HotStuff Sa Adaptivnim Pacemaker-om

Boris Čuljak
Predmet: Primenjeni Algoritmi u Upravljačkim Sistemima
E2 66/2024
25. decembar 2025.

***Sažetak*—U ovom radu proučavamo HotStuff, vizantijski otporan (*Byzantine Fault Tolerant*, BFT) protokol konsenzusa za replikaciju mašina stanja u modelu delimične sinhronosti. Prvo sistematizujemo njegove bezbednosne mehanizme i meahizme živosti (eng. *liveness*) koji omogućavaju potvrđivanje i konačno usvajanje blokova u prisustvu vizantijskih grešaka. Zatim implementiramo HotStuff u podesivom simulatoru sa modelom mreže koji obuhvata stabilne i nestabilne periode kašnjenja, kao i scenario sa vizantijskim vođom. Predlažemo adaptivni *pacemaker* koji dinamički podešava vremenska ograničenja za promenu mandata na osnovu lokalno posmatranih statistika mrežnog kašnjenja (npr. povratnog vremena i vremena formiranja *QC*-a). Predložena izmena utiče isključivo na politiku vremenskih ograničenja i ne menja pravila glasanja, zaključavanja niti potvrđivanja, čime se zadržavaju izvorne garancije bezbednosti, uz cilj smanjenja nepotrebnih promena mandata pri promenljivom mrežnom kašnjenju. Efekti osnovnog i adaptivnog *pacemaker*-a upoređuju se kroz simulacije u navedenim uslovima.**

I. Uvod

Od savremenih distribuiranih servisa se očekuje da ostanu dostupni čak i kada pojedinačne mašine otkažu ili postanu privremeno nedostupne. Zbog toga se servis često replicira na više čvorova, tako da sistem može nastaviti sa radom sve dok je dovoljan broj replika ispravan. Međutim, u prisustvu mrežnih kašnjenja i nepredvidive isporuke poruka, replike mogu posmatrati zahteve različitim redosledom, što dovodi do razilaženja stanja i narušavanja konzistentnog ponašanja servisa. Ovaj problem se u praksi rešava tako što se replike primoravaju da se slože oko jednog globalnog redosleda komandi, odnosno blokova.

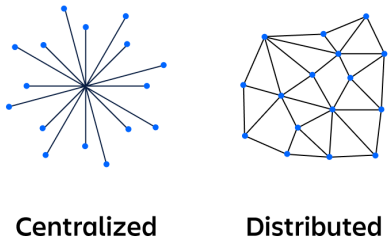
Konsenzus protokoli obezbeđuju mehanizam dogovora o tom redosledu. U okruženjima u kojima čvorovi mogu odstupati od protokola na proizvoljan način, potrebno je koristiti vizantijski otporan (*Byzantine Fault Tolerant*, BFT) konsenzus, koji ima cilj da očuva bezbednost (da ispravne replike ne donesu konfliktne odluke) i živost (da sistem na kraju donosi nove odluke) uprkos prisustvu neispravnih i potencijalno zlonamernih replika.

BFT protokoli zasnovani na vođi su naročito privlačni jer u uobičajenom slučaju mogu postići visoke performanse: vođa predlaže sledeći blok, a replike glasaju. Napredovanje protokola, međutim, zavisi od vođe. Kada je vođa spor ili neispravan, replike moraju preći u novi mandat sa novim vođom, kroz proceduru promene mandata (*view change*). Klasični dizajni, kao što je PBFT (eng. *Practical Byzantine Fault Tolerant*), postižu dobre performanse pod stabilnim vođom, ali procedura promene mandata može imati značajan komunikacioni trošak kako sistem raste [1]. HotStuff posebno obrađuje ovaj problem, sa ciljem da zadrži jednostavnu i efikasnu promenu mandata uz očuvanje jakih garancija bezbednosti [4].

HotStuff je BFT protokol za replikaciju mašina stanja u modelu delimične sinhronosti [2], [4]. Njegov dizajn teži linearnom komunikacionom trošku po mandatu i odzivnom napredovanju nakon stabilizacije mreže (*optimistic responsiveness*), tako da brzina odlučivanja prati stvarna mrežna kašnjenja umesto konzervativno izabranih fiksnih vremenskih ograničenja [4].

Doprinosi ovog rada su sledeći:

1) Sistematično objašnjenje HotStuff mehanizama koji obezbeđuju bezbednost i živost u delimično sinhronom modelu.
2) Implementacija HotStuff protokola u podesivom simulatoru sa parametrizovanim mrežnim modelom i mogućnošću modelovanja ispravnih i vizantijskih replika, uključujući scenario sa vizantijskim vođom.
3) Predlog adaptivnog *pacemaker*-a koji dinamički podešava vremenska ograničenja za promenu mandata na osnovu lokalno posmatranih statistika mrežnog kašnjenja, pri čemu se pravila glasanja, zaključavanja i potvrđivanja ne menjaju.
4) Evaluacija osnovnog i adaptivnog *pacemaker*-a u uslovima stabilnih i nestabilnih kašnjenja, uz analizu uticaja vizantijskog ponašanja vođe na napredovanje protokola.



Slika 1: Centralizovane naspram distribuiranih arhitektura. Centralizovani dizajni se oslanjaju na jedan koordinacioni čvor, dok distribuirani dizajni uklanjaju jedinstvenu tačku kontrole, ali otežavaju koordinaciju i konzistentnost.

II. Pozadina i postavka problema

Ova sekcija uvodi terminologiju i sistemske pretpostavke koje se koriste u ostatku rada. Ukoliko nije drugačije navedeno, pretpostavljamo *permissioned* okruženje sa autentifikovanom komunikacijom, fiksan skup od n replika od kojih do f može biti vizantijsko ($n \geq 3f+1$), kao i delimično sinhroni mrežni model sa nepoznatim trenutkom *Global Stabilization Time* (GST).

A. Distribuirani sistemi

Distribuirani sistem je skup nezavisnih računara (čvorova) koji komuniciraju razmenom poruka preko mreže kako bi zajednički realizovali servis. Za razliku od programa na jednoj mašini, distribuirani sistem se suočava sa dva osnovna izvora neizvesnosti: otkazima i vremenom. Otkazi obuhvataju rušenje čvora, nedostupnost ili neispravno ponašanje, dok vremenska neizvesnost potiče od mrežnih kašnjenja, gubitaka, dupliranja i isporuke poruka van redosleda, kao i od nesavršenog usklađivanja satova.

Ciljevi poput dostupnosti, konzistentnosti i otpornosti na otkaze često su u međusobnom kompromisu, a protokoli konsenzusa su direktan primer takvog kompromisa.

B. Replikacija i State Machine Replication (SMR)

Replikacija je osnovna tehnika za otpornost na otkaze: sistem pokreće više replika istog servisa, pa preostale replike mogu nastaviti rad kada deo sistema otkaže. Međutim, u prisustvu mrežnih kašnjenja replike mogu obrađivati zahteve različitim redosledom, što dovodi do razilaženja stanja i nekonzistentnog ponašanja servisa.

State machine replication (SMR) rešava ovaj problem modelovanjem servisa kao determinističke mašine stanja. Ako sve ispravne replike krenu iz istog početnog stanja i izvršavaju iste komande istim redosledom, ostaju u istom stanju i proizvode iste izlaze. Zbog toga SMR svodi problem replikacije na jedan glavni zahtev: replike moraju postići saglasnost o jednom globalnom redosledu komandi.

Koristan pogled na SMR je replikovani log: svaka pozicija k u logu sadrži jednu vrednost (komandu ili grupu komandi), a protokol konsenzusa se ponavljano izvršava kako bi sve ispravne replike upisale istu vrednost na svaku poziciju.

C. Konsenzus

Konsenzus je zadatak da distribuirani čvorovi postignu saglasnost o jednoj vrednosti. U SMR-u se rešava iterativno kako bi se dobio niz vrednosti. Dve osobine su ključne:

1) Bezbednost (*safety*): ispravne replike nikada ne odlučuju o konfliktnim vrednostima za istu poziciju u logu.
2) Živost (*liveness*): sistem na kraju odlučuje nove vrednosti, tako da klijenti ne čekaju beskonačno.

Bezbednost se tipično zahteva nezavisno od mrežnih kašnjenja, dok živost zahteva dodatne pretpostavke o vremenu.

D. Modeli grešaka i autentifikacija

U modelu *crash-fault* neispravan čvor može prestati da odgovara, ali ne šalje netačne poruke. U vizantijskom modelu grešaka, neispravan čvor može da se ponaša proizvoljno: da šalje konfliktne poruke različitim replikama, namerno odlaže ili uskraćuje poruke, ili koordinisano deluje sa drugim neispravnim čvorovima.

Pošto vizantijski čvorovi mogu lagati, BFT protokoli tipično pretpostavljaju autentifikovanu komunikaciju, kako bi primalac mogao da proveri identitet pošiljaoca svake protokolske poruke. U praksi se to ostvaruje digitalnim potpisima ili *MAC*-ovima, čime se sprečava falsifikovanje poruka u ime poštenih replika. Autentifikacija ne sprečava vizantijsku repliku da laže pod sopstvenim identitetom, ali sprečava imitaciju drugih replika, što je ključno za dokaze kvoruma (npr. sertifikate kvoruma).

