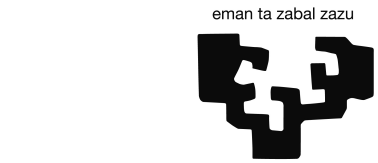


Universidad del País Vasco
Euskal Herriko Unibertsitatea

Informatika Ingeniaritza

Karrera Amaierako Proiektua

TrustedPals plataformarako middlewarearen inplementazioa eta simulazioa

---

Egilea
Aingeru Itoiz Zabalza

2014

Laburpena

Sistema banatuak zenbait konputagailu edo gailu autonomoaz osaturiko sareak dira, non algoritmo *banatuen* bidez partaide guztien lana koordinatzen da entitate bakarra izatearen irudia emanez. Eredu honi esker sistemaren sendotasuna handitzen da, posible baita sistemak aurrera jarraitzea zenbait partaidek huts egin arren.

Sistema banatuak diseinatzeak badu zenbait zailtasun, prozesu guztien arteko koordinazioa lortu behar baita. Erronka nagusietako bat adostasuna edo *consensus* lortzea da; hau da, prozesu guztiak ados jartzea zerbait erabaki behar dutenean. Ingurune desberdinetan planteatu badaiteke ere, lan honetan *Byzantine* ingurunean egingo da. Ingurune honetan partaideen hutsegiteak ausaz gerta daitezke eta edozein momentutan. Horrez gain, hutsegite horiek edozein motakoak izan daitezke, hala nola, prozesu bat bertan behera geratzea edota prozesu baten eskaera okerra edo lekuz kanpokoa egitea.

Aurkeztutako *consensus* arazoa garrantzi handikoa da sistema banatuen arloan, honen bitartez beste hainbat helburu lortu baitaitezke. Horien artean *Secure Multy-party Computation* (SMC) dugu, non sare banatu bateko partaide guztiek adostasuna lotu behar dute partaide bakoitzaren informazioa gainontzekoei ezkutatuz. Horren adibide bezala "aberatsaren arazoa" azaldu ohi da, non partaide guztiek aurkitu behar dute zein den beraien artean aberatsena, partaide bakoitzak gainontzekoen "aberastasuna" ezagutu ahal izan gabe. SMC erabili daiteke soluzioa emateko planteamendu bera jarraitzen duten aplikazio erreal askori, hala nola, enkante pribatuak edo bozketak.

SMC inplementatu ahal izateko TrustedPals izeneko plataforma dugu, non diseinu modularra jarraituz *smartcard* bat eta algoritmo banatuak konbinatzen dira lehenengo consensus eta ondoren SMC lortzeko.

Karrera amaierako proiektu honen helburua TrustedPals proposamenaren alde praktikoa jorratzea izango da. Horretarako proposamenaren algoritmo banatuak inplementatu eta simulatuko dira zenbait probetako kasuetan. Simulazioak bideratzeko gertaera diskretuko NS-3 simulagailuan erabiliko da. Simulazio eszenario desberdinak inplementatuko dira eta ondoren emaitzak aztertuko dira.

Eskerrak

Lehenik eta behin proiektu hau garatzen lagundu nauten bi pertsonak eskertu nahi ditut elkartzean jasotako harrera bikainarengatik, gomendioengatik, pazientziarengatik eta laguntzarengatik. Roberto Cortiñas eta Mikel Larrea, eskerrik asko.

Nire alboan egon zareten lagun guztiei, batez ere hain hurbil egon diren horiei, Ioseba eta Lander eskerrik asko.

Maiteri, beti nire ondoan egoteagatik eta hainbeste pazientzia izateagatik. Zure animoak eta presentzia amaitzera lagundu dute, eskerrik asko.

Eta nola ez, nire eskerrik beroenak etxekoei. Aita eta Ama eskerrik asko. Zuen presentzia beti igarri dut eta emandako berotasuna ezinbestekoa izan da proiektua aurrera eramateko. Eskerrik asko zuen laguntzarengatik.

Aingeru

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1 KAPITULUA

Sarrera

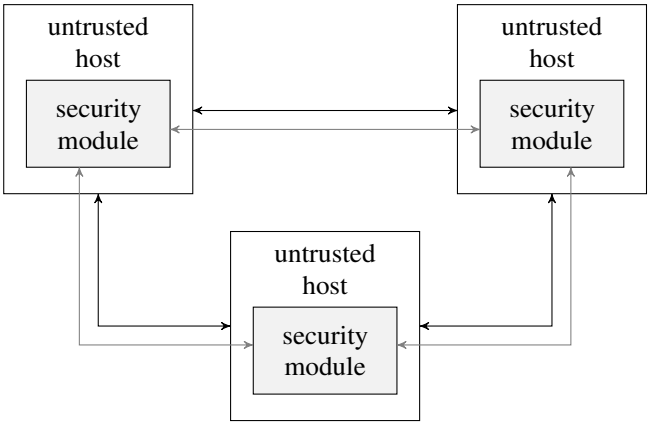
Gaur egun gizartean hain barneratuta dagoen Internet-en, segurtasun arazo ugari ager daitezke. Denbora pasatzen den ahala, jendeak fisikoki egiten zituzten operazioak, erosketak edo banku kontuen eguneraketa edo transakzioak adibidez, Internet bidez egitera pasa dira gaur egun. Hauek burutzeko, egin beharreko transakzioak segurtasunez burutu behar dira eta pribatutasunaren bermatzera behartzen gaitu. Hauetaz gain, beste kasu ugari aipa daitezke, hala nola, bozketa banatuak, segurtasun agirien edo desenkriptatze funtzioen sakabanatzea, enkanteak eta beste kasu askoren abar luze bat. Parte hartzaile guzti hauen artean pribatutasuna bermatzen dela ziurtatu behar da, partaideen artean funtzio komun bat exekutatu behar delako, adibidez, erosketa, bozketa edo adostasuna. Parte hartzaileek beraiengan ez dute konfiantza, ezta komunikazio kanaletan ere, ondorioz, eragiketak burutzeko zailtasun gehiago aurkitzen dira. Arazo honi *Secure Multi-party Computation* (SMC) [Yao, 1982] bezala ezagutzen zaio.

Zoritxarrez, SMC arazoari ebazpen bat lortzea (suposaketa estrarik hartu gabe) oso garestia da, komunikazio terminoetan (mezu kantitateengatik), egokitzapenean (erredundantziarengatik) eta denboran (sinkronizazio txanda kantitateak direla eta). SMC arazoaren konplexutasunaz gain, aintzat hartu behar da exekutatu beharreko sistemaren ezaugarriek are eta gehiago zaildu dezaketela soluzioaren bilaketa. Proiektu honetan planteatuko den kasuan, sisteman prozesu batzuk huts egin dezakete, hau da, prozesu guztiak ez direla *zuzenak*. Horrez gain, prozesamendu eta komunikazio denboran malgutasun puntu bat onartzen da.

Gaur egunean SMC arazoa konpontzeko ugari diren tekniken artean, *Trusted Third Party* (TTP) dugu. Teknika honen erabilerak SMC arazoaren soluzioa errazten du. Kriptografian, *Trusted Third Party* (TTP), bi parte hartzaileak fidatzen diren entitate bat da, eta ondorioz beraien arteko interakzioak sinplifikatzen dira. Modu deskriptiboago batean azaltzearren, TTP teknikaren erabilerak sare erreal baten gainean alegiazko sare seguru baten sorrera izango litzateke.

TrustedPals, [Fort et al., 2006] Secure Multy-party Computation (SMC) arazoa modu eraginkor batean ebazteko smartcard-etan oinarritutako framework bat da. TrustedPals, sistema banatu batean maltzurkerien aurkako segurtasun modulu batez hornituta dauden prozesuak kontsideratzen ditu (ikusi 1.1 irudia). Prozesu hauen artean SMC arazoari ebazpen bat lortzeko, prozesu bakoitzaren segurtasun moduluez baliatuz, *Trusted Third Party* (TTP) bat

simulatzean datza.



1.1 Irudia: Maltzurkerien aurkako segurtasun moduluak dituzten prozesuak.

Segurtasun modulu hauekin, prozesuek edukiko dituzten erroreak *crash* edo bertan behera geratzea eta mezu omisioak izatera murriztuko dira, arbitrarioki edo maltzurkeriaz dauden atakeak saihestuko direla bermatuz. Honez gero, TrustedPals-ek SMC segurtasun arazotik, hutsegite-tolerantzia sinkronizazio problema batera murriztuko du. *Secure Failure Detection and Consensus in TrustedPals* artikuluan [Cortiñas et al., 2012] ikusten denez, sinkronizazio arazo ebaztea ez da soluzio nabaria, batez ere hain ez-ohikoa diren mezu omisio erroreen ingurune batean. Murrizte honek, konprobaketa berri batzuk eskatuko ditu errore-tolerantzia algoritmoan, segurtasun moduluan barneratutakoa. Hau soluzio nabaria izateaz nahiko urrun dagoena.

Proiektuaren helburua

Secure Failure Detection and Consensus in TrustedPals [Cortiñas et al., 2012] artikuluan, TrustedPals-en moldaketa batzuk proposatzen dira SMC-ren arazoa modu eraginkorrago batean ebazteko. TrustedPals-en diseinu modularra aurkezten da, *Consensus* eta *Failure Detector* modulu bezala erabilita. Moldaketa berri honekin, hutsegite mota murrizketa sortzen duen hutsegite-tolerantzia sinkronizazio arazoa, sistema asinkronoago batean nola lan egin daitekeen ikusiko da.

Diseinu berri honen ingurunea aztertu eta ezagutu ondoren, bertan azaldutako kontzeptu guztien eta garatutako algoritmo berrien inplementazioa burutuko da. Aurrera eramateko, merkatuan dauden simulazio sistemen azterketa eta konparaketa bat egingo da, gure ezaugarrietara gehien moldatzen dena aukeratuz. Ondoren, simulagailuan aztertutako deseinu berriaren sistema osoa inplementatuko da eta proba kasu batzuk erabiliz algoritmoen portaerak aztertuko dira, irteera parametroei interpretazio bat emanaz.

Memoriaren egitura

Memoriaren gainontzeko edukia ondorengo eran egituratuta dago. 2 kapituluan proiektuaren helburuak eta horiek lortzeko aztertutako atazak aurkezten dira, dagozkien estimazioekin eta planifikazioarekin batera. 3 kapituluan proiektuaren gaiaren testuingurua eta aztertu beharreko algoritmo banatuak azaltzen dira, simulatu beharreko eszenario teorikoen deskribapenarekin batera. Ondoren, 4 kapituluan aukeratutako simulagailuaren eta bere aukeraketaren nondik norakoak azaltzen da. 5 eta 6 kapituluek egindako inplementazioaren analisia eta simulatutako kasuak (lortutako emaitzekin batera) aurkezten dituzte hurrenez hurren. Azkenik, 7 atalean proiektuaren ondorioak eta hobespenak azaltzen dira. Memoriaren bukaeran zenbait eranskin gehitu dira informazio gehigarria eskuragarri uzteko: jorratutako simulagailuaren erabilera gida (A eranskina), erabilitako terminoen aurkezpena (B eranskina) eta kontsultatutako erreferentzia zerrenda gaika sailkatuta (C eranskina).

2 KAPITULUA

Proiektuaren helburuen dokumentua

2.1. Proiektuaren deskribapena

Proiektuaren helburua *Secure Failure Detection and Consensus in TrustedPals* [Cortiñas et al., 2012] lanean garatutako algoritmo banatuaren inplementazioa, simulazioa eta simulazioaren funtzionamendua aztertzean datza.

Proiektua aurrera eramateko eman beharreko pauso guztiak atal honetan azalduko dira: sistemaren ezagutzarako burutu beharreko ikasketak, eraikitako sistemaren nondik-norakoak, beharrezko izango diren tresnen ikerketa, inplementazio lanak, proba kasuen sorrera eta proba irteeren interpretazioa. Pauso guzti hauen inplementazioa, 1 atalean azaldutako egiturarekin antolatu dira.

Jada aipatu den bezala, *Secure Failure Detection and Consensus in TrustedPals* lanean proposatzen diren algoritmoen inplementazioa eta probak egikarituko dira, jarraian eraikitako proben ondorioak argitaratuz. Proiektu honetan, *Secure Multy-party Computation* (SMC) [Yao, 1982] Byzantine ingurune batean TrustedPals bidez soluzio eraginkorrago bat lortzeko egin beharreko pausoak erakutsiko dira. Diseinu berri honetan, errore detektatzaile batez baliatuz, *general omission* modeloan *consensus* soluzio bat lortzea ikertu da. *General omission* hutsegite modeloan, prozesuek bertan behera geratuz eta jasotako edo bidalitako mezuak ezikusi eginez huts egin dezakete. Ingurune berri honetan ikertutakoa aurrera eramateko, *consensus* eta $\Diamond\mathcal{P}(om)$ *failure detector*-aren definizio berriak aztertuko dira eta $\Diamond\mathcal{P}(om)$ *faiulre detector*-aren zein *consensus*-aren inplementazioak aurkeztuko dira, sinkronizazio suposaketa ahul batzuk harturiko sistemara egokituta. Hutsegite detektatzailea TrustedPals framework-ean barneratzeko, pribatutasunaren hobekuntza tekniken baliabideak erabiliko dira.

2.2. Proiektuaren helburuen deskribapena

Bi helburu nagusi planteatu daitezke lan hau aurrera eramateko. Lehenengo proiektuaren lan praktikoa edo kasu honetan bezala, algoritmo batzuen inplementazioa, simulazioa eta proba kasuen azterketa eta bigarrena, inplementazio lan praktiko hau aurrera eramateko egin

beharreko pausoen azterketa eta erabiliko den metodologia.

Bigarren helburua oso garrantzitsua da lehenengoa modu aproposean eramateko, beraz lehenik eta behin proiektua nola eraman den azalduko da, ondoren proiektuaren funtsa azaldu eta amaitzeko emaitzak aurkeztuko dira.

Inplementazioa egiteko lehenik proiektu honen helburu nagusia aztertu behar da eta hortaz, aurrera eramateko baliabideak ere. *Secure Failure Detection and Consensus in TrustedPals* artikuluan oinarritzen da eta simulagailu batekin bertan aurkezten diren algoritmoen kodetzea egin behar da. Aurreko esaldia berriz irakurtzen bada, bi funtsezko atal ikusten dira, *Secure Failure Detection and Consensus in TrustedPals* eta hau simulatzeko gailua. Lehenengoak ulermen sakona eskatzen du, gero modu fidagarri batean egikaritzeko: zein ingurutan aplikatzen den, zein baldintzatan eta nola inplementatzen diren. Artikuluan aurkezten diren elementuak hobe ulertzeko, bertan aipatzen diren beste lanen irakurketa ere komenigarria izango da. *Secure Failure Detection and Consensus in TrustedPals* sistema banatuetan aplikatzen den teknika denez eta arlo honetan sakonago sartzea beharra dudanez, baita ere prestakuntza egin behar izan dut. Horretaz gain, *SMC & Consensus* teknikaren ezagutza beharrezkoa izango da, ondorioz ikasketarako denbora gehiago beharko da.

Simulazioaren helburua burutzearren simulagailu desberdinen azterketa egin beharko da eta nola ez, hauen erabileran trebetasuna hartu. Aurrerago, aztertuko diren simulagailuen artean zein aukeratu den eta zergatia azalduko da (4 atala). Zein simulagailu aukeratu den aurreratuko dut ondorengo pausuak azaltzeko; NS-3 edo *Network Simulator 3* izan da behin-betiko aukera. Simulagailu honekin lan egiteko C++ eta *Phython* lengoaiak erabil daitezke. Edozein bi lengoaiatan lan egiteko, lehenik aurreprestakuntza bat egitea beharrezkoa zenez ikasketa prozesu bat planteatu behar zen. Azkenik, C++ lengoaian lan egitea erabaki nuen. NS-3 Simulagailua C++ lengoaian garatu da eta ikasketarako eskaintzen diren adibide gehienak lengoaia honetan idatziak daude eta *Python* lengoaian berriz, adibideak nahiko eskasak dira. Aldi berean, simulagailua C++ lengoaian garatua egoteak, simulagailuaren nondik norakoak aztertzen ziren bitartean C++ lengoaian ikasketak burutzeko lagungarria izango zela pentsatu nuen ere.

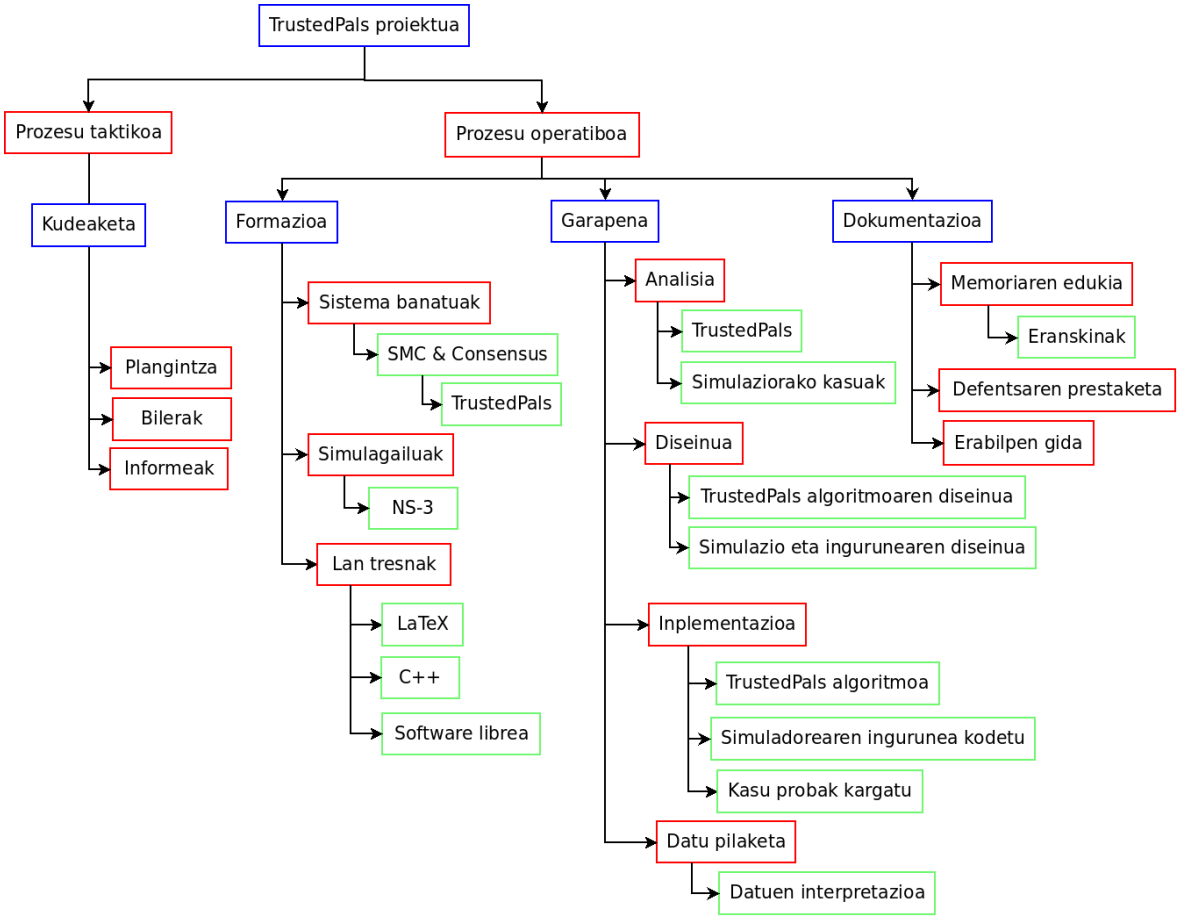
Behin aurrekariak aztertuta, proiektuan zer egin behar den erabaki daiteke. Simulazioa aurrera eramateko tresna guztiak ditugu, beraz, simulazioak egiteko erak ikasita *Secure Failure Detection and Consensus in TrustedPals* algoritmoen inplementazioa egin behar da, honek dakarren proba kasuen aukeraketekin. Hau guztia, sakonago ikusiko da ondorengo ataletan, zein erabaki hartu diren argitaratuz eta simulazioa egiteko pausoak nola egin behar diren (Eranskina A, gida liburua).

Proiektua dokumentatzerako garaian, baita ere zein tresnekin lan egingo den erabaki behar da. Zuzendariek LaTeX tresnarekin dokumentatzea proposatu zidaten, dokumentazio tresna ahaltsua delako eta amaierako itxura txukuna eskaintzen duelako. LaTeX erabiltzea erabaki

nuen baina honetarako ere trebakuntza fase bat beharra nuen. *Software librea*-rekin lan egitea gustukoa dut eta dokumentazioan agertu behar diren grafiko eta beste irudien sorrerarako eskura dauden tresnak aztertu behar izan dira. Hona hemen erabilitako tresnak: Irudiak marrazteko *Dia* marrazte tresna erabiliko da. *Dia*-k LaTeX dokumentazio tresnarekin oso ongi elkar eragiten dute, marrazkiak LaTeX tresnaren formatura esporta daitezkeelako. Denbora estimazioa egiterakoan eta modu grafiko batean ikusteko, *Gantt* diagrama erabiliko da. Honetan ere, *software libreko* GanttProject [GanttProject, 2014] tresna aukeratu da.

2.2.1. Azpi-atazen zerrenda

Proiektu hau egituratzerako garaian, goragoko lerroetan azaldutako atazak kontutan hartuta, eta software eraketan ditugun aurre ezagutzak erabiliz, atazen banaketa egin da. Ikuspuntu grafiko bat edukitzearren, software ingeniaritzako bizi zikloa aplikatu dugu LDE diagrama baten bidez (ikusi 2.1 irudia).



2.1 Irudia: LDE diagrama

- **Prozesu Taktiko - K Kudeaketa**
  - K1 - Plangintza / PHD

    Proiektuaren helburuen dokumentuaren lehen zirriborroa idatzi, bertan, proiektuaren helburuak eta burutuko den lanaren deskribapena jasoz.
  - K2 - Bilerak

    Zuzendariekin egingo diren bilerak.
  - K3 - Informeen elaborazioa

    Zuzendariekin egindako bileren aktak idatzi eta memoriaren jarraipena idazten joan.

- **Prozesu operatiboak - F Formakuntza**
  - F1 - Sistema Banatuak

    Sistema banatuen ezagutza eskasarengatik, hauen funtzionalitatea eta ezaugarriak ikasi.
    * F1.1 - *SMC & Consensus*

      Sistema banatuen munduan badaude arlo ugari. Artikuluko algoritmoan jorratzeko, erlazionatutako artikuluak irakurri eta kontzeptuak ikasi.
      · F1.1.1 -*TrustedPals*

        TrustedPals soluzioaren inguruan dauden ikerketak irakurri, beharrezko kontzeptuak ulertu eta algoritmoaren funtsa barneratu.
  - F2 - Simulagailuak

    Badiren simulagailuen azterketa egin, proiektu honetarako hoberena zein den aukeratzeko.
    * F2.1 - Network Simulator 3 (NS-3)

      Behin simulagailua aukeratu dela, erabiltzen ikasi dokumentazioak dituen tutorial eta adibideen bidez.
  - F3 - Lan tresnak

    Proiektuan zehar erabiliko diren tresnen ikasketa.
    * F3.1 - LaTeX

      Dokumentazioa LaTeX-en egitean erabaki denez eta erabilera jakintasun ezarengatik, ikerketa eta ikasketa egin beharko da.
    * F3.2 - C++

      Simulagailuak C++ lengoaia erabiltzen du eta lengoaian oso trebea ez naizenez, tutorialak eta manualak erabiltzea beharrezkoa izango da ikasketarako.

    * F3.3 - Software Libre
      Proiektu software libreaz egiteko aukera aurkeztu denez, sistema hautan lan egiteko dauden tresnak aztertu eta erabiltzen ikasi.

- **Prozesu operatiboak - G Garapena**

  – G1- Analisia

    * G1.1 - *TrustedPals*
      TrustedPals algoritmoaren analisian emandako datuekin, inplementazioaren norakoak definituko dira.
    * G1.2 - Simulaziorako Kasuak
      Simulagailuaren erabilpena analizatu eta baliabideak aukeratu.

  – G2 - Diseinua

    * G2.1 - *TrustedPals* Algoritmoaren diseinua
      TrustedPals algoritmoaren diseinua C++-en inplementatzeko eta simulagailuen zein baliabideekin erabili erabaki.
    * G2.2 - Simulazio eta ingurunearen diseinua
      Simulagailuan, sarearen definizioa erabaki eta eskaintzen diren aplikazioen artean zein erabili erabaki.

  – G3 - Inplementazioa.

    * G3.1 - *TrustedPals* Algoritmoa
      Simulagailuaren moduluak erabiliz, sare protokoloena, aplikazioa lagungarriak... eta guk sortutako moduluarekin bat *TrustedPals* osatuko duten aplikazioaren inplementazioa.
    * G3.2 - Simulazioaren ingurunea kodetu
      Edozein sare banaturekin lan egiteko algoritmo orokorra egin, ondorengo simulazio probak exekutatzeko
    * G3.3 - Kasu Proben Karga
      Kasu probak exekutatuko dituen script-aren inplementazioa eta fitxategien sorrera.

  – G4 - Datu Pilaketa.

    * G4.1 Datuen Interpretazioa
      Egindako simulazioen datuen pilaketa egin eta ondoren balorazioak atera.

- **Prozesu operatiboak - D Dokumentazioa**

- D1 - Memoriaren edukia

  Proiektuaren lana dokumentatzen duen idatzia prestatu beharko da, proiektua amaitzerakoan entregatuko dena.

  * D1.1 Eranskinak
    Proiektuaren kontzeptuak errazago ulertzeko erabilgarri izan daitezkeen eranskinen dokumentazioa.

- D2 - Erabilpen gida

  Aplikazioaren erabiltzaileei zuzendutako gida prestatu beharko da, baita erabilitako programa batzuen erabilerari buruzko gida (Simulagailua). Eranskin gisa sartuko da.

- D3 - Defentsaren prestaketa

  Proiektu hau defendatzeko beharrezkoak ikusten diren elementuen sorrera eta prestakuntza.

2.2.2. Plangintza

Behin proiektuko atazak antzeman ditugula, hauen plangintza denboran zehar erabaki beharko da. Plangintzan, ezinbesteko da atal bakoitzari eskainiko zaion denbora zehaztea baliabide guztiak ondo kudeatzeko. Lehenengo hurbilpen honetan, bakoitzak duen lan egiteko trebetasunaz baliatuz, ataza bakoitzari denbora estimazio bat eman behar zaio kudeaketa ahalik eta egokiena izateko. Plangintza hau, proiektuko zuzendariekin edukitako lehen bileran adostu zen. Bilera horretan ere, proiektuaren ezaugarriak eta betebeharrak zehaztu ziren. Bigarrengo bileran, proiektu honetan egin beharreko lana erabaki zen.

Denbora Plangintza

Proiektua aurrera eramateko, lehenik ikaste eta ikertze atal handi bat dago. Sistema banatuetan ezagutza sakona ez izateak eta simulagailuen azterketak denbora luzea emango du. Sistema banatuetan kontzeptu berri ugari ikasi behar dira eta batez ere Secure Failure Detection and Consensus in TrustedPals [Cortiñas et al., 2012] eta erlazionatutako beste lanetan agertzen diren kontzeptuak. Simulatzeko tresnak bilatzea eta hauen ezaugarriak aztertzea denbora emango du. Inplementazioaren atalak beste ordu kopurua handia eskatu du. Ez da kodetzeak ekar ditzakeen arazoak bakarrik kontutan hartu behar, algoritmoak simulagailu baten gainean inplementatu behar da eta simulagailuaren nondik norakoak ikasteak ere denbora eskatuko du. Amaitzeko simulazioaren emaitzak aztertu behar dira eta simulazio datuei interpretazio bat eman zaie. Deskribatutako ataza guzti hauek 400 ordu kopuruko lan karga suposatuko dute, hau da, aurreikusitako lan karga 400 ordukoa da. Ondorengo taulan, ataza bakoitzari esleitutako ordu kopuruak azaltzen da (2.1 taula).

Ataza	Orduak
Prozesu Taktiko	35
Plangintza	20
Bilerak	10
Informeak	5
Prozesu operatiboak	245
Formazioa	130
Sare Banatuak	30
TrustedPals eta Consensus	35
Simuladorearen ikasketa	30
C++	15
LaTeX Ikasi	20
Garapena	115
Analisia	15
Simulazioaren kasuak analizatu	15
Diseinua	20
Simulazioaren Kasuen erabaki	20
Inplementazioa	50
Algoritmoaren kodeketa	20
Simulazioan ingurunea kodetu	20
Kasu proben kodeketa	10
Datu Pilaketa	30
Dokumentazioa	120
Memoriaren edukia	90
Defentsaren prestakuntza	25
Erabilpen gida	5
Guztiara	400

2.1 Taula: Estimatutako ordu plangintza.

Karrera amaierako proiektu bat 15 kredituz osatzen da. Kreditu bakoitzak 25 lan ordu izaten dira. Hortaz, 375 orduko lana suposatzen da proiektuaren atal desberdinak aurrera eramateko. Proiektuko atal bakoitza ataza bat egokituko zaio estimatutako ordu kopuru bat esleituz. Estimazioak egiteko bakoitzaren lan karga eta zailtasuna kontutan hartu beharko da, atal guztiak ezaugarri desberdinak baitituzte eta horien arabera beharrezkoa diren orduen estimazio bat egingo da. Aurreko paragrafoan azaldu bezala, proiektu honek 400 ordu aurreikusi dira eta ez dator bat suposatzen den proiektu bateko 375 orduko aurrekontuarekin. Proiekturen kredituen ordu kopuruetara egokitzeko ataza bati edo batzuei denbora gutxiago eskaini beharko zaie, hortaz, ondorengo erabakia hartu zen: LaTeX ikasketari denbora gutxiago eman zaio eta oinarrizko kontzeptu batzuk edukita nahiko izango, ondorioz, 10 orduko lan karga edukiko du. C++ lengoaia ikasteari baita orduak kenduko zaizkio. Oinarrizko kontzeptuak ezagututa, simulagailua C++ lengoaian idatzita dagoenez, aztertzen den heinean C++ lengoaian jorratzen joango gara. C++ lengoaia ikasketak 10 orduko lan karga esleituko zaio. Azkenik, datu pilaketa atazaren denbora ere murriztuko da. Kasu honetan 20 orduko lan karga esleituko zaio. Esleipen berri hauen ostean, proiektu bat eskatzen dituen ordu kopuruarekin bat dator (ikusi 2.2).

