525944065)

[Слика 2.5.1.4. - VoteMessage 15](#_Toc525944066)

[Слика 2.5.1.5. – VoteMessageHandler 16](#_Toc525944067)

[Слика 2.5.1.6. - DecisionMessage 16](#_Toc525944068)

[Слика 2.5.1.7. - DecisionMessageHandler 17](#_Toc525944069)

[Слика 2.5.1.8. – HasVoteMessage 17](#_Toc525944070)

[Слика 2.5.1.9. – HasVoteMessageHandler 17](#_Toc525944071)

[Слика 2.5.1.10. – NewRoundStepMessage 18](#_Toc525944072)

[Слика 2.5.1.11. – NewRoundStepMessageHandler 18](#_Toc525944073)

[Слика 2.5.2.1. – GossipRoutine 19](#_Toc525944074)

[Слика 2.5.2.2. – InfoMessagesSender 20](#_Toc525944075)

[Слика 2.5.2.3. – DecisionMessagesSender 21](#_Toc525944076)

[Слика 2.5.2.4. – ProposalMessagesSender 22](#_Toc525944077)

[Слика 2.5.2.5. – VoteMessageSender 23](#_Toc525944078)

[Слика 4.1.1. – Ток порука кроз мрежу 25](#_Toc525944079)

[Слика 4.2.1. – Рачунање времена пристизања поруке 26](#_Toc525944080)

[Слика 5.1. – Просечно трајање консензус инстанце 27](#_Toc525944081)

[Слика 5.2. – Приказ односа дуплираних Prevote порука 28](#_Toc525944082)

[Слика 5.3. – Приказ односа дуплираних Precommit порука 1 29](#_Toc525944083)