E. Kvorumi i uslov $n \geq 3f+1$

U autentifikovanim BFT okruženjima standardni uslov je da sistem, radi tolerisanja f vizantijskih replika, ima najmanje $n \geq 3f+1$ replika. Protokoli se oslanjaju na kvorume veličine $2f+1$ (ekvivalentno $n-f$), jer se bilo koja dva takva kvoruma seku u najmanje $f+1$ replika, pa samim tim i u najmanje jednoj ispravnoj replici.

Tabela I: Performanse odabranih BFT protokola nakon GST (delimična sinhronost): složenost autentifikatora i odzivnost (prilagođeno iz [4]).

Protokol	**Složenost autentifikatora**			**Odzivan**
	Ispravan vođa	*Otkaz vođe (*view change*)*	*f otkaza vođe*	
DLS [2]	$O(n^4)$	$O(n^4)$	$O(n^4)$	–
PBFT [1]	$O(n^2)$	$O(n^3)$	$O(fn^3)$	×
SBFT	$O(n)$	$O(n^2)$	$O(fn^2)$	×
Tendermint / Casper	$O(n^2)$	$O(n^2)$	$O(fn^2)$	–
Tendermint* / Casper*	$O(n)$	$O(n)$	$O(fn)$	–
HotStuff [4]	$O(n)$	$O(n)$	$O(fn)$	✓

Svojstvo preseka kvoruma je centralno za sprečavanje dve konfliktne odluke.

F. Modeli mrežnog vremena

Garancije napredovanja zavise od pretpostavki o mrežnom kašnjenju poruka. U sinhronom modelu postoji poznata gornja granica kašnjenja, dok u asinhronom modelu takva granica ne postoji, pa deterministički konsenzus ne može biti garantovan čak ni uz samo *crash* greške [3].

Mnogi praktični protokoli koriste delimično sinhroni model [2]. Postoji nepoznat trenutak *Global Stabilization Time* (GST) nakon kojeg sve poruke između ispravnih replika stižu unutar ograničenog kašnjenja Δ, dok pre GST poruke mogu kasniti proizvoljno. U ovom modelu, bezbednost treba da važi uvek, a živost nakon GST. U praksi se to postiže *timeout*-ima koji vremenom postanu dovoljno veliki da pokriju post-GST kašnjenje.

G. Konsenzus zasnovan na vođi i promena mandata

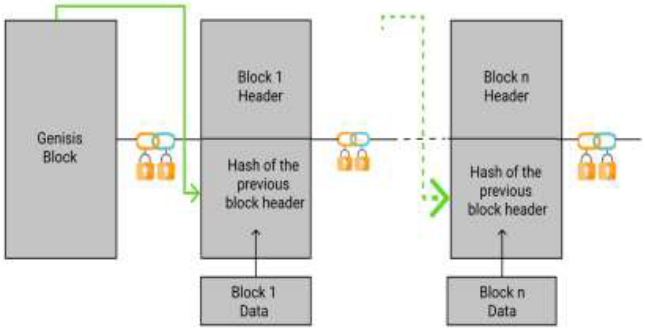
Uobičajen dizajn za konsenzus visokih performansi je pristup zasnovan na vođi. Vreme se deli na mandate, a svaki mandat ima dodeljenog vođu. Vođa predlaže sledeću komandu ili blok, a replike glasaju.

Kada je vođa spor ili neispravan, replike prelaze u novi mandat sa novim vođom. Ova procedura naziva se promena mandata (*view change*) i neophodna je za živost, ali mora biti projektovana tako da se bezbednost ne naruši predlaganjem konfliktnog nastavka istorije.

H. Skalabilnost

U BFT protokolima tipične mere troška su: broj poruka po odluci, ukupna količina prenetih podataka i kriptografski trošak (npr. provere potpisa ili izračunavanja *MAC*-ova). Big-O notacija opisuje kako ovi troškovi rastu sa brojem replika n.

Razmena „svako sa svakim“ među replikama košta $O(n^2)$ poruka. Promene mandata mogu biti skuplje ako se informacije prikupljene od mnogih replika moraju ponovo redistribuirati, što potencijalno uvodi troškove višeg reda. Jedan od ciljeva HotStuff-a je da promena mandata ostane jednostavna i efikasna korišćenjem kompaktnih sertifikata kvoruma i strukturisanog protokolskog jezgra [4].



Slika 2: Blokčejn kao replikovani log. Svaki blok sadrži grupu komandi i pokazivač na roditelja, formirajući lanac od *genesis* bloka. Saglasnost o istom lancu ekvivalentna je saglasnosti o jednom globalnom redosledu istorije.

I. Permissioned *blokčejn kao replikovani log*

HotStuff se često objašnjava terminologijom blokčejna, ali suštinski predstavlja SMR sa uređenim logom. Blok je kontejner koji grupiše jednu ili više klijentskih komandi (transakcija) i sadrži pokazivač na prethodni blok (roditelja), pa se dobija lanac koji počinje od *genesis* bloka. Saglasnost o istom lancu ekvivalentna je saglasnosti o istom redosledu komandi.

Razlika između *permissioned* i *permissionless* okruženja odnosi se na članstvo u konsenzusu. U *permissioned* blokčejnu validatori su poznat i fiksan skup čvorova sa uspostavljenim identitetima, što omogućava determinističku finalnost putem BFT glasanja. U *permissionless* okruženju članstvo je otvoreno, pa se koristi zaštita

od *Sybil* napada (npr. PoW/PoS), a finalnost je tipično probabilistička. HotStuff cilja *permissioned* slučaj i teži determinističkoj finalnosti u prisustvu vizantijskih grešaka i delimične sinhronosti [4].

III. Pregled HotStuff protokola

Ova sekcija objašnjava HotStuff na dva nivoa. Najpre opisujemo protokol konceptualno kroz ključne objekte i mehanizme bezbednosti i živosti. Zatim te ideje povezujemo sa konkretnim fazama i tokom poruka u *Basic HotStuff*-u, kao i sa protočnim pravilom potvrđivanja u *Chained HotStuff*-u.

A. Sistemski model

HotStuff pretpostavlja fiksan komitet od n replika (validatora) sa poznatim identitetima i autentifikovanom komunikacijom (videti Sekciju 2). Protivnik može kompromitovati do f replika i učiniti da se ponašaju vizantijski, uključujući slanje nekonzistentnih poruka, uskraćivanje poruka ili koordinaciju sa drugim neispravnim replikama, dok ispravne replike prate protokol.

Protokol je dizajniran za delimično sinhroni model [2], sa trenutkom *Global Stabilization Time* (GST) nakon kojeg se poruke između ispravnih replika isporučuju unutar ograničenog kašnjenja Δ. U standardnom autentifikovanom BFT podešavanju, HotStuff pretpostavlja $n \geq 3f + 1$, uz kvorume veličine $n - f = 2f + 1$, čije svojstvo preseka predstavlja osnovu za sprečavanje da dve konfliktne istorije obe postanu potvrđene.

U našem simulatoru pratimo isti model: fiksan skup od n replika, do f vizantijskih replika, autentifikovana razmena poruka i mrežni raspoređivač koji prelazi iz nestabilnog perioda (pre GST) u isporuku sa ograničenim kašnjenjem (posle GST). Modelujemo rotaciju vođe po mandatima i uključujemo protivnička ponašanja za neispravnog vođu, kao što su: (i) *equivocation* (slanje različitih predloga različitim replikama u istom mandatu), (ii) *withholding* (nepredlaganje ili prekid nakon predlaganja), i (iii) selektivno odlaganje ili odbacivanje poruka koje vođa treba da prosledi (npr. glasova ili sertifikata) kako bi usporio napredovanje.

B. Osnovni objekti

HotStuff predstavlja napredak protokola kao rastuće stablo predloga. Svaki predlog je blok i sadrži: (i) grupu klijentskih komandi, (ii) pokazivač na roditeljski blok i (iii) *opravdanje* (*justification*) u obliku sertifikata kvoruma.

Sertifikat kvoruma (*Quorum Certificate*, QC) je kompaktan dokaz da je kvorum od $2f + 1$ replika podržao konkretan blok u konkretnoj fazi protokola. U našoj implementaciji, QC predstavljamo kao skup od $(n - f)$ pojedinačno autentifikovanih glasova; kao optimizacija, *threshold signatures* mogu obezbediti da QC ostane konstantne veličine i kada n raste.

Vreme je podeljeno na mandate, a svaki mandat ima dodeljenog vođu. Vođa predlaže novi blok i emituje ga replikama, a replike odgovaraju slanjem glasova. Kada vođa prikupi glasove iz kvoruma od $2f + 1$ različitih replika za isti blok i fazu u datom mandatu, ti glasovi se kombinuju u QC.

QC ima dve ključne uloge: (i) sažima najveći poznati napredak tokom normalnog rada i (ii) služi kao prenosiv dokaz napretka kroz promenu mandata, tako što replike prosleđuju svoj najveći poznati QC vođi narednog mandata.

C. Mehanizmi bezbednosti

Bezbednost HotStuff-a se obezbeđuje kombinacijom sertifikata kvoruma, pravila zaključavanja i predikata bezbednog glasanja. Intuitivno, kada replika vidi dovoljno jak dokaz da je protokol napredovao na nekoj grani, ona se zaključava na tu granu i odbija glasanje za konfliktnu granu.