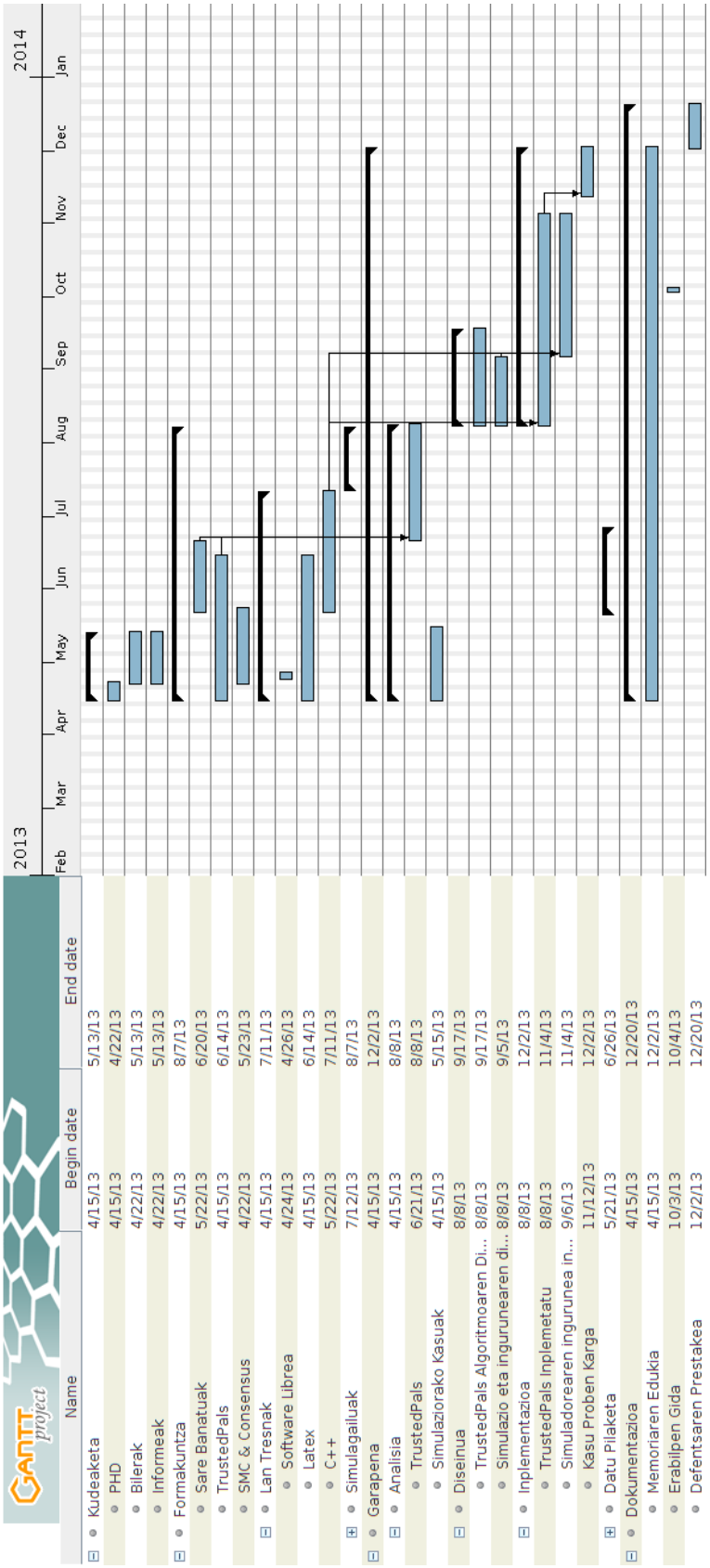
Ataza	Orduak
Prozesu Taktiko	35
Plangintza	20
Bilerak	10
Informeak	5
Prozesu operatiboak	220
Formazioa	115
Sare Banatuak	30
TrustedPals eta Consensus	35
Simuladorearen ikasketa	30
C++	10
LaTeX Ikasi	10
Garapena	105
Analisia	15
Simulazioaren kasuak analizatu	15
Diseinua	20
Simulazioaren Kasuen erabaki	20
Inplementazioa	50
Algoritmoaren kodeketa	20
Simulazioan ingurunea kodetu	20
Kasu proben kodeketa	10
Datu Pilaketa	20
Dokumentazioa	120
Memoriaren edukia	90
Defentsaren prestakuntza	25
Erabilpen gida	5
Guztiara	375

2.2 Taula: Kostu plangintza.

Behin lan orduak definituta daudela, astean zehar nola banatuko diren aztertu behar da. Zuzendariek proiektua bat erritmo onean eramateko, egunean 4 orduko kargarekin nahiko zela gomendatu zidaten, baina hau bakoitzaren gain dagoen erabaki bat da. Ikasleak lan karga horiek egunean zehar nahi dituen moduan bana ditzake. Baita gerta daiteke egunen batean ez usteko batengatik ordu horiek ez sartzea edo lan ez egitea, ordu horiek modu batean edo bestean errekuperatzea beharrezkoa izango da, egunean lan karga gehiago sartuz. Kasu hauek, aurrerago aipatuko den bezala, kontutan hartzea oso garrantzitsua izango da.

Bestalde, lan egunak astean zehar bost izango dira. Printzipioz astelehenetik ostiralera lan egingo da. Asteburuetan atsedena hartuko da, baina asteburu batean astean zehar galdutako orduak berreskuratu nahi badira edo lana aurreratu nahi bada, ikaslearen esku egonen da.

Proiektuak 375 ordu behar baditu eta egunean 4 orduko lan karga ezartzen bada, 94 egun behar dira proiektua aurrera eramateko. Lan asteak astelehenetik ostiralera direnez, lan asteak 5 egunekoak izango dira, beraz 19 aste beharko dira, jaiegunak eta ustekabeko egoerak kontutan hartu gabe. Kasu hauetan ere ustekabeko egoerak kontsideratu behar dira, hala nola ikaslearen gaixotzea, erabiltzen den materialarekin arazoak edukitzea (ordenagailua haustea, segurtasun kopiak galtzea). Hau kontsideratzen bada, beste aste bat edo bi egokitzea ez legoke gaizki. Hortaz, kalkuluak orokortuz, proiektuaren lan egunak osora 104 egunez osaturik egongo da.



2.2 Irudia: Gantt diagramaren estimazioa

Proiektuaren plangintza aurrera eramateko azpi-ataza eta fase bakoitzari eskainiko dizkiogun denbora eta ordu kopurua ondorengo Gantt-diagraman definituko da (ikusi 2.2 irudia).

Esan bezala, hau hurbilpen bat da eta bakoitzak duen lan karga eta gerta daitezkeen ustekabeak kontutan hartzen dira, baina maila logiko batean. Amaieran beste Gantt diagrama bat sortuko da, benetan emandako ordu kopuruekin eta honekin alderatuko da, gertatutako eta balorazioak ateraz.

Arrisku plangintza

Proiektu hau aurrera eramateko ezinbestekoa izango da arrisku plangintza egoki bat egitea. Plangintza honetan gerta daitezkeen arazo posible guztiak aurreikusi beharko dira eta arazo horientzako irtenbideak edo soluzioak bilatu, batik bat, hauek gertatzeko probabilitatea edota eragin ditzaketen galerak handiegiak direnean. Gure proiektuan izan daitezkeen arrisku nagusiak gertaera hauek dira:

- **Denboraren antolamendu ez egokia**

  Informatika gai berri baten aurrean aurkitzen garenez eta egin beharreko ikasketen zailtasunarengatik, aurreikusitako orduen estimazio desegokia edota ez zehatza egitea gerta daiteke. Esperientzia ezak, proiektua atzeratzera eraman gaitzake, ataza batzuetan estimatutakoa baino denbora gehiago pasa behar izateagatik. Adibide bat jartzearren, programaren kodeketan suerta daitezke traba ugari aurkitzea eta ondorioz estimazioak aldatu behar izatea, eta ataza honetaz dependentzia duten beste atazak ere atzeratu behar izatea horrek dakarren proiektuaren atzerapenarekin.

  Probabilitatea: Ertaina-Altua

  Soluzioa: Ataza bakoitza bukatzerakoan lan horretarako aurreikusitako denbora eta erabilitako denbora alderatuko da eta geratzen diren atazen artean denbora banandu egingo da. Proiektuarekin amaitzean denborak aztertuko dira eta helburuetatik zein ez den bete eta zergatik erabakiko da.

- **Informazio Galera**

  Proiektu honetan erabili ditugun gailu elektroniko guztiak matxuratzeko edota funtzionamendu arazoak izateko aukera daukate. Arazo horietako bat informazio galera dugu, proiektuaren atzerapen luzeak ekar ditzakeenak. Adibidez, disko gogorretan gordetzen ditugun datuak, gailuaren arazoengatik galdu daitezke.

  Probabilitatea: Ertaina

  Soluzioa: Datuak egunero lantokiko ordenagailuan, disko eramangarrian edo Network-Attached Storage (NAS) batean gorde eta gaur egun hainbeste erabiltzen diren kode bertsio kontrolen bat erabiltzea komenigarria izango da (adibidez, Git, SVN, Mercurial).

Baita, gaur egunean Internet munduak eskaintzen dituen hodei plataformak erabili daitezke, Google Drive edo Dropbox adibidez. Azken finean, helburua datu kopiak edukitzea da ezarri nahi den denbora tartearekin. Gutxienez egunean behin kopia egitea ez legoke gaizki.

Lan Metodologia

Gaur egunean, Internet-en hain modan dauden lan metodologiak probestu gaitezke, *Getting Things Done* (GTD), Pomodoro, Focus... teknikak adibidez. Orohar helburu bera dute, lana atazetan zatitu eta hauek aurrera eraman bakoitzak duen garrantziaren arabera.

Funtsa behin barneratuta dugula, proiektuan zehar beharreko atazak atera behar dira. LDE-n (ikusi 2.1 irudia) ikusten denez lan hori jada burutu dugu, baina hauek oso ataza orokorrak dira. Proiektuaren atazen banaketa egin dugun bezala, ataza hauek beste azpiatazetan banatu beharko dira, ataza atomiko edo zatiezinak eta *erraz* burutzeko atazak lortu arte. Adibidez, kodeketa barnean funtzio konkretu bat burutzea. Klase osoa inplementatzea jartzen bada, nondik hasi ez jakitea gerta daiteke eta ondorioz denbora estimazioak txarrak izatera eraman gaitzake.

Behin ataza zatiezinak lortu ditugula, garrantzi eta dependentziaren arabera ordenatuko dira eta hauek egikarituko dira. Teknika hauen filosofian, helburua, ataza bat hartzen bada amaitu arte ez uztea da. Gerta daiteke kodeketa batean trabatzea edo beste dokumentu baten beharra edukitzea, ondorioz, ataza hori denbora batez utziko dugu beste momentu batean buru argiago edo zuen dependentziaren dokumentazioarekin itzultzeko. Bestalde, arazoa ez bada konpontzen estimatutako ordu kopuruetan, ataza bukatu arte ordu gehiago sartu beharko dira.

Ataza bat burutzean gertaera ugari moztu gaitzake, telefono deiak edo mail-en iristeak adibidez. Hauek saihesten saiatuko gara, gerorako lan ez garrantzizkoetan gordez. Denbora libre edo lan eguna amaitzerakoan egin daitezkeenak.

Garrantzizkoa ere izango da gerorako, ataza bakoitzean erabilitako denbora, guk hasieran estimatutako denborarekin alderatzeko eta ondorioak ateratzeko.

Bi astetan behin proiektuaren nondik-norakoak aztertuko dira. Proiektuaren ikuspuntu generiko bat egin beharko da eta ez orain arte bezala, atazaka. Lehenago proposatutako helburuak betetzen joaten badira ez dago arazorik, baina zerbaitengatik hauek ez badira betetzen, helburuak betetzearren zergatia eta nola konpontzea erabakiko da.

Bileren Kudeaketa

Hasieran, proiektuaren funtsa ulertzeko haina bilera beharko dira. Kontzeptu ugari agertuko dira eta modu egokian aurrera eramateko hauen ulermena ezinbesteko da. Gero e-mail bidez suertatutako kezkak galdetuko dira eta behin lan pakete edo zati bat eginda dagoelarik, bilera

fisiko bat egin daiteke. Hau proiektuaren martxaren arabera erabakiko da. Bilera e-mail bidez adostuko da eta irakurtzeko edo aztertzeko materialik balego, elkarrekin lan egiteko Dropbox sortutako karpetara igoko dira fitxategiak.

Fitxategien kudeaketa

Simulagailu gainean egingo den inplementazioa eta proiektuaren dokumentazioa egunero erabiltzen den ordenagailuan erabiliko da. Gailu elektronikoek suerta ditzaketen arazoak direla eta, disko eramangarri, USB pendrive edo zerbitzari batean kopiak egingo dira. Baita Dropbox bezalako hodeiko plataforman, edozein tokitatik eskuragarri edukitzeko. Lehen esan bezala, egunean behin kopia egitea ez legoke gaizki baina komenigarriena ataza bat bukatzean kopia egitea litzateke.

Bideragarritasuna

Proiektua 2013 urteko apirilean egokitzapena egin zen, ordurako ikasleak kreditu guztiak gaindituak zituelarik, beraz urte horretako uztailan aurkezteko egoeran egoteko espero dugu. Kasu honetan bideragarritasunarekin izan dezakegun arazo nagusia programazio lengoaian trebatzean eta simulagailuaren erabilpena izan daiteke. Ikasleak lengoaia berria eta simulagailuaren erabilpena ikasi eta menperatu beharko du aplikazioa inplementatzen hasi aurretik.

Proiektuaren gaian trebatzea, sare banatuak eta hutsegite detektoreak, urteetan eduki duen hezkuntzarengatik ez luke arazo handirik suposatu beharko.

LaTeX erabiltzeko ere ikasketak egin behar dira, baina ez du arazo handirik suposatuko.

Emangarriak

Entregatutako Dokumentazioa: Proiektuaren memoria paperean entregatzeaz gain, honen iturburu kodea entregatuko da. Hurrengo estekatik iturburu kodea eskura daiteke: [Itoiz, 2014].

Bertan ondorengo materiala egongo da:

- Proiektua eraikitzeko erabili diren moduluak.
- Erabilitako script-ak.
- Kasu proben fitxategiak.

3 KAPITULUA

Proiektuaren aurrekariak

Kapitulu honetan proiektuaren sakontasunean sartu aurretik eskuratu beharreko ezagutzak azalduko dira. *Secure in Failure Detection and Consensus in TrustedPals* artikuluaren azalpena egingo da, honetaz gain algoritmoak ulertu eta inplementatzeko beharrezko diren ezagutzak argituko dira. Behin beharrezkoa den ezagutza eskuratu dela, ondorengo kapituluan proiektuaren helburuak diren atalak zein metodologiarekin eramango diren azalduko da.

3.1. Sarrera

Secure Multi-party Computation (SMC) [Yao, 1982], gaur egun oso maiz agertzen den segurtasun arazo horietako bat da. Elkar konektatutako n partaideek $F(f_1, f_2, \ldots f_n)$ funtzioa konputatu nahi dute. Partaide bakoitzak funtzio hau egikaritzeko bere sarrera (pribatua) edukiko du. Hau gainontzeko partaideen ikuspuntutik sekretua izango da, F funtziotik jaso ditzakeen emaitzak izan ezik.

Adibide batez azaltzeko, Andrew Chi-Chih Yao-k bere lanean aurkeztutako adibideaz baliatuko gara. Demagun bi pertsona aberatsen artean apustu bat egin dutela, zein den aberatsagoa jakiteko. Aberatsek, pribatutasuna bermatzearren beraien diru kantitatea ezagutzea ez dute nahi. Argi ikusten dira aurreko paragrafoan aurkeztutako kontzeptuak, funtzioa *zein den aberatsena* litzateke eta biek pribatutasuna bermatu nahi dutela (beraien arteko konfiantza ezagatik). *Aberatsen arazoa* bezala ezagutzen den adibide hau lehen aipaturiko kasuetara (enkanteak, erosketak...) ere estrapola daiteke.

Beraz, SMC arazoari ebazpen bat lortzeko, ondorengo propietateak bete beharko dira:

- *SMC-Validity*. Prozesu batek F funtzioaren emaitza jasotzen badu, erabakia konputatu eta jaso duten prozesu zuzen guztien emaitza izango da.
- *SMC-Agreement*. p_i-k r_i F-emaitza jasotzen badu eta p_j-k r_j F-emaitza jasotzen badu, $r_i = r_j$.
- *SMC-Termination*. Azkenean, prozesu zuzen guztiek F-emaitza jasoko dute.

- *SMC-Privacy*. Huts-egindako prozesu guztiek ez dute prozesu zuzenen balioaz ezer ikasten.

TrustedPals, [Fort et al., 2006] Secure Multy-party Computation (SMC) arazoa modu eraginkor batean ebazteko smartcard-etan oinarritutako framework bat da. Praktikara eramanaz, prozesuak Java mahai-gaineko aplikazio bat bezala inplementatuta daude eta segurtasun modulua Java Card Technology-rekin aktibatutako SmartCard-ekin [Chen, 2000] eginda dago. Prozesu hauen artean SMC arazoari ebazpen bat lortzeko, prozesu bakoitzaren segurtasun moduluez baliatuz, *Trusted Third Party* (TTP) bat simulatzean datza. *TrustedPals* nola funtzionatzen duen adibide batekin, *Trusted Third Party*-n funtzionamendua argiago ikusiko da.

SMC TrustedPals framework-an ebazteko, F Java-n kodetutako funtzioa bat da eta hau, hasieraketa fasean sarearen zehar sakabanatzen da. Ondoren prozesu bakoitzak bere sarrera balioa segurtasun moduluari emango dio, jarraian framework-ak sarrera balioen sakabanatze segurua burutzeko. Azkenik, segurtasun modulu guztiek F egikarituko dute eta emaitza beraien prozesuei pasako diete. Segurtasun moduluen sarea sortzean, konfidentzialtasuna eta haien arteko komunikazio kanalen autentifikazioa ziurtatzen denez, sakabanatze garaian *geruza seguru* batean lan egiten dutela esan daiteke.

3.2. Aztertu beharreko sistema: TrustedPals

3.2.1. Analisia

TrustedPals-en hasierako definizioan eta inplementazioan, sarea sinkrono dela suposatuko da, non adibidez, eraikitako sareko denbora parametroak ezagunak eta lehenetsiak diren. Suposaketa hauek oso sentikorrak dira atzerapenak kontutan edukitzen baditugu, gerta daiteke sare trafiko ugariagatik mezu bat atzeratzea eta ezarritako denbora borneak gainditzea, ondorioz, Internet bezalako sare batean aplikatzeko ez da eredu oso aproposa. *Secure Failure Detection and Consensus in TrustedPals* artikuluan [Cortiñas et al., 2012], *TrustedPals* sinkronizazio gutxiagoko ingurunetan aplikatzeko azterketa egiten da, beste modu batean esanda, gaur egungo hutsegite-tolerantzien sistema banatuetan oinarrituta, *SMC bertsio asinkrono* batean nola ebatz daitekeen azaltzen da: adostasun algoritmo *asinkrono* bat erabiliko da eta denbora suposaketa batzuk deskribatuko dira. Bestalde, dispositibo berri bat barneratuko da, *failure detector* edo hutsegite detektatzailea bezala ezagutzen dena, *Chandra and Toueg* ikerketan agertutakoa [Chandra and Toueg, 1996].

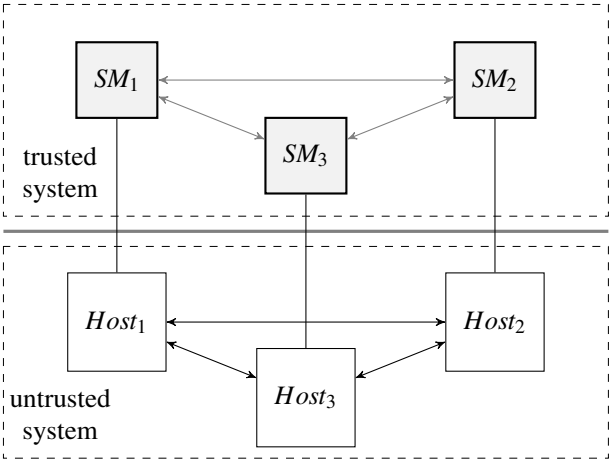
Failure detector-aren kontzeptua, hutsegiteak dituzten ingurunetan aztertua izan da batez ere [Freiling et al., 2011]. Sistema horretan prozesu zuzenek (adibidez, kolisiorik ez duten prozesuak) kolisioa eduki duten prozesuetan behin-betiko susmatu behar dute. Prozesu zuzenak

definitzerakoan, modu aunitzetan egin daiteke, adibidez, inoiz huts-egiten ez duten prozesuak edo kolisioak jasaten ez dituztenak baino mezu omisioak eduki ditzaketenak. *Secure Failure Detection and Consensus in TrustedPals* artikuluan prozesu onez hitz egin beharrean, *well-connected*-tzat hartuko dira, konputatu eta beste gainontzeko prozesuekin komunika daitezkeen prozesuak.

Ondorengo lerroetan, *TrustedPals* eredua hobe ulertzeko modu formal batean egin beharreko suposaketak aurkeztuko dira.

3.2.2. Sistemaren modeloa

Trusted (fidagarri) eta Untrusted (ez fidagarri) sistemak



3.1 Irudia: Trusted eta Untrusted sistemak.

Sarrerako kapituluan ikusi denez (1 kapitulua), prozesuak bi klasetan bereiz daitezke: *untrusted host* edo *host* ez-fidagarria eta *secure module* edo segurtasun modulua. Antzematen denez, sistema hibrido bat eraikitzen da, non *host* guztien arteko komunikazio topologia bat existitzen den, adibidez, elkar-konektaturiko bi prozesuak ondokoak dira. *n host*-ek sistema osatuko dute eta beraien artean komunikazioak egiteko ez dute konfiantzarik. *Host* bakoitzari (h_a), konfiantzazko segurtasun modulu (sm_a) bat atxikituko zaio. Segurtasun modulu guztiek, beste segurtasun moduluengan konfiantza dute edozein komunikazio egiteko. Irudian ikusten denez (3.1 irudia), h_a eta sm_a elkar komunikatzeko erabiliko duten lotura batez konektatuta daude. Segurtasun moduluek eta hauek dituzten komunikazioa loturei *trusted system* edo sistema fidagarria deituko zaio. *Host*-ak eta hauen komunikazio kanalek osatzen duten sistemari berriz, *untrusted system* edo sistema ez fidagarria deituko zaio. *Untrusted system*-ko prozesuek dagokien aplikazioa exekutatuko dute, e-voting, e-banking,... Maila honetan dagoen komunikazio konfiantza ezagatik, *trusted system* maila erabiliko dute programa hauen exekuzioa burutzeko. *Trusted system* azkenean, segurtasun moduluek exekutatzen duten sistema da. Hauen artean eragiketak egiteko konfiantza badago, lehendik segurtasun neurri batzuk pasa dituztelako

izango da. Adibide bat aipatzearren, jada aipaturiko SmartCard-ek [Chen, 2000] *Trusted system* bat sor dezakete Internet bezalako *Untrusted system* sare modelo batean. Bi sistemen arteko komunikazio burutzeko, public-key egituradun sistema erreferentzia bezala hartuko da, non, bien arteko konfidentzialtasuna, mezu osagarritasuna eta autentifikazioa bermatzen den. Segurtasun maila honekin, elkarren arteko transakzioak eta aldagaien trukaketa modu seguru batean egiten dela bermatuko da. *SMC*-n ez bezala, operazio bakoitzean segurtasun neurriak pasatzea ez da beharrezkoa. Eredu honetan lehenengo deialdian segurtasun neurriak gainditzea nahikoa izango da, behin hauek gaindituta segurtasun moduluak egoeraren kontrola hartuko duelako.

Untrusted system eta *trusted system*-ren arteko konexioa, bi mailen arteko prozesu deietan datza, hau da, *untrusted system*-ko prozesu bakoitzak *trusted system* mailako bere segurtasun moduluari deituko dio. Ikusten denez, makina bakoitzak bere baitan segurtasun modulua du, beraz suposatuko da bi-noranzko komunikazio kanala dagoela.

3.2.3. SMC asinkronoa Trustedpals-ekin zehazten

Sarrerako kapitulua aipatu bezala (1 kapitulua), *SMC*-ek, *host* multzo bati, hauen aldagai lokalean, komunean duten F funtzioen konputazioa ahalbidetzen du. Hurbilketa honetan, sistema modu asinkrono batean ebazteko aukera aurkezten da. *SMC* modu sinkrono edo asinkronoan ebaztearen desberdintasunak, sarrera guztiek F funtzioa ez konputatzean datza. Nahikoa litzateke F funtzioa gutxienez $n-f$ prozesuek atzitzea, non f hutsegite egiten duten prozesu kopurua den, adibidez prozesu maltzurrak mota honetakoak lirateke. Prozesu bat ona da, baldin eta *crash*-ik ez duen egiten eta ezarritako protokoloak jarraitzen baditu (prozesu *Byzantine*-ak ez izatea).

Formalizatuz, demagun $x_1,\ldots,x_n$ *host*-aren sarrera pribatuak direla. *TrustedPals* asinkrono ereduan, r emaitza C azpimultzoaren sarreren bidez konputatutako balioa da, non $n-f$ *host*-ek parte hartu duten, adibidez, $r = F(y_1,\ldots,y_n)$ konputazioan, i-garren *host*-a C azpimultzoaren barne badago, $x_i = y_i$ balioa jasoko du bestela, lehenetsitako balio bat (adibidez $y_i = 0$) edukiko du. r emaitza modu seguru eta fidagarri batean konputatuko da, *host* guztiek *TTP* erabili izan balute bezala. Honekin, *host* bakoitzak, gainontzeko *host*-ekin bere pribatutasuna mantendu duela eta prozesu maltzurrek emaitzaren aukeraketan zein konputazioan ez dutela parte hartu esan daiteke. Ondorioz, sarreran aipaturiko *SMC* propietateak betetzen direla bermatzen da. Birgogoratu ditzagun:

- **SMC-Validity**. *Host* batek emaitza bat jaso badu, emaitza hori, C azpisarea osatzen duten *host*-engatik konputatutako F funtzioaren emaitza da. Gutxienez, $n-f$ *host*-ek parte hartu dute $r = F(y_1,\ldots,y_n)$ konputazioan, eta i-garren *host*-a C azpimultzoaren barne badago, $x_i = y_i$ balioa jasoko du bestela, lehenetsitako balio bat (adibidez $y_i = 0$) edukiko du.

- **SMC-Agreement**. Segurtasun moduluek itzultzen dituzten emaitzetatik, ez daude bi desberdinak direnak.
- **SMC-Termination**. Azkenean, *host* ez maltzur guztiek, bere segurtasun moduluengandik emaitza bat jasotzen dute.
- **SMC-Privacy**. *Host* maltzurrek ez dute beste *host* ez maltzurrengandik ezer ikasten, azken *r* emaitza eta beste *host* maltzurren sarreretatik izan ezik.

Untrusted System

Aurreko atal batean aipatu bezala, *untrusted system*-eko *host* pare bakoitza, zentzu bakarreko bi kanalengatik lotuta daude, bat noranzko. Sisteman, kanal fidagarri minimo batzuk egotea espero da eta gainontzekoak ez fidagarriak izan daitezke, galerak dauzkatenak adibidez. Kanal fidagarri batean bidalitako mezuak bere helburura iritsiko direla ziurtatuko du.

- *Hutsegite modeloa*

  Untrusted system-an, *Byzantine* eredu [Lamport et al., 1982] ematen dela suposatuko da. *Byzantine* hutsegitea, sistema banatuetan prozesu baten exekuzioan gerta daitekeen hutsegite edo portaera arbitrario da. *Byzantine* prozesu batek maltzurkeri bat egitearren, helburu desberdinetara mezu desberdinak bidali ditzake baina mezuak berdinak direla antzemanaz. Hauek mezu *mutanteak* lirateke. Honen proposamena adibidez, sistema bertan behera uztea litzateke. *Byzantine* hutsegitea, sistema modu ez-ohiko batean portatzea helburua luke. Kasu honetan adibidez, prozesu oker guztiak protokoloren exekuzioaren informazio guztia bildu dezakete, sistema bertan behera uzteko estrategia bat erabakitzeko. Ere suposatu behar da, *host*-ek eragiketak konputatzeko garaian mugak dituztela, adibidez, komunikazio kanaletan indarkeriazko *erasoak* ez direla posible. *Untrusted system*-an gehiengo *host*-ak ez maltzurtzat hartuko dira.

- *Denbora*

  Host bakoitzak erloju lokal bat edukiko du, baina honek ez du esan nahi hau sarearen erlojuarekin sinkronizatuta dagoela. *Untrusted system partzialki sinkronotzat* hartuko da, prozesu ez maltzurren prozesatze kapazitatea eta komunikazio atzerapenen menpe dagoelako.

Trusted System

Trusted system edo sistema fidagarria, aurreko atalean aurkeztu bezala, sistema ez fidagarriaren gainean dagoen geruza bat bezala antzeman daiteke. Nodoek, haien artean konektatzeko, *host*-ak lotzen dituzten kanal berdinak erabiliz, kanal logiko edo birtual baten bidez egingo

da. Hortaz, segurtasun moduluek gainontzekoekin komunikatzeko aukera dute. Goiko geruza hau *Smart Card* baten bidez sor daiteke. Adibidez, demagun smart card-a dispositiboan sartzerakoan, konektaturiko nodoen arteko *VPN* bezalako sare birtual bat sortzen duela edo *SSL tunneling* bezalako konexio bat egiten duela.

- *Modelo fidagarria*

  Sistema fidagarrian komunikazioan parte hartzen duten bi parteak mezu konfidentzialtasuna, mezu integritatea eta mezu kautotzea bermatzen dute. Hau, bien artean giltza kriptografikoen trukaketarekin egin daiteke hasieraketa fasean, ala aurreko sarreran aipatutako VPN edo beste hainbat segurtasun protokoloak erabiliz. Modelo fidagarrian ere izenpea bermatzen da, hau da, nodoek gainontzeko nodoen autoretza ziurtatzen dute eta badakite inork ez duela bere izenpea haustu. Aipatu bezala, izenpe eta konexioaren pausoak hasieraketan egitearekin nahiko da, honela F funtzioak prozesatzean ez da denbora galtzen berriz kodetzean...

- *Denbora*

  Segurtasun moduluek ez dute erlojurik, hauek pauso zenbaketan oinarritzen dira. Pauso bat mezu baten konputazioa litzateke, hau da, lokalean agindu batzuk exekutatu eta bidaltzea izan daiteke pauso bat. Denbora pasa, emandako pauso hauen zenbaketa izango da. Aurreko modeloan bezala, partzialki sinkronoak direla esango dugu, nodo bakoitzaren eta konexioaren menpe egongo delako denbora pauso baten balioa.

  Sistema fidagarriak eta ez-fidagarriak komunikazioa kanal fisiko beretik egiten dutenez, bi sistematarako denbora portaera berdina hartuko da. Bi sistemak *partzialki sinkronoak* direla suposatzeak, azkenean sareko parametro garrantzitsu guztien borneak (mezu jasotze atzerapena, prozesaketa denbora guztiak,...) beteko direla bermatzen du. Eredu hau Dwork, Lynch eta Stockmeyer-ek [Dwork et al., 1988] argitaratutako laneko bariante bat da. Kasu honetan aurreko lanarekin alderatuta, kanalak fidagarriak direla hartuko da.

  Dena den, ondorengo atal batean azalduko den bezala, mezu bat *garaiz iritsi* dela suposatuko da, baldin eta ezarritako denbora parametroen barnean jasotzen bada bestela hutsegitea edo maltzurtzat bezala hartuko da. Denbora mugetaz baliatuz, mezu horien omisioak modu fidagarri batez detekta daitezke.

- *Hutsegite modeloa*

  Untrusted system-an, *trusted system*-an bezala, eraso maltzurra ere jasa ditzake. Segurtasun modulua dela eta, eraso hauek kanal seguruengatik nahiko murrizten dira. Maltzurkeriak honelakoak izan daitezke:

- Segurtasun moduluaren suntsitzea.
- Segurtasun moduluaren eta kanalaren arteko mezuen oztopatzea.
- Pauso kontagailuaren aldaketa maltzurra.

Honezkero, *trusted system*-an *general omission* eta *untrusted system*-an maltzurkeriarekin lotutako *host*-en segurtasun moduluek jasan ditzaketen egoera asinkronoak hartuko dira hutsegite modelo bezala (adibidez, *host Byzantine* portaera baten gainean dagoen segurtasun modulua). Mezu bat bidaltzean (*send omission edo bidaltze omisioa*) edo jasotzean (*receive omission edo jasotze omisioa*) bertan behera geratzea suerta daiteken hutsegitea da *omisio* hutsegitearen kontzeptua. *Trusted system*-an hartutako *general omission* modeloan, segurtasun moduluek mezu erortzeak (*crash*) edo jasotze zein bidaltze omisioak jasan ditzake. *Secure Failure Detection and Consensusin TrustedPals* artikuluan, ere *transient* omisioak onartzen dira, hau da, prozesu batek une batez mezuak bertan behera uzten ditu, geroago, modu fidagarri batean, berriz mezuak entregatzen hasteko. Badago ere betirako *omission* moduan egotea.

Gerta daitezkeen omisioez aparte, *trusted system*-a arbitrarioki moteldu daiteke (asinkronoa) nahiz eta fisikoki sistemaren gainean lan egiten duten, *untrusted system*, partzialki sinkronoa da. Modelo honek, *host* maltzurren beste bi eraso hauek jasa ditzakete:

- *Timing Attack*. Segurtasun moduluen denbora pasa, mezuen trukaketan oinarritzen da, beraz segurtasun moduluaren pauso kontaketa, dagokion *host*-aren menpe dago. *Timming attack* batean, *host* maltzur batek arbitrarioki alda dezake pauso kontaketaren abiadura, hau handituz edo motelduz, baina beti *host*-aren erlojuaren menpe egonen da. Honen ondorioz, segurtasun modulua asinkronoa bihurtuko litzateke eta honekin bat komunikazio kanal birtual horiek ere.
- *Buffering Attacks*. *Host* maltzur batek segurtasun moduluaren eta kanalaren arteko mezuak hartu edo oztopatu ditzake. Hau azken finean *omission* bat gertatzea litzateke, bai bidaltzeko garaian edo jasotzeko garaian. *Buffering attack* batean, *host* batek ez ditu mezuak ezabatzen edo bertan behera uzten, hauek buffer batean sartuko lituzke denbora tarte bat pasa eta gero modu arbitrario batean kanalean sartzeko. Kasu honetan ez da omisiorik gertatuko, baina sistema fidagarriko kanala asinkronoa bihurtuko da.

Kasu bietan, denboraren aldaketa izango lirateke, komunikazioa asinkronoa bihurtuz. *Bufferin attack*-en kasuan, *Trustedpals* algoritmo honen berdiseinaketan, mezuen ordenazioan ez du eraginik izango, mezu bakoitzak bere sekuentzia zenbaki bat eta bakarra dena garraiatzen duelako, hortaz, mezu bat ez da bidaliko espero ez bada. Buffer baten overflow-a ere saihestu daiteke *Smart Card*-ak mezu konkretu batzuen harrera ez eginez.