Svaka replika održava dve ključne lokalne promenljive:

- *highQC*: najveći QC koji replika trenutno poznaje (najjači dokaz napretka, tipično poređen po broju mandata);
- *lockedQC*: QC na kojem je replika trenutno zaključana (tačka nakon koje odbija prelazak na konfliktnu granu).

Kada vođa predlaže blok, on uključuje *opravdanje* (QC). Replike primenjuju pravilo bezbednog glasanja (često izraženo kao predikat `safeNode`). Neformalno, replika glasa za predlog ako važi makar jedno od sledećeg:

1) predloženi blok nastavlja granu na kojoj je replika trenutno zaključana, ili
2) predlog nosi *opravdanje* (QC) iz strogo većeg mandata od trenutnog zaključavanja replike, što predstavlja jači dokaz da je protokol bezbedno prešao dalje [4].

Ovo pravilo sprečava ispravne replike da daju glasove koji bi mogli omogućiti dve konfliktne potvrde. Svojstvo preseka kvoruma je ključno: da bi dve konfliktne istorije obe prikupile kvorum, bar jedna ispravna replika bi morala da glasa nekonzistentno, što je onemogućeno predikatom bezbednog glasanja.

D. Promena mandata

U HotStuff-u, ako replike dovoljno dugo ne uoče napredak, pokreću promenu mandata. Tokom promene mandata replike prelaze u viši mandat i šalju poruku `new-view` vođi narednog mandata, zajedno sa svojim najvećim poznatim QC-om.

Glavni izazov živosti je da novi vođa predloži blok na grani za koju su ispravne replike spremne da glasaju. Ako vođa predloži na zastareloj grani, ispravne replike mogu odbiti glasanje zbog svojih zaključavanja, što vodi ponovljenim neuspesima i izgubljenim mandatima. HotStuff rešava ovo tako što novi vođa bira najveći QC koji nauči (označen kao *highQC*) i predlaže blok koji nastavlja blok sertifikovan tim QC-om, čime se nakon GST tipično postiže brza konvergencija na jednu granu.

E. *Faze* Basic HotStuff-*a*

Basic HotStuff vodi isti blok kroz tri faze glasanja: `prepare`, `pre-commit` i `commit`. U svakoj fazi, vođa prikuplja glasove iz kvoruma i formira odgovarajući QC, koji zatim emituje kao dokaz prelaska u sledeću fazu.

Svaka replika prati svoj trenutni mandat (*viewNumber*), kao i:

- *highQC*: najveći QC koji poznaje (koristi se kao dokaz napretka i prosleđuje se u `new-view`);
- *lockedQC*: QC na kojem je zaključana (koristi se za sprečavanje glasanja za konfliktnu granu).

U svakoj fazi replike čekaju očekivanu poruku do isteka *timeout*-a koji kontroliše *pacemaker*. Ako *timeout* istekne, replika uvećava svoj broj mandata i šalje poruku `new-view` za sledeći mandat. Nakon GST, *timeout*-i vremenom postaju dovoljno veliki da obuhvate ograničenje mrežnog kašnjenja, čime se omogućava da ispravan vođa prikupi kvorume i pokreće protokol napred.

1) Pregled jednog mandata: Na početku mandata v, replike šalju vođi poruku `new-view` koja sadrži njihov najveći poznati QC (*highQC*). Vođa bira najveći QC koji primi (takođe označen kao *highQC*). Intuitivno, *highQC* identifikuje najaktuelniju granu podržanu kvorumom koju vođa može bezbedno nastaviti.

Koristeći *highQC* kao *opravdanje*, vođa predlaže novi blok koji nastavlja *highQC.block* i vodi blok kroz tri faze glasanja: `prepare`, `pre-commit` i `commit`. Nakon prikupljanja kvoruma u svakoj fazi, vođa formira QC za tu fazu i emituje ga kako bi prešao u sledeću fazu. Kada se formira *commitQC*, replike mogu finalizovati odluku i izvršiti potvrđene komande. U ovom prikazu, vođa emituje eksplicitnu poruku `decide` koja nosi *commitQC* kao implementacionu pogodnost.

2) `prepare` faza: Vođa kreira novi blok koji nastavlja *highQC.block* i emituje `prepare` poruku koja sadrži blok i *opravdanje* (QC). Po prijemu predloga, replika proverava da li je predlog bezbedan pomoću predikata bezbednog glasanja (npr. `safeNode`). Ako je prihvaćen, replika šalje `prepare-vote` vođi. Kada vođa prikupi $(n-f)$ `prepare-vote` glasova, kombinuje ih u *prepareQC* za predlog.

3) `pre-commit` faza: Vođa emituje *prepareQC* u poruci `pre-commit`. Replike verifikuju QC i odgovaraju sa `pre-commit-vote`. Kada vođa prikupi $(n-f)$ `pre-commit-vote` glasova, kombinuje ih u *precommitQC*.

4) `commit` faza (zaključavanje): Vođa emituje *precommitQC* u poruci `commit`. Replike verifikuju QC i odgovaraju sa `commit-vote`. Kada replika obradi validan *precommitQC* za trenutni predlog, zaključava se na taj QC postavljanjem *lockedQC* $\leftarrow$ *precommitQC* [4]. Nakon zaključavanja, replika neće glasati za konfliktnu granu osim ako kasnije ne uoči *opravdanje* (QC) iz višeg mandata koji prolazi predikat bezbednog glasanja.

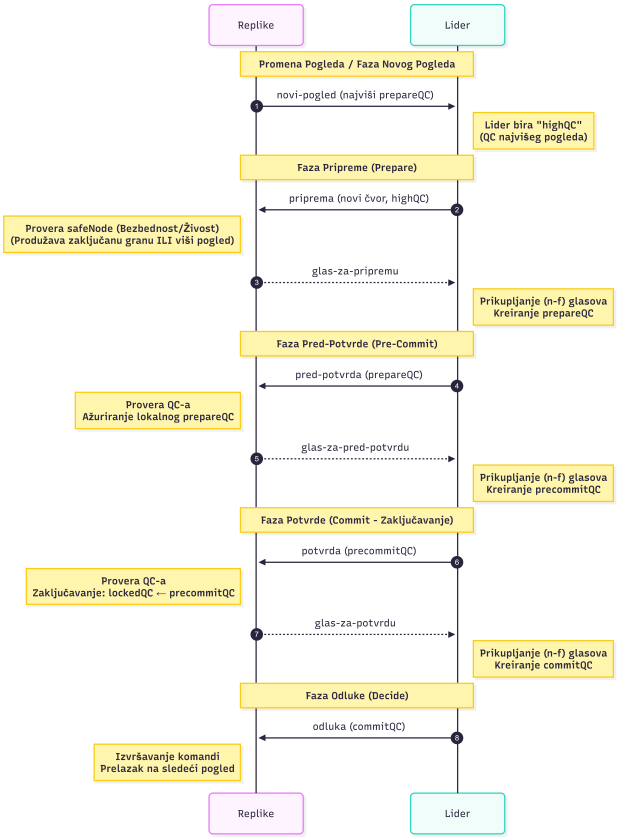
5) `decide` faza (finalizacija i izvršavanje): Kada vođa prikupi $(n-f)$ `commit-vote` glasova, kombinuje ih u *commitQC* i emituje ga u poruci `decide`. Po prijemu validnog `decide`, replike tretiraju blok koji implicira *commitQC* (i njegove pretke) kao potvrđen, izvršavaju novopotvrđene komande po redu i prelaze u sledeći mandat.

F. Chained HotStuff

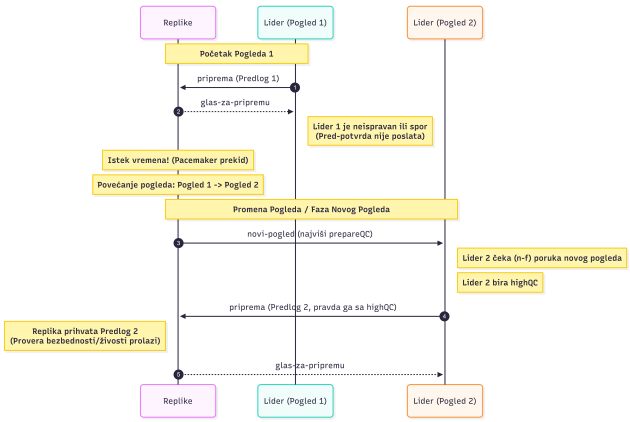
Chained HotStuff je varijanta HotStuff-a visokih performansi koja protočno potvrđivanje raspoređuje preko uzastopnih blokova. U *Basic HotStuff*-u, vođa vodi isti blok kroz `prepare`, `pre-commit` i `commit` pre nego što ga finalizuje. U *Chained HotStuff*-u, svaki mandat proizvodi QC za najnovije predloženi blok, a ti QC-ovi implicitno unapređuju status potvrde ranijih blokova na istoj grani. Slike 4 i 5 daju vizuelno poređenje.

Konkretno, u mandatu v vođa predlaže blok b_1 i formira $\mathrm{QC}(b_1)$. U mandatu $v+1$, vođa predlaže b_2 koji nastavlja b_1 i formira $\mathrm{QC}(b_2)$. U mandatu $v+2$, vođa predlaže b_3 koji nastavlja b_2 i formira $\mathrm{QC}(b_3)$. Kada postoji 3-lanac $b_1 \leftarrow b_2 \leftarrow b_3$ sa QC-ovima na b_2 i b_3 u strogo rastućim mandatima, protokol potvrđuje b_1 [4]. Intuitivno, QC-ovi na potomcima pokazuju da je kvorum nastavio da proširuje istu granu više puta; u kombinaciji sa zaključavanjem i presekom kvoruma, ovo čini b_1 nepovratnim.

Praktični efekat je da mandati nastavljaju da proizvode nove blokove čak i dok raniji blokovi još nisu potvr-

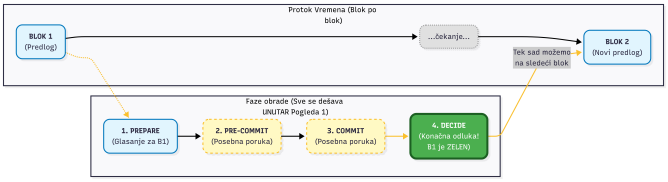


(a) Tok u normalnom slučaju unutar jednog mandata. Replike šalju `new-view` poruke, vođa bira *highQC*, predlaže u `prepare`, prikuplja glasove da formira *prepareQC*, zatim ponavlja u `pre-commit` i `commit` da formira *precommitQC* i *commitQC*. Replike se zaključavaju kada uoče *precommitQC* i izvršavaju nakon `decide`.



(b) Otkaz vođe i oporavak. Vođa u mandatu v prikupi deo glasova, ali otkaže pre prelaska faza. Replike isteknu preko *pacemaker*-a, prelaze u mandat $v+1$, šalju `new-view` poruke sa svojim najvećim QC-om, a novi vođa bira *highQC* da bezbedno nastavi predlažući novi blok opravdan tim QC-om.

Slika 3: Sekvencijski dijagrami za *Basic HotStuff*: (levo) uspešno napredovanje kroz faze unutar jednog mandata; (desno) promena mandata (*view change*) usled neispravnog/sporog vođe i bezbedan oporavak u sledećem mandatu korišćenjem *highQC*.



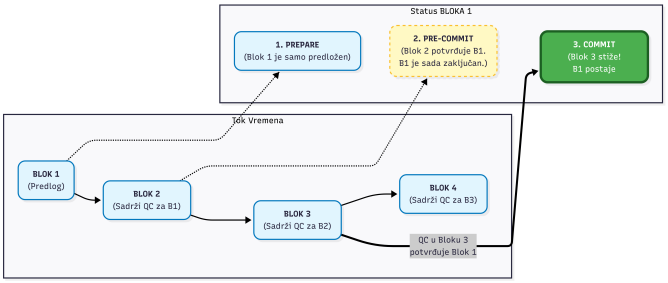
Slika 4: Tok *Basic HotStuff*-a. Jedan blok se vodi kroz eksplicitne fazne poruke (`prepare`, `pre-commit`, `commit`, `decide`) pre nego što protokol pređe na sledeći blok.

đeni. Finalnost se dobija kada je blok zakopan ispod dva potomka sertifikovana QC-om. Nakon popunjavanja cevovoda, protokol može (približno) potvrditi jedan blok po mandatu u stabilnom režimu.

G. Pacemaker *u HotStuff-u*

HotStuff razdvaja bezbednosno-kritično jezgro od mehanizma živosti kroz modul koji se uobičajeno naziva *pacemaker*. Bezbednosno jezgro definiše za šta replike smeju da glasaju (bezbedno glasanje, zaključavanje i pravila potvrđivanja), dok je *pacemaker* odgovoran za napredovanje mandata, izbor vođe i pokretanje promene mandata pomoću *timeout*-a.

Loše podešen *pacemaker* može smanjiti performanse ili odložiti napredovanje, ali ne bi trebalo da naruši bezbednost sve dok se pravila glasanja i zaključavanja striktno sprovode. Ovo razdvajanje takođe čini HotStuff pogodnim za kontrolisane varijacije. U sledećem odeljku



Slika 5: Tok *Chained HotStuff*-a. Potvrđivanje je protočno preko uzastopnih blokova; svaki novi blok nosi QC za svog roditelja, a blok postaje potvrđen kada je "zakopan" ispod dva potomka sertifikovana QC-om (pravilo 3-lanca).

uvodimo adaptivni *pacemaker* koji menja samo politiku *timeout*-a (kada replike odustaju od vođe i prelaze u viši mandat), dok bezbednosno jezgro HotStuff-a ostaje nepromenjeno.

IV. Implementacija

Naša implementacija je realizovana kao diskretna simulacija u jeziku Python. Kodna baza prati principe *Domain-Driven Design* (DDD) i obezbeđuje jasnu separaciju između (i) domenskih modela i protokolskog jezgra, (ii) mehanizma živosti (*pacemaker*), (iii) mrežnog/simulacionog sloja i (iv) orkestracije eksperimenata. Ovakva organizacija olakšava testiranje, ponovljivost i kontrolisano poređenje različitih strategija *pacemaker*-a bez menjanja bezbednosnih pravila HotStuff-a.

A. Slojna arhitektura

Sistem je organizovan kroz slojnu arhitekturu sa ciljem jasnog razdvajanja odgovornosti i jednostavnijeg testiranja. Kodna baza je podeljena na sledeće logičke slojeve:

- **Domenski sloj**: sadrži osnovne modele podataka (npr. `Block`, `QuorumCertificate`, `Vote`). Domenski modeli ne zavise od simulacije niti od mrežnog sloja.
- **Protokolski sloj**: enkapsulira konsenzusnu logiku i obradu protokolskih poruka. Klasa `Replica` predstavlja glavnog agenta, dok su pravila faza i tok poruka izdvojeni u `BasicHotStuffHandler`, a bezbednosne provere u `SafetyRules`.
- **Sloj *pacemaker*-a**: apstrahuje mehanizam živosti (rotaciju vođe, upravljanje *timeout*-ima i iniciranje promene mandata). Različite strategije (npr. fiksni *timeout*, adaptivni *timeout*) mogu se zameniti bez promena u bezbednosnom jezgru.
- **Infrastrukturni sloj**: obuhvata simuliranu mrežu (`SimulatedNetwork`), pomoćne komponente (logovanje, metrika) i zajedničke interfejse ka okruženju.
- **Simulacioni sloj**: orkestrira izvršavanje. `SimulationEngine` upravlja virtuelnim vremenom (`SimulationClock`) i raspoređivačem diskretnih događaja (`DiscreteEventScheduler`), obezbeđujući determinističko izvršavanje (uz fiksirano seme generatora pseudoslučajnih brojeva).

B. Simulacija mreže

Za razliku od produkcionog okruženja zasnovanog na TCP/IP, evaluacija se izvršava unutar diskretne simulacije. Ovaj pristup omogućava strogu kontrolu kašnjenja i redosleda isporuke poruka, kao i deterističku reprodukciju scenarija. Poruke se modeluju kao objekti i ubacuju u globalni prioritetni red događaja, indeksiran po planiranom vremenu isporuke.

Mrežni podsistem je implementiran u klasi `SimulatedNetwork` (Listing 1). Umesto serijalizacije bajtova i slanja preko soketa, metoda `send` izračunava kašnjenje, određuje vreme isporuke i planira događaj u prioritetni red.

```python
# hotstuff/network/simulated_network.py

class SimulatedNetwork(NetworkInterface):
    """
    Simulated network with configurable
    delays.
    """

    def send(self, message: BaseMessage,
    target_id: ReplicaId,
             current_time: int) -> int:

        # 1. Fault Injection: Check if target
    is partitioned
        if target_id in
    self._blocked_replicas:
            return -1

        # 2. Latency Simulation
        delay = self._random.randint(
            self._delay_min_ms,
            self._delay_max_ms
        )
        delivery_time = current_time + delay

        # 3. Schedule Event (Priority Queue)
        self._message_queue.enqueue(
            message=message,
            target_id=target_id,
            delivery_time=delivery_time
        )
```

```
        return delivery_time
```

Listing 1: Simulirana isporuka poruka

Ova apstrakcija omogućava simulaciju mrežne izolacije (*partition*) tako što se identifikator replike doda u skup `_blocked_replicas`, nakon čega mreža odbacuje poruke namenjene toj replici. Komponenta `SimulationEngine` periodično proverava red događaja i isporučuje poruke čije je vreme isporuke dospelo u trenutnom virtuelnom vremenu.

Svaka replika održava isključivo sopstveno lokalno stanje (stablo blokova, promenljive zaključavanja i skladište QC-ova), bez deljenja stanja između agenata. Time se zadržava vernost modelu distribuiranog sistema u kojem replike komuniciraju isključivo razmenom poruka.