Laburbilduz, *Trusted system*-eko prozesuek bakarrik hutsegin dezakete *crash* edo *mezu omisioen* kasuetan. Beste maltzurkeri bat gertatzen bada, gerta daitekeen gauza bakarra, kanala asinkronoagoa bihurtzea da.

TrustedPals-engatik harturiko segurtasun moduluko erabakiak direla eta, antzeman daiteke *untrusted system*-eko Byzantine hutsegite modelotik, segurtasun moduluak jarrita, *general-omision* modelo batera pasa dela, hutsegite motak kontrolagarriagoak izanaz eta hauen kasuak murriztuz.

- Trusted system-*eko prozesu klase motak*

  Prozesu bat (*untrusted system*-an zein *trusted system*-ean) *correct* edo *zuzena* da hutsegiten ez badu. Prozesu bat *faulty* edo *okerra* da, zuzena ez bada, lehenago definitutako protokolo parametroak betetzen ez baditu adibidez. Gehiengo prozesuak zuzentzat hartuko dira bi sistematan, *untrusted system*-an zein *trusted system*-ean. Kontutan hartzekoa da, segurtasun modulu oker batek *host* oker bat inplikatzen du, baina *host* oker batek ez du beharrez segurtasun modulu oker bat inplikatu behar. *Secure Failure Detection and Consensusin TrustedPals* artikuluan, aurrerago azalduko den bezala, prozesu zuzenak deitu beharren *well-connected* kontsideratuko dira, adibidez, prozesu gehiengoarekin komunika edo konputatu daitezkeen prozesu horiek.

  Sistema hibrido hurbilketa honen motibazioa, sistema erasokorra den ingurune batean exekutatzen delako da. Baina, segurtasun moduluaren suposaketa eta komunikazioak kanal fidagarrietan burutzen direla suposatzeak, sistema fidagarrian erasotzaileak sistema bertan behera uztearen probabilitatea murrizten da.

  Sistema fidagarrian omisio modeloa aplikatzea erabaki denez, *transient* bidaltze eta jasotze omisioak agertzea suerta daiteke. Demagun bi prozesuk beraien artean mezu bat bidali nahi dutela, p-k q-ri m mezua bidaltzen dio. p-tik q-ra bidalitako m mezua q-k ez badu jasotzen, zer pasa da? p-k *send-omission* eduki du?, q-k *receive-omission* eduki du? Ikus daitekeenez bietako batek hutsegin du, baina ezin da jakin bietatik zein. Biak, p eta q, okertzat hartzea oso murriztailea izatea litzateke, honela ba, omisio modeloko prozesu zuzena/okerra bezala klasifikatzea birplanteatu daiteke. Adibidez, prozesu batzuk beste prozesu zuzen batzuekin ez badute omisio hutsegiterik, hauek ontzat hartuko dira, ondorioz, adostasuna bezalako sistema banatuko protokoloetan parte hartu dezakete prozesu zuzen horien bitartez komunikazioa eginez. Kasu honetan, prozesu motak hauek *well-connected*-tzat hartuko dira, azkenean komunika daitezkeelako beste prozesuekin baina modu ez zuzenean.

  Adibidean ikusi bezala, prozesu baten *konektibitatea* omisioen menpe dago, gerta daiteke *send-omission*-ak ongi egitea baina gero *receive-omission*-ak edukitzea. Beraz, motibazio hau kontutan edukita klasifikazio bat egin daiteke:

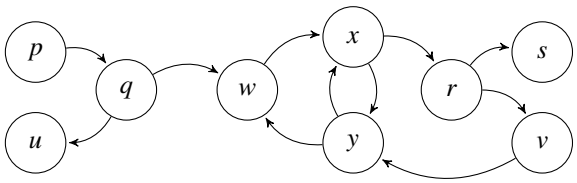
  Definizioa 1. *Prozesu bat* InConnected *da, baldin eta soilik baldin:*

- *p prozesu* well-connected *bada edo*
- *p huts-egiten ez badu eta q prozesu bat existitzen den, non q prozesua InConnected den eta azkenean, q-k p-ri bidalitako mezu guztiak p-k garaiz jasotzen baditu*. (Adibidez, *q*-k *p*-rekin ez du bidaltze omisioa edukiko eta *p*-k *q*-rekin ez du jasotze omisioa edukiko)

Definizioa 2. *Prozesu bat* OutConnected *da, baldin eta soilik baldin:*

- *p prozesu* well-connected *bada edo*
- *p huts-egiten ez badu eta q prozesu bat existitzen den, non q prozesua OutConnected den eta azkenean p-k q-ri bidalitako mezu guztiak q-k garaiz jasotzen baditu*. (Adibidez, *p*-k *q*-rekin ez du bidaltze omisioa edukiko eta *q*-k *p*-rekin ez du jasotze omisioa edukiko)

Definizioetan antzematen denez, prozesu zuzenak aldi berean *InConnected* eta *OutConnected* dira. Baita antzeman daiteke, *InConnected* eta *OutConnected* prozesuak errekurtsiboak direla. Prozesu zuzen guztietatik *InConnected* motako prozesu guztietara garaiz iristeko omisiorik gabeko bide bat dago. Baita, *OutConnected* prozesu guztietatik prozesu zuzenetara garaiz iristeko omisiorik gabeko bide bat dago, ondorioz, *InConnected* prozesu guztietara.



3.2 Irudia: Prozesuen adibidea.

Jarraian adibide batez argiago ikusiko da 3.2 irudian. Prozesu batetik besterako marraztutako arkuak, mezuak garaiz iristen diren omisiorik gabeko loturak adierazten dituzte. *w*, *x*, *y*, *r* eta *v* prozesu *well-connected* dira, ondorioz, aldi berean *InConnected* eta *OutConnected* dira. *OutConnected* definizioan oinarritzen bagara, *q*-k bi baldintzak betetzen ditu. *q* prozesu zuzena da, arkuak *m* mezua garaiz eta omisiorik gabe bidaliko duela adierazten duelako. *q*-k huts-egiten ez duenez eta *w* prozesu zuzenak, (*InConnected* eta *OutConnected* dena) garaiz jasoko dituenez *q*-k bidalitako mezuak, *q* ere *OutConnected* da. *p* baita ere *OutConnected* da. *p* prozesu zuzena da, ez du huts-egiten eta bada *q* prozesua *OutConnected* dena. *s* prozesua *InConnected* da, arkuak mezuak omisiorik gabe eta garaiz iritsiko direla adierazten duelako. Hutsegiterik ez du edukiko eta *r InConnected* prozesuak *s*-ri bidaliko dizkion mezuak garaiz iritsiko direnez, *s* ere *InConnected* klasekoa izango da. Azkenik, *u* prozesua ez da *InConnected* ezta *OutConnected* ere. Izatekotan *InConnected* izan liteke, arkuak, *u*-k *q*-ren mezuak

omisiorik gabe eta garaiz jasoko dituela adierazten duelako, baina *q InConnected* ez denez, bigarren baldintza ez da betetzen.

Aurreko puntuan aipa den bezala, prozesu zuzenak, *well-connected*-tzat kontsideratuko dira, *InConnected* eta *OutConnected* aldi berean prozesu zuzenak direlako. *InConnected* eta *OutConnected* adierazten dute komunikabide bat dagoela, horregatik *well-connected* terminoa.

Jarraian algoritmoaren diseinuaren eta beste zenbait kontzeptu azalduko dira.

3.2.4. Uniform Consensus

Consensus edo adostasun arazoaz hitz egiterakoan, burura etortzen dena honakoa litzateke: prozesu guztiek balio bat proposatzen dute eta prozesu zuzen guztien artean, proposatutako balio horiekin, amaieran behin-betiko balio bat aukeratuko da. Kolisio ereduan, prozesu zuzen guztiek behin-betiko erabakian parte hartu behar dute. Propietate honi adostasunaren *Termination* deritzo. *Consensus* eta *Uniform Consensus*-aren arteko diferentzia, *uniform agreement* propietatean datza, zuzenak ez diren prozesuak erabaki bat hartu beharrean badaude, prozesu zuzenak erabakitako balioaren berdina izan behar da.

Consensus omisio modelora egokitzeko, *Secure Failure Detection and Consensus in TrustedPals* artikuluan bakarrik *Termination* propietate birdefinitzea proposatzen da. Chandra and Toueg [Chandra and Toueg, 1996] lanean, prozesu zuzen guztiek azkenean erabaki bat hartu behar dutela aipatzen da. Oraingo definizioak, *InConnected* prozesu guztiak tartean sartzen ditu, nahiz eta omisio batzuk jasan ditzaketen (*send-omission*-ak), *InConnected* prozesuek, erabakia jaso dezaketen prozesuak dira. *Uniform Consensus*-aren propietateak omisio modeloan honakoak izango dira:

- *Termination*. *InConnected* prozesu guztiek azkenean erabaki bat hartuko dute.
- *Integrity*. Prozesu guztiek gehienez behin erabakitzen dute.
- *Uniform agreement*. Ez daude bi prozesu erabaki desberdinarekin
- *Validity*. Prozesu batek *v* erabakitzen badu, *v* balioa prozesu batek proposatua duelako da.

3.2.5. SMC asinkronoa TrustedPals ebazten

TrustedPals-en jatorrizko lanean [Fort et al., 2006], SMC arazoa sistema fidagarritan *Uniform Interactive Consistency*-ra (UIC) murrizten da. SMC protokoloan, segurtasun-kritikoko akzio batzuk sistema fidagarrian erabiltzen den UIC-ren esku utziko ditu. *SMC-Privacy* propietatea

betetzearren, protokolo honetan kontutan hartu beharreko segurtasun propietateak analizatu beharko dira. *Content Secrecy* eta *Control flow secrecy* betetzen direla bermatu behar da. Prozesuak eta kanalak atalean (3.2.6 atala) definitzen diren komunikazio kanaletetan oinarrituz, *trusted system*-an bi propietateak beteko lirateke, non UIC erabiltzen den. Komunikazio kanalak fidagarriak direnez, ezingo da transmititzen den mezua zelatatu, ezta mezuaren patroia informazioa lortu.

UIC dela eta, *trusted system*-an, prozesu fidagarrien artean bakoitzaren sarrera datuak partekatzen dituzte, F funtzioa prozesatzen dute eta emaitza *untrusted system*-ra bidali baino lehen, haien arteko sinkronizazioa burutzen da. Inplementazioa, Parvédy and Raynal [Parvédy and Raynal, 2004] *uniform consensus*-erako proposatutako algoritmoan oinarrituta dago. Jarraian argudiatuko da SMC-ren bariante asinkrono bat, *uniform consensus* bidez *TrustedPals*-en ebatz daitekeena. Segidan, pseudokode bat prozedura moduan aurkeztuko da: (3.1 irudia).

Algoritmo 3.1: SMC-ren ebazpena *uniform consensus* erabiliz.

```
1 Procedure SMC(x_i)
2   send (i,x_i) to all other trusted processes
3   wait until (for ⌈(n+1)/2⌉ processes q: received (j,x_j) from q)
4   V_i ← {all (j,x_j) received}
5   V ← uniform_consensus(V_i)
6   forall the j ∈ {1,...,n} do
7     if (j,x_j) ∈ V then y_j ← x_j else y_j ← 0  { some default value }
8   r ← F(y_1,...,y_n)
9   return r
```

SMC prozeduran, lehenik p prozesuak (i,x_i) balioa gainontzeko prozesu fidagarriei jakinaraziko die, bera koordinatzailea dela esanaz. $\lceil \frac{(n+1)}{2} \rceil$ q prozesuetatik (j,x_j) jasotzean, balio horiek V_i bektorean gordeko dira, ondoren hauekin $uniform\_consensus(V_i)$ exekutatuko da V bektorean balioak gordez. Jasotako emaitza, V bektorearen aldagaien barne badago (bozketan parte hartutako prozesuak) dagokien balioa ezarriko zaizkie eta gainontzekoei, okerrak edo parte hartu ez dutenak, lehenetsitako balio bat ezarriko zaie, gure algoritmoaren kasuan 0 balioa. Behin balioak ezarrita daudela, hau emaitza bezala itzuliko da.

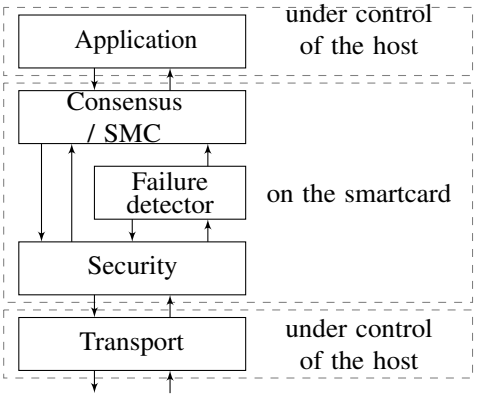
Aurreko algoritmoan arreta jartzen badugu (3.1 irudia), *host* zuzenek (ez maltzurrak) erabaki bat har dezakete *trusted system*-an. Gogora dezagun *host* zuzenak, *well-connected host*-ak eta *InConnected host*-ak osatzen duten multzoa dela.

Uniform consensus-aren lehenengo propietatea hartzen badugu, *SMC-Termination*, nahiz eta sistema asinkronoa izan, prozesu zuzenek gainontzeko gehiengo prozesu zuzenen mezuak jasotzen badituzte, prozesuak amaiera batera iritsiko dira. Ondorioz, *trusted system*-ko *host* zuzenek *uniform consensus*-erako balio bat proposatuko dute. *Uniform consensus*-eko *Termination* propietatetik, lehen deskribatutako prozesu motek, behin-betiko erabaki bat hartuko dute eta denen artean erabakitako balioa *host*-ari pasako zaio.

SMC-Validity propietatean ausnartu dezagun orain. *Host* zuzen guztien segurtasun modelo

guztiek *uniform consensus*-ean parte hartzen dutenez, denek bere V_i balioa proposatuko dute. *Validity* propietatea dela eta, V_i horien artean, V behin betiko balioa erabakiko dute. *Uniform consensus*-aren *Agreement* (adostea) propietateak, erabakitako balioa sarrera bektorean bertan konputatutako balioa dela bermatuko du.

SMC-Privacy propietatea probatzea konplexuagoa da *trusted system*-aren portaeraren menpe dagoelako (*Trustedpals* adibidez). *Truted system*-ak komunikazioan beharrezkoa den informazioa besterik ez du erabili behar, x_i sarreran eta r emaitza irteeran. Hau bermatzeko, pribatutasun garapenen tekniken beharra beharko da, beharrezkoa den konfidentzialtasuna lortzeko eta kanalak ahalik eta pribatuenak izateko. Ondorengo atal batean hau bermatzeko ideiaren bat azaldu da.



3.3 Irudia: Sistemaren Arkitektura.

TustedPals-entzat proposatuko den arkitektura modularra irudian laburbiltzen da, bere geruza eta interfazeekin.(3.3 irudia)

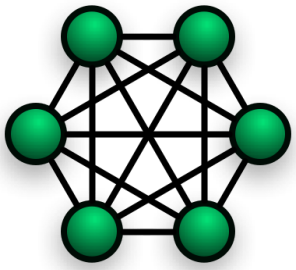
Mezuen partekatzea, *untrusted system*-aren menpe dago, *transport* geruzan burutzen delako. SmartCard-aren barnean (segurtasun modulua), mezuak enkriptatzeko/dekriptatzeko segurtasun mekanismoa, *Security* geruzan exekutatzen da. *Consensus/SMC* eta *Failure Detector* geruzek, *Consensus/SMC* eta *Failure Detector* algoritmoak exekutatzen dituzte hurrenez hurren. Azkeniz, *Application* geruzan, *untrusted system*-ko *host*-en esku dagoena, *Consensus/SMC*-rekin lan egiteko eskaintzen den software aplikazioaren interfazea da.

3.2.6. Trusted system-aren formalizazioa

Orain arte azaldu bezala, *SMC* adostasunera murrizteak, *trusted system*-ak *uniform consensus* algoritmoa edukitzea suposatuko du. Algoritmoaren ulermena errazagoa izateko, sakonean azaltzera pasa baino lehen, orain arte erabilitako kontzeptuak sakonago azalduko ditugu, algoritmoa ulergarriago izan dadin.

Prozesuak eta kanalak

Sistema banatu baten modeloan oinarritzen denez, $n > 1$-eko elkar-konektaturiko prozesu multzo batez osatuko da $\Pi = p_1, p_2, \ldots, p_n$. Prozesuak bi noranzko loturez hornituriko sistema da, non denen artean *fully connected* (3.4 irudia) topologia eraikiko duten. Kanal batez elkar konektaturiko bi prozesu, ondokoak direla esango da eta prozesuak p,q,r... badira, p eta q arteko loturarik c_{pq} bezala.



3.4 Irudia: Fully Connected Network

Algoritmoak eta gertaerak

A algoritmoa automata deterministaz osaturiko multzo batean datza, non bat prozesuko dagoen. Algoritmoa gertaera notazioetan oinarritzen da, beraz , bakoitzak lokalki gertaeren FIFO ilara bat edukiko dutela ustekoz hartzen da. Pausu exekuzio bakoitzean gertaera ilaratik prozesuek gertaerak hartuko dituzte ondoren dagokion aginduak exekutatzeko, mezu bidalketa edo dena delakoa. Mezu jasotze gertaerak ere tratatzen dira, adibidez, mezu bat jasotzean gertaera ilaran atxikituko da. Sistemako prozesu guztiek, algoritmoa infinitu aldiz exekutatuko dutela suposatuko da, *crash* errorea edukitzen dutenak izan ezik.

Erloju globala

Sistemaren sinplifikazio egiteko, erloju global diskretu bat suposatuko da. Prozesuek ez dute erloju globalaren atzipena egingo, beraz alegiazkoa bezala izango litzateke.

Pausoak, gogora dezagun gertaera baten exekuzioa dela, erloju global bati lotuta dago. Gertaeren exekuzioa, modelo lineal batean egiten dela suposatuko da.

Prozesu hutsegiteak

Aurreko atalean azaldu bezala, prozesuek honako hutsegiteak jasan ditzakete:

- *Crash Failure.* $F_c \subset \Pi$ prozesuen azpimultzo bat da. *Crash* eduki duten prozesu guztiak edukiko ditu.

- *Send-Omission.* $F_{so} \subset \Pi \times \Pi$ erlazioen azpimultzoa da. p-k q-rekiko bidaltze omisioa eduki badu, multzoan egongo da.
- *Receive-Omission.* $F_{ro} \subset \Pi \times \Pi$ erlazioen azpimultzoa da. p-k q-rekiko jasotze omisioa eduki badu, multzoan egongo da.
- *Denbora errorea.* $F_{ap} \subset \Pi$ asinkronia jasaten duten prozesuak dira. Multzo honen barnean ez badaude, sinkronoak direla esan nahi du.

Prozesu bat hauetako edozein multzoan badago, *prozesu ez zuzena* dela esango da. $\Pi = F_c$,F_{so},F_{ro},F_{ap}

Prozesu zuzena berriz, aurreko hutsegiteren bat edukitzen ez duten prozesuak dira. Algoritmoan ikusiko dugun bezala, ez da beharrezkoa suposatzea prozesu gehienak prozesu zuzenak izatea. Gerta daiteke, prozesu zuzenik ez egotea.

Bidaltze eta jasotzeak

Prozesuek mezuak bidal ditzakete *Send* primitiba erabiliz. *Send(p,m,q,t)* gertaera honela interpreta daiteke: t denbora momentuan, p prozesuak q-ri bidalitako m mezua. Hau da, p prozesuak *send-omission* bat jasaten ez badu, m mezua kanalean sartuko du. Kanal fidagarritik bidali bada, gertaera q prozesuaren ilara lokalean sartuko dela bermatuko da. Nahiz eta *buffering attack* edo kanalak asinkronia jasan, ez dago denbora mugarik q-ren gertaera ilaran sartzeko. Kanala ez fidagarria bada edo *receive-omission* bat jasan badu m mezua kanaletik hartuko da eta ez da q-ren gertaera ilaran sartuko. Gertaera prozesatu denean, m mezua t' denbora momentuan jaso dela esango da, *Receive(p,m,q,t')*. Prozesuei mezu itxoite selektiboa onartzen zaio.

Prozesuek mezuen iristea entzuten itxoin dezakete.

Mezuak ez dute derrigorrez helburura bidaltzen diren orden berdinean iritsi behar, kanalaren portaerarengatik edo dana dalakoarengatik, mezuak atzeratzea gerta liteke. Mezuak desordenean iristen badira, hauen trataera egokia egiteko suposaketa bat definituko da. Mezuen kontrolerako suposaketa honekin, *send-omission* edo *receive-omission* bat izan den detektatzeko balioko digu. p-tik q-ra bidaltzen diren m mezu guztiak, p egokitutako *sekuentzia zenbaki* bat edukiko dute eta hau bakarra izango da. Mezu batean, espero genuen sekuentzia zenbaki bat agertzen ez bada eta espero dugun sekuentzia zenbakia bornatutako denbora gainditzen badu, *send-omission* edo *receive-omission* bat suertatu dela antzeman daiteke.

Kanalen hutsegitea

Demagun c_{pq} kanala dugula, p-tik q-ra m mezua garaiz iristen dela esango da, baldin eta soilik baldin, bidaltzea eta jasotzearen artean Φ denbora tartea baino gutxiago pasa bada,

non Φ denbora tartearen muga den. Ondorioz, c_{pq} kanalean mezuak azkenik garaiz iristen direla esango da, baldin eta t denbora bat existitzen den non p-tik q-ra bidalitako m mezuak t denboran jada helburuetara iritsi diren.

Φ denbora tartean ez badira iristen, kanalaz edo prozesuaz susmatuko da.

Partzialki sinkronoa den sistema modelo honetan, Φ-ren balioa ez da ezagutzen.

Hasieran kanal bakoitzari lehenetsitako balio bat emango zaio. Komunikazio batean, denbora tarte hori gainditzen ez bada, kanala hutsegiteak ez dituela suposatuko da, baina gerta liteke, lehenetsitako balioa altua izatea sarearen ezaugarrietarako eta nahiz eta denbora tartean mezua garaiz iritsi, hau maltzurkeri edo kanalaren errore bat izatea. Komenigarria da hasierako Φ balioa baxua izatea eta mezuak garaiz iristen ez diren bitartean, tartea handituz joatea, baina zentzuzko modu batean, pixkanaka. Denbora tartea gainditzen den bitartean, kanala susmagarritzat hartuko da. Behin bornea ez bada gainditzen, eta kanaletik datozen gehiengo mezuak garaiz iristen badira, Φ denbora tartea mantenduko da eta kanala fidagarritzat hartuko da. Azken borne hau, gainditzen duten mezu bidalketak susmagarritzat hartuko dira eta ondorioz, kanala ere.

Zeharkako komunikazioa

Bi prozesu p eta q c_{pq} kanala bidez komunika daitezke edo beste prozesuen bidez komunikatu daitezke. Zeharkako komunikazioa kontsideratzen den momentutik, p-k q-ra bidalitako mezuak garaiz iristen dira nahiz eta c_{pq} komunikazio kanalean mezuak garaiz ez diren iristen eta kanala fidagarria ez izan. Beste modu batean esanda, nahiz eta c_{pq} komunikabide zuzena susmagarria izan eta bidalitako mezuak garaiz ez iritsi, zeharkako komunikazioaren bidez bada komunikazio bide bat non p-tik q-ra bidalitako mezuak garaiz iritsiko diren.

Well-Connected prozesuak

Demagun prozesuak eta hauen arteko komunikazio erlazioak dituen grafo bat. Suposatuz gehiengo *host*-ak zuzenak direla eta bi prozesu zuzenen arteko bi noranzko grafoko komunikazio erlazioa existitzen bada, sistemako gehiengo prozesuak konektaturik daude. Ezaugarri hauek betetzen dituzten prozesuak *well-connected* prozesutzat hartuko dira. Hau da, multzo zuzeneko prozesu pare bat hartzen badira, elkarrekin modu fidagarrian, denborazkoan eta omisio gabeko komunikazioa edukiko dute.

InConnected eta OutConnected prozesuak

Well-connected diren prozesu guztiak *InConnected* ere dira, eta p prozesua *InConnected* izango da, baldin eta existitzen da q prozesu bat, non *well-connected* barnean dagoen eta badago komunikazio erlazioa bat q eta p artean.

Well-connected diren prozesu guztiak ere *OutConnected* dira, eta p prozesua *OutConnected* izango da, baldin eta existitzen da q prozesu bat *well-connected* multzoaren barnean eta p eta q-ren arteko komunikazio erlazioa existitzen bada.

$\Diamond\mathcal{P}(om)$ *Hutsegite detektatzailea*

Trusted system-eko prozesu klase motetan oinarrituta, $\Diamond\mathcal{P}(om)$ omisio modeloan bete-behar dituen propietateak birdefinitu behar dira. Prozesu zuzena/okerra klasifikazioa susmagarrien lista batez definituta dagoenez eta susmagarria den ala ez jakin nahi izanez gero, oso argi ikusten da lista honetan begiratuz. Omisio modeloan aritzeko beste bi balio beharko dira: $I\_am\_InConnected_p$ balio boolearra eta $OutConnected_p$ prozesuen lista, ondorioz $\Diamond\mathcal{P}(om)$ hutsegite detektorearen heina $R = 2^{\Pi} \times \{\text{TRUE,FALSE}\}$ da. Hutsegite detektatzaileren $\Diamond\mathcal{P}(om)$ klaseak, Chandra-Toueg [Chandra and Toueg, 1996] lanean definitutako propietate hauek bete beharko ditu omisio modelo batean:

- *Strong Completeness*. Amaieran, *InConnected* prozesu guztiek *OutConnected* ez diren prozesu guztiak betiko ez *OutConnected*-tzat kontsideratu behar dituzte.
- *Eventual Strong Accuracy*. Amaieran, *InConnected* prozesu guztiek *OutConnected* diren prozesu guztiak betiko *OutConnected*-tzat kontsideratu behar dituzte.
- *InConnectedness*. Amaieran *InConnected*-tzat kontsideratzen diren prozesuak, bere burua betiko *InConnected*-tzat kontsideratuko dute.

Ezaugarri hauek *InConnected* prozesuetan arreta jartzen da, *OutConnected* direnen prozesuen multzo erabakitzen dutelako. Honela, prozesu guztiek *InConnected* diren ala ez bermatu behar dute (*InConnectedness*). Denbora edo buffer erasoak, edo kanalaren asinkroniarengatik, prozesu batzuk *OutConnected*-tzat kontsideratuko dira nahiz eta hauek ez izan, ondorioz ez da posible *OutConnected* multzo zehatza sortzea *InConnected* prozesuentzako. Bestalde, *Eventual Strong Accuracy* propietateak, *OutConnected* diren prozesuak amaieran betirako *OutConnected*-tzat hartuko dira eta beraz fidagarriak direla bermatuko da. Eta amaitzeko, *Strong Completeness* propietateak, *OutConnected* ez diren prozesuak betiko fidagarritasuna galduko dute.

3.3. *TrustedPals birdiseinaketaren algoritmoak*

3.3.1. $\Diamond\mathcal{P}(om)$*-n oinarritutako* Trusted system*-eko* consensus

TrustedPals arkitekturaren irudia aurrean dugula (3.3 irudia), *consensus* geruzan arreta jarriko dugu orain. Bertan, definitutako $\Diamond\mathcal{P}(om)$ hutsegite detektatzailea erabiliz, *uniform consensus* inplementatzen da (3.1 irudia). Berez *consensus* algoritmoa *asinkronoa* denez, fase asinkrono

arbitrarioak eduki ditzake. Ondorengo lerroetan, *consensus* algoritmoaren analisi sakonago bat egingo da.

Consensus *Algoritmoa*

Ondorengo 3.2 eta 3.3 algoritmoetan, $\Diamond\mathcal{P}(om)$ hutsegite detektore klasea erabiliz, omisio modeloan oinarritutako *consensus* arazoa ebazten da. Chandra-Toueg lanaren $\Diamond\mathcal{S}$ adostasun algoritmoan oinarritzen da [Chandra and Toueg, 1996]. Chandra-Toueg lanean argi azaltzen den bezala, $\Diamond\mathcal{S}$ $\Diamond\mathcal{P}$ baino ahulagoa da, hortaz, $\Diamond\mathcal{S}$-rekin funtzionatzen badu $\Diamond\mathcal{P}$-rekin ere bai, $\Diamond\mathcal{P} \succ \Diamond\mathcal{S}$. Ondorengo bi baliabideek, p prozesu guztiengatik $\Diamond\mathcal{P}(om)$-ren erabilera adierazi-ko dute: $I\_am\_InConnected_p$ balio boolearra, *InConnectedness* propietatea bermatzen du eta $OutConnected_p$ *Strong Completeness* eta *Eventual Strong Accuracy* propietateak bermatzen ditu.

Algoritmoa koordinatzaile errotazioaren paradigman oinarrituta dago, txandaka exekutatzen da. Txanda bakoitzean prozesu bakar bat koordinatzaile moduan funtzionatzen du. Koordinatzaileak gainontzeko prozesuei balio bat egokitzen saiatuko dira. Gainontzeko prozesuek balio hori onartzen badute, momentuan koordinatzen dagoen prozesua ez susmagarritzat hartzen dutela esanahiko du. Onartutako balio hori adosten bada, koordinatzaileak beste prozesu guztietara *reliable-broadcast* bat bidaliko du eta prozesuek bidalitako adoste emaitza bere baitan hartu behar dute. Berriz, koordinatzaileaz susmatzen bada, hurrengo txanda batera pasa beharko da, non beste prozesu batek koordinatzaile rola hartuko duen. Txanda bakoitza lau fasetan zatitzen da: bozketa fasea, proposizio fasea, proposamenaren onartze fasea eta erabakitze fasea. $\Diamond\mathcal{P}(om)$ hutsegite detektatzailea modu egokian erabiltzeak, azkenean, adostasuna *well-connected* den prozesu batek erabakiko du.

Segidan, Chandra-Toueg algoritmoan oinarritutako algoritmoaren funtzionamendua azalduko da eta bertan egindako aldaketak deskribatuko dira (propose(v_p) 3.2 algoritmoan). Aurkezten den prozedura hau prozesu guztiek exekutatuko dute.

Consensus erabakia hartzen ez den bitartean, koordinatzaileen txandaketa egingo da. Koordinatzailea zein izango den behin erabakita, lehenengo fasean momentuko koordinatzaileak gainontzeko prozesuei estimatutako balioa jakinaraziko die.

- **Hasieraketa fasea**. $estimate_p$, p uneko prozesuak proposatutako balioa izango da. $state_p$ erabakia zein egoeran aurkitzen den adieraziko du, *undecided* edo *decided* balioak edukiko dituena. Hasieraketan erabakirik ez denez hartu, *undecided* egoeran egongo da. r_p ($round_p$), uneko p prozesuaren txanda da, hasieraketan 0-ko balioa ezarriko zaio. p prozesuak $estimate_p$ balioa aldatu duen azken txanda ts_p ($timestamp_p$) izango da, hasieraketan 0-ko lehenetsiko zaiona.

Algoritmo 3.2: $\Diamond\mathcal{P}(om)$ erabiliz *Consensus* arazoa omisio modeloan ebazten : algoritmo nagusia.

```
{Every process p executes the following}
1  Procedure propose(v_p)
2    estimate_p ← v_p                                   {estimate_p is p's estimate of the decision value}
3    state_p ← undecided
4    r_p ← 0                                            {r_p is p's current round number}
5    ts_p ← 0                                           {ts_p is the last round in which p updated estimate_p, initially 0}

     {Rotate through coordinators until decision is reached}
6    while state_p = undecided do
7      r_p ← r_p + 1
8      c_p ← (r_p mod n) + 1                            {c_p is the current coordinator}
9      Phase 1: {All processes p send estimate_p to the current coordinator}
10     send (p, r_p, estimate_p, ts_p) to c_p

11     Phase 2: { The current coordinator tries to gather ⌈(n+1)/2⌉ estimates. If it succeeds,
                  it proposes a new estimate. Otherwise, it sends a NEXT message to all }
12     if p = c_p then
13       wait until ( not (I_am_InConnected_p) or
                      (for ⌈(n+1)/2⌉ processes q: received (q, r_p, estimate_q, ts_q) from q) )
14       if for ⌈(n+1)/2⌉ processes q: received (q, r_p, estimate_q, ts_q) from q then
15         success_p ← TRUE
16         msgs_p[r_p] ← {(q, r_p, estimate_q, ts_q) | p received (q, r_p, estimate_q, ts_q) from q}
17         t ← largest ts_q such that (q, r_p, estimate_q, ts_q) ∈ msgs_p[r_p]
18         estimate_p ← select one estimate_q such that (q, r_p, estimate_q, t) ∈ msgs_p[r_p]
19         send (p, r_p, estimate_p) to all
20       else
21         success_p ← FALSE
22         send (p, r_p, NEXT) to all

23     Phase 3: {All processes wait for the new estimate proposed by the coordinator}
24     wait until ( not (I_am_InConnected_p) or (c_p ∈ Π − OutConnected_p) or
                    received [ (c_p, r_p, estimate_{c_p}) or (c_p, r_p, NEXT) ] from c_p )
25     if received (c_p, r_p, estimate_{c_p}) from c_p then
26       estimate_p ← estimate_{c_p}
27       ts_p ← r_p
28       send (p, r_p, ACK) to c_p
29     else
30       send (p, r_p, NACK) to c_p

31     Phase 4: { If the current coordinator sent a valid estimate in Phase 2, it waits for replies of
                  out-connected processes while it considers itself as in-connected. If ⌈(n+1)/2⌉
                  processes replied with ACK, the coordinator R-broadcasts a decide message }
32     if (p = c_p) and (success_p = TRUE) then
33       wait until [ not (I_am_InConnected_p) or
                      for each process q: ( received (q, r_p, ACK) or
                                            received (q, r_p, NACK) or
                                            q ∈ Π − OutConnected_p ) ]
34       if for ⌈(n+1)/2⌉ processes q:received (q, r_p, ACK) then R-broadcast(p, r_p, estimate_p, decide)
```

Algoritmo 3.3: $\Diamond\mathcal{P}(om)$ erabiliz *Consensus* arazoa omisio modeloan ebazten: erabakia hartzen.

```
{If p R-delivers a decide message, p decides accordingly}
35 when R-deliver(q, r_q, estimate_q, DECIDE) do
36   if state_p = undecided then
37     decide(estimate_q)
38     state_p ← decided
```

- **Lehenengo fasea**. Behin koordinatzailea erabaki dela, c_p, prozesu guztiek, koordinatzailea barne, bakoitzaren estimazio balioa bidaliko diote momentuko koordinatzaileari.

Ondoren aurkitzen diren faseak, proposatutako balioa baten erabakia hartzen ez den bitartean, koordinatzaileen txandaketa egingo da. (Hiru fase hauetan Chandra-Toueg algoritmoarekin alderatuta aldaketa batzuk daude)

- **Bigarren fasea**. Uneko koordinatzaileak, gainontzeko prozesuen estimazioak beharko dituenez, mezuak jasotzen dituen bitartean eta hau ez blokeatzearren, bere burua *InConnected*-tzat hartuko du. Uneko koordinatzaileak $\lceil \frac{(n+1)}{2} \rceil$ estimazio jasotzen saiatuko da eta lortzen baditu, estimazio berri bat proposatuko du, hau gainontzekoei bidaliz. Estimazio kopurua hori lortzen ez badu, koordinatzaileak prozesu guztiei NEXT mezua bidaliko die txanda hori bideragarri ez dela esanez.

- **Hirugarren fasea**. p prozesu guztiek koordinatzaileak proposatutako duen balioaren zain egongo dira. Aldi berean, p prozesua ez blokeatzearren bere burua *InConnected*-tzat hartuko du eta koordinatzailea *OutConnected*-tzat. p prozesuak koordinatzailearengatik baliozko estimazio bat jasotzen badu, ACK mezu batez erantzungo dio koordinatzaileari, bestela NACK mezua bidaliko dio.

- **Laugarren fasea**. Koordinatzaileak bigarren fasean baliozko estimazio bat bidali badu, *OutConnected* prozesuen erantzunen zain geratuko da eta ez blokeatzearren bere burua *InConnected*-tzat hartu du. Koordinatzaileri $\lceil \frac{(n+1)}{2} \rceil$ prozesuek (gehiengo prozesuak) ACK mezua bidali badiote, koordinatzaileak *reliable-broadcast* bat egingo du erabakitako balioarekin [Hadzilacos and Toueg, 1993], prozesu guztiek adostutako balioa jakin dezaten.

Azkenik, algoritmo nagusia exekutatu dela, prozesu bakoitzak emaitza jasoko du.

- **Egokitzapen fasea**. Behin erabakia eta *reliable-broadcast*-a burutu direla, erabakian parte hartutako prozesu bakoitzak erabakitako balioa jasoko du (R-deliver) eta erabakiaren egoera *decided* egoerara pasako da.

p prozesuak m mezua q-ri bidaltzen dionean honakoa transmisioa egoerak pasako dira:

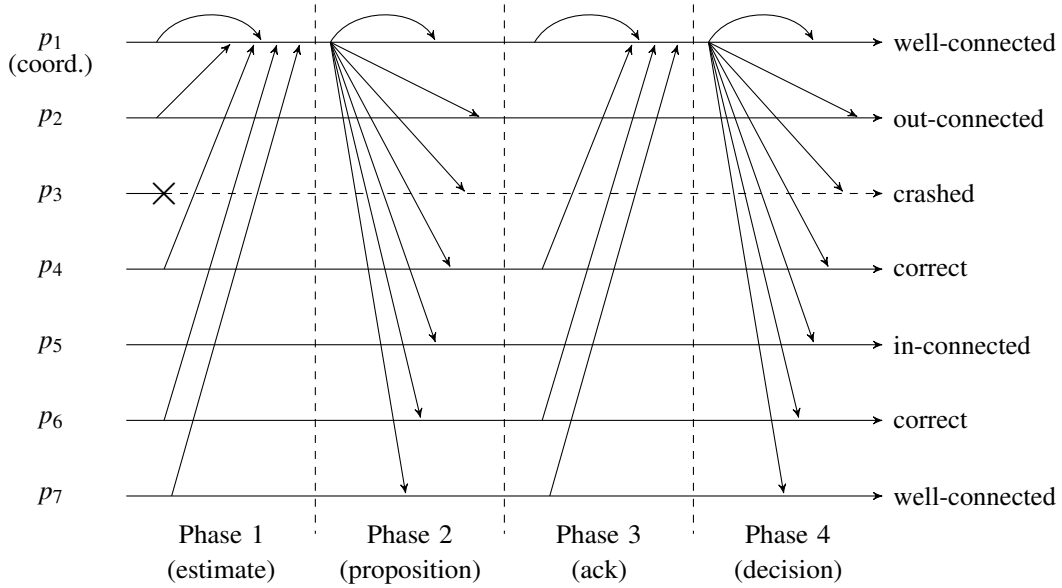
1. p-k m mezua prozesu guztietara, q barne, bidaltzen du, p bere buruari izan ezik.
2. p-ek m mezua jasotzen duenean eta mezu horren helburua beste q prozesu bat bada, p-ek segituan mezu hori gainontzeko prozesu guztiei bidaliko die, bere burua eta jaso duen prozesuari izan ezik.

Hurbilketa honekin, $\Diamond\mathcal{P}(om)$ hutsegite detektatzaileak inplementatuta duen *all-to-all*-az baliatuko da mezu extra gehiago ez sortzeko. Hutsegite detektatzaileak aldizka bidaltzen dituen *all-to-all* komunikazio modelokoak sortzen dituen mezuak aprobetxatuko ditu berarentzat ez diren mezuak birbideratzeko.

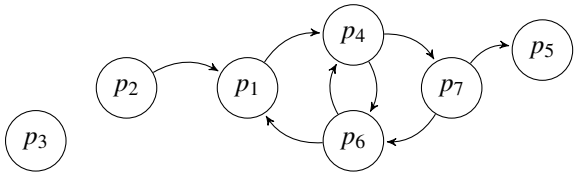
Proposatutako algoritmoaren zuzentasuna ikusi nahi bada, prozesuen itxoite momentuan blokeaketa gerta daitekeen ala ez frogatu beharko da. Bigarren, hirugarren eta laugarren faseetatik ideia orokorra bat atera daiteke: Itxoiteak soilik *InConnected*-tzat dauden bitartean gauzatzen dira. Hortaz, itxoite puntuetan prozesua *InConnected* bada, komunikazioa badu eta beste prozesuen mezuak jaso ditzake, ondorioz ez da prozesua blokeatuko. Zuzentasun froga sakonagoa aztertu nahi bada, *Secure Failure Detection and Consensus in TrustedPals* [Cortiñas et al., 2012] artikulura joan.

Consensus *azalpena adibide baten bidez*

Consensus algoritmoa errazago ulertzeko, irudi batekin azalduko da. *Consensus* adibidean 3.5 eta prozesuen grafoan 3.6 arreta ipiniko da orain:



3.5 Irudia: *Consensus* adibidea



3.6 Irudia: Prozesuen grafoa *Consensus* adibiderako

Algoritmoa jarraituz, lehenik koordinatzailearen aukeraketa egin behar da, adibide honetan p_1 prozesuak kargu hori hartuko du. Txanda honetarako prozesu denek koordinatzaile bera eta bakarra edukiko dute. Behin koordinatzailea erabaki dela, lehenengo fasera pasako da.

- **Lehenengo fasea**. p_1 koordinatzailea bere burua *InConnected*-tzat hartzen duen bitartean, $p_1 \dots p_7$ prozesu guztiek p_1 koordinatzaileari bakoitzaren estimazioak bidaliko dizkiote. Ikusten denez koordinatzailea bere buruari ere bidaltzen dio, prozesu zuzenen balio guztiak eduki behar direlako *consensus* batera iristeko. Adibidean adierazten denez, prozesu zuzen guztiek eta *OutConnected* diren prozesuek, estimatutako balioa bidaltzen dute. p_5 prozesua soilik *InConnected* denez, ezin du estimazioa bidali. p_5-k gainontzeko prozesuekin bidaltze-omisioa jasango du eta bere estimazioa ez da garaiz iritsiko, ondorioz, bere balioa ez da kontutan hartuko *consensus*-aren emaitzarako. *OutConnected* direnak berriz, prozesu zuzen guztiek eta p_2, ez dute bidaltze-omisiorik jasango, beraz estimazioak garaiz iritsiko dira eta balio hauek *consensus*-aren emaitzarako kontutan edukiko dira. p_3 prozesuak hutsegite edo *crash* bat eduki du. Ezin du estimazioa bidali eta adostasunaren erabakian ez du parte hartuko. Behin p_1 koordinatzaileak $\lceil \frac{(n+1)}{2} \rceil$ estimazio baino gehiago lortu dituela, hurrengo fasera pasako da. Adibidearen kasuan, 4 prozesuren erantzunarekin aski luke eta fase honetan 5 prozesuek erantzun dutenez, hurrengo fasera pasako da.

- **Bigarren fasea**. p_1 koordinatzaileak jasotako estimazioak baloratuta, proposamen bat erabakiko du eta proposatutako balioa prozesu guztiei bidaltzen die, bere burua barne. Kasu honetan *InConnected* motako prozesuek proposamena jasoko dute, jasotze omisiorik jasango ez dutelako. Adibidearen kasuan p_1, p_4, p_5, p_6 eta p_7 prozesuak. Koordinatzaileak behin proposamena bidali duela, hau ez blokeatzearren bere burua *InConnected*-tzat hartuko du eta prozesuei proposatutako balioarekin *consensus* erantzunaren zain geratuko da.

- **Hirugarren fasea**. p_1 koordinatzailea berriz bere burua *InConnected*-tzat hartzen duen bitartean, *OutConnected* diren prozesuek *consensus* erantzuna bidaliko diote. Proposatutako balioarekin ados badaude koordinatzaileari *ACK* batekin erantzungo diote, *NACK* bestela. Mezuen gehiengoa *NACK* izanez gero, berriz beste balio ezberdin bat proposatuko beharko luke, baina koordinatzaile tokatzen zaion hurrengo txanda batean. p_2-ren kasuan nahiz eta *OutConnected* izan, kasu berezia da. Proposatutako balio bidali duenean, *InConnected* motako prozesua ez denez, jasotze omisioa eduki du, beraz, fase horretan proposatutako balioa ez daki zein den. Ondorioz, ezin du parte hartu azkeneko bozketan, honegatik aurreko fasean p_1 koordinatzaileari *NACK* balio bidaliko dio . Koordinatzaileak behin gehiengo *ACK* erantzun dituenean, hurrengo fasera pasako da.

- **Laugarren fasea**. Koordinatzaileak behin erabakia hartu duela, erabakitako balioarekin prozesu guztietara *R-Broadcast* bat egingo du. Ondoren *InConnected* diren prozesuek emaitza jasoko dute eta erabakita dagoela esango dute.

3.3.2. Trusted system-*ko hutsegite detektatzailea*

TrustedPals-eko analisi atalean ikusi den bezala, garraio geruzan sinkronizazio ahul edo partzial sistema modelo bat hartzen du. Jarraian hutsegite detektatzailearen $\Diamond\mathcal{P}(om)$ klasearen inplementazioa azalduko da, *TrustedPals* arkitekturako hutsegite detektatzailearen geruzan arreta jarriz (3.3 irudia).

Algoritmo honek *consensus* algoritmoaren oinarriak sortzeaz gain(3.3.1 irudia), aurreko atalean definitutako konektibitate erlazio eta propietateak zehazten ditu (3.2.6 sekzioa).

Algoritmoa, prozesuek beste prozesuei aldizka bidaltzen dizkieten mezuetan oinarritua dago. Planteamendu honek mezu bidalketa atzeratua eskaintzen du eta ondorioz, prozesuen arteko zeharkako komunikazioak ahalbidetzen ditu.

Hutsegite detektatzaile algoritmoa

3.4, 3.5 eta 3.6 algoritmoak $\Diamond\mathcal{P}(om)$ klasearen inplementazioa da, aurreko atalean definitutako propietateak betez (3.2.6 atalean). Algoritmoak prozesu guztiak *OutConnected* multzo batez hornitzen ditu, $OutConnected_p$, eta $I\_am\_InConnected_p$ balio boolearraz. p prozesua *InConnected* bada, $I\_am\_InConnected_p$ balio boolearra $TRUE$ izango da. $OutConnected_p$ aldagaiak uneko eta betiko $OutConnected_p$ prozesu guztiak eguneratuak edukiko ditu baldin eta p-ren $I\_am\_InConnected_p$ balioa *TRUE* bada.

Algoritmoa, prozesu guztiak gainontzeko prozesu guztietara aldizka bidaltzen diren komunikazio mezu trukeetan oinarritua dago (*heartbeat*). Prozesuen arteko mezu horietan konektibitate informazioa garraiatuko da. p prozesuak mezu bat jasotzean, bertan garraiatzen den konektibitate informazioaz baliatuz eta gainontzeko prozesuen ikuspuntuak edukita, bere ikuspuntua eguneratzeko erabiliko du. Honela, p prozesuak duen informazioarekin eta mezuan jasotakorekin, sistemaren konektibitate ikuspuntua eguneratua edukiko du. Ikuspuntu berri hau ondorengo *heartbeat*-ean gainontzeko prozesuei mezuetan bidaliko die. Goazen algoritmoa azaltzera:

p prozesu guztiak M_p $n \times n$-ko matrize batez baliatuko dira konektibitate informazioa gordetzeko (adibidez, 3.2 edo 3.6 irudietan marraztutako arkuak gordeko dira). p prozesuaren *heartbeat* mezu bakoitzeko, mezu omisioak detektatzeko sekuentzia zenbaki bat eta M_p matrizea garraiatuko da. Mezuak jasotzen diren heinean, hauei dagokien sekuentzia zenbakia eta gainontzeko informazioarekin FIFO motako buffer batean gordeko dira. p prozesuak, q prozesuaren mezu bat jasotzean, p-ren sarrera kanalen arabera eta jasotako M_q matrizearen arabera, bere M_p matrizea eguneratuko du. Hasieran prozesu guztiak zuzenak direla suposatuko denez, M_p matrizeko elementuek 1 balioa edukiko dute. q prozesuak bidalitako mezu guztiak p prozesuari garaiz iritsi baldin bazaizkio, $M_p[p][q]$-ren balioa 1 mantentzen jarraituko du, bestela $M_p[p][q]$-ri 0 balioa egokituko zaio.

M_p matrize adjazente irauli bat da, norantz grafoaren (0,1)-dun matrizea da eta $M_p[p][q]$ balio bakoitzak q-tik p-ra bideren bat badagoen adieraziko du. M_p matrizeak $OutConnected_p$ multzoa eta $I\_am\_InConnected_p$-ren balio kalkulatzeko beharrezkoa den informazioa eskaintzen du. Algoritmo nagusian antzeman daiteke prozesuek bere burua ez dutela monitorizatzen, ondorioz, matrizearen diagonala beti 1-eko balioa edukiko du.

Algoritmoa p eta q prozesuen arteko edozein luzerako bide bat aurkitzeko M_p matrizearen potentziaz baliatuko da ($A_p = (M_p)^n$), prozesuen zuzeneko edo bitartekariko konexioa izan daitekeena. p eta q prozesuen arteko komunikazioan, bidalitako mezua garaiz iristeko bide bat dagoen ala ez esango du, $q \xrightarrow{*} p$. $A_p[p][q] \neq 0$ den balioren bat itzultzen badu, bi prozesuen arteko omisiorik ez dela egon esanahi du, eta ondorioz zuzeneko edo zeharkakoa bidea dagoela.

$OutConnected_p$ prozesu multzoa eta $I\_am\_InConnected_p$ aldagaia, M_p matrizea aldatzen den bakoitzean, *update_Connectivity*() prozedura bidez eguneratuko dira (3.5 irudia). p prozesua bakoitzarentzat oso garrantzitsua da bere *InConnectivity*-a kontrolatzea M_p-k duen informazioaren edukia balizkoa izateko. p prozesuaren *InConnectivity*-a *update_Connectivity*() prozeduran egiaztatzen da eta irteera $I\_am\_InConnected_p$ aldagaiaren balioa izango da. *InConnectivity*, $\lfloor n/2 \rfloor$ prozesuk baino gehiagok p prozesuarekin modu egokian komunika daitezkeela esanahi du.

Algoritmo nagusian, p prozesu guztiek ondorengo hiru ataza hauek exekutatu behar dituzte:

- **1.go Atazan**(14 lerroa), p prozesuak gainontzeko prozesuei aldizka *heartbeat* mezu bat bidaltzen die, M_p matrizea eta sekuentzia zenbakia barne dituelarik. Mezua bidali denean, helburura lotutako sekuentzia zenbakia handitzen da.

- **2. Atazan** (21 lerroa), p prozesuak q prozesuari dagokion mezua garaiz jasotzen ez badu (dagokion $next\_receive_p[q]$ sekuentzia zenbakian), $M_p[p][q]$ balioa 0-ra pasa beharko da.

- **3. Atazan** (28 lerroa), jasotako mezuak prozesatzen dira. p prozesuak beste q prozesu batetik jasotzen dituen mezuak FIFO buffer batean sartzen ditu (29 lerroa) eta $next\_received_p[q]$ bufferra sekuentzia zenbakiaren arabera hustuko da (31 lerroan). Behin q prozesutik espero diren mezuak prozesatuta daudela, q prozesutik datozen mezuen bufferra hutsik dagoela esan nahi du eta ondorioz $M_p[p][q]$ balioa 1-era pasako da. Honekin, p prozesuak, dagokion matrize lerroa beteko du, espero ziren beste prozesuen mezuak garaiz jaso dituela esanaz. Sarearen egoera desberdinengatik, gerta daiteke espero ez den zenbaki sekuentzia bat bufferretik hartzea, agian omisio bat gertatu delako. Kasu hauetan, ilara korritzen jarraituko da edo espero den sekuentzia zenbakia iritsi arte itxaron beharko da.

Algoritmo 3.4: $\Diamond\mathcal{P}(om)$ omisio modeloan: algoritmo nagusia.

```
1  Procedure main()
2    OutConnected_p ← Π
3    I_am_InConnected_p ← TRUE
4    forall the q ∈ Π − {p} do
5      Δ_p(q) ← default time-out interval          {Δ_p(q) denotes the duration of p's time-out interval for q}
6      next_send_p[q] ← 1                         {sequence number of the next message sent to q}
7      next_receive_p[q] ← 1                      {sequence number of the next message expected from q}
8      Buffer_p[q] ← ∅
9    forall the q ∈ Π do
10     forall the u ∈ Π do
11       M_p[q][u] ← 1                            {M_p[q][u] = 0 means that q has not received at least one message from u}
12     Version_p[q] ← 0                           {Version_p contains the version number for every row of M_p}
13   UpdateVersion_p ← FALSE
14 || Task 1: repeat periodically                 {Sending heartbeats}
15   if UpdateVersion_p then                      {p's row has changed}
16     Version_p[p] ← Version_p[p] + 1
17     UpdateVersion_p ← FALSE
18   forall the q ∈ Π − {p} do
19     send (ALIVE, p, next_send_p[q], M_p, Version_p) to q   {sends a heartbeat}
20     next_send_p[q] ← next_send_p[q] + 1         {p updates its sequence number for q}
21 || Task 2: repeat periodically                 {Checking time-outs}
22   if ( p did not receive (ALIVE, q, next_receive_p[q], M_q, Version_q)
        from q ≠ p during the last Δ_p(q) ticks of p's clock ) then   {next message not received timely}
23     if M_p[p][q] = 1 then
24       Δ_p(q) ← Δ_p(q) + 1
25       M_p[p][q] ← 0                            {the potential omission is reflected in M_p}
26       UpdateVersion_p ← TRUE
27       call update_Connectivity()
28 || Task 3: when receive (ALIVE, q, c, M_q, Version_q) for some q   {Processing msgs. in order}
29   insert (ALIVE, q, c, M_q, Version_q) into Buffer_p[q]
30   while (ALIVE, q, next_receive_p[q], M_q, Version_q) ∈ Buffer_p[q] do   {it is the next expected message from q}
31     call deliver_next_message(q, M_q, Version_q)   {the message is delivered}
32     remove (ALIVE, q, M_q, next_receive_p[q], Version_q) from Buffer_p[q]
33     next_receive_p[q] ← next_receive_p[q] + 1
34   if Buffer_p[q] = ∅ then
35     M_p[p][q] ← 1                              {so far, p has received all messages from q}
36     UpdateVersion_p ← TRUE
37   if M_p has changed then call update_Connectivity()
```

Algoritmo 3.5: $\Diamond\mathcal{P}(om)$ omisio modeloan: update_Connectivity() prozedura.

```
38 Procedure update_Connectivity()
39   A_p ← (M_p)^n                                {A_p is the n-th power of the M_p matrix}
40   forall the u, v ∈ Π do
41     if A_p[u][v] > 0 then A_p[u][v] ← 1
42   Out ← ∅
43   forall the q ∈ Π do
44     if (Σ_{i=0}^{n−1} A_p[i][q] ≥ ⌈(n+1)/2⌉) then Out ← Out ∪ {q}
45   OutConnected_p ← Out
46   I_am_InConnected_p = (Σ_{i=0}^{n−1} A_p[p][i] ≥ ⌈(n+1)/2⌉)
```

deliver_next_message() prozedurak (3.6 irudia), mezuak garraiatutako informazioarekin, M_p matrize adjazentea eguneratzeko erabiltzen da. Prozeduran, p prozesuak q prozesutik jasotako M_q matrizearen q-garren lerroa M_p matrizean kopiatzen du. Honela, p-k q-ren sarrera konektibitateari buruz informazioa lortzen du. Gainontzeko u prozesuak kontutan

Algoritmo 3.6: $\Diamond\mathcal{P}(om)$ omisio modeloan: deliver_next_message() prozedura.

```
47 Procedure deliver_next_message()
48   forall the v ∈ Π do M_p[q][v] ← M_q[q][v]          {q's row of M_q is systematically copied into M_p}
49   Version_p[q] ← Version_q[q]
50   forall the u ∈ Π − {p,q} do
51     if Version_q[u] > Version_p[u] then               {q's information about u is more recent than p's}
52       forall the v ∈ Π do M_p[u][v] ← M_q[u][v]
53     Version_p[u] ← Version_q[u]
```

hartzeko, bertsio zenbaki mekanismo bat erabiltzen da u prozesuen sarrera konektibitatearen informazioko kopia zaharrak ekiditeko. p prozesuak M_p-n M_q-ren u-garren lerroa bakarrik kopiatuko luke, baldin eta lerroari lotutako bertsio zenbakia, altuago bada.

Matrize trukatzearen periodikotasunaren mekanismoak, prozesuen konektibitate informazioa modu zeharkako batean ezagutzeko aukera ematen digu, dagokion komunikazioa erlazioa denbora borne barnean egingo dela esanaz. Kasu hau, *well-connected* eta *InConnected* edo *OutConnected* diren prozesuetan emango da. Bestalde, p *OutConnected* ez den prozesu bat, denbora/buffering erasoak eta kanalaren asinkronia dela eta, *OutConnected* bezala joka dezake, baldin eta p prozesua eta *well-connected* prozesu multzoren arteko bide ordezko bat existitzen bada, non komunikaziorako ezarritako denbora muga ez da gainditzen. Berdina esan genezake *InConnected* prozesuentzako, *InConnected* ez den prozesu bat *InConnected* bezala joka dezakeela, baina nahiz eta hau gerta daitekeen sinplifikatzearren egoera hau deuseztatuko da.

Zuzentasun froga aztertu nahi bada, *Secure Failure Detection and Consensus in TrustedPals* [Cortiñas et al., 2012] artikulura joan.

4 KAPITULUA

Simulazioaren diseinua

Sare banatuen eta *TrustedPals* ingurunea 3 kapituluan landu ondoren, aurkeztutako algoritmoen inplementazioa eta probaketa egin behar da. Hau aurrera eramateko algoritmoak simulatzea erabaki da, sare erreal batean neurtzea oso garestia izango litzatekeelako, bai lan orduetan eta baita ekipamenduan ere. Atal honetan algoritmoen simulazioa aurrera eramateko eman beharreko pausoka zehaztatuko dira. Lehenik algoritmoak simulatzeko betebeharren azterketa eginen da. Betebeharrak zehaztuak daudelarik, simulagailu desberdinen arteko azterketa baten ondoren gure betebeharrak asetzen dituena aukeratuko dugu. Behin aukeratu dela, *Consensus* eta *hutsegite detektatzaile* algoritmoak aukeratutako simulagailuan nola garatuko diren azalduko da.

4.1. Algoritmoak simulatzeko betebeharren azterketa

4.1.1. Sare Simulagailua

Gaur egunean sareen hedapena dela eta, proiektu aunitz aurki daitezke aplikazioek sarean duten portaera aztertzeko. Trenak hauek simulagailuak dira. Beharren arabera bat edo beste aukeratu daiteke, bakoitzaren ezaugarriak aztertuz gure nahiak beharrak asetzeko. Badira makina fisikoetan zuzenean sareak simulatzen dituzten simulagailuak eta beste batzuk makina birtualak erabiliz sareen portaerak simulatzen dituztenak. Batez ere, proiektu honen betebeharrak asetzen dituzten simulagailuetan arreta jarriko da (PlanetLab, NS-3).

Aztertuko diren sistemak, kronologia sekuentzia bat jarraitzen dituzten gertaerak irudikatzen dituen gertaera diskretuak simulagailuak dira. Gertaera bakoitza, momentu konkretu batean gertatzen dena, sistemaren egoera aldaketa bat suposatzen du (gertaera bat adibidez, pakete baten bidalketa izan daiteke edo pakete baten erorketa). Sarearen simulazioan zehar, fitxategi batzuk sortzen dira, gerorako, behin prozesaturik daudela, sarearen ezaugarriak eta gertatutakoak analiza daitezkeenak; hala nola, atzerapena, pakete galerak edo omisioak.

4.1.2. Betebeharrak

Sare simulagailuaren software-ak jarraian aurkezten diren betebehar guztiak edo gehienak bete behar ditu. Ezaugarri hauetako baten bat ez badu betetzen hausnarketa egingo da, irteera bide bat pentsatuko da, moduluaren bat inplementatuko da edo dana dalako.

- **Lizentzia**

  Proiektuaren garapena software librean egitea pentsatu da, horregatik ezaugarri hau kontutan hartzekoa litzateke, baina ez da derrigorrekoa. Baita, lizentzia akademikoa eskaintzen dituzten sistemak ere onartuko dira, hau da, libreak direnak helburu komertzialentzat izan ezik. Aipatzekoa da, murrizketa hauekin, aplikazio mota hauek software libretzat ez direla hartzen.

- **Sistema eragilea**

  Hobe plataforma anitzetan egikaritu ahal izatea, baina aurreko filosofiarekin jarraitzearren software librean oinarritutako sistemak (GNU/Linux) baloratuko dira. Betebehar hau nahitaezkoa izango da.

- **Sare protokoloak**

  TrustedPals Interneten erabiliko den middlewarea da, beraz Internetek betetzen dituen protokoloak bete beharko ditu. Nodoak elkarrekin lotuak egongo direnez, *full connected network* bat sortzeko, peer-to-peer lotura ahalbidetzea ere derrigorrezkoa litzateke. Protokoloen artean, UDP/IP ere eduki beharko du, komunikazioak konexio gabekoak izango direlako.

- **Nodo multi-interfazeduna**

  Sareko nodo bakoitzak interfaze anitzen routerren beharra izan beharko dute, nahiz eta hau oinarrizkoa ezaugarri bat izan. Era zuzenean ezin bada lortu, zeharkako modu batean lortzeko aukera eskaini beharko luke, adibidez tarteko azpi nodo baten sorrerarekin.

- **Omisioak simulatzeko ahalmena**

  Sarean omisioak simulatzeko ahalmena eskaini behar du. Bai *send/receive omission* eta *crash*.

- **Erasoen simulazioa**

  Sare kanalari erasoak burutzeko aukera eman beharko du. Adibidez, aurreko ataletan azaldu bezala, *buffering* eta *asinkronia* bezalako gertaeren simulazioa eskaini behar du.

- **Sare banatua**

  TrustedPals middleware-a sare banatu batean inplementatzeko pentsatua dago, beraz derrigorrezkoa da sistema banatu bat simulatzea.

- **Protokolo/Aplikazio errealen erabilera**

  Simulagailu gehienak, integrazioaren arrazoiengatik, protokoloak zerotik eraikitzen dituzte. Gerta liteke, protokoloen garapena ez zehatzarengatik emaitzek zehaztasuna galtzea. Hau garrantzi handiago du aplikazio errealak exekutatzen direnean.

  Honela, murriztapen hauek direla eta, aplikazio errealak sareetan simulatzearen murriztapenak garrantzia handia dute. Sare errealeko protokoloen erabiltzeko (TCP eta UDP normalean) posibilitatea kontutan hartuko da, hau da, aplikazioen moldaketa egin gabe simulagailuan exekutatzeko aukera.

- **Emulazioa modua**

  Nahiz eta hau simulagailu baten funtzionalitateetan normalean ez egon, baloratuko den ezaugarri bat da. Sare emuladore batek, sare errealekin interakzioa ahalbidetzen du. Beharrezkoa duten aplikazioentzako, emaitza zehatzagoak jaso daitezke.

- **Komandoak**

  Komando bidezko euskarri edukitzea garrantzitsua izango da. Komando bidez simulazioak automatizatuagoak egiteko *script*-ak eraiki daitezke. Sare kasu batzuen portaera ikusi nahi bada, automatizazioaren aukera hau interesgarria izango da.

- **Simulagailuaren bizitasun egoera**

  Puntu honetan, simulagailuak zein laguntza euskarri eskaintzen dituen eta komunitatearen egoera ere baloratuko da. Simulagailuaren eguneraketak eta mantenua oso garrantzitsua da, eskaini ditzakeen funtzio berriak, optimizazioengatik edo errore trataerengatik. Laguntza aurkitzeko, komunitate aktibo bat edo kontaktu zuzeneko foru bat edukitzea ere eskertzekoa da.

4.2. Aztertutako simulagailuak

Aurreko atalean azaldutako betebeharretan oinarritua, ondorengo simulagailuak aztertuko dira: GTNetS, OPnet, OMNet++, PlanetLab eta Network Simulator 3 (NS-3). Gure betebeharrak gehien asetzen duten simulagailuei azterketa sakonago bat egingo zaie. Badira beste simulagailu pila, interesgarria izan daitekeena OneLab da. Munduko Testbed edo proba banku multzo bat biltzen ditu, PlanetLab barne duelarik. TestBed edo banku proba garapen handiko proiektuen esperimentaziorako plataforma bat da. Banku probek teoria zientifiko, elementu konputazional eta beste teknologiak konprobaketa zehatz eta garden frogatzeko modu bat da. Filosofia antzekoa denez, proiektu honetan PlanetLab bakarrik aztertuko da, baina geroko proiektu batzuentzat kontutan eduki daitekeen sistema da, ematen dituen simulazio aukeren desberdinengatik.

Sare erreal batean simulazioa egiteko ere aukera egon da, Informatika Fakultateko laborategi batean simulazio sistema eraikiz. Hau atzera bota zen laborategian egoteko aukera zailengatik eta simulagailu batek edozein tokitatik lan egiteko aukera ematen duelako.