C. *Implementacija* Basic HotStuff-*a*

Protokol *Basic HotStuff* enkapsuliran je u klasi `BasicHotStuffHandler`. Handler implementira tok poruka kroz četiri tipa faznih poruka (`PREPARE`, `PRE-COMMIT`, `COMMIT`, `DECIDE`) u skladu sa specifikacijom protokola [4]. Ulaz handler-a su dolazne poruke i trenutno lokalno stanje replike, dok su izlaz planirani događaji (npr. slanje glasova) i ažuriranja lokalnih promenljivih (npr. `highQC`, `lockedQC`).

1) Strukture podataka: Osnovna struktura podataka je `Block` (Listing 2), koji predstavlja čvor u stablu predloga. Da bi se efikasno podržao predikat bezbednog glasanja (`safeNode`), svaki blok čuva heš roditelja. To omogućava proveru da predlog nastavlja odgovarajuću granu, kao i navigaciju kroz lanac predaka.

2) Bezbednosna pravila: Ključni predikat bezbednog glasanja `safeNode` implementiran je u klasi `SafetyRules`. Ova komponenta proverava da li replika sme da glasa za predloženi blok na osnovu trenutnog zaključavanja (`locked_qc`) i QC-a koji dolazi uz predlog (ovde označen kao `justify_qc`). U skladu sa opisom iz Sekcije 3, replika može glasati ako važi makar jedan od uslova: (i) predlog nosi QC iz strogo većeg mandata u odnosu na zaključavanje, ili (ii) predloženi blok nastavlja zaključanu granu. Listing 3 prikazuje ovu proveru i pomoćnu funkciju za proveru odnosa predak/potomak.

3) Upravljanje stanjem replike: Centralni dokaz napretka u HotStuff-u je `QuorumCertificate` (QC), koji predstavlja kriptografski dokaz da je najmanje $2f+1$ replika glasalo za određeni blok u određenom mandatu i fazi. U simulaciji koristimo pojednostavljeni model QC-a (agregacija skupa glasova), umesto *threshold signatures*. Logičko značenje QC-a ostaje isto (dokaz podrške kvoruma), dok se veličina i trošak verifikacije razlikuju.

```python
# hotstuff/domain/models/block.py

class Block(BaseModel):
    """
    Represents a block (node) in the HotStuff
    tree.
    """
    parent_hash: Optional[BlockHash] = Field(
        default=None,
        description="Hash of the parent block"
    )
    command: Command = Field(
        description="The client command in
    this block"
    )
    height: int = Field(ge=0)
    proposer_id: ReplicaId
    view_number: ViewNumber

    @computed_field
    @property
    def block_hash(self) -> BlockHash:
        # Deterministic hash computation
        content =
    f"{self.parent_hash}|{self.command}|..."
        return BlockHash(
            hashlib.sha256(content.encode())
            .hexdigest()[:16]
        )
```

Listing 2: Implementacija modela `Block`

Replike održavaju sledeće ključne promenljive, koje direktno odgovaraju konceptima iz HotStuff opisa:

- `high_qc` (**replika/vođa**): najveći QC koji replika poznaje. U promeni mandata, replika ga šalje kroz `NEW-VIEW`, a vođa novog mandata bira maksimum preko pristiglih poruka i koristi ga kao *opravdanje* sledećeg predloga.
- `locked_qc` (**replika**): QC na kojem je replika zaključana; `SafetyRules` zabranjuje glasanje za predloge koji su u konfliktu sa ovim zaključavanjem, osim ako je predlog opravdan QC-om iz višeg mandata.

Napomena: u implementaciji se mogu pojaviti pomoćne promenljive kao što je `prepare_qc` (npr. „poslednji poznati QC u tekućem toku faza“), ali semantika napretka i prenosa preko `NEW-VIEW` poruka se oslanja na `high_qc` kao jedinstvenu „najbolju“ informaciju koju replika trenutno poseduje.

4) Faze protokola i obrada poruka: `BasicHotStuffHandler` orkestrira faze protokola. Listing 4 ilustruje obradu `PREPARE` poruke: replika

```python
# hotstuff/protocol/safety_rules.py

def is_safe_node(
    self,
    block: Block,
    justify_qc: Optional[QuorumCertificate],
    locked_qc: Optional[QuorumCertificate]
) -> bool:
    """
    Implements safeNode(node, qc) from Algorithm 1.
    """
    # 1. Liveness Rule: If valid highQC > lockedQC
    if (locked_qc is None or
        (justify_qc and justify_qc.view_number > locked_qc.view_number)):
        return True

    # 2. Safety Rule: Node extends from lockedQC.node
    if self._extends_from(block, locked_qc.block_hash):
        return True

    return False

def _extends_from(self, block: Block, ancestor_hash: BlockHash) -> bool:
    """
    Recursively checks if block is a descendant of ancestor_hash.
    """
    if block.block_hash == ancestor_hash:
        return True

    current = block
    while current.parent_hash is not None:
        if current.parent_hash == ancestor_hash:
            return True
        current = self._block_registry.get(current.parent_hash)
        if current is None:
            break

    return False
```

Listing 3: Implementacija predikata `safeNode`

najpre proverava predikat `safeNode`, i tek ako je predlog bezbedan, šalje glas vođi. Time se sprečava da ispravne replike glasaju za konfliktnu granu u situacijama kada to bezbednosna pravila ne dozvoljavaju.

Analogno `PREPARE` fazi, handler obrađuje `PRE-COMMIT` i `COMMIT` poruke, formira odgovarajuće QC-ove nakon prikupljanja kvoruma glasova i ažurira zaključavanje (`locked_qc`) kada replika uoči validan `precommitQC` (u skladu sa Sekcijom 3 i [4]). Finalizacija i izvršavanje se modeluju kroz `DECIDE` poruku, nakon koje replika obeležava odgovarajuće blokove kao potvrđene i izvršava komande redom.

D. Implementacija pacemaker-*a*

Iako HotStuff formalizuje modul `Pacemaker` za rotaciju vođe i iniciranje promene mandata, konkretna strategija *timeout*-a ostaje apstraktna. Zbog toga uvodimo modularni interfejs *pacemaker*-a, tako da se različite strategije mogu menjati bez uticaja na bezbednosno jezgro. Implementirane su dve varijante: (i) osnovni *pacemaker* sa fiksnim *timeout*-om i (ii) adaptivni *pacemaker* namenjen promenljivim mrežnim uslovima.

1) Adaptivni pacemaker*:* Klasa `AdaptivePacemaker` (Listing 5) dinamički podešava trajanje *timeout*-a na osnovu posmatranog trajanja mandata. Implementacija koristi eksponencijalni klizni prosek (EMA) i bezbednosnu marginu β:

$$T_{next} = \beta \cdot (\alpha \cdot T_{obs} + (1 - \alpha) \cdot T_{prev}).$$

U slučaju uzastopnih neuspeha (istek *timeout*-a), primenjuje se eksponencijalni *backoff* sa gornjom granicom, čime se modeluje tipično ponašanje BFT implementacija

```
# hotstuff/protocol/basic/handler.py

def handle_prepare(
    self,
    message: PrepareMessage,
    current_view: ViewNumber,
    locked_qc: Optional[QuorumCertificate],
    ...
) -> ...:
    block = message.block

    # Verify safety before voting
    if not self._safety_rules.is_safe_node(
        block, message.high_qc, locked_qc
    ):
        return [], block, last_voted_view, PhaseType.PREPARE

    # Send Vote to Leader
    vote = MessageFactory.create_prepare_vote(
        sender_id=self._replica_id,
        view_number=current_view,
        block_hash=block.block_hash,
        target_id=self._leader_scheduler.get_leader(current_view)
    )
    self._network.send(vote, ...)

    return [vote_event], block, current_view, PhaseType.PREPARE
```

Listing 4: Obrada `PREPARE` faze

u delimično sinhronom modelu (postepeno „nadmašivanje“ nepoznatog Δ nakon GST).

E. Simulirane greške

Da bismo proverili BFT osobine protokola, implementacija replike podržava kontrolisano ubrizgavanje grešaka. Modelujemo tri ponašanja: *crash fault* (replika prestaje da obrađuje poruke), *silent* režim (replika ne emituje odgovore), i *random drop* (replika nasumično odbacuje deo poruka). Scenariji se aktiviraju kroz metodu `inject_fault` i skriptuju u okviru eksperimentalnog okruženja.

Listing 6 prikazuje primer implementacije u obradi poruka. U *silent* režimu replika i dalje može da „proguta“ poruku bez slanja izlaznih događaja (tj. bez glasova), dok *random drop* uvodi stohastično gubljenje poruka radi simulacije nestabilne isporuke ili lokalnog kvara u komunikaciji.

F. Ograničenja simulacije i odstupanja u odnosu na literaturu

a) Simulirana mreža naspram realnog TCP/IP: Originalne evaluacije HotStuff-a se oslanjaju na realan mrežni stek i sistemsko vreme. U našem okruženju koristi se virtuelno vreme, pri čemu se mrežno kašnjenje modeluje kao promenljiva koja se dodaje na vremensku oznaku isporuke. Ovo značajno povećava reproduktivnost i kontrolu, ali metričke vrednosti poput „propusnosti“ odražavaju kapacitet simulacionog modela, a ne fizičke performanse u realnom sistemu.

b) Bez grupisanja komandi: Literatura tipično koristi grupisanje komandi (*batching*) kako bi se amortizovao kriptografski i komunikacioni trošak. Naš simulator prati životni ciklus pojedinačnih komandi radi granularne provere korektnosti, pa blokovi trenutno sadrže jednu komandu. Zbog toga propusnost (operacije/sekundi) nije direktno uporediva sa rezultatima koji se oslanjaju na agresivno grupisanje u blokovima.