4.2.1. GTNetS

Georgia Tech Network Simulator (GTNets) [GTNetS, 2014] BSD lizentziapean (software librea) garatutako simulagailua da. Eskala handiko sare topologiak diseinatzeko eta hauen portaera aztertzeko erabili oi den simulagailua da. GTNets-ren lan inguruneak simulazio sare topologiak sortzeko ahalbidetzen du (nodoez eta hauen arteko komunikazio loturak osatutakoa), eta erabiltzaileak hauen gainean deskribatutako aplikazioaren datu-fluxua azter dezake.

Simulagailuak oinarrizko protokoloak garatuak ditu eta horretaz gain simulagailuaren azken eguneraketa 2008-koa denez baztertutako simulagailua da. Simulagailuak erabiltzaile murritzeko komunitatea du eta honek erabilera ikasketarakoan zailtasunak ekar ditzake.

4.2.2. OMNet++

OMNet++ [OMNet++, 2014] arkitektura modular eta hedagarriko simulazioa plataforma bat da, hain hedagarria da, ingurune desberdinetan erabiltzen den simulagailua dela, adibidez, sare, protokolo, multiprozesadore edo hardware arkitektura modelaketan. Berez ez da sare simulagailua, oro har modela daiteken edozein gertaera diskretu eta mezu trukatze sistemarako erabilgarria den simulagailua da.

OMNet++ objektuei zuzendutako sistema da, sinple eta modularra, ondorioz eskala handiko sistemak erraz eraiki daitezke. *OMNet++*en oinarrizko entitatea modulua da, atomikoa edo beste azpi-moduluez osatuta daitekeena. Egitura konplexuak sortzeko moduluen konbinaketa bidez egiten da. Sarearen topologia eraikitzeko, garatzaileak garatutako NED (NEtwork Description) lengoaiarekin definitzen da, moduluen definizioez hornituta dagoena, sareak eta kanalak.

Nahiz eta *OMNet++*ek sare simulaziorako osagai zehatzak ez eskaini, badira simulagailuaren nukleotik kanpo garatzen diren ingurune bereziak, zeintzuk bere garapen zikloak dute. Sare simulazioaren ingurunean interesa eduki dezaketenen artean aipagarriena INET [INET, 2014] da, gainera eguneratuena da.

- **Lizentzia**. Garatzaileak garatutako *Academic Public License*-pean dagoen sistema da, soilik ingurune akademikoetarako librea dena. Bestalde, INET lan ingurunea software librea da.
- **Sistema Eragilea**. GNU/Linux edo Microsoft Windows sistemetan erabili daiteke.

- **Lengoaia**. C++ eta NED (NEtwork Description), sare topologiak eraikitzeko garatutako sistema propioa.

- **Protokoloak**. INET, hari eta hari gabeko sistematarako garatutako TCP/IP-n oinarritako funtzio multzoa da eta urteak pasatzen diren heinean estandarizatzen diren beste protokoloak batzuk dira.

- **Nodo multi-interfazeduna**. Nodo bakoitzari nahi diren interfaze kopuruak ezar zatzaizkio. Tresna oso osatua da puntu honetan.

- **Omisioak simulatzeko ahalmena**. Simulatzeko aukera asko ematen ditu. Socket-en erabilera egin daiteke omisioak simulatzeko, sarrera edo irteera socket-a suntsituz eta sortuz. Beste aukera ugari daude.

- **Erasoen simulazioa**. Badira aplikazio ugari eta sarea trafikoa handitzeko edo beste zereginak egiteko jada inplementatuak daude. Baliagarria izan daiteke pakete injekziorako edo beste funtzio batzuetarako.

- **Sare banatua**. Sare banatu bat eraikitzeko ahalmena ematen du. Bai Ethernet sare batean edo peer-to-peer konexioaz osaturiko sare batean. Simulagailu honek erabilera anitzak eskaintzen ditu.

- **Protokolo/Aplikazio errealen erabilera**. Lehen esan bezala, protokolo ugari dauzka inplementatuta eta hauek mundu errealeko protokoloekin lan egiteko ahalmena eskaintzen dute, hau *OppBSD* [OppBSD, 2014] bidez lortu daiteke. *OppBSD*-k *FreeBSD* kernel-eko TCP/IP protokoloa OMNet-era egokitzen du. Aplikazio errealekin lan egite posible da, baina hauen konplexutasuna sinplea izan behar du [realworldOMNeT, 2014].

- **Emulazioa modua**. Emulatzeko aukera ematen da.

- **Komandoak**. Komando bidezko lan egiteko aukera ematen du, honetaz gain badu interfaze grafikoaren lan egiteko aukera.

- **Simulagailuaren bizitasun egoera**. Simulagailua zein moduluak garatzen dihardute, bakoitza erritmo desberdinean INET eta OMNet++ garatzaileak desberdinak direlako.

  Batez ere mundu akademikoan oso hedatua dagoen simulagailua da, batik bat eskaintzen den moldagarritasunarengatik Baliabide ugari daude eskura, hala nola tutorialak, manualak eta artikuluak. Badira ere foruak erabiltzaileen kezkak ebazteko.

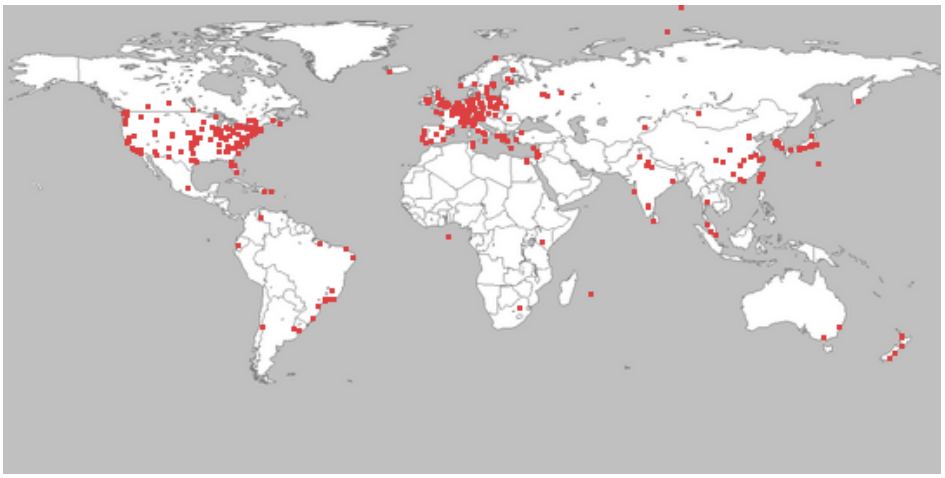
- **Balorazioa**. OMNet++ simulazio tresna oso interesgarria da, diseinu modularrarekin, non erabiltzaileak programazio ingurunetik simulatu nahi duen dena kontrolpean eduki dezakeena, NED fitxategien bitartez edo C++ moduluen bitartez. Euskarri oso garatua du bai komando lerroan zein interfaze grafikoan.

Aspektu negatiboen artean, mundu errealeko aplikazioekin duen interakzioa urria da. Simulatutako aplikazioa etorkizunean mundu errealean probatu nahi bada, hau konplexutasun gutxiko izan beharko da.

4.2.3. PlanetLab

PlanetLab [PlanetLab, 2014] sare ikerketa global bat da, sare zerbitzu berrien garapena eskura ematen dituena. 2003-n hasi zen proiektua eta geroztik 1000 ikerketa baino gehiago egin dira. Erakunde akademiko askotan eta industria munduko laborategi askotan ikertzeko erabili den tresna da, adibidez, biltegi banatuak, network mapping edo peer-to-peer sistemak.

Gaur egun PlanetLab 512 lekutan dauden 1172 nodoz osaturik dago.



4.1 Irudia: PlanetLab

- **Lizentzia**. BSD motako lizentzia garatutako sistema da eta kontribuzioren bat egin nahi bada BSD motako lizentzia betetzen dela bermatu beharko da. Sistema erabiltzeko, erabiltzaile partikularrak ezin du zuzenean erabili, horretarako erakunde baten parte-hartzailea izan behar da eta hau PlanetLab konsortzioaren barnean egon behar du. Erabileraren arabera suerta daiteke ordaindu beharra, baina akademiko moduan lan egiteko ez da diru aportaziorik egin behar.

- **Sistema Eragilea**. PlanetLab osatzen duten nodoek *Linux vserver* sistemaz osaturiko sarea da, gaur egun ofizialki, LXC sistematara migratzen hasi dira, Linux kernelean birtualizatutako edukiontzi birtualen inplementazioa. Hau exekutatzeko API ugari daude eta edozein sistema eragilearekin lan egin daiteke.

- **Lengoaia**. Garapena, XML-RPC edo antzeko liburutegiak dituzten lengoaiekin lan egin daiteke. Adibidez, C++, Java edo Python.

- **Protokoloak**. BSD oinarritutako sistematan garatutako tresna denez, gaur egungo protokolo nagusienak erabili daitezke eta komunitateak geroz eta funtzionalitate gehiago sartzen dituzte.

- **Nodo multi-interfazeduna**. Nodo errealak balira bezala jokatzen dute, Linux sistema-duna. Beraz arazorik ez da edukiko.

- **Omisioak simulatzeko ahalmena**. Portaera erreala bezala duenez, omisioak sortzeko aukera ere bada.

- **Erasoen simulazioa**. Portaera erreala bezala duenez, erosoak sortzeko aukera dago.

- **Sare banatua**. PlanetLab sistema, sare banatuekin lan egiteko garatu da. Sarreran aipatu bezala, munduko leku desberdinetan kokatutako konputagailuetan dagoen sistema da, sare birtual bat eraikitzen dutenak.

- **Protokolo/Aplikazio errealen erabilera**. Protokoloak esan bezala, linux birtualak erabiltzen direnez sisteman inplementatuta daude. Ondorioz, linux birtualen erabilera dela eta, edozein BSD sistema motarekin bateragarritasuna duen aplikazio erreala sor daiteke.

- **Emulazioa modua**. Mundu errealean eraikitako sistema da, beraz hau beteko da beti.

- **Komandoak**. Sistemara konektatzeko komandoz egin behar da, ssh bidez konexioa egin behar delako. Gero bere baitan lan egiteko, PlanetLab shell dugu. Beraz, script-en sorrera posiblea izango da eta automatizazioa ere.

- **Simulagailuaren bizitasun egoera**. Aurreko puntu batean esan bezala, sistema bizi bizirik dago eta gaur egun LXC sistemara migratzen dihardute. Euskarrien aldetik kurtso eta tutorial ugari daude, eta kezka edo arazoren bat badago ere email lista badago.

- **Balorazioa**. Gure betebeharrak asetzen ditu eta abantailetako bat da testbed-aren erabilera da. Emaitza zehatzak eman ditzake, mundu errealeko sistema batean gainean lan egiten duelako, nahiz eta nodo birtualak izan. Aplikazioa errealen erabilera ere egin daiteke. Desabantaila nagusia, kontu bat edukitzako eta erabiltzaile izateko, erakunde baten parte-hartzailea izan behar duzula, hau da, PlanetLab Konsortzioan egon.

4.2.4. Network Simulator 3 (NS-3)

NS-3 [NS-3-Consortium, 2014], gertaera diskretuen simulagailua bat da, NS-2-ren ondorengoa izatera pasa dena. NS-3, hasieran Mathieu Lacage lan batean hasi zen yans simulagailuan (Yet Another Network Simulator), bertan ns-2-ren akats multzo bat aurkitu zituen eta balorazio

batzuen ondoren simulagailua zerotik inplementatzea erabaki zuten. Erroreen artean aldakortasun eza zegoen, moduluen arteko dependentzia eta objektuei zuzendutako programazio teknikaren erabilera urriarengatik.

Bere aurrekariarekin alderatuta, NS-3 soilik C++en garatutakoa da, eta lengoaia honetan ere erabiltzen da, baina Python lengoaiaren erabilera ere eskaintzen du. NS-3ren garapena 2006-ean hasi zen eta oraindik garatzen ari dira, bug-ak konpontzen eta funtzionalitate berriak eransten. Beraz goazen azterketa sakonago bat egitera.

- **Lizentzia**. GNU GPL v2 erabiltzen du. Software librea da eta edozeinek erabil, aztertu, banatu eta alda dezake.
- **Sistema Eragilea**. POSIX bezalako sistemak, GNU/Linux, BSD, OS X, eta Microsoft Windows (Cygwin edo MinGW-rekin).
- **Lengoaia**. C++-en garatutakoa da eta simulazioak C++ edo *Python*-en egin daitezke. Modulu multzo batzuk badira *Python*-en lan egiteko, baina ez guztiak. Funtsezkoenak daude eta pixkanaka garatzen doaz.
- **Protokoloak**. Protokolo komunenak barneratuta dauzka, TCP, UDP..., eta urteak pasatzen diren heinean estandarizatzen diren protokoloak barneratzen dira.
- **Nodo multi-interfazeduna**. Nodo bakoitzari nahi diren interfaze kopuruak ezar zatzaizkio. Tresna oso osatua da puntu honetan.
- **Omisioak simulatzeko ahalmena**. Simulatzeko aukera asko ematen ditu. Socket-en erabilera egin daiteke omisioak simulatzeko, sarrera edo irteera socket-a suntsituz eta sortuz. Beste aukera ugari daude.
- **Erasoen simulazioa**. Badira aplikazio ugari inplementatuak eta hauen artean trafikoa injektatzen dutenak. Oso baliagarriak izan daitezke *Buffering* bezalako erasoak simulatzeko, nahi den momentuan, nahi den denbora eta nahi den kanala datuez bete dezakeelako. Bestalde sinkroniaren aldaketa egiteko, badago denboran zehar kanalen parametroak aldatzeko aukera.
- **Sare banatua**. Sare banatu bat eraikitzeko ahalmena ematen du. Bai Ethernet sare batean edo peer-to-peer konexioaz osaturiko sare batean. Simulagailu honek erabilera anitzak eskaintzen ditu.
- **Protokolo/Aplikazio errealen erabilera**. Lehen esan bezala, protokolo ugari dauzka inplementatuta, baina honetaz gain, API ugari bere baitan ditu protokolo zein aplikazio errealekin lan egiteko. BSD socket-ekin lan egiteko aukera ematen du, NSC-ren erabilerako API-a garatu da eta TCP bezalako protokolo errealekin lan egiteko aukera ematen du. Urteak pasa diren ahala, API berriak ateratzen dira funtzionalitate gehiago sartuz.

- **Emulazioa modua**. Aurreko balorazioan esan bezala, API ugariren garapena egiten ari dira. Bada aukera emulazioa egiteko eta sare errealetara datuak bidaltzeko aukera. *Testbed*-ekin lan egiteko aukera ematen du eta ondoren aztertuko dugun simulagailuarekin (PlanetLab) lan egiteko ere aukera ematen du, hala nola beste simulagailu ugarirekin.

- **Komandoak**. Komando bidezko aukera ematen du, operazio guztiak komando lerroan egiten bai dira, ondorioz aukera dago *script*-ak egiteko. Badu *shell* aukera.

- **Simulagailuaren bizitasun egoera**. Komunitatea simulagailu garatzen dihardu, funtzionalitate gehiago erantsiz eta beste hainbat arazo konpontzen. Geroz eta hedapen handiagoa du simulagailu honek munduan. Komunitatea oso handia egin da eta proiektu askotan erabiltzen den tresna da. NS-3-k simulagailuaren arduradunekin hizketan ibiltzeko korreo lista bat du eta bestalde, bakoitzaren kezkak galdetzeko *Google Groups*-en talde bat sortu da. Azkartasunez erantzuten dute moderatzaileek. Oso lagungarri eta eskertzekoa da egiten duten esfortzua. Software librea dela eta, komunitatea asko hedatu da eta moderatzaileaz gain, beste pertsona ugarik, bakoitzaren ezagutzaren arabera, proiektuaren garapenerako zein erantzuteko laguntza eskaintzen dute.

  Badira ere NS-3 tutorial ugari, manualak eta beste hainbat euskarri, simulagailuaren kontrola errazagoa izateko. Sareen inguruko eta simulagailuan nola inplementatzeko artikulu ugari ere badira.

- **Balorazioa**. Gure betebeharrak betetzen dituen simulagailua da (software librea, garapen irekia, akademi komunitatearen laguntza...). Garapena aurrera doa eta geroz eta euskarri eta funtzionalitate gehiago daude eskura. Proiektu honetan garatu nahi den sarea sortzeko arazorik ez du eta nahi izanez gero, sare errealetan emulatzeko aukera ere bada, aldaketa gutxi batzuk eginda (Testbed edo banku proba, Planetlab, Orbit [orbit, 2014] sistematan erabiliz). Azken funtzio honen erabilera emaitza zehatzagoak eman ditzake eta agian modu simulatuan exekutatzen den aplikazio batek emaitza berdinak ematen ditu simulazio bakoitzean, zehaztasun pixka bat galduz.

4.2.5. OPNET

OPNET [OPNET, 2014], Alain Cohen-en sare kurtso bateko graduko proiektua da [OPNETWiki, 2014]. Denbora baten ondoren, software komertziala egitea erabaki zen eta *OPNET Technologies, Inc.* enpresa jaio zen. 2012ko Abenduan Riverbed [riverbed, 2014] enpresak *OPNET Technologies, Inc.* erosi zuen.

OPNET, softwarea, ezaugarri eta baliabide ugariz osatutako sareak simulatzeko gertaera diskretuko software tresna da. Protokoloak defini daitezke, nodo modeloa erabiltzen da sare osagaiak definitzeko, portaera partikularra egokitu daitezke, interfaze grafiko bidezko sareen sorrera egin daiteke eta simulazioaren paketeak kaptura daitezke gero hauek aztertzeko.

Simulazio tresna oso interesgarria da baina betebeharren artean software librean garatzea aipatu da. Unibertsitate ikerketarako programa bat badago eta baita edizio akademiko bat ere. Edizio akademikoan 6 hilabetez erabil daiteke behin web erregistratua eta 6 hilero lizentzia berritu daiteke.

Soluzio hau baztertuko da, software librea ez izateagatik eta honek dakarren moldaketa murriztapenengatik. Adibidez, NS-3 simulagailuan, software librea izateagatik, nahi haina ezaugarri moldatu daitezke. Bestalde, euskarri eta kolaborazio eskasa eskaintzen da, berriz beste software libreko simulagailuetan komunitate aktiboak badira.

4.3. Aukeratutako simulagailua

Sakontasunean aztertutako hiru sistemek proposatutako betebeharra betetzen dituzte, OMNet++, PlanetLab eta NS-3. Proiekturako NS-3 simulagailua aukeraketa da.

PlanetLab baztertzearen arrazoi nagusia erabiltzeko betebeharrengatik izan da, konsortzioaren barnean egon beharra dago. EHU-k ez du erakunde honetara sarrera zuzenik, beraz pisuko traba da hau. Aipatu beharra dago proiektuko zuzendariek PlanetLab-en lan egiteko aukera zegoela aipatu zutela (tartekari baten bidez).

NS-3 simulagailuak duen hedapenarengatik eta hain bizi dagoen komunitatearengatik, erabakia hartzerakoan simulagailu honetara ere eraman nau. Gaur arte eraikita dauden protokolo eta aplikazioekin (erabiltzeko eta eredu bezala edukitzeko) nahikoa da proiektu honen garapena egiteko. OMNet++ simulagailuak hedapen handia ere badu, eta INET framework-a erabiltzen duen komunitate bizia ere badago.

Beste puntu garrantzitsu bat erabakia hartzerakoan sistemaren kontrola da. Algoritmoak sare erreal batean emulatzen badira, hauek sarean kargatzean ezusteko arazoak aurkitzea gerta daiteke, adibidez, nodoa edo bideragailua matxuratuta egotea, sistemaren eguneraketa eta itxoiten egotea... Honen konponketa ez dago gure menpe, PlanetLab-en kasuan sistema honen mantentze arduradunen menpe egongo litzateke eta hauek konpondu arte ezingo litzateke probarik egin. Berriz, NS-3 edo OMNet++ bezalako simulagailuetan kontrola erabiltzailearen menpe dago, nodoak etab. birtualak eta erabiltzaileak sortutakoak direlako.

Hiru simulagailuetan C++ lengoaia erabiltzen da eta NS-3 eta PlanetLab simulagailuetan *Python*-ekin lan egin daiteke. OMNet++en sareen topologia definitzeko NED lengoaia propioa erabiltzen da. C++ lengoaian jorratu behar zenez eta NS-3 simulagailua C++-en garatua dagoenez, hau aukeratzera eraman nau. NS-3 simulagailuaren iturburu kode guztia eskura dago, PlanetLab eta OMNet++ simulagailuak ez bezala (INET framework-aren kodea eskura dago). Honek beste abantaila ematen du ikasketa prozesuan, tutorialetan erabiltzen diren adibideez aparte beste aplikazio ugari eskura daude, hortaz, simulagailuaren nondik norakoak errazago ulertzeko balio dezake.

Seguruena PlanetLab sistemak, sare erreal baten gainean inplementatua dagoenez, emaitza zehatzagoak emango lituzke, baina oro har NS-3 simulagailuan eta TestBed-ean simulatutako eszenario batean emaitza berdineko kasu ugari daude, nahiz eta batzuetan TestBed-etan sareak eduki dezakeen portaera desberdinengatik emaitza desberdinak eman [Baldo, 2014]. Ikusten den bezala, NS-3-k Testbed-ekin lan egiteko modulu bat badu eta jada PlanetLab-ekin lantzeko modulua garatu da, ondorioz mundu errealeko emaitzak lortu nahi badira, modulu hauen erabilerarekin PlanetLab edo edozein Testbed sistematan erabiltzeko aukera dago.

4.4. Simulazioaren arkitektura

Aurreko atalean azaldu den bezala, NS-3 simulagailuan lan egingo da. Simulagailuaren analisia bat burutuko da lan egiteko modua aztertuz. Ondoren inplementatu beharreko algoritmoak simulagailuan nola garatuko diren azalduko da.

4.4.1. Simulagailuaren metodologia

Simulazio bat egiteko pauso batzuk eman behar dira. Ondorengo lerroetan simulazioa aurrera eramateko eman beharreko pausoak azalduko dira.

Simulagailuaren oinarrizko egitura

NS-3 simulagailuak sare egitura bat sortzeko tresna ugari eskaintzen ditu, nodoak, protokoloak, kanalak. Oinarrizko elementuak ezagutzeko joan dokumentazioan eranskin bezala dagoen NS-3 erabilpen gidara [A.2.1].

Simulazio bat egiteko ondorengo pausoak eman behar dira hurrenez urren:

- *Nodoen sorrera*. Sareak zenbat nodoek osatuko duten adierazi behar da.
- *Interfazeen instalazioa*. Simulagailuak eskaintzen dituen interfazeen artean erabaki bat hartu behar da.
- *Protokoloak instalatu*. Zein sare protokolo erabili behar diren adierazi behar da.
- *Bideraketa taulen sorrera eta instalazioa*. Sarea osatzen duten nodoekin bideratze taula sortu.
- *Aplikazioen instalazioa*. Simulazioan exekutatu nahi den aplikazioa.
- *Sarearen portaeraren aztertzailea konfiguratu*. Irteera daturik nahi bada, irteera parametroak definitu.
- *Simulazioaren exekuzioa*. Simulagailua martxan jarri.

Algoritmoen simulazioaren egitura

Behin egitura azaldu dela, planteatu nahi den simulazioaren ezaugarriak aukeratuko dira aurreko egitura mantenduz.

- *Nodoen sorrera*. Simulatu nahi den sarearen arabera nahi haina nodo sortuko dira.
- *Interfazeen instalazioa*. Peer-to-peer edo puntuz-puntuko konexio interfazea instalatuko da. Jada ikusi den bezala, full-connected sare bat sortuko da.
- *Protokoloak instalatu*. UDP/IP protokoloa erabiliko da.
- *Bideraketa taulen sorrera eta instalazioa*. Sarea osatzen duten nodoekin bideratze taula sortuko da non nodoei IPv4 helbide motak egokituko zaizkie.
- *Aplikazioen instalazioa*. Simulazioan exekutatu nahi den aplikazioa, kasu honetan inplementatu beharreko *Consensus* eta *hutsegite detektatzaile* algoritmoak.
- *Sarearen portaeraren aztertzailea konfiguratu*. Sarearen nondik norakoak aztertzeko, trazak aztertzea ezarriko da
- *Simulazioaren exekuzioa*. Simulagailua martxan jarri. Nodo guztiak aldi berean hasiko dira. Hasiera eta amaiera denbora ezarri beharko zaio eta eszenarioan definitutako nodo guztiak aldi berean aplikazioak exekutatuko dituzte.

Behin simulazioa exekutatu dela, lortutako emaitzen lanketa eta aurkezpena burutuko da, traza eta denborekin ondorioak ateratzeko.

4.4.2. Simulazio egoerak

Simulagailuan omisioak eta crash-ak simulatzeko nodoen egoerak erabiliko dira. Egoerak, nodo batek omisio edo crash-en bat jasan behar duten adierazten dute. Jarrain egon daitezkeen aukerak aurkeztuko dira:

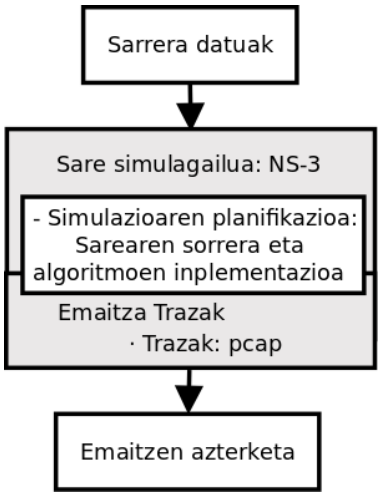
- **Normal (0).** Kasu honetan nodoa ez du omisiorik ezta crash-ik edukiko, komunikazioan bidaltzen diren mezuak beti garaiz eta dagokien helburura iritsiko dira. Jasotako mezuak ere bornatutako denbora mugan ailegatuko dira.
- **Send Omission (1).** Balio hau hartzen duen momentuan, uneko nodoak *send omission*-ak edukiko ditu dagokion nodoarekin, hau da, bakarrik komunikatutako mezuak garaiz jasotzen direla berma daiteke. Bidalitakoak berriz, omitituko dira edo esperotako denbora tartetik kanpo iritsi daitezke.

- **Receive Omission (2).** Balio hau hartzen duen momentuan, uneko nodoak *receive omission*-ak edukiko ditu dagokion nodoarekin. Jasotzaileak komunikatutako mezuak garaiz jasoko dituela berma daiteke. Jasotako mezuak berriz, omitituko dira edo esperotako denbora tartea gainditzen duen mezu baten jasotzea izan daiteke.

- **Send-Receive Omission (3).** Uneko nodoak helburura bidalitako edo jaso beharreko mezuak garaiz iritsiko direla ez da bermatzen. Mezuak ez jasotzea edo ez bidaltzea ere gerta daiteke, ondorioz gainontzeko nodoek matxuratuta balego bezala antzeman dezakete nodoa. Kasu honetan nodoa *bizirik* mantenduko da nahiz eta mezurik garaiz ez jaso edo bidali, baina ondorengo denbora tartean batean egora berri bat har dezake.

- **Crash(4).** Nodoa bertan behera geratzen da, ez du mezurik bidaltzen ezta jasotzen ere. *Send-Receive Omission*-ekin aldaratuta, egoera berri batera aldatzea ezinezkoa da. Nodoa matxuratuta dagoela esan daiteke.

4.4.3. Algoritmoak simulatzeko erabakitako soluzioa



4.2 Irudia: Simulaziorako erabakitako arkitektura

NS-3 simulagailuan aplikazioan (algoritmoen inplementazioa) burutzeko, ondorengo ezaugarriak dituen sistema modularra eraikiko da(ikusi 4.2 irudia):

Sarrera parametroak gertaera diskretuko sare simulagailuari bidaliko zaizkio sarea eraikitzeko eta simulazioa hasteko. Simulazioa hastean inplementatu beharreko algoritmoak exekutatuko ditu eta aldi berean sarearen trazak sortuko ditu. Simulazioa amaitzean, sortutako irteera parametroekin, datuen analisia eginen da eta ondorioak aterako dira.

Erabiltzaileak sortutako sarrera datuak simulagailuari sartu behar zaizkio, hauek nodo kopurua eta nodoen egoeren fitxategiak izango dira besteak beste. Simulagailua martxan jartzeko, aurreko atalean zehaztutako egiturarekin aplikazio bat sortuko du. Sortu beharreko beste parte

bat simulatu nahi diren algoritmoak dira. Hauek simulagailuak aplikazioa bezala identifikatzeko modulu berri batean sortu behar dira, bertan *Consensus* eta hutsegite detektatzailea egonen dira.

5 KAPITULUA

Inplementazioa

Dokumentazioko atal honetan, inplementazioan sakonduko da. A.2.1 atalean NS-3 simulagailuen klase nagusiak aurkezten dira eta 5.1 atalean proiektu honetarako aplikazioa sortzeko erabili diren klaseak agertzen dira, non *Consensus* eta hutsegite detektatzaile algoritmoen inplementazioak barne egonen dira. Bestalde 4.4.3 atalean azaldutako arkitekturaren inplementazio azalduko da. Azkenik, *Consensus* eta hutsegite detektatzaile inplementazioa egiterakoan hartu behar izandako erabaki batzuk azalduko da, GST [5.3.1] eta Reliable-Broadcast [5.3.2].

5.1. Domeinuaren klase diagrama

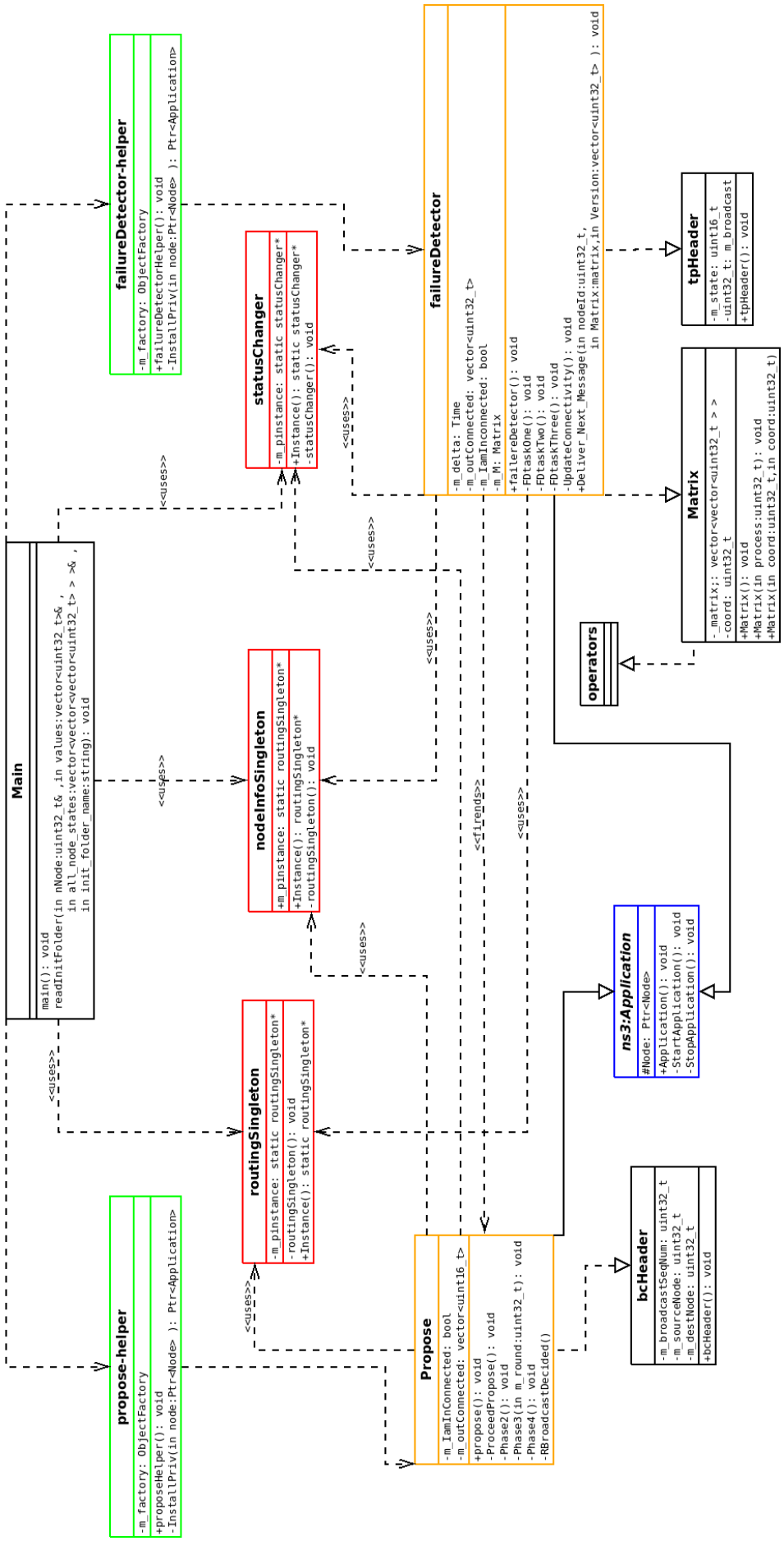
Simulatu beharreko algoritmoaren inplementazioa hobeto azaltzeko klase diagrama baten bidez egingo da. Bertan klasearen eginkizunak eta berezitasunak argituko dira. (5.1 irudia ikusi)

- **Main**

  Klase honetan simulazioa egikaritzeko beharrezkoak diren elementuak definitzen dira. Sare egitura, simulazioaren hasieraketak eta honen exekuzioaz arduratuko den klasea. Sarrera parametroen bidez, nahi den sare bat eraiki eta simulatu daiteke.

- **propose-helper eta failureDetector-helper**

  Bi klase hauek funtzio berdinak betetzen dituzte. Simulazioaren konfigurazioa sortzean, prozesu edo nodoei exekutatu nahi diren aplikazioak esleitu behar zaizkie. Aplikazioak zuzenean ezin direnez esleitu, bitartekari baten beharra dago eta hau *helper* klaseen funtzioa da. Aplikazioa abiarazteaz, gelditzeaz eta beharrezko parametroen bidalketez arduratuko dira.



5.1 Irudia: Klase Diagrama

- **ns3:Application**

  Simulagailuan aplikazio bat sortzean *ns3:Application* klase abstraktuaren herentzia egin behar da. Klase honek simulazioa hasi eta amaitzeko heredatu behar diren funtzioen definizioak izango ditu.

- **Singletons**

  Mota honetako hiru klase dauzkagu. Bakoitzak bere funtzioa espezifikoak ditu, baina hauen erabilera inplementatutako algoritmoen eta *main* programa orokorraren arteko datuen trukaketarako erabiltzen dira. Klase diagraman gorriz marraztutako klaseak dira.

  – *routingSingleton.* Sarearen informazioa gordeko da. *Main* programak simulaziorako datuak beteko ditu eta hauek, inplementatutako *propose* eta *failureDetector* algoritmoek erabiliko dituzte. Edukien artean, sare nodoen IP helbideak gordeko dira.

  – *nodeInfoSingleton.* Nodoaren informazioa gordeko da, adibidez, simulazio eredu honetan partekatuko dituzten proposamen balioak.

  – *statusCharger.* Nodo bakoitzaren egoera gordeko da. Nodo bakoitzak dagozkien egoerak berreskuratuko dituzte eta honela simulazioan zehar omisioren edo crashen bat eduki duten jakinen dute.

- **failureDetector**

  Klase hau NS-3 simulagailuaren aplikazio baten inplementazio bat da, hutsegite detektatzaile algoritmoaren inplementazioa hain zuzen ere [3.4 algoritmoa]. *Consensus* algoritmoarekin elkar lan egiteko, *InConnected* eta *OutConnected* balioak kalkulatu ondoren, *propose* algoritmoarekin komunean dituen aldagaietan informazioa partekatuko du.

- **propose**

  Beste aplikazioa bat da, hain zuzen ere *consensus* algoritmoaren inplementazioa da [3.2 algoritmoa]. Singleton-etatik eta failureDetector klasetik jasotako datuekin *consensus* batera ailega daitekeen aplikazioa da. Simulazioaren amaieran, behin-betiko emaitza pantailaratuko du.

- **bcHeader**

  Propose klasean, komunikazioak burutzeko datuak sare mailara egokitu behar dira. Sarean *broadcast* bat egin behar denean, bidali beharreko paketeari bururako bat erantsi behar zaio. Bururakoaren serializatzeko edo deserializatzako funtzioak gordetzen dituen klasea da.

- **tpHeader**

  failureDetector klasean erabiltzen den bururakoa da. Aurrekoan bezala, bururako paketearen serialiatzeaz eta deserializatzeaz arduratuko da.

- **Matrix**

  Klase honek matrizeen trataerarako funtzioak ditu. Nodoen egoerak antzemateko *failureDetector* klaseak erabiliko ditu. *Operator* klasean *Matrix* matrize funtzio gehiago inplementatu dira.

5.2. Simulazio arkitekturaren inplementazioa

4.4.3 atalean azaldu bezala, behin NS-3 simulagailua erabaki dela aplikazioaren inplementazioa burutzeko, 4.2 irudiko ezaugarriak dituen sistema modularra eraikiko da.

1. 5.1 ataleko *main* klasea, sare topologia sortu eta simulazioa martxan jartzen duen klasea da. Klase honi, *script* batek exekutatu nahi den proba direktoriotik fitxategiak kargatuko dizkio, klaseak fitxategi hauekin simulazio eszenario eraikitzeko. Simulazio bakoitzeko bi fitxategi mota sortuko dira. Lehenengoan, simulazio denboran zehar nodoek edukiko duten egoerez osatuta egongo da, **node-status-** hasierarekin. Fitxategi hauetan, denboran zehar nodo bakoitzak beste nodoekiko zein egoera edukiko duten adieraziko dute. Sareko nodo bakoitzeko, mota honetako fitxategi bat egongo da. Egoera aldaketa, guk simulazioa exekutatzean ezarritako denbora izango da. Bigarren fitxategian, nodo bakoitzak daukan estimazioa balioa egonen da, *estimations* izenarekin. Estimazioa, nodoak *consensus* bat lortzeko proposatzen duen balioa da. Proba bakoitzeko fitxategi hauek direktorio batean sartuko dira eta hau erabiltzaileak eraikitako *script*-ari parametro bezala pasako zaizkio *-f* aurrekariarekin.

   Bestalde, denborekin zerikusia duten beste hautazko parametro batzuk badira. Lehenengoa, egoeraren aldaketa denbora tartea izango da, *-ts* (time-status) aldagaia. Bigarrengo bat, aurreko atal batean (ikusi 3.2.6 atala) definitutako Φ balioarekin erlazionatuta dago. *-td* (time-delta) hasieran lehenetsitako itxoite denbora ezarriko du. Esan bezala, balio *nahiko* altua ematen bazaio, omisio detektatze ezagatik emaitza ez zehatza itzul dezake. Azkenik, hutsegite detektatzailearekin erlazionatuta dagoen *-thb* (time-failure detector) aldagaia. Aldagai honekin, hutsegite detektatzailearen *heartbeat* mezuen bidaltze frekuentzia zehaztuko da.

script-ak aurreko parametroak lehenetsiak ditu, hortaz, aplikazioa martxan jartzeko hauek zehaztea ez da beharrezkoa. Kasu berezi bat zehaztu nahi bada, hauek aurreko parametroekin molda daitezke.

2. Simulazioaren programa nagusiak simulatzeko parametroak jaso ondoren, nahi diren simulazio kopuruak egingo ditu. Hau simulagailuak eskaintzen dituen baliabideekin inplementatutako *Consensus* eta hutsegite detektore algoritmoen exekuzioak izango dira. Exekutatzen den heinean simulazioen trafiko informazioa trazetan gordetzen da ".pcap" luzeradun fitxategietan. Bestalde, geroko ebaluazioa egiteko irteera mezuak LOG fitxategi batean gordetzen dira.

3. Behin simulazioak egin direla, emaitza aztertuko dira eta ondorioak aterako dira.

5.2.1. Script-aren erabilera

Aurreko puntuan definitutako sarrera parametroez gain beste batzuk ere badira. Bestalde proba eta helburu desberdinetarako *script* desberdinak ere eraiki dira.

- *consensus.sh* - Simulazioaren informazioa irteera estandarrean inprimatzen du.
- *consensus-to-file.sh* - Simulazioaren informazioa exekutatzen den probaren fitxategian gordetzen du.

Bi *script* hauek helburu desberdinetarako pentsatuak daude. Simulazio estandar batean datu gutxi pantailaratzen badira, *consensus* baten lortzea eta hau lortzeko emandako denbora, *consensus.sh* erabiliko da. Hau irteera estandarretik irakurgarria da, baina ez da modu aproposena aplikazioaren nondik norako guztiak jakin nahi badira. Kasu honetarako *consensus-to-file.sh* script-a sortu da. Informazioa probaren izen berdineko irteera fitxategi batera bidaliko da, adibidez, *proba1.out*.

Simulaziotik jaso nahi den informazioa *-d* parametroaren bidez uneko *script*-ri adierazi diezaiokegu. Aukera desberdinak ditugu:

1. *INFO*. Simulazioaren informazio minimoa pantailaratuko da. *Consensus* lortu den ala ez, eta lortu bada zein denbora momentuan.

2. *DEBUG_FD*. Exekuzioan zehar hutsegite detektatzaile algoritmoaren nondik norakoa ezagutu nahi badira hau aukeratuko da. Matrizeen eta beste hainbat parametro interesgarrien momenturoko pantailaratzea.

3. *DEBUG_PROPOSE*. Exekuzioan zehar *propose* algoritmoaren nondik norakoa ezagutu nahi badira hau aukeratuko da. Aldagai interesgarrien pantailaratzea.

4. *DEBUG*. Exekuzioan zehar *propose* eta hutsegite detektatzaile algoritmoen nondik norakoen pantailaratzea.

5. *DEBUG_ALL*. Exekuzioan zehar dauden maila guztien pantailaratzea. *Propose* eta hutsegite detektatzaile algoritmoez gain, garraio geruza edo beste geruzen informazio guztia pantailaratuko da.

Bi *script*-ek INFO parametroa lehenetsita dute, beraz informazioa pantaila estandarrean irakur daiteke. Beste aukeraren bat aukeratuz gero, fitxategi batetik eroso irakurtzeko, aproposena *consensus-to-file.sh* script-a aukeratzea litzateke.

Exekuzio adibide bat:

```
# ./consensus.sh -d DEBUG_FD -f proba6 -ts 0.8 -thb 1 -td 0.7

# ./consensus-to-file.sh -d DEBUG_FD -f proba6 -ts 0.8 -thb 1 -td 0.7
```

Proba kasuen kapituluan azalduko den bezala (ikusi 6.2 atala), kasu desberdinetarako mota bakoitzerako beste bi script eraiki dira: *consensus-2.sh*, *consensus-3.sh*, *consensus-to-file-2.sh* eta *consensus-to-file-3.sh*. Funtzionamendua orain arte azaldutakoaren berdina da baina sare mota desberdinetarako.

5.3. Algoritmoa inplementatzerakoan hartutako erabakiak batzuk

Eskainitako dokumentazioan agertzen diren algoritmoez aparte, beste erabaki batzuen aukeraketa egin behar izan da. Algoritmoan, badira kontzeptu batzuk modu orokorrean deskribatzen direnak eta hauek inplementatzerakoan zehaztu beharrekoak. Aipagarrienak *GST* eta *R-Broadcast* kontzeptuak ditugu. Ondorengo lerroetan hauen deskribapena eta hartutako erabakia azalduko da.

5.3.1. GST edo Self-Stabilitation

Self-Stabilitation hutsegite-toleratzailetan agertzen den kontzeptu bat da. Sistemaren egonkortasuna lortzean datza. Sistema banatu bat egonkorra da baldin eta egoera arbitrario batetik hasten den sistema bat, pauso batzuen ondoren egoera legitimo batera iristen bada. Geroztik, egoera horretan mantenduko da. Egoera legitimoa da baldin eta legitimitate hau hasten denetik algoritmoaren zehaztapenak asetzen baditu.

Denbora konplexutasuna

Algoritmo egonkor baten denbora konplexutasuna neurtzeko, asinkronia dela eta, txandak edo zikloak erabiltzen dira.

- *Txanda*, exekuzio pauso batean exekutatzen den exekuzio traza motzena da.
- *Zikloa*, errepikatzen den iterazio oso batean exekutatzen den exekuzio traza motzena da.

Egonkortzearen denbora, sistema egonkortu arte emandako denbora izango da, pasatako txanda asinkrono kopurua.

Definizioa

Sistema bat egonkorra izango da baldin eta soilik baldin ondorengo baldintzak betetzen baditu.

1. Edozein hasiera egoera delarik, azkenean sistema egoera zuzen batera iritsiko dela bermatzen bada.
2. Sistema egoera egonkorrean dagoenetik, nahiz eta edozein akats suertatu egoera zuzen batean geratuko dela bermatzen bada.

Sistema bat ausazko egonkorra dela esaten da, baldin eta soilik baldin sistema egonkortzen bada eta egoera egonkorrera iristeko espero diren pausoak k konstanteaz mugatua badago.

Gure algoritmora egonkortzearen kontzeptua eramaten bada, komunikazio egoki eta garaizko bat izateko beharrezkoak diren pausoen neurketa izango da. Kontzeptua hau hutsegite detektatzailean agertzen da. Prozesu baten bizitasuna neurtzeko erabiltzen den mekanismoa da.

Gure algoritmoan $\Delta_p(q)$ balioarekin adierazten da. $\Delta_p(q)$, komunikazioa egokia eta garaizkoa dela ziurtatzen duen parametroa da. Hutsegite detektatzaile algoritmoan (3.4 algoritmoa) zehar hiru lerrotan aurki daiteke: 5. lerroa, 22. lerroa, 24. lerroa.

Simulatzerakoan, erabiltzen diren nodoen komunikabideak zuzenekoak edo zeharkakoak dira. Honen ondorioz gerta liteke nodo batek beste batzuekiko $\Delta_p(q)$ balioa desberdinak izatea. Komunikazio zuzen batek edo zeharkako komunikazioa batek ez dute $\Delta_p(q)$ berdina zertan eduki behar.

Txandaren denbora ez da ausaz jartzen. Lehen aipatu bezala edozein egoeratik abia daiteke eta egoera egonkor batean amaituko da. Hasieraketan txandaren suposaketa bat egingo da, normalean sarearen egitura ezagututa balio koherente minimo bat lehenetsiko da. Demagun bi nodoren artean komunikazio 10 milisegundokoa dela eta hau baino gehiagoko komunikazioak okerrak direla. Ez du zentzu handirik $\Delta_p(q)$-ren hasierako balioa 500 milisegundokoa izatea, ez genituzke suerta zitezkeen akatsak detektatuko. Honegatik aproposagoa izango litzateke hasiera denbora baxuagoaz bekatzea, handiagoaz baino. Hasieraketan balio baxuago bat ezartzen bada, $\Delta_p(q)$ balio batera konbergituko du.

Eguneratzea aldizkako kontrol batez eginen da. Komunikazio gehienak denbora tarte horretan egiten badira, muga ezarriko da eta hauek baino denbora handiagoak direnak, hutsegiteak bezala hartuko dira. Momentu honetan bornatua legoke eta $\Delta_p(q)$-k egonkortasuna bermatuko du.

Jarraian egonkortzearen kontzeptua agertzen diren lerroak azalduko dira, kontzeptu berean sakontzeko.

- **Hutsegite detektatzaile algoritmoaren 5. lerroa**

  Lerro honetan hasieraketa burutuko da. Lehen azaldu bezala, txanda denbora minimoa izango da, ahalik eta denbora baxuena eta koherentea.

- **Hutsegite detektatzaile algoritmoaren 22. lerroa**

  Komunikazioa ongi burutu den ala ez kontrolatu behar da, horregatik lerro honetan mezua garaiz iritsi den ala ez kontrolatzen da. Mezua garaiz iritsi ez bada eginbeharrekoak egin beharko dira eta bertan $\Delta_p(q)$ balioa eguneratzea izan daiteke.

- **Hutsegite detektatzaile algoritmoaren 24. lerroa**

  Lerro honetan $\Delta_p(q)$ aldagaiaren eguneraketa ematen da. Baldintzak ez badira bete, $\Delta_p(q)$ balioa behin eta berriz eguneratuko da egonkortasuna ematen duen borne bat aurkitu arte.

Eguneraketak

Egoera egonkor bat lortu arte hasierako balioa eguneratu behar da. Definizioan esan bezala, puntu batera konbergitu behar du, non borne puntu honek prozesatutako eragiketa hutsegitea den ala ez adieraziko du.

Oso garrantzitsua izango da hasieraketari emandako balioa. Komenigarria da balio baxu bat jartzea, baina oso baxua bada egonkortasun egoera hori lortzeak denbora gehiegi eramatea gerta liteke. Hau ere eguneraketa moduaren menpe egonen da. $\Delta_p(q)$ aldagaia modu askotan eguneratzea dago, beraz azkartasuna eta zehaztasuna izango dira eguneraketa moduaren helburuak.

$\Delta_p(q)$ parametroa eguneratzeko aukera pila daude. Jarraian bi aurkeztuko dira, ulertzeko errazak direnak.

- Eguneraketa konstantea

  Kasu honetan eguneraketa balio finko batez egiten da. Demagun $\Delta_p(q)$ eguneratzean 100 milisegundotan egitea erabaki dela.

$$\Delta_p(q) = \Delta_p(q) + 100ms$$

Esan bezala, eguneraketa hau sarearen arabera handia izan daiteke eta gero hutsegiteak ez detektatzea edo oso baxua izatea eta bornatu arte denbora asko iragatea. Azkenean helburua beti balio batera konbergitzea da.

- Portzentajezko eguneraketa

  Eguneraketa aldagai finko batez egin beharrean portzentajeekin jokatzen da. Demagun $\Delta_p(q)$ eguneratzean $\Delta_p(q)$-ren $\%20$ milisegundotan egitea erabaki dela.

  $$\Delta_p(q) = \Delta_p(q) + \%20$$

  Portzentajeen erabilerak denboran zehar balio ezberdinen batura ekartzen du.

Beste eguneratze mota desberdin ugari egin daitezke, azkenean borne bat lortu eta ezartzea da. Bi hauek planteatu eta gero hauetako bat aukeratu beharko da.

Hartutako erabakia

Gure algoritmoaren kasuan helburua mezu hutsegiteak detektatzea da. Detektatzeko, komunikazio denborari borne bat ezartzen zaio eta hau gainditzen bada, mezua hutsegite bezala hartuko da. Helburua lortzeko, eguneraketa beti inkrementala izango da. Behin gehiengo mezua batzuk ezarritako azken eguneraketa muga gainditzen ez badute, lortutako balio hori finkatuko da borne bezala. Borne hau gaindituz gero, komunikazioa hutsegitea izango da.

Aurkeztutako bi eguneraketa moduak argituta daudelarik, hauetatik bat aukeratu beharko da algoritmoaren inplementaziorako. Erabakitako kasu honetan *portzentajezko eguneraketa* hartu da soluzio bezala. Erabaki honen zergatia adibide batez ikusiko da.

Demagun $\Delta_p(q)$-ren hasiera balioa $100ms$-koa dela eta mezu bat komunikatzeko $150ms$ behar dira, ondorioz $150ms$ baino gehiagoko komunikazioak hutsegiteak izango dira. Goazen ba argitutako bi eguneraketa moduak aztertzera:

- Eguneraketa konstantea

  Demagun eguneraketa zehatzagoa izatearren, pixkanaka eguneratzea erabakitzen dela, adibidez $1ms$-ko eguneraketa.

  1. $100ms + 1ms = 101ms$
  2. $101ms + 1ms = 102ms$

     ...
  3. $149ms + 1ms = 150ms$

  Bornea ezartzeko 50 eguneraketa behar dira. Hau sarearen edo komunikazio kopuruaren arabera arazoa izan daiteke. Borne egokia lortu arte 50 mezu baztertu dira hutsegiteak izan direla suposatuz.

Demagun orain eguneraketa $15ms$-koa dela.

1. $100ms + 15ms = 115ms$
2. $115ms + 15ms = 130ms$
3. $130ms + 15ms = 145ms$
4. $145ms + 15ms = 160ms$

Kasu honetan bornea azkarrago ezarri da, 4 eguneraketetan, baina bornearen zehaztasun eza ekarri du. Bornea mezu frekuentzia baino handiago da eta suerta daiteke $157ms$-an iristen diren mezuak akastunak bezala hartu behar izatea. Kasu honetan errorea ikusezina litzateke.

Azkenik eguneraketari gaikuntza handiago bat egingo zaio, adibidez, $100ms$-koa

1. $100ms + 100ms = 200s$

Lehenengo eguneraketan bornatu da, baina mezuak normalean $150ms$ iristen badira eta $190ms$-an mezu bat badator, kasu honetan egokitzat hartuko luke nahiz eta errealitatean hutsegitea izan. Azkar lortu da bornatzea baina zehaztasuna galdu da.

- Portzentajezko eguneraketa

  Portzentajeekin lan egingo da aukera honetan. Aurrekoan ez bezala, konstante bat batu beharrean nahi dugun ehuneko portzentaje bat batuko zaio aurreko balioari. Momentuko $\Delta_p(q)$ balioaren menpe egonen da, honen gain egindako portzentajea delako.

  Demagun kasu honetan momentuko $\Delta_p(q)$ balioaren %10-a batzen dela, portzentaje konstante bat.

  1. $100ms + \%10 = 110ms$
  2. $110ms + \%10 = 121ms$
  3. $121ms + \%10 = 133,1ms$
  4. $133,1ms + \%10 = 146,41ms$
  5. $146,41ms + \%10 = 161,051ms$

  Bornea azkar lortzen da, baina azkar lortzeak zehaztasunaren arazoa dakar, $150ms$-ak gainditu dira.

  Hurrengo kasuan portzentaje konstante bat batu beharrean *portzentaje aldakorra* batuko da. Demagun lehenengo eguneraketan %10-koa batzen dela eta hurrengo eguneraketan %9-a ...

  1. $100ms + \%10 = 110ms$

2. $110ms + \%9 = 119,9ms$
3. $119,9ms + \%8 = 129,492ms$
4. $129,492ms + \%7 = 138,55644ms$
5. $138,55644ms + \%6 = 146,8698264ms$
6. $146,8698264ms + \%5 = 154,21331772ms$

Denboraren aldetik aurreko txandak berberetan bornatu da,

Denbora aldetik 6 txandetan bornatu da baina kasu honetan batuera txikitzen joatek zehaztasun maila handitzea ekarri du, nahiz eta $150ms$ balioa gainditu.

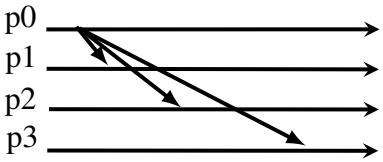
Borneak azkar lortzeak zehaztasun falta ekar dezake eta ondorioz mezuen susmagarriak diren ala ez detektatzean zailtasunak edukitzea. Bestalde, zehaztasuna lortzeko inkrementua unitate txikitan egiten bada, borneak ezartzeko denbora gehiegi pasatzea posible da eta ondorioz, egokiak ziratekeen mezuak, hutsegiteak bezala hartzea suerta liteke. Azken finean bi kontzeptu hauen artean oreka lortu behar da eta hau sare egoeraren menpe egonen da.

Algoritmoa inplementatzerakoan *portzentaje konstante* baten batura aukeratu da. Sarearen egitura ezagutzen ez bada, denbora eguneraketa konstante batez egiten bada arazoak ekar ditzake bornea lortzerakoan. Hau dela eta, hasieran lehenetsitako denbora balioarekin jokatzeak, borneak sare egituraren araberakoak izango direla ziurtatuko da.

```
...
m_deltaVector[nodeId] = Seconds(
        m_deltaVector[nodeId].GetSeconds()
                + (m_deltaVector[nodeId].GetSeconds() * float(0.4)));
...
```

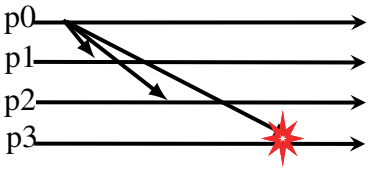
5.3.2. Reliable Broadcast (R-Broadcast)

Prozesu parte-hartzaileek *consensus* batera iristean, uneko koordinatzaileak erabakitakoa behin-betiko emaitza jakinarazi behar du. Honetarako *broadcast* teknika erabiltzen da. *Broadcast*-ak honakoa bermatuko du: prozesu batek *m* mezua bidaltzen badu, beste gainontzeko prozesuek azkenik *m* mezua jasoko dute (ikusi 5.2 irudia).



5.2 Irudia: Broadcast adibidea

5.2 irudian ikuste denez, hau sistema ideal bateko *broadcast* bat da. Baina, zer gertatuko da prozesuren batek akasten bat badu? (5.3 irudia ikusi) nola bermatuko da *m* mezua prozesu guztiei iristen zaiela? Bidaltzaileak nola jakinen du beste prozesuei bidalitako *m* mezua iritsi zaiela? Nola moldatu daiteke?



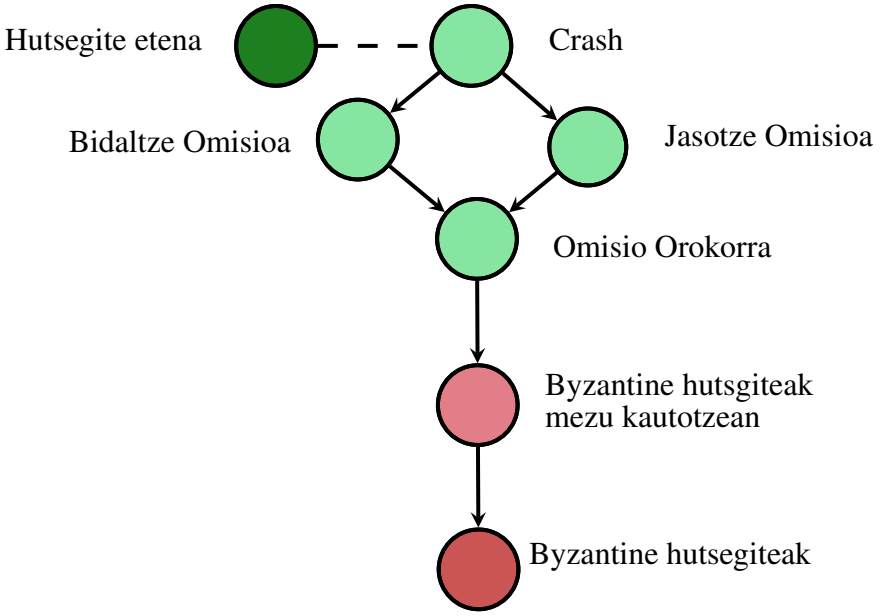
5.3 Irudia: Broadcast hutsegite adibidea

Hutsegiteak, sistema banatuetan askotan agertuko den arazoa da. Broadcast-ei irtenbidea emateko *reliable broadcast*-ak erabiltzen dira. Sakontasunean analizatzeko, lehenik kontzeptu batzuk gogoratzea komeni da.

5.4 irudian ikusten denez, hutsegite mota aunitz suerta daitezke sare bateko komunikazioan, 3.2.6 atalean azaldu dira hauek. Eszenarioa sistema asinkrono baten gainean egin denez, hutsegite hauek gainditzeko hutsegite-detektatzaile bat erabiltzen da. Detektatzaileak eskaintzen duen informazioarekin prozesuen egoera identifikatuko da eta sarearen egitura lortuko da. *Reliable-broadcast* bat egiterakoan, prozesuek bakoitzaren hutsegite-detektatzailetik jasotako informazioaz baliatuko dira *reliable-broadcast*-a exekutatzeko.

Reliable-broadcast-ek omisio modeloan funtzionatzen duela ziurtatzeko, omisio modeloko propietateak bete beharko ditu. Ondorengo lerroetan 3.2.4 atalean azaldutakoa propietateak betetzen diren ikusiko dugu.

- *Validity.* Mezu bidaltzailea zuzena bada eta *m* mezua sarean zehar sakabanatzen badu (broadcast egin), azkenik prozesu zuzen guztiek *m* mezuak jasoko dute.
- *Agreement.* Prozesu zuzen batek *m* mezua jasotzen badu, beste prozesu zuzen guztiek *m* mezua jasoko dute.



5.4 Irudia: Hutsegite motak

- *Integrity.* Prozesu zuzen guztiek gutxienez mezu bat jasoko dute. Bidaltze hutsegitea jasan ez duen prozesu batetik m mezua jaso badu, prozesuren batek m mezuaren broadcast-a bidali duelako da.
- *Termination.* Prozesu zuzen guztiek beti mezuren bat jasoko dute.

Antzematen denez, *reliable-broadcast*-ak *uniform consensus* batean bermatu behar diren propietate berdinak betetzen ditu.

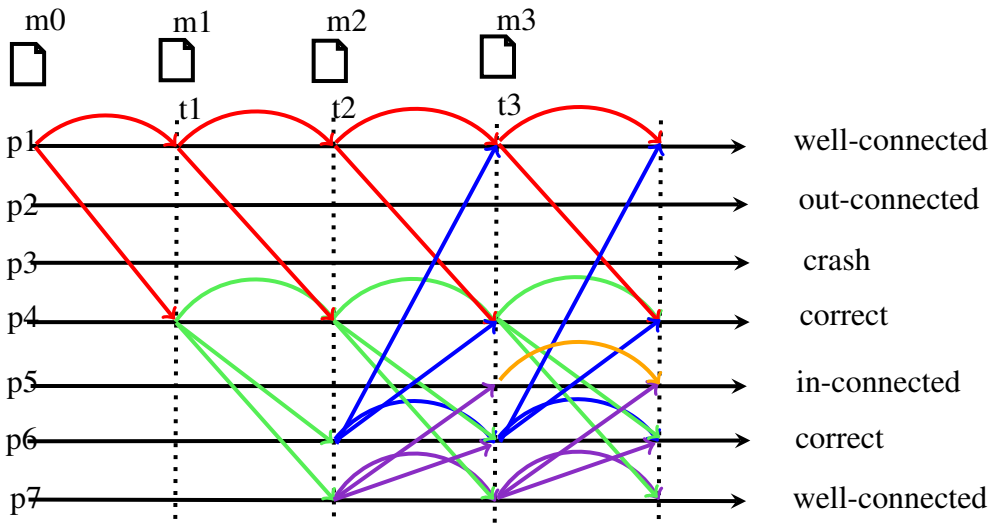
Prozesu guztiak *consensus* batera iristen direnean, koordinatzaileak bidalitako m behin-betiko emaitza mezua jasotzean, hurrengo txandan beste gainontzeko prozesu guztiei jasotako m birbidali beharko die, beraz beste *reliable-broadcast* berri bat egingo da.

Reliable-Broadcast plana

Reliable-broadcast bat egiterakoan, prozesu zuzen guztiek m mezua jaso arte beste n reliable-broadcast egingo dira.

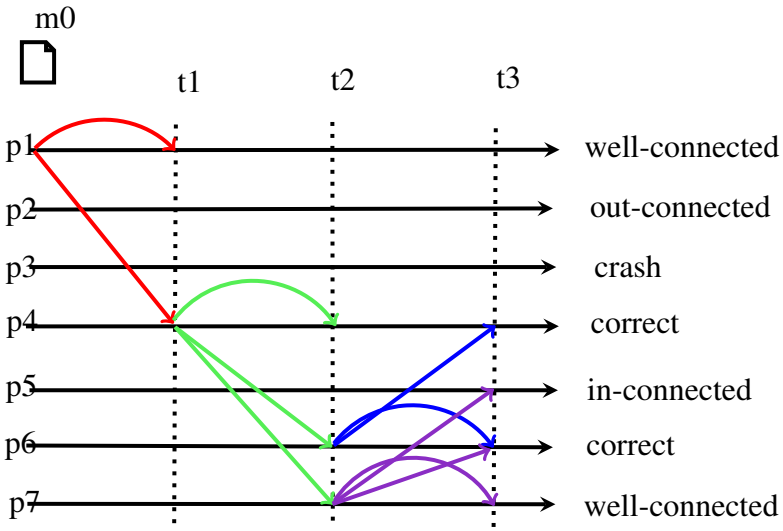
3.6 irudiaren grafo erabiliko da ondorengo azalpena ulertzeko. 5.5 irudian, $p1$ prozesuak uneoro bidaltzen dituen mezuen trantsizioa irudikatzen da.

Reliable-broadcast-a, zuzeneko konexioa duen prozesuekin egiten du. Kasu honetan $p4$ prozesuarekin du zuzeneko konexioa, hortaz $p4$-ren menpe egonen da gainontzeko prozesuei dagokion mezua bidaltzea. Hau ere, aurreko kasuan bezala, konexio zuzeneko prozesuez baliatuko da broadcast mezua bidaltzekoa. $t3$ txanda denboran, $p1$, $p4$ eta $p6$ prozesuek bi txanda lehenago $p1$-ek bidalitako $m0$ mezuaren broadcast-a jasoko dute. Kasu honetan mezu hau ez da berriz bidaliko. Irudikatutako denbora trantsizioan, $t3$.garren txandan jada denek



5.5 Irudia: R-broadcast

jaso dute $m0$ mezua. *well-connected*, *correct* eta *InConnected* diren prozesuek bakarrik jaso dute *reliable-broadcast* seinalea.



5.6 Irudia: R-Broadcast erabakia

5.6 irudian *consensus*-ean hartutako erabakiaren emaitza mezuaren broadcast-a irudikatzen da, 3.6 grafoaren egoera irudia kontutan hartuz. Aurreko paragrafoan azaldu bezala, $p1$ prozesuak bidalitako $m0$ mezua $t3$ txandan iritsi zaie jaso zezaketen prozesu guztiei. 5.5 irudiarekin alderatuz, behin erabakia hartu dela ez dira *consensus*-erako beste mezu gehiago bidaltzen, ez *reliable-broadcast* ezta *heartbeat* mezuak ere.