Kompletan izvorni kod simulatora i eksperimenata dostupan je onlajn [5].

```python
# hotstuff/pacemaker/adaptive_pacemaker.py

class AdaptivePacemaker(PacemakerInterface):
    """
    Adjusts timeout based on EMA of view duration.
    """
    def on_view_success(self, view_number, duration_ms):
        # 1. Reset backoff
        self._consecutive_timeouts = 0

        # 2. Update EMA
        ema = (self._alpha * duration_ms +
               (1 - self._alpha) * self._current_timeout)

        # 3. Set new timeout with safety margin
        self._current_timeout = int(ema * self._safety_margin)

    def on_timeout(self, current_time):
        # Exponential Backoff on failure
        self._consecutive_timeouts += 1
        mult = min(2 ** self._consecutive_timeouts, 4)
        self._current_timeout = min(
            self._current_timeout * mult,
            self._max_timeout
        )
        return self._current_view + 1
```

Listing 5: Adaptivna logika *timeout*-a

```python
# hotstuff/protocol/replica.py

def handle_message(self, message: BaseMessage, ...) -> List[dict]:
    # 1. Crash Fault: Stop processing entirely
    if self._is_faulty and self._fault_type == FaultType.CRASH:
        return []

    # 2. Silent Fault: Process but suppress outgoing actions
    if self._is_faulty and self._fault_type == FaultType.SILENT:
        self._logger.warning("SILENT: suppressing response")
        handler = handlers.get(message.message_type)
        _ = handler(message, current_time)  # state may be updated internally
        return []

    # 3. Random Drop: Drop 50% of messages
    if self._is_faulty and self._fault_type == FaultType.RANDOM_DROP:
        if random.random() < 0.5:
            return []

    # Normal processing...
    handler = handlers.get(message.message_type)
    return handler(message, current_time)
```

Listing 6: Ubrizgavanje grešaka u obradi poruka replike

V. SIMULACIJA I EVALUACIJA

Evaluaciju sprovodimo korišćenjem diskretno-događajne simulacije opisane u Sekciji IV. Svi eksperimenti koriste identičnu implementaciju HotStuff jezgra i isti simulacioni mrežni model. Konfiguracije se razlikuju isključivo u veličini komiteta, aktiviranim modelima grešaka i odabranoj strategiji *pacemaker*-a. Tokom pokretanja posmatramo (i) signale korektnosti

Tabela II: Ključni parametri simulacije i izvori konfiguracije.

Parametar	**Vrednost / izvor**
Trajanje pokretanja	kontrolisano preko `max_views` (`benchmark_config.yaml`); stvarno trajanje: `simulation_duration_ms`.
Broj ponavljanja R	`runs_per_config` (`benchmark_config.yaml`).
Kašnjenje poruka	`network_delay_min_ms`=10ms, `network_delay_max_ms`=50ms (`hotstuff/config/settings.py`).
Rotacija vođe	Round-Robin: $leader(v) = v \bmod n$ (`hotstuff/protocol/leader_scheduler.py`).
Fiksni timeout	`base_timeout_ms` (`benchmark_config.yaml`).
Adaptivni PM (EMA)	α=`adaptive_alpha`=0.125 (`hotstuff/config/settings.py`).
Sigurnosna margina	`safety_margin`=1.5 (hardcoded) (`adaptive_pacemaker.py`, linija 131).
Maks. timeout	`adaptive_max_timeout_ms`=5000ms (`hotstuff/config/settings.py`).

(bezbednost: odsustvo konfliktnog komitovanja) i (ii) signale performansi (propusnost, mrežno kašnjenje i dinamiku promena mandata).

A. Metodologija

Simulator izvršava replike i mrežni sloj u jednom procesu, upravljajući konkurentnim akterima pomoću globalnog prioritetnog reda zakazanih događaja. Vreme je izraženo u **logičkim milisekundama** i koristi se isključivo za zakazivanje i isporuku poruka u simulaciji. Slučajnost (npr. kašnjenja poruka i stohastičke odluke u modelima grešaka) generiše se preko **seedovanog** generatora pseudoslučajnih brojeva na nivou simulatora, čime se obezbeđuje reproduktivnost za istu konfiguraciju i isti *seed*.

Za svaku konfiguraciju izvršavamo R nezavisnih pokretanja i agregiramo rezultate. Osim ako nije drugačije navedeno, protokolsko jezgro i politika rotacije vođe su identični u svim scenarijima, dok se menjaju mrežni parametri, tip i broj grešaka, i strategija *pacemaker*-a.

1) Reproduktivnost i parametri simulacije: Tabela II sumira ključne parametre simulacije koji utiču na rezultate. U zagradi navodimo izvor parametra (konfiguracioni fajl ili modul) radi lakše proverljivosti.

2) Metrike i definicije: Za svako pokretanje simulacije prikupljamo sledeće metrike (implementaciona referenca: `hotstuff/benchmark/runner.py` i `results.py`):

- **Stopa uspeha (%)**: jedno pokretanje je *uspešno* ukoliko komituje bar jedan blok, tj. ako važi `total_blocks_committed` > 0. Stopa uspeha je procenat uspešnih pokretanja preko svih ponavljanja R za datu konfiguraciju.
- **Propusnost (blokovi/s)**: brzina komitovanja blokova tokom trajanja simulacije. Pošto u našem modelu važi **1 blok = 1 komanda**, propusnost u blokovima/s direktno odgovara komande/s. Intuitivno, u stabilnim uslovima broj komitovanih blokova je usko povezan sa brojem mandata kroz koje sistem uspešno prođe.
- **Mrežno kašnjenje (ms)**: vreme od trenutka kada vođa predloži blok (emitovanje `PREPARE` poruke za taj blok) do trenutka kada blok postane komitovan (`DECIDE`/commit događaj). Prikazujemo srednju vrednost i metrike p50/p95/p99.
- ***Timeout*-i**: ukupan broj `TIMEOUT` događaja koje je *pacemaker* emitovao tokom pokretanja (implementaciona referenca: `hotstuff/metrics/collector.py`). Ova metrika je *brojač svih timeout okidanja* i zato može biti visoka čak i u režimu bez ubrizganih grešaka. Tumači se kao indikator učestalosti timeout mehanizma u izabranoj konfiguraciji, a ne nužno kao broj „realnih“ neuspeha lidera.

B. Evaluacioni scenariji

Protokol procenjujemo kroz četiri scenarija, kako bismo obuhvatili skaliranje, ponašanje pod greškama, uticaj adaptivnog *pacemaker*-a i pragovno ponašanje pri prekoračenju BFT uslova.

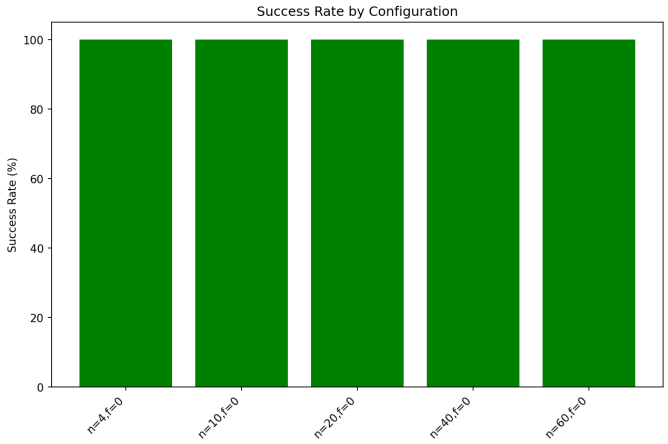
1) Scenario 1: Skaliranje bez ubrizganih grešaka: Merimo propusnost i mrežno kašnjenje dok veličina komiteta raste od $n = 4$ do $n = 60$, u režimu bez ubrizganih grešaka (sve replike su korektne). Cilj scenarija je da se karakterišu performanse u najboljem slučaju, kada nema protivničkog ponašanja i nema aktiviranih modela grešaka. Rezultati su sumirani u Tabeli III.

a) Analiza: Sve testirane konfiguracije su završile uspešno. U ovom scenariju nisu uočene povrede bezbednosti: nije detektovano konfliktno komitovanje, a komitovana istorija ostaje konzistentna između korektnih replika.

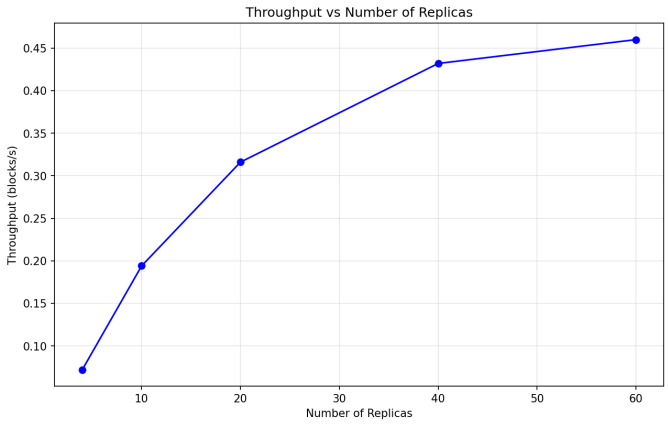
Slika 7 pokazuje da propusnost raste sa n (od 0.07 do 0.46 blokova/s), ali sa opadajućim dobitkom posle približno $n \approx 40$. Ovaj trend ukazuje na zasićenje usled koordinacionog troška po mandatu (obrada poruka

Tabela III: Scenario 1 (bez ubrizganih grešaka): stopa uspeha i propusnost kroz različite veličine komiteta.