6 KAPITULUA

Proben plana

Software ingeniaritzan aplikazio batek bete behar dituen betebeharrak analizatzeko proba kasuak erabiltzen dira. Proba kasuak mugatuak diren inguruneetan denak probatu daitezke eta horrekin demostratu daiteke aplikazioa zuzena dela. Sistema banatuen kasuan, berriz, sarrera-parametroen balio posibleen konbinazioa eta exekuzioan eman daitezkeen egoerak infinituak dira eta, beraz, ezinezkoa da denak simulatzea. Hortaz, simulazioaren bitartez azpimultzo bat baino ezin izango da egiaztatu. Aplikazio ondo ibiltzea horiekin ez du inplikatzen simulatu gabe gelditzen diren guztiekin ondo joango denik.

Simulazio sistemaren errendimendua neurtzeko eta esperotako datuekin alderatzeko balioko du. Errendimenduaren neurketa, sarean sortze den trafiko sorrera, latentzia eta sarearen egonkortasuna izango da.

6.1. Kasu proben eraiketa

Proba kasuak aurrera eramateko bi fitxategiez baliatuko gara, *estimazioak* eta *nodoen egoerak* gordetzen dituzten fitxategiak hain zuzen ere (ikusi 5.2 atala). Gogora dezagun, nodoak *consensus* bat lortzeko proposatzen duen balioa estimazioa dela eta nodo batek omisio edo crash-en bat jasan behar duen egoerak adierazten duela. Fitxategi hauetan gordeko den informazioa nahi diren kasu probak eraikitzeko erabiliko dira. Jarraian, bi fitxategien nondik norakoak deskribatzen dira:

- **Estimazio fitxategia**

  Estimazio fitxategiaren barnean nodo bakoitzak edukiko duen estimazio balioa gordeko da. Fitxategiko lerro bakoitzeko nodoaren *ID* edo indizea agertuko da eta jarraian bakoitzak edukiko duen estimazio balioa. Hurrengo taulan (6.1 taula) adibide bat ikus daiteke.

- **Egoera fitxategia**

  Fitxategi honetan, nodo bakoitzak denboran zehar gainontzeko nodoekin edukiko dituen egoerak gordeko dira. Nodo bakoitzeko fitxategi bat egongo da eta bertan gainontzeko

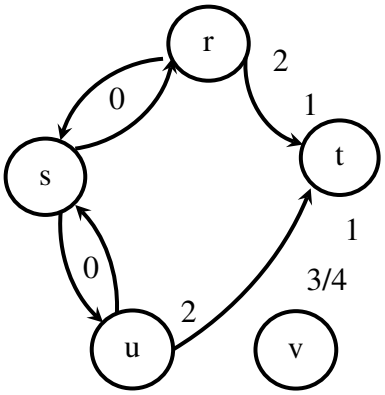
Nodo ID	Estimazioa
0	4
1	14
2	9
3	8
4	11

6.1 Taula: Estimazioa fitxategiaren edukia.

nodoekiko denboran zehar edukiko dituen egoerak. *Estimazioen* eta *egoeren* fitxategietako nodo kopuruak berdinak izan behar dute. Simulazio denboran, nodo batek beste nodo batekiko egoera konstantea manten dezake, baina denboran zehar egoerak aldakorrak izatea ere posiblea izango da. Fitxategian, denboran zehar edukiko dituzten egoerak *hutsune* batez bereiziak egonen dira.

Simulazioa egiterakoan, egoeren aldaketa denbora tartea *time-state* aldagaiaren bidez ezarriko da, gure *script*-aren exekuzioan *ts* parametroarekin ezarriko dena. Esan bezala, aldagai honek egoeren aldaketa denbora ezartzen du, hau da, nodoak egoera berri bat hartzeko pasa behar den denbora tartea. *ts* parametroari 1 balioa ematen bazaio, nodoek segundo bakoitzean egoera mantendu edo berria bat hartuko dute.

Goazen adibide batez azaldutako kontzeptuak finkatzera. Hurrengo grafoan (6.1 irudia), denbora konkretu batean ematen den egoerak aztertuko ditugu.



6.1 Irudia: Kasu proba baten adibidea.

Grafoan ikusten denez, $u \leftrightarrow s$ eta $s \leftrightarrow r$ nodoen artean ez dira omisiorik egonen, horregatik arkuetan zein fitxategiko dagokion tokian 0-ko balio jarriko da.

$u \leftrightarrow t$ eta $r \leftrightarrow t$ arkuetan berriz, t nodoari mezuak garaiz iritsiko zaizkiola bakarrik bermatuko da. t nodoak *Send-omission*-ak jasango ditu eta ondorioz, r eta u nodoekin dituen arkuak 1 egoera balioa hartuko dute. r eta u nodoen kasuan, t nodoarekiko *receive-omission* egoera dute, horregatik 2-ko balioa hartuko dute t nodoari dagokien arkuan.

v nodoaren kasuan bi balio ezarri dira adibide honetan. Berez balio bakarra edukiko luke, baina kontzeptu guztiak azaltzearren bi ezarri zaizkio. Demagun *Send-Receive*

Omission (3) kasuan gaudela. Kasu honetan mezuak garaiz jaso eta bidaltzea ez da bermatzen. *Send-Receive Omission*-ak berezitasuna badu, geroago beste egoera batera pasa daitekeela eta ondorioz hurrengo momentu batean mezuak garaiz jaso edo bidali egitea gerta liteke, hau da, hurrengo denbora tarte batean u nodoaren egoera normala (0) izan daiteke. *Crash* (4) kasuan berriz, mezuak ez ditu jasoko ezta bidaliko ere egingo. Aurreko egoeran ez bezala, momentu honetatik aurrera nodoak ezingo du egoera berri bat jaso, betiko *crash* egoeran mantenduko da nodoa matxuratu delakoa.

$s \leftrightarrow t$ nodoen artean ez dago erlazio zuzenik, beraz egoeren fitxategian 3 balioa ezarriko zaio nodo bakoitzari, geroko momentu batean egoera berri batez hornitu daitekeelako eta nodoa matxuratu ez dagoelako. Nodoa matxuratzen bada, nodo guztiekiko egoera *crash* (4) izango da.

Nodo guztiek bere buruarekiko edukiko duten egoera normala edo 0-koa izango da.

Ondorengo tauletan (6.2, 6.3, 6.4, 6.5, 6.6), jada azaldu dugun adibidearen taula egoera ikus daiteke. Zutabe bakoitzean, nodoak denbora une konkretu batean dituen egoerak irakur daitezke.

Nodo ID					
r	0	0	0	0	0
s	0	0	0	0	0
t	2	2	2	2	2
u	3	3	3	3	3
v	3	3	3	3	3

6.2 Taula: *r* nodoaren egoera fitxategiaren edukia.

Nodo ID					
r	0	0	0	0	0
s	0	0	0	0	0
t	3	3	3	3	3
u	0	0	0	0	0
v	3	3	3	3	3

6.3 Taula: *s* nodoaren egoera fitxategiaren edukia.

Nodo ID					
r	1	1	1	1	1
s	3	3	3	3	3
t	0	0	0	0	0
u	1	1	1	1	1
v	3	3	3	3	3

6.4 Taula: *t* nodoaren egoera fitxategiaren edukia.

Simulazioaren exekuzioa ahalik eta automatizatuena izateko, proba bakoitzeko fitxategi guzti hauek karpeta berean gordeko dira. Estimazio fitxategiari *estimationst.txt* izena emango zaio. Karpetan berean n nodoen egoera fitxategiak gordeko dira. *node-status-n.txt* izena emango zaie, non n nodoaren ID-a izango da.

Nodo ID					
r	3	3	3	3	3
s	0	0	0	0	0
t	2	2	2	2	2
u	0	0	0	0	0
v	3	3	3	3	3

6.5 Taula: *u* nodoaren egoera fitxategiaren edukia.

Nodo ID					
r	3	3	3	3	3
s	3	3	3	3	3
t	3	3	3	3	3
u	3	3	3	3	3
v	0	0	0	0	0

6.6 Taula: *v* nodoaren egoera fitxategiaren edukia.

6.2. Aztertutako proba kasuak eta balorazioak

Erabakitako azterketa kasuak azalduko dira orain, esperotako eta jasotako emaitzak alderatuz. Hauek eraikitako aplikazioan egikarituko dira eta errazteko sortutako *script*-ez baliatuko gara (*consensus.sh*). Lehenengo simulazioak soilik algoritmoek sortutako mezu-trukeen trafikoa edukiko dute eta ondorengoetan sarea egituran aldaketa batzuk sartuko dira.

Exekutatuko den lehenengo proba sortan [Proba 1,Proba 2,Proba 3,Proba 4], hasieratik egora berdinak edukiko dituzte nodoek. Bigarren proba sortan [Proba 5,Proba 6,Proba 7] denbora zehar alda daitezke nodoen egoerak. Azkenik [Proba 8], jada 3.2.3 atalean aztertutako grafoaren portaera ikusiko da.

Nota 1: Nodoak *consensus* batera iritsi badira, erabakia hartu den momentua (koodinatzailearen kasua) edo erabakitako balioa jasotzen den momentuan (erabakian parte hartutakoak) pantailaratuko da erabakia hartu duen koordinatzailea, erabakitako balioa eta zein txandan erabakirekin batera.

Nota 2: Simulazioko irteera denborak 2 segundo baino handiagokoak izango dira. Simulagailuan behin sarea eraiki dela, aplikazioa noiz hasi eta geratuko den espezifikatu behar da. Horregatik, aplikazioan ezarri den bezala, simulazioa guztiak 2-garren segundotik aurrera izango dira. Simulazioak amaitzeko denbora 100-garren segundoan ezarri da. *Consensus* batera ailegatu bada eta nodoak *consensus* erabakian parte hartu badu, 100-garren segundora "NODE X GET CONSENSUS!!!" mezua pantailaratuko da. *Consensus* bat lortu bada eta simulazioa 100-garren segundora iristen bada, erabakia hartzen parte hartu ez duten nodoek, aplikazioa exekutatzeari utziko diote eta "NODE X DOESN'T GET CONSENSUS!!!" pantailaratuko da. Bestalde gogoratu [3.2.3 atala], nahiz eta hartutako behin-betiko erabakia ez jaso, lehenetsitako erabaki balioa edukiko duela.

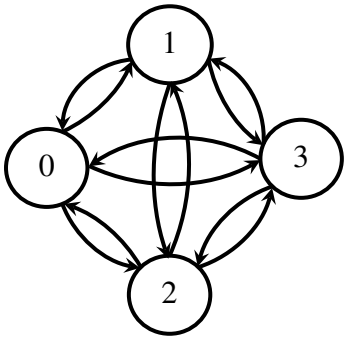
Nota 3: Simulazioak exekutatzean lehenetsitako aldagai batzuk daude eta ondorengoak izango

dira. hutsegite detektatzailearen *heartbeat* seinalea 0,1 segundokoa da, hasierako $\Delta(t)$ 0,3 segundokoa eta egoera aldaketaren denbora tartea 1 segundokoa. Beste balio batzuekin exekutatu nahiko balitz 5.2 atalean azaldutako parametro balioak erabiliko genituzke.

Kasu proba bat exekutatzeko adibidea:

```
# ./consensus.sh -d INFO -f proba1

# ./consensus-to-file.sh d INFO -f proba1
```

6.2.1. Proba 1



6.2 Irudia: Proba 1.

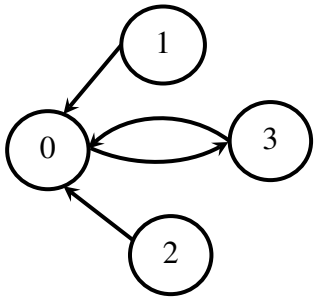
Nodo guztiak egoera normalean daude, mezuak garaiz jaso zein bidaliko dituzte ezarritako helburuetara. Ondorioz, nodo guztiak parte hartuz *consensus* lortuko dute eta aldi berean denei behin-betiko emaitza iritsiko zaie.

Simulazioaren exekuzioa:

```
================================
Executing simulation proba1.....
================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.341s)
Node 1 : All nodes arrived to a consensus and the estimation decided is 8 in round 1
Consensus time is 2.01505
Node 0 : All nodes arrived to a consensus and the estimation decided is by coordinator 1 with
     value 8 in round 1
Consensus time is 2.02007
Node 2 : All nodes arrived to a consensus and the estimation decided is by coordinator 1 with
     value 8 in round 1
Consensus time is 2.02007
Node 3 : All nodes arrived to a consensus and the estimation decided is by coordinator 1 with
     value 8 in round 1
Consensus time is 2.02007

NODE 0 GET CONSENSUS!!!
NODE 1 GET CONSENSUS!!!
NODE 2 GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
```

Simulazio emaitza: Nodo guztiek *consensus* erabakitzen parte hartu dute eta ikusten denez, nodo denei estimatutako balio berria heldu zaie.

6.2.2. Proba 2



6.3 Irudia: Proba 2.

Aurreko proba kasuarekiko aldaketa batzuk sartu dira. Egoera momentu honetan 0 eta 3 nodoek gainontzeko nodoak *OutConnected*-tzat kontsideratuko dituzte hauen *heartbeat* seinaleak jasotzen dituztelako. 3.garren nodoaren kasuan hasieran 1 eta 2 nodoak ez *OutConnected*-tzat hartu ditzake, baina denbora pasa ahala, zeharko komunikazioa eta denbora borneen eguneraketa dela eta, bi nodo hauek *OutConnected* multzoan edukiko ditu. Beraz, esan daiteke 0 eta 3 nodoek bere burua *InConnected*-tzat hartzen dutela. 1 eta 2 nodoetatik ezin da suposatu. Bi nodo hauek gainontzekotik *receive-omission* jasaten dute. Honez gero, ezin dute sarearen egoera eguneratu bere *InConnected*-tzat hartzen ez dutelako.

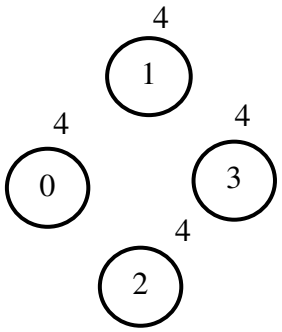
Azkeneko bi nodo hauek *heartbeat*-ak ezarritako denboran bere estimazioak bidaliko dituzte, gainontzeko nodoak bezala. Arazoa uneko koordinatzailea proposamena bidaltzerakoan ematen da. Bi nodo hauek ez dute proposamena jasoko eta ondorioz ez diote uneko koordinatzaileari erantzuna emanen eta ez dute *consensus* erabakian parte hartuko. *Consensus* batera ailegatzeko uneko koordinatzaileak gutxienez proposatutako estimazio berriaren $\lceil \frac{(n+1)}{2} \rceil$ onartze jaso behar ditu, kasu honetan 3 erantzun jaso beharko lituzke. Antzematen denez, 4 nodoetatik 2 bakarrik *InConnected*-tzat hartzen dute bere burua eta hauek izango dira erabakian parte hartuko dutenak, beraz ez da baldintza betetzen. Emaitzan, *consensus* ez dela lortu azalduko da.

Simulazioaren exekuzioa:

```
================================
Executing simulation proba2.....
================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.284s)

NODE 0 DOESN'T GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 DOESN'T GET CONSENSUS!!!
```

Simulazio emaitza: Esan bezala *consensus* ez da lortu. *InConnected* multzoko nodoak $\lceil \frac{(n+1)}{2} \rceil$ baino gutxiago direnez, ezingo da *consensus* batera iritsi.

6.2.3. Proba 3



6.4 Irudia: Proba 3.

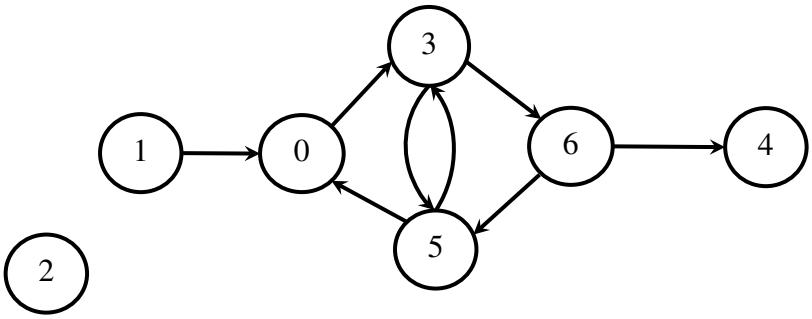
Hau kasu berezi bat da. Denak *crash* egin dute, ondorioz nodoak ez dute komunikaziorik ezta egoera aldaketarik. Nodo guztiak matxuratuta baleude bezala antzemango da. Hortaz, ez da *consensus* batera ailegatuko.

Simulazioaren exekuzioa:

```
================================
Executing simulation proba3.....
================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.291s)

NODE 0 DOESN'T GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 DOESN'T GET CONSENSUS!!!
```

Simulazio emaitza: *consensus* ez da lortu nodoak matxuratuta daudelako.

6.2.4. Proba 4



6.5 Irudia: Proba 4.

Aurreko azpiatalean erabilitako adibide bera dugu (38 atala). Azpiatal horretan azaldu bezala, *consensus* batera ailegatzea lortuko da. Adibide honen analisi sakonago bat edukitzeko, nahikoa da atalean deskribatutako irakurtzea.

Bertan azaltzen den bezala, nahiz eta nodoek zuzeneko mezuak ez bidali edo jaso, zeharkako komunikazioaz baliatuko dira. Komunikazio honetaz eta nodoen arteko estimatutako denbora borneak eguneratuz, komunikazioak modu egokian burutuko dira eta nodoek sarearen egoera jasoko dute.

Simulazioaren exekuzioa:

```
================================
Executing simulation proba4.....
================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.286s)
WARNING: Ignoring blank row.
Node 3 : All nodes arrived to a consensus and the estimation decided is 9 in round 3
Consensus time is 3.02022
Node 5 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.0253
Node 6 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.0253
Node 0 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.03025
Node 4 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.0303

NODE 0 GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
NODE 4 GET CONSENSUS!!!
NODE 5 GET CONSENSUS!!!
NODE 6 GET CONSENSUS!!!
```

Simulazio emaitza: Nodoak *consensus* batera iristen dira, baina ez dute nodo guztiek behin betiko emaitza jasotzen. Kasu honetan $\lceil \frac{(n+1)}{2} \rceil$ *InConnected* nodo baina gehiago daudenez *consensus* batera iritsiko da. 0, 3, 6 eta 5 nodoek hartuko dute erabakia, hauek bere burua *InConnected*-tzat hartzen dutelako eta uneko koordinatzaileak proposatutako balio berriari erantzun bat emateko kapazak izango direlako.

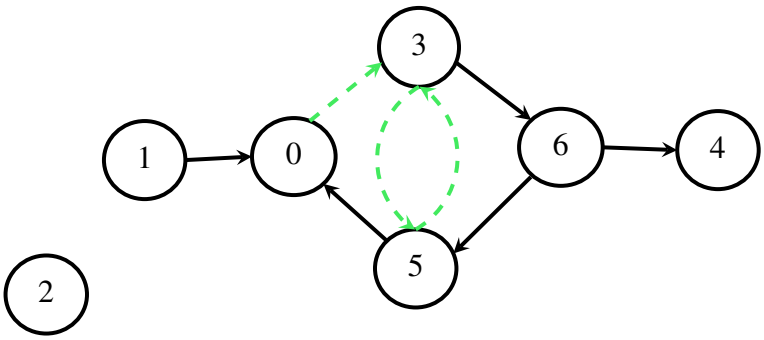
Erabakitako behin-betiko emaitza sakabanatzean, adostasunera iritsi diren nodoez aparte, 4 nodoak bere burua *InConnected*-tzat hartzen duenez, nahiz eta erabakia hartzen ez parte hartu, behin-betiko erabakia ere jasoko du *InConnected* baldintzarengatik.

1 eta 2 *InConnected* ez direnez, beren lehenetsitako estimazioarekin geratuko dira, erabakitakoa jakin gabe. 1 nodoa, 2 nodoa ez bezala, proposatutako balioa aukeratutakoa izan daiteke, *hearbeat* bakoitzeko uneko koordinatzaileari eskaintzen diolako. Esan dezakegu, erabakian parte hartu dezakeela, nahiz eta behin-betiko balioaren jabetza ez eduki. *OutConnected* multzoan dagoenez eta hau *well-connected* multzo batekin komunikatuta dagoenez, *well-connected* multzokoek *OutConnected*-tzat hartuko dute.

Simulazioan erabakitako emaitza nodo bakoitzari noiz iritsiko zaion ere antzeman daiteke.

3.garren nodoa koordinatzailea da *consensus* batera ailegatzen den momentuan. Koordinatzaileak *OutConnected* multzoko nodo guztiei bidaltzen die behin-betiko emaitza, baina denbora momentu desberdinetan iritsiko zaie nodoei. Erabakitako behin-betiko emaitza sakabanatzean, ikus daiteke nola 5 eta 6 nodoek lehenengo txanda batean emaitza jasotzen duten, koordinatzailearekin zuzeneko komunikabidearengatik. Berriz, gainontzeko nodoei bidaltzeko zeharkako komunikazioaren erabilpenaren beharra dela eta, beranduago iritsiko zaie. Hau inprimatutako denboretan antzeman daiteke. 0 eta 4 nodoak *OutConnected* multzoan daude baina uneko koordinatzaileak (3.garren nodoa) egoera fitxategian 3-ko balioa duenez zeharkako komunikazioaz baliatuko da emaitza sakabanatzeko eta honegatik, erabakitako *broadcast* metodoaren arabera, behin-betiko erabakia hurrengo txanda batean iristen zaie.

6.2.5. Proba 5



6.6 Irudia: Proba 5.

Simulazio proba kasu honetan elementu berri bat sartu da: nodoek denboran zehar egoera alda dezakete. Orain arte simulazio osoan zehar nodoek egora berdina mantentzen zuten. Hau irudikatzeko koloreaz baliatuko gara. Beltzez marraztutako geziak egoera konstantea adieraziko dute eta berdez marraztutakoak berriz, egoera aldaketa jasaten duten komunikabideak dira. Geziaren norantzarekin zein egoerara pasa den adieraziko da.

Proba kasu honetan, egoera aldaketa 20.garren segundoan emango da. Sarearen parada dela eta, algoritmoa ez da *consensus* batera ailegatuko lehenengo 20 segundoetan. 20.garren segundoan egoera berri batera pasako da. Komunikabide berriak sortzen dira eta ondorioz baldintzak aldatzen dira. Une honetan aurreko proban emandako simulazio eszenario bera dugu (6.5 adibidea), hortaz, aurrerago azaldu den bezala, nodoek *consensus* batera iritsiko dira.

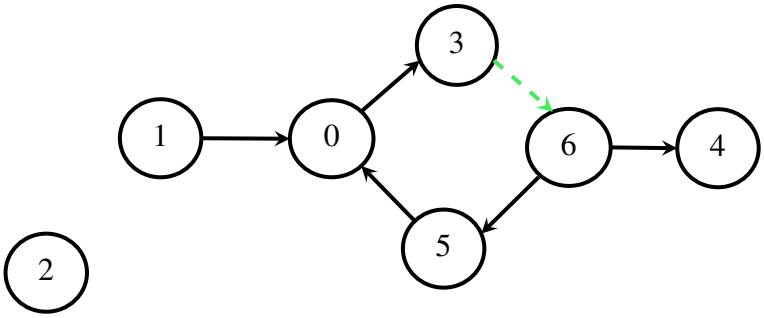
Simulazioaren exekuzioa:

```
================================
Executing simulation proba5.....
================================
Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.269s)
WARNING: Ignoring blank row.
Node 3 : All nodes arrived to a consensus and the estimation decided is 9 in round 38
Consensus time is 20.5202
Node 5 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.5253
Node 6 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.5253
Node 0 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.5303
Node 4 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.5303
NODE 0 GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
NODE 4 GET CONSENSUS!!!
NODE 5 GET CONSENSUS!!!
NODE 6 GET CONSENSUS!!!
```

Simulazio emaitza: Simulazioko irteeran antzematen denez, uneko koordinatzaileak 20.garren segundoa pasata *consensus* erabakia hartuko du. Ordura arte ez dira betebeharreko baldintzak betetzen. Komunikabide berriak sortzean, egoera aldaketa ematen denez *consensus*-a lortzen da (aurreko proba kasuaren eszenario berdina da eta). Koordinatzaileak erabakia hartzen duenetik azkeneko nodoak emaitza jaso arte denbora tarte bat dago, emaitza sakabanatze prozesuarengatik. Lehenago edo geroago jasotzearen arrazoia komunikazio bide eta sarearen abiaduraren menpe egonen da.

6.2.6. Proba 6

Aurreko proba kasuaren ildo berdinari jarraituz, berdez irudikatutako marra etenak denbora tarte baten ondoren aktibatuko den komunikabidea dugu. Kasu honetan berriz ere 10 segundo pasa eta gero. Hasieran ez dago *consensus*-a lortzea, *InConnected* diren nodo kopuruak nahiko badira, baina hauek aldi berean ez daude *OutConnected* multzoan. 10.garren segundoan, komunikabide berriengatik mezuak sarean zehar sakabanatzeko bideak sortzen dira. Hortaz, *consensus*-a lortzeko baldintzak betetzen badira, emaitza bat lortuko da. Simulazio honetan 10.garren segundotik aurrera betebeharreko baldintzak betetzen direnez, *consensus* batera



6.7 Irudia: Proba 6.

ailegatuko dira nodoak.

Simulazioaren exekuzioa:

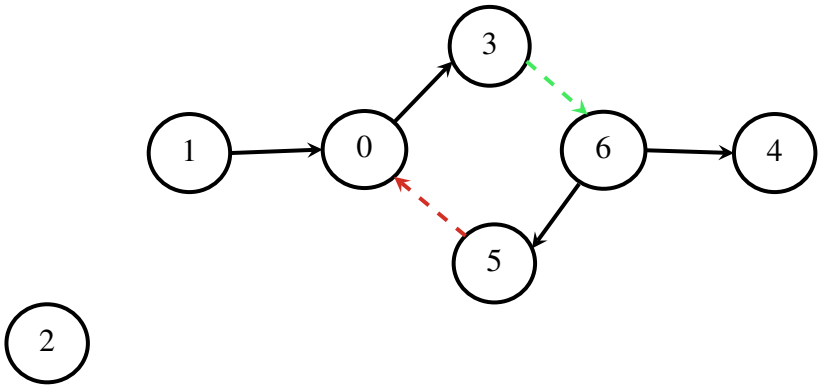
```
================================
Executing simulation proba6.....
================================
Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.297s)
WARNING: Ignoring blank row.
Node 5 : All nodes arrived to a consensus and the estimation decided is 10 in round 19
Consensus time is 10.5353
Node 0 : All nodes arrived to a consensus and the estimation decided is by coordinator 5 with
     value 10 in round 19
Consensus time is 10.5403
Node 3 : All nodes arrived to a consensus and the estimation decided is by coordinator 5 with
     value 10 in round 19
Consensus time is 10.5454
Node 6 : All nodes arrived to a consensus and the estimation decided is by coordinator 5 with
     value 10 in round 19
Consensus time is 10.5504
Node 4 : All nodes arrived to a consensus and the estimation decided is by coordinator 5 with
     value 10 in round 19
Consensus time is 10.5554
NODE 0 GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
NODE 4 GET CONSENSUS!!!
NODE 5 GET CONSENSUS!!!
NODE 6 GET CONSENSUS!!!
```

Simulazio emaitza: 10.garren segundoa arte ez dago *consensus*-ren berririk. Egoera aldaketa suertatu bezain pronto nodoak *consensus* batera iristen dira. Beste adibidetan bezala, badira nodo batzuk *consensus*-a lortzen parte hartu ez dutenak, baina hauei erabakitako emaitza iritsiko zaie baldin eta soilik baldin *Inconnected* badira eta *well-connect* nodo batekin lotura badute. Exekuzioak adierazten duen bezala, kasu honetan *consensus*-a lortzeko parte hartu duten nodoak 0, 3, 5 eta 6 izango dira. Uneko koordinatzaileak proposatutako balioen artean 1 nodoko balioa ere egon daiteke. Erabakitako emaitza 4 nodoari iritsiko zaio *InConnected*

delako eta *OuConnected* nodo batekin komunikabide bat duelako.

Simulazioaren irteeran igarri daiteke erabakia hartu denetik azkeneko nodoari emaitza iristen zaion momentua arte denbora tarte bat pasa dela. Hau egindako *reliable-broadcast* edo sakabanatzearengatik eta zeharkako komunikazioarengatik da.

6.2.7. Proba 7



6.8 Irudia: Proba 7.

Simulazio kasu honetan bi egoera aldaketa ematen dira. Berdez dagoen marra, 10.garren segundoan emango den aldaketa adierazten du, marra gorria berriz 20.garren segundoan emango den aldaketa. Lehenengo aldaketaren kasuan *consensus*-a lortuko betebeharreko baldintzak betetzen ez direnez, ez dute erabaki bat hartuko. Nahiz eta beharrezkoak diren *InConnected* nodo kantitatea egon, ez da nahikoa izango $\lceil \frac{(n+1)}{2} \rceil$ *OutConnected* nodo baino gutxiago daudelako. 20.garren segundoak emandako egoera aldaketa berriarengatik komunikazio bide berri bat sortzen da. Kasu honetan aurreko adibidean (6.7 irudiko proba kasua) emandako baldintza berdinak ematen dira berriz ere. Beraz 20.garren segundotik aurrera nodoak *consensus* batera iritsi beharko dira.

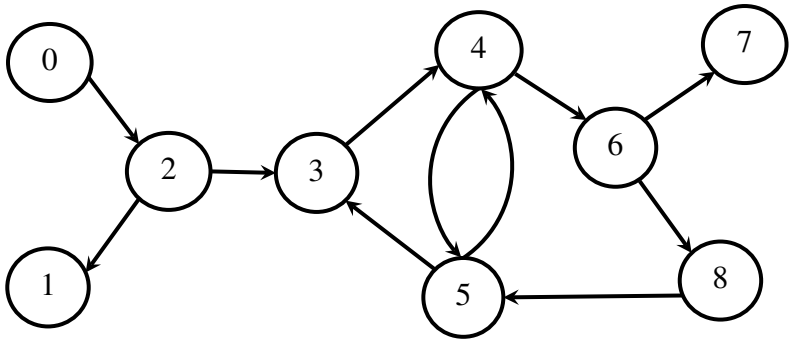
Simulazioaren exekuzioa:

```
================================
Executing simulation proba7.....
================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.283s)
WARNING: Ignoring blank row.

Node 3 : All nodes arrived to a consensus and the estimation decided is 9 in round 38
Consensus time is 20.0302
Node 6 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.0353
Node 4 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.0403
Node 5 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.0403
Node 0 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 38
Consensus time is 20.0453
NODE 0 GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
NODE 4 GET CONSENSUS!!!
NODE 5 GET CONSENSUS!!!
NODE 6 GET CONSENSUS!!!
```

Simulazio emaitza: Simulazioan ikusten den bezala, 20.garren segundoa arte ez da *consensus* batera iritsi. Lehenengo egoera aldaketan, 10.garren segundoan, nodo batzuetan aldaketa badago, baina ez nahikoak *consensus* batera iristeko. 4,5 eta 6 nodoek bere burua *InConnected*-tzat hartzen dute eta *OutConnected* matrizea eguneratu dute. 0, 1 2, 3 nodoek berriz, bere burua ez dute *InConnected*-tzat hartzen, beraz $\lceil \frac{(n+1)}{2} \rceil$ *InConnected* nodoen baldintza ez da betetzen momentu honetan. Hurrengo egoera aldaketan berriz, 20.garren segundoan, beharrezkoa diren baldintza betetzen direnez, aldaketa gertatu bezain pronto nodoak *consensus* batera ailegatuko dira.

6.2.8. Proba 8

Ondorengo adibidea beste atal batean agertutako kasu bat da (3.2.3 atala). Sistema elkarkonektaturiko 9 nodoetatik, 5 *well-connected* nodoak dira (3, 4, 6, 8, 5). Hauek *consensus* batera iritsiko dira eta behin emaitza erabaki dela, *reliable-broadcast*-ekin sarean zehar sakabanatuko da. Kasu honetan *well-connected* nodoez aparte, *reliable-broadcast* 7.garren nodora iritsiko da, hau bere burua *InConnected*-tzat duelako eta R-Broadcast-arekin sakabanatuko



6.9 Irudia: Proba 8.

duen 6.garren nodoarengatik. 0, 1 eta 2 nodoek erabakitze proposamen bakoitzean, norberaren estimazioa proposatuko dute *OutConnected* multzoaren parte izango direlako. Gerta liteke hauen estimazioa aukeratua izatea behin-betiko emaitza bezala, baina azkeneko erabakia ez dute inoiz jasoko bere burua *InConnected*-tzat hartzen ez dutelako.

Simulazioaren exekuzioa:

```
==================================
Executing simulation proba8.....
==================================

Waf: Entering directory `/home/aingeru/workspace/src/ns-3.18/build'
Waf: Leaving directory `/home/aingeru/workspace/src/ns-3.18/build'
'build' finished successfully (1.273s)
Node 3 : All nodes arrived to a consensus and the estimation decided is 9 in round 3
Consensus time is 3.55537
Node 4 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.56044
Node 5 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.56547
Node 6 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.56549
Node 7 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.57052
Node 8 : All nodes arrived to a consensus and the estimation decided is by coordinator 3 with
     value 9 in round 3
Consensus time is 3.57054

NODE 0 DOESN'T GET CONSENSUS!!!
NODE 1 DOESN'T GET CONSENSUS!!!
NODE 2 DOESN'T GET CONSENSUS!!!
NODE 3 GET CONSENSUS!!!
NODE 4 GET CONSENSUS!!!
NODE 5 GET CONSENSUS!!!
NODE 6 GET CONSENSUS!!!
NODE 7 GET CONSENSUS!!!
NODE 8 GET CONSENSUS!!!
```

Simulazio emaitza: Kasu honen deskribapenean esan bezala, *well-connected* diren nodoez aparte, 7.garren nodoak behin betiko emaitza jasoko du. Nodo honek ez du erabakian parte hartuko *OutConnected* multzoaren barne ez delako egongo, baina bere burua *InConnected*-tzat hartzen duenez eta *well-connected* nodo batekin komunikabide kanala duenez, behin-betiko emaitza helduko zaio. Berriz 0, 1 eta 2 nodoak beren estimazioak proposatu arren, ez dute behin-betiko erabakia jaso, bere burua *InConnected*-tzat hartzen ez dutelako.

Simulazio sorta burutu ondoren, aurkeztutako algoritmoaren portaera *NS-3* simulagailuan aztertu da. Sare erreal batera eramaten bada, agian portaera desberdinak izan ditzake, adibidez *consensus* batera ailegatzeko behar den denboran. Simulagailuan konexioak oso egonkorra direla ematen dute, exekutatuko aplikazioak behin eta berriz exekutatzen direnean *consensus* denborak beti berdinak direlako. Berriz, sare erreal batek ziurrena egonkortasun maila hori ez dela bermatzen eta hortaz, aplikazioaren exekuzioak emaitza desberdinekoak izango lirateke. Trafikoarekin simulatzean, ez du arazorik eman nahiz eta kanalak datuaz beteta egon. Sare erreal batean sareak *buffer-overflow* bat jasan lezake eta hortaz, algoritmoaren portaera ere bai. Simulatutako kasu desberdinak ikusita, sare erreal batean *over-flow*-ren bat gertatzen bada algoritmoak emaitza egokia erabakiko luke.

7 KAPITULUA

Ondorioak eta etorkizuneko lerroak

7.1. Helburuak

Proiektua hasterakoan ezarritako helburuak bete dira: Algoritmoen inplementazioa eta simulazioa. Simulazioa aurrera eramateko lehenik simulagailua aukera behar izan da, NS-3 simulagailua kasu honetan. Bertan sistema banatu bat inplementatzea eta simulatzea zen helburua eta horrek dakarren *consensus* eta hutsegite-detektatzaile algoritmoen inplementazioa. Helburua lortua da. Ondoren, eraikitako aplikazioan eszenario ugari simulatu daitezkeela ikusi da, honela simulazioaren helburua bete dela bermatuz.

7.2. Denborazko planifikazioa

Jarraian, denbora plangintzan egindako aldaketak kontuan hartuta gure kostu plangintza 7.1 taulan agertzen da

Kanpoko faktoreak eraginda denborari buruzko arazoarekin aurkitu gara. Aurreikusitako denboraren kudeaketa behin baino gehiagotan aldatu behar izan dugu, zenbait kasutan ezin izan ditugulako atazei estimatutako denbora orduak mantendu. Denboraren faktorea, besteak beste, proiektuaren gaiaren esperientzi ezarekin bat dator. Kasu honetan, ikasketa ataza batzuk espero zenaren baino denbora gehiago eman dute. Hasieran, sistema banatuen arloaren esperientzia eskasak eta zenbait kontzeptu eta algoritmoen ulermenak estimatutako denbora gehiago eman dute. Hau ongi barneratua egon behar zen proiektua aurrera eramateko. Bestalde, simulagailuaren ikasketak ere denbora gehiago eman du, batez ere kontzeptuen barneratzea eta inplementatzeko era. Denbora estimazioak erabakitzean, formazioari lan ordu gehiago egokitu behar izan nion.

Zuzendariekin edukitako komunikazio eskasak ere zerikusia eduki proiektuaren atzerapenean. Proiektu kudeaketan ezarritako datak ez dira bete batez ere kudeaketan estimatutako datetan zuzendariekin argudiatzeko material gutxi zegoelako. Ondorioz, proiektuaren aurrera eramatea erritmoa motela izan da.

Garapen atalean denbora gehien eraman duen atala inplementazioa izan da. Programaren di-

Ataza	Orduak
Prozesu Taktikoa	35
Plangintza	20
Bilerak	10
Informeak	5
Prozesu operatiboak	375
Formazioa	205
Sare Banatuak	40
TrustedPals eta Consensus	60
Simuladorearen ikasketa	70
C++	20
LaTeX Ikasi	15
Garapena	170
Analisia	15
Simulazioaren kasuak analizatu	15
Diseinua	20
Simulazioaren Kasuen erabaki	20
Inplementazioa	115
Algoritmoaren kodeketa	80
Simulazioan ingurunea kodetu	20
Kasu proben kodeketa	15
Datu Pilaketa	20
Dokumentazioa	145
Memoriaren edukia	110
Defentsaren prestakuntza	25
Erabilpen gida	10
Guztiara	555

7.1 Taula: Kostu plangintza erreala.

seinua simulagailuan nola egin erabakitzea eta inplementatzea izan da arazo nagusia. Jarraian sakonean aztertuko dira atal honetan izandako arazoak:

- C++ lengoaian trebakuntza hartzeak eta simulagailua kontrolatzeak, logikoa zen bezala bere denbora eman du, baino igarritakoa gaindituz. Kodeketan arazo aunitz aurkitu dira, seguruena lengoaiaren ezagutza eskasarengatik eta simulagailuaren kontrol ezagatik. C++ lengoaia sakonago ulertzea eta ikasteak buru-hauste ugari ekarri ditu eta ondorioz denbora gehiago sartu behar izatea.
- Zuzendariekin edukitako komunikazio eskasarengatik, hasierako inplementazio modu okerrean hasi zen. Bilera baten ondoren garapen lana berriz bideratu zen. Behin simulazio diseinu berria erabakita eta ordurarte ikasitakoarekin, inplementazioak aurrera egin zuen. Argi ikusten da inplementazioan sartutako orduen luzapena.

Proiektuan zehar aurkitutako beste arazo bat, informazio galera izan zen. Proiektuarekin hastean segurtasun kopiei garrantzi handia ez zen eman. Hasieran, kopia frekuentzia egunekoak edo astekoak ziren. Informazio galera bat egon zen eta egun batzuen lana galdu zen. Geroztik kopientzako beste sistema bat inplementatzea erabaki zen, egunerokoa eta bertsio kontrolarekin.

Azkenik, dokumentazioari estimatutako denbora ere luzeagoa izan da, batez ere simulazioak ongi burutu arte eta kasu proben exekuzioa eduki arte ezin zelako amaitu. Bestalde, LaTeX tresnan lan egiteko ikasketa orduak sartu behar izan direlako.

Ondoren ikusiko dugun Gantt-diagrama honetan (7.1 irudia), gure proiektuan zehar azpi-ataza edo fase bakoitzari eskainitako denborak zehaztuko dira. Proiektuarekin hasi aurretik egindako denbora plangintzarekin alderatuta (2.2 irudia) honakoa ondorioztatu dezakegu:

Nota: Bilerak *Gantt* diagraman bloke bakar batean adierazten dira nahiz eta 4 bilera fisiko eduki proiektuan zehar.

7.3. Etorkizuneko lerroak

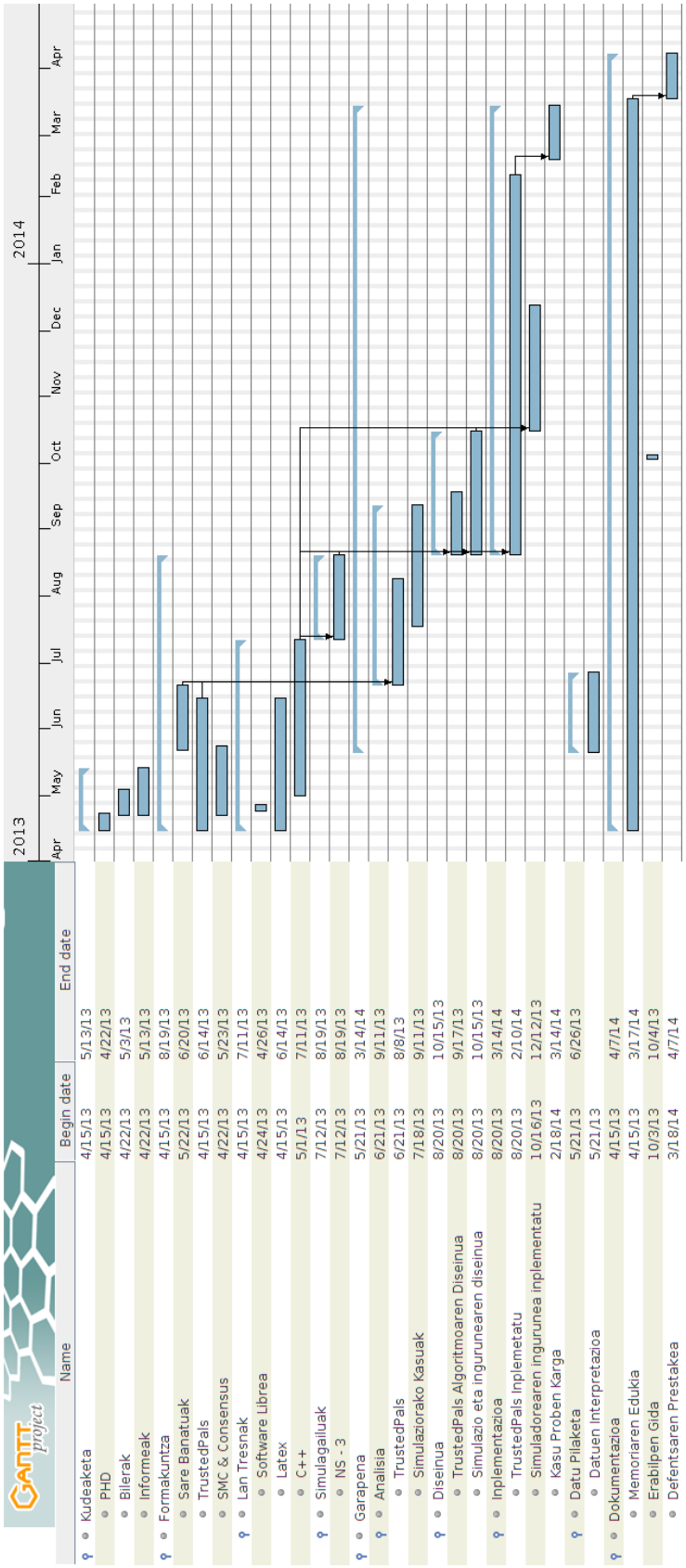
6.2 atalaren amaieran aipatu bezala, NS-3 simulagailua eta mundu errealeko sare baten portaerak desberdina izan daitezke. NS-3 algoritmoa inplementatzeko eta simulazioan funtzionamendu egokian egikaritzen dela ikusteko oso baliagarria da, honez gero interesgarria izango litzateke ingurune erreal batean sortutako aplikazioa probatzea.

NS-3-ren analisian aztertu den bezala [4.2.4], NS-3 simulagailuak *TestBed*-ekin lan egiteko aukera ematen du eta aukeren artean *PlanetLab* dago. *PlanetLab*-entzako API-a garatu da. Gogoratu *PlanetLab* simulatzeko tresnen artean edukitako aukera bat izan dela (ikusi 4.2.3 atala). Behin algoritmoen inplementazioa NS-3 simulagailuan garatu dela, *PlanetLab* API-aren erabilerarekin sare errealeko portaera aztertu daiteke. Aldaketa gutxi egin beharko lirateke. Garatutakoaren egitura berdina mantendu eta metodologia berrerabiltzen bada, *main* klase nagusian *PlanetLab* deiak barneratzea besterik ez da. Ideia interesgarria litzateke simulagailuaren eta sare errealaren portaeren arteko alderaketa.

Sare erreal bat saturatzean zer gertatuko da? Simulagailuan, algoritmoak ez du errorerik eman sarea gainkarga jasatean. Simulagailuan sare egitura guztia kontrolpean dago eta guk sartzen ez ditugun hutsegiteak ez dira sortuko. Simulazioan sarearen portaera berezi bat simulatu nahi bada, lehenik aurreikusi behar da gero simulazioan aplikatzeko. Adibidez, sare erreal batean ausaz gerta daitezkeen *buffer-overflow* eta kolisioak bezalako erroreak simulatzeko aurreikusi behar dira, simulazioak duen muga da. *PlanetLab* proba bankua bezalako saretan, dispositiboen kontrola gure esku ez dagoenez, ustekabeko erroreak sor daitezke, eta hauen portaera ikustea interesgarria litzateke.

7.4. Ondorio pertsonalak

Proiektu honekin helburu pertsonal asko bete dira. Lan metodologi bat ikastera eta inplementatzea eraman nau, hemendik aurrera bizitzan garrantzitsua izango dena. Tresna berri ugarietan ikasketak ere egon dira, hala nola C++ lengoaiaren ezagutza sakonagoa, LaTeX dokumentazio tresnaren ikasketa eta simulagailu baten erabilpena. Bestalde, komunikazioaren garrantziaz



7.1 Irudia: Gantt diagramaren 2 .estimazioa

ohartzera ere eraman nau. Nahiko eskasa izan dela esan behar da eta horrek ekarritako arazoek komunikazioak duen garrantziaz ohartzera eraman nau.

Gaur, nire ikasketa zikloko fase berri bat amaitzen da eta nire bizitzako beste proiektu pertsonal bat.

Gaur Informatika Ingeniaritza ikasketak amaitzen dira eta gauza asko ikasi ditut, oso baliozkoak izan direnak. Lan egiteko metodologiak, informatika munduko ezagutzak eta gure garaian komunikazioak duten garrantzia, ezagutza guzti hauek nire bizitzaren hurrengo faseetan aurkituko ditudan lanetarako baliozkoak izango direnak.

Urte hauetan ezagututako jende anitza eta elkarrekin partekatutako pentsamendu eta interesak ere gogoan izango dira. Jende asko pasa da urte hauetan eta hauek ezagutzeak bizitzari beste ikuspuntu bat ekarri dio.

Amaitzeko, eskerrak eskatu nahi ditut urte hauetan hainbeste irakatsi didaten irakasle eta pertsonei, bai informatika inguruko ikasketetan eta baita pertsonaletan.

Eskerrik asko,

Aingeru

Bibliografia

[Alvisi, 2014] Alvisi, L. (2014). Reliable-broadcast azalpen gardenkiak. http://www.cs.utexas.edu/users/lorenzo/corsi/cs371d/08F/notes/week8.pdf. Azkenengo bisita: 2014an.

[Baldo, 2014] Baldo, N. (2014). Validation of the ieee 802.11 mac model in the ns3 simulator using the extreme testbed. http://scholar.google.es/scholar_url?hl=es&q=http://www.researchgate.net/publication/228348933_Validation_of_the_IEEE_802.11_MAC_model_in_the_ns3_simulator_using_the_EXTREME_testbed/file/9fcfd5076dd088d532.pdf&sa=X&scisig=AAGBfm01lv9QfrHG7qxNmm462bAx2EhFbA&oi=scholarr&ei=8NGVU7uuK-W_0QXFloGIBA&ved=0CCoQgAMoADAA. Azkenengo bisita: 2014an.

[C++, 2014] C++ (2014). C++ lengoaiarako dokumentazioa. http://www.cplusplus.com/. Azkenengo bisita: 2014an.

[Chandra and Toueg, 1996] Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267.

[Chen, 2000] Chen, Z. (2000). *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley Longman Publishing Co., Inc.

[Code-Project, 2014] Code-Project (2014). Code project - c++ artikuluak. http://www.codeproject.com/. Azkenengo bisita: 2014an.

[Cortiñas et al., 2012] Cortiñas, R., Freiling, F. C., Ghajar-Azadanlou, M., Lafuente, A., Larrea, M., Penso, L. D., and Soraluze, I. (2012). Secure failure detection and consensus in trustedpals. *IEEE Trans. Dependable Sec. Comput.*, 9(4):610–625.

[Dijkstra, 2014] Dijkstra, E. W. (2014). Self-stabilizing systems in spite of distributed control. http://www.cs.utexas.edu/~EWD/ewd04xx/EWD426.PDF. Azkenengo bisita: 2014an.

[Dwork et al., 1988] Dwork, C., Lynch, N. A., and Stockmeyer, L. J. (1988). Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323.

[Eclipse, 2014] Eclipse (2014). Eclipse cdt. http://www.eclipse.org/cdt/. Azkenengo bisita: 2014an.

[Eclipse-Config, 2014] Eclipse-Config (2014). Eclipse cdt konfigurazioa ns-3-rako. http://www.nsnam.org/wiki/index.php/HOWTO_configure_Eclipse_with_ns-3. Azkenengo bisita: 2014an.

[Fort et al., 2006] Fort, M., Freiling, F. C., Penso, L. D., Benenson, Z., and Kesdogan, D. (2006). Trustedpals: Secure multiparty computation implemented with smart cards. In *ESORICS*, volume 4189 of *LNCS*, pages 34–48. Springer.

[Freiling et al., 2011] Freiling, F. C., Guerraoui, R., and Kuznetsov, P. (2011). The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9.

[GanttProject, 2014] GanttProject (2014). Ganttproject tresna. http://www.ganttproject.biz/. Azkenengo bisita: 2014an.

[GTNetS, 2014] GTNetS (2014). Georgia tech network simulator (gtnets). http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/. Azkenengo bisita: 2014an.

[Hadzilacos and Toueg, 1993] Hadzilacos, V. and Toueg, S. (1993). *Distributed systems*, chapter Fault-tolerant broadcasts and related problems, pages 97–145. ACM Press/Addison-Wesley Publishing Co., second edition.

[INET, 2014] INET (2014). The inet framework - omnet++. http://inet.omnetpp.org/. Azkenengo bisita: 2014an.

[Itoiz, 2014] Itoiz, A. (2014). Proiektuaren iturburu kodea. https://www.dropbox.com/s/125chajjttvobdp/TrustedPals%20Kodea.zip. Azkenengo bisita: 2014an.

[Lamport et al., 1982] Lamport, L., Shostak, R. E., and Pease, M. C. (1982). The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

[Mercurial, 2014] Mercurial (2014). Mercurial kontrol sistema. http://mercurial.selenic.com/. Azkenengo bisita: 2014an.

[NetBeans-Config, 2014] NetBeans-Config (2014). Netbeans konfigurazioa ns-3-rako. http://www.nsnam.org/wiki/index.php/HOWTO_configure_NetBeans_with_ns-3. Azkenengo bisita: 2014an.

[NS-3-Consortium, 2014] NS-3-Consortium (2014). Network simulator 3. http://www.nsnam.org/. Azkenengo bisita: 2014an.

[NS-3-Doxygen, 2014] NS-3-Doxygen (2014). Network simulator 3, klaseen eta moduluen dokumentazioa. https://www.nsnam.org/doxygen/index.html. Azkenengo bisita: 2014an.

[NS-3-Forum, 2014] NS-3-Forum (2014). Network simulator 3, ns-3 erabiltzaileen forua. https://groups.google.com/forum/#!forum/ns-3-users. Azkenengo bisita: 2014an.

[NS-3-Tutorial, 2014] NS-3-Tutorial (2014). Network simulator 3, tutoriala eta erabilpen gida. http://www.nsnam.org/docs/release/3.19/tutorial/singlehtml/index.html. Azkenengo bisita: 2014an.

[NS-3-Tutorial-First.cc, 2014] NS-3-Tutorial-First.cc (2014). Network simulator 3 tutorialeko first.cc adibidea. http://www.nsnam.org/docs/release/3.14/tutorial/singlehtml/index.html#a-first-ns-3-script. Azkenengo bisita: 2014an.

[NS3-Bertsioak, 2014] NS3-Bertsioak (2014). Network simulator 3 bertsioen esteka. http://code.nsnam.org/. Azkenengo bisita: 2014an.

[NS3-Prerequisites, 2014] NS3-Prerequisites (2014). Network simulator 3 aurrekariak. http://www.nsnam.org/wiki/index.php/Installation#Prerequisites. Azkenengo bisita: 2014an.

[NS3-Release, 2014] NS3-Release (2014). Ns-3 tarball azken bertsioa. http://www.nsnam.org/releases/. Azkenengo bisita: 2014an.

[OMNet++, 2014] OMNet++ (2014). Omnet++. http://www.omnetpp.org/. Azkenengo bisita: 2014an.

[OPNET, 2014] OPNET (2014). Opnet technologies, inc. http://www.opnet.com/. Azkenengo bisita: 2014an.

[OPNETWiki, 2014] OPNETWiki (2014). Opnet wikipedia informazioa. http://en.wikipedia.org/wiki/OPNET. Azkenengo bisita: 2014an.

[OppBSD, 2014] OppBSD (2014). Oppbsd. https://svn.tm.kit.edu/trac/OppBSD/. Azkenengo bisita: 2014an.

[orbit, 2014] orbit (2014). Orbit. http://www.orbit-lab.org/. Azkenengo bisita: 2014an.

[Parvédy and Raynal, 2004] Parvédy, P. R. and Raynal, M. (2004). Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *SPAA*, pages 302–310. ACM.

[PlanetLab, 2014] PlanetLab (2014). Planetlab web orri ofiziala. http://www.planet-lab.org/. Azkenengo bisita: 2014an.

[QT-Config, 2014] QT-Config (2014). Qt creator konfigurazioa ns-3-rako. http://www.nsnam.org/wiki/index.php/HOWTO_configure_QtCreator_with_ns-3. Azkenengo bisita: 2014an.

[realworldOMNeT, 2014] realworldOMNeT (2014). Integrating real world applications into omnet++. http://telematics.tm.kit.edu/english/article.php?publication_id=290&language_id=2. Azkenengo bisita: 2014an.

[Reliable-Broadcast, 2014] Reliable-Broadcast (2014). Reliable-broadcast wikipediako informazioa. http://en.wikipedia.org/wiki/Terminating_Reliable_Broadcast. Azkenengo bisita: 2014an.

[riverbed, 2014] riverbed (2014). Riverbed. http://www.riverbed.com/products/performance-management-control/network-performance-management/network-simulation.html. Azkenengo bisita: 2014an.

[Self-Stabilitation, 2014] Self-Stabilitation (2014). Self-stabilitation wikipediako informazioa. http://en.wikipedia.org/wiki/Self-stabilization. Azkenengo bisita: 2014an.

[Stackexchange, 2014] Stackexchange (2014). Stackexchange - tex munduko kontsultarako forua. http://tex.stackexchange.com/. Azkenengo bisita: 2014an.

[Stackoverflow, 2014] Stackoverflow (2014). Edozein garapen kontsultarako forua. http://stackoverflow.com/. Azkenengo bisita: 2014an.

[Wikibook, 2014] Wikibook (2014). Latex wikibook. http://en.wikibooks.org/wiki/LaTeX. Azkenengo bisita: 2014an.

[Yao, 1982] Yao, A. C.-C. (1982). Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE.

Eranskinak

A ERANSKINA

Network Simulator 3 - Erabilpen gida

Network Simulator 3 [NS-3-Consortium, 2014] edo NS-3 gertaera diskretuen simulagailu bat da, NS-2 simulagailuaren ondorengotzat hartu da. C++ lengoaian garatutako simulagailua da baina *Python* lengoaiarekin lan egiteko aukera dago.

Python-en lan egin nahi bada, simulagailuak garatuta dituen *Python*/C++ loturekin programak inplementatuko dira. *Python* lotura hauek, NS-3 simulagailuan C++-en garatutako programen deiak izango dira.

Simulagailuaren sakontasunean sartu aurretik, hau erabiltzeko beharko diren tresnak eta hauen instalazioa azalduko dira.

A.1. Sistemaren konfigurazioa

NS-3 simulagailua, GNU/Linux plataformetarako garatutako sistema da, non oinarrizko simulazio bat egikaritzeko gcc edo clang konpilatzaileak eta Python interpretatzailea beharrezkoak dira.

Ondorengo sistema eragile eta konpilatzaileekin erabil daiteke:

- Linux x86 and x86_64: gcc versions 4.1 through 4.7 and, 3.4.6, clang versions 3.0 and 3.1.
- FreeBSD x86 and x86_64: gcc version 4.2, clang versions 3.0 and 3.1.
- Mac OS X ppc and x86: gcc versions 4.0 and 4.2.

Windows sisteman erabiltzeko, edozein Linux sistema birtual bat erabili beharko da edo Wubi bezalako tresnak. Bada MinGW edo Cygwin komando interpretatzaileen erabilera, baina mantenu ofiziala ezagatik simulagailuaren portaera egokia ez da bermatzen.

Simulagailuarekin lan egiteko garapen ingurune (IDE) ugari daude. Garatzaile bakoitzak bere gustukoen garapen ingurunea erabil dezake, baina proposamen bat jartzearren eta nahiz eta hauek ofizialak ez izan, garatzaile askok hauekin lan egiten dute. Adibidez: Eclipse, QtCreator edo NetBeans.

Nik proiektuan lan egiteko eraili ditudan tresnak hauek dira.:

- Ubuntu 12.04.
- Eclipse IDE Kepler
- Network Simulator 3 v3.19

A.1.1. Aurre betebeharrak

Aurreko atal batean azaldu bezala, NS-3 simulagailuaren *core*-ak *gcc/g++* 3.4 edo berriago den konpiladorearen beharra du eta *Python* 2.4 bertsioa baino handiago izan behar du. Hau oinarrizko moduluak exekutatzeko betebeharrak dira, baina simulagailuaren beste aukera gehiago erabiltzeko, hurrengo estekan beste paketeen instalazioa burutzeko komandoak daude, bai Debian/Ubuntu sistemarentzako hala nola beste sistementzako, Fedora edo FreeBSD

- NS-3 aukeren esteka begirada eman [NS3-Prerequisites, 2014]

Nahiz eta nahi diren aukerak instalatu daitezkeen, *mercurial* [Mercurial, 2014] kontrol sistemen plataformaren instalazioa gomendagarria da simulagailuaren azkeneko bertsioa egonkorra edukitzeko.

A.1.2. Instalazioa

Simulagailua gure konputagailuan instalatzerakoan bi instalazio aukera ematen dira: *mercurial* edo *tarball* bidez. Gida honetan *mercurial* bidezko instalazioa azalduko da baina *tarball* instalazioa nahiago bada NS-3 *tarball* azken bertsio jaitsi [NS3-Release, 2014] eta behin hau deskonprimitu dela *mercurial* instalazioaren azkeneko agindua exekutatu.

Instalazioarekin hasi baino lehen, *mercurial*-ek lokalki sortuko dituen errepositorioen fitxategiak gordetzeko, direktorioa sortzea da. *Mercurial* instalaturik dagoela suposatuz, *ns-3-allinone* direktorioaren kopia bat egiteko Linux shell-ean ondorengoa idatzi beharko da:

```
cd
mkdir repos
cd repos
hg clone http://code.nsnam.org/ns-3-allinone
```

Behin *hg* edo *mercurial* komandoa exekutatu dela, ondorengoaren antzeko testu bat agertu beharko da:

```
destination directory: ns-3-allinone
requesting all changes
adding changesets
adding manifests
adding file changes
added 26 changesets with 40 changes to 7 files
7 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Behin fitxategi guztiak sinkronizatu direla, *repos* direktorioan ondorengo fitxategiak aurkituko dira (ls komandoa erabili):

```
build.py*  constants.py  dist.py*  download.py*  README  util.py
```

Ikusi *Python* script batzuk sortu direla. Ondorengo pausoa simulagailuaren jaitsiera eta instalazioa da. Simulagailuaren bertsio konkretu bat eduki nahi bada, hau jaisteko aukera dago. NS-3 bertsioen estekan [NS3-Bertsioak, 2014], eskura dauden bertsio eta bestelako pakete aukerak agertzen dira. Garatzaileek garatzen ari diren simulagailua jaitsi nahi bada, *download.py* script-a exekutatzearekin nahikoa da, baina bertsio egonkor batekin lan egiteko, azkeneko bertsioaren jaitsiera adieraziko dugu, kasu honetan 3.19 bertsioa. Ondorengo komandoaren exekuzioa egin:

```
./download.py -n ns-3.19
```

Behin sistema sinkronizatu dela eta gure konputagailuan simulagailuaren moduluen kopia dugula, ondorengo komando exekutatuz sistema eraikiko da:

```
./build.py
```

Denbora tarte baten ondoren eta sistemaren eraikitzea amaitzen denean, dena ongi joan bada antzeko irteera agertuko da.

```
Build finished successfully (00:02:37)
Leaving directory `./ns-3.19'
```

Behin sistema eraiki dela, *ns-3-allinone* direktorioan sinkronizatutako *script*-ak ez dira gehiago erabiliko eta hemendik aurrera NS-3 simulagailuaren direktorioaren barnean konpilazio berriak egiteko, *./waf* komandoarekin lan egingo da. Kasu honetan, edozein konpilazio berri egiteko ns-3.19 direktorioan kokatu beharko gara.

A.1.3. Waf konfigurazioa

Behin oinarrizko sistema eraiketa egin dela, beste aukera batzuk ere aktiba daitezke. Gure aplikazioak garatzen goazen heinean, arazo ugari gerta daitezke eta hauek *debug*-etzeko tresna badago. *waf* komandoa bidez konfiguratuko da. Jarraian, proiektu honetarako erabili den konfigurazioa:

```
./waf -d debug configure --enable-sudo --enable-examples --enable-tests
```

Lehenengo *debug* aukerarekin, eskura dugun debugger-ekin lan egiteko emango da. Gure garapenean suerta daitezkeen arazoak errazago bilatzeko erabiltzen da, adibidez, *gdb* debug tresnarekin.

Bigarren aukerak, *enable-sudo*, modulu batzuen exekuzioa egiteko super-erabiltzaile aukera emango digu. Hau exekutatu da, nahiz eta proiektu honetan ez erabili, etorkizuneko lanak garatzeko honen beharra eman daiteke, adibidez, TestBed edo PlanetLab bezalako tresnak erabiltzeko.

enable-examples eta *enable-tests* aukerek, adibideen eta test batzuen konpilatzea egiten da. *enable-examples* moduluarekin tutorialetan agertzen diren adibideak instalatzen dira, honela tutorialarekin bat, tresnaren erabileran trebetasuna hartuko da.

Konfigurazioa sakonagoa egin nahi bada, honako komandoa erabiliz eskura dauden aukera guztiak inprimatuko dira:

```
./waf --help
```

Behin konfigurazioa komandoa exekutatu dela, zein modulu konpilatuko diren agertuko dira. Gure interesa duen beste modulu baten beharra badago, beharrezkoak diren konfigurazioa aukerak exekutatu beharko dira. Dudarik balego, aurre betebeharren ataleko estekara joan.

A.1.4. Sistema ondo funtzionatzen du?

Sistema modu egokian exekutatuko dela bermatzeko, NS-3 simulagailuaren direktorioan dagoen "*./test.py*" komandoa exekutatuz konproba daiteke. Moduluen test bat egingo da eta okerrik balego sistemak abisatuko du. Beste aukera bat, jada garatutako adibideren bat exekutatzea da, adibidez:

```
./waf --run scratch-simulator
```

A.1.5. Eclipse IDE

Proiektu hau garatzeko Eclipse IDE [Eclipse, 2014] aukeratu dut eta hau konfiguratzeko esteka honetan azaltzen da: Eclipse Konfigurazioa [Eclipse-Config, 2014].

Aurreko atal batean esan bezala, guk nahiago dugun IDE-aren gainean gara dezakegu eta aurkeztutako beste IDE-en konfigurazioa egiteko, ondorengo estekez baliatu gaitezke: QtCreator Konfigurazioa [QT-Config, 2014] eta NetBeans Konfigurazioa [NetBeans-Config, 2014].

Aukeratutako IDE-aren konfigurazioa egin aurretik, sistema konfiguratua edukitzea gomendagarria da.

A.1.6. Konpilazioa eta exekuzioa

Behin konfigurazioa eginda