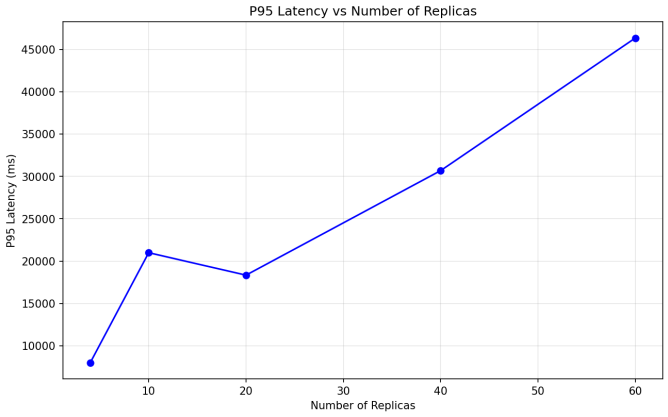
n	**Stopa uspeha**	**Prosečna propusnost (blokovi/s)**	**Komitovani blokovi**
4	100%	0.07	∼7
10	100%	0.19	∼19
20	100%	0.31	∼32
40	100%	0.43	∼43
60	100%	0.46	∼46



Slika 6: Scenario 1: sva pokretanja su završena uspešno (100% stopa uspeha).



Slika 7: Scenario 1: propusnost u funkciji veličine komiteta n (bez ubrizganih grešaka).



Slika 8: Scenario 1: repna latencija (p95) u funkciji veličine komiteta n (bez ubrizganih grešaka).

i prikupljanje glasova od strane vođe), posebno u režimu bez grupisanja komandi (1 blok = 1 komanda).

Istovremeno, mrežno kašnjenje raste sa n. Slika 8 pokazuje da p95 mrežno kašnjenje raste približno sa 8 s ($n = 4$) na oko 46 s ($n = 60$) u logičkom vremenu. Veći komitet povećava verovatnoću da makar jedna spora putanja (npr. kasna isporuka ili zakasneli glas) produži kritičnu putanju mandata. Nemonotoni pad oko $n = 20$ verovatno je posledica varijanse u raspoređivanju događaja u simulaciji, dok dominantni trend rasta ostaje izražen za veće n.

2) Scenario 2: Tolerancija grešaka i oporavak: Procenu živosti pod greškama vršimo konfigurisanjem komiteta sa $n = 10$ replika i ubrizgavanjem $f = 3$ neispravnih replika (maksimalno tolerisano za uslov $n \geq 3f + 1$). Upoređujemo tri ponašanja grešaka:

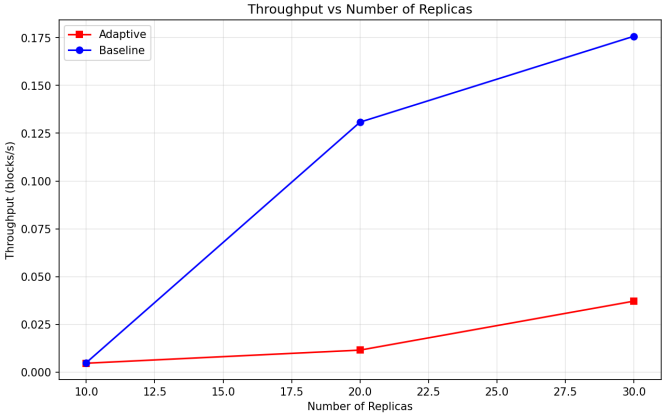
- ***Crash***: neispravne replike potpuno prestaju da obrađuju i šalju poruke.
- ***Silent***: neispravne replike potiskuju učešće u protokolu i ne emituju poruke.
- ***Random Drop***: neispravne replike nasumično odbacuju 50% izlaznih poruka.

Merimo stopu uspeha i sposobnost sistema da komituje blokove u svakoj varijanti.

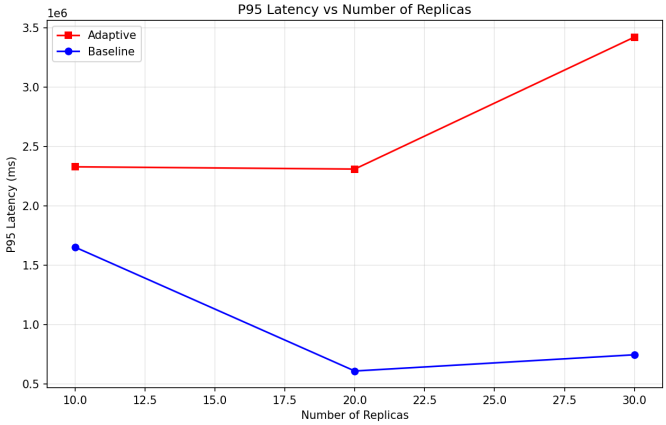
a) Analiza: Tabela IV poredi performanse za $n = 10$ pod različitim tipovima grešaka sa referentnim režimom bez ubrizganih grešaka. U svim slučajevima nisu uočene povrede bezbednosti. Degradacija performansi zavisi od ponašanja greške: *crash* i *silent* varijante najviše smanjuju propusnost, jer sistem češće ne uspeva da prikupi kvorum i napreduje tek nakon rotacije mandata. *Random Drop* u proseku degradira propusnost manje,

Tabela IV: Scenario 2: uticaj tipa greške na performanse ($n = 10$). Referenca je režim bez ubrizganih grešaka za $n = 10$ iz Scenarija 1.

Tip greške	**Stopa uspeha**	**Prosečno blokova**	**Prosečno *timeout*-a**	**Uticaj na propusnost**
Nema (referenca)	100%	∼20	∼975	0%
Crash ($f = 3$)	100%	3.4	699	-83.0%
Silent ($f = 3$)	100%	4.8	995	-76.0%
Random Drop ($f = 3$)	100%	15.0	985	-25.0%



Slika 9: Scenario 3: propusnost pod `RANDOM_DROP` (50% gubitka). Varijanta sa fiksnim *timeout*-om postiže konzistentno veću propusnost.



Slika 10: Scenario 3: repna latencija (p95) pod `RANDOM_DROP` (50% gubitka). Adaptivni *pacemaker* pokazuje znatno veću repnu latenciju.

što je konzistentno sa time da povremeno učešće ipak omogućava da se sporadično dostigne kvorum.

3) Scenario 3: Adaptivni naspram fiksnog pacemaker*-a pod gubitkom poruka:* Upoređujemo `AdaptivePacemaker` sa baznom varijantom sa fiksnim *timeout*-om u uslovima nestabilne isporuke, koristeći `RANDOM_DROP` model (50% gubitka). Testiramo konfiguracije sa $n \in \{10, 20, 30\}$ i fiksnim $f = 3$, kako bismo izolovali efekat strategije *timeout*-a.

a) Rezultati: Suprotno početnoj hipotezi, ***pacemaker* sa fiksnim *timeout*-om** nadmašuje adaptivnu varijantu za sve testirane veličine komiteta. Bazna varijanta postiže veću propusnost i nižu repnu latenciju u svim tačkama.

b) Analiza: U režimu stohastičkog odbacivanja poruka, duže čekanje ne povećava pouzdano verovatnoću formiranja kvoruma. Adaptivni *pacemaker* izostanak glasova/sertifikata interpretira kao povećano kašnjenje i reaguje povećanjem *timeout*-a kroz EMA, što u ovom režimu može povećati prazno čekanje i smanjiti frekvenciju pokušaja. Nasuprot tome, fiksni *timeout* efektivno realizuje strategiju „brzo odustani“, proizvodeći više pokušaja kroz češće rotacije mandata, što povećava šansu da se kvorum formira uprkos gubicima.

4) Scenario 4: Prekoračenje praga grešaka: Da bismo ilustrovali granice BFT živosti, konfigurisali smo komitet od $n = 10$ replika sa $f = 4$ replike sa *crash* greškom, čime se krši uslov $n \geq 3f + 1$.

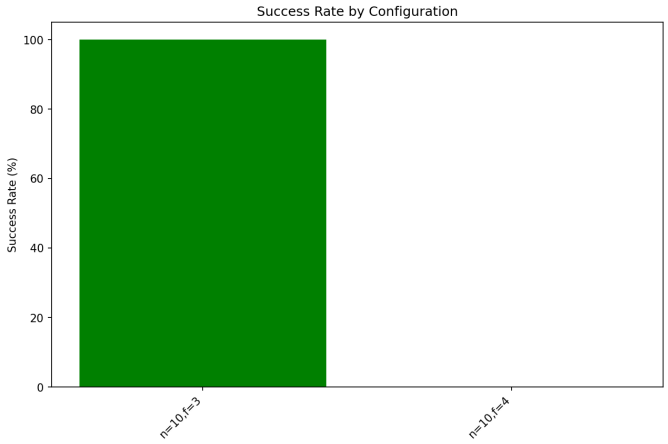
$$f_{\max} = \left\lfloor \frac{n-1}{3} \right\rfloor = \left\lfloor \frac{9}{3} \right\rfloor = 3.$$

a) Rezultati: Slika 11 pokazuje pragovno ponašanje: sa $f = 3$ simulacija završava uspešno, dok sa $f = 4$ sistem ne komituje nijedan blok kroz pokretanja.

b) Analiza: Ishod sledi direktno iz zahteva za kvorum. Sa $n = 10$ i $f_{\max} = 3$ potreban je kvorum od $2f_{\max} + 1 = 7$ glasova. Kada $f = 4$ replike prestanu da odgovaraju (*crash*), preostaje najviše 6 replika koje mogu glasati, pa nijedan vođa ne može sakupiti dovoljan broj glasova da formira QC. Zato sistem kontinuirano rotira mandate (TIMEOUT događaji) bez komitovanja blokova (otkaz živosti), bez narušavanja bezbednosti.

Dodatak A
Interaktivni interfejs za evaluaciju

Radi podrške evaluaciji implementirali smo veb-kontrolnu tablu (*dashboard*) za simulator. Interfejs omogućava podešavanje parametara pokretanja i kontrolu



Slika 11: Scenario 4: stopa uspeha kolabira nakon prekoračenja praga grešaka. Konfiguracija $n = 10, f = 4$ ne ostvaruje napredak.

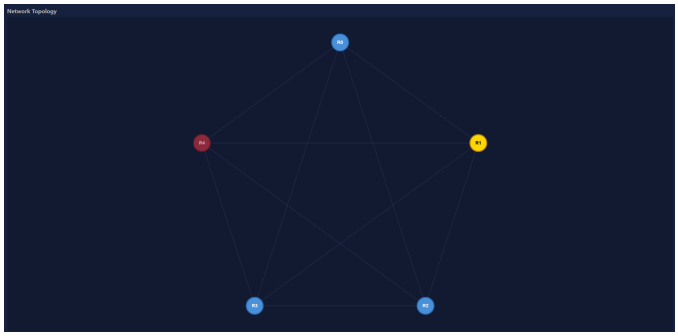
izvršavanja, pregled aktivnosti poruka preko mrežne topologije u realnom vremenu i inspekciju lokalnog protokolskog stanja po replici.

A. *Konfiguracija*

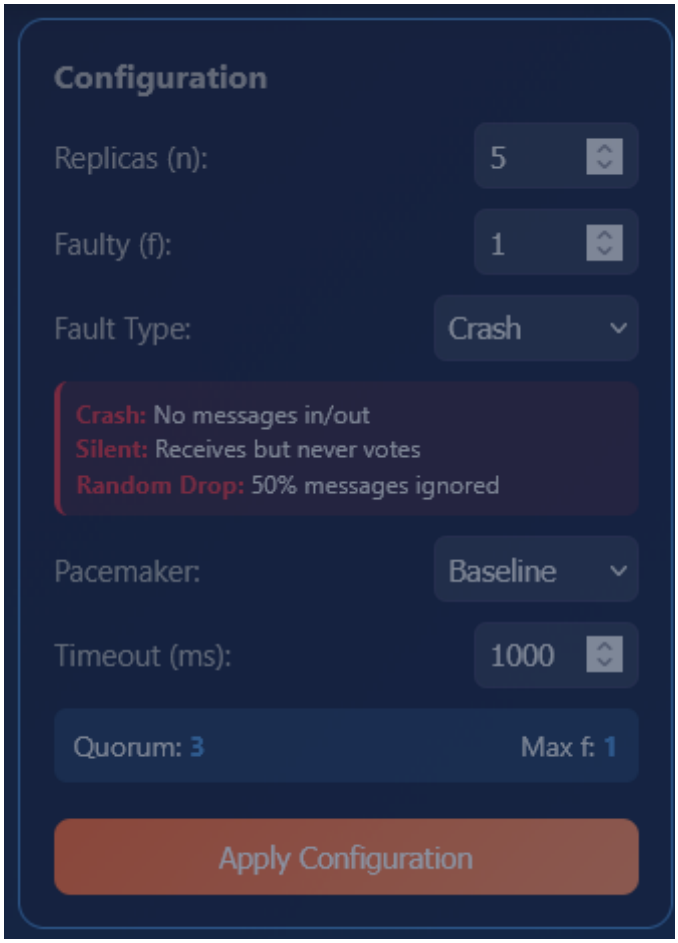
Korisnik može da podesi veličinu komiteta N, količinu neispravnih replika f, model greške (npr. `Crash`, `Silent`, `RandomDrop`), strategiju *pacemaker*-a i osnovni *timeout*. Izvršavanje se može pokrenuti, pauzirati ili resetovati. Simulacija se može napredovati korak-po-korak ili u više koraka odjednom, dok automatski režim kontinuirano obrađuje red događaja podešivom brzinom.

B. *Vizuelizacija*

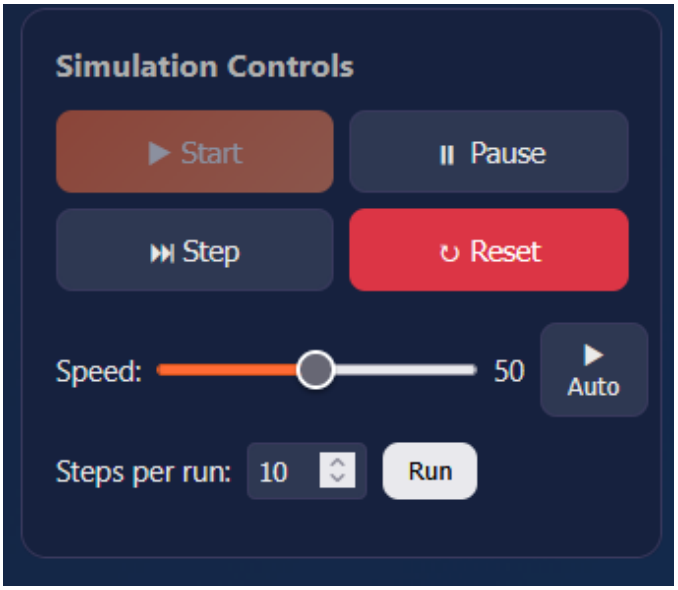
Prikaz topologije ističe trenutnog lidera (žuto), neispravne replike (crveno) i obične replike (plavo), i prikazuje prenose poruka preko obojenih strelica. Inspektor replike prikazuje lokalno stanje (npr. mandat, posmatrane QC-ove, poslednji glas i brojače komitovanja). Log beleži isporuke poruka i tranzicije (pošiljalac, primalac, tip poruke, vremenska oznaka).



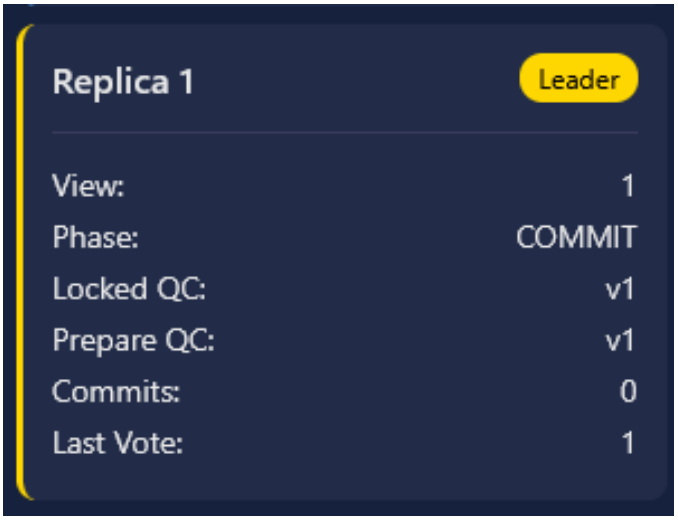
Slika 12: Prikaz mrežne topologije. Trenutni lider je istaknut žutom bojom, neispravne replike crvenom, a obične replike plavom. Ivice predstavljaju autentifikovane komunikacione kanale.



Slika 13: Panel za konfiguraciju parametara pokretanja (npr. N, f, model greške, tip *pacemaker*-a i osnovni *timeout*).



Slika 14: Kontrole izvršavanja tokom eksperimenata: pokretanje/pauza/reset, determinističko izvršavanje korak-po-korak i automatski režim sa podešivom brzinom.



Slika 15: Inspektor replike koji prikazuje lokalno protokolsko stanje (npr. *view*, fazu, posmatrane QC-ove, poslednji glas i brojače komitovanja).

Slika 16: Log koji prikazuje aktivnost poruka sa vremenskim oznakama, pošiljaocem/primaocem i tipom poruke.

LITERATURA

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, pages 173–186, 1999.

[2] Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[3] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, 1983.

[4] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC '19)*, pages 347–356. ACM, 2019.

[5] Boris Čuljak. Adaptive-pacemaker-hotstuff: Discrete-event hotstuff simulator with an adaptive pacemaker. GitHub repository, 2025. Accessed: 2025-12-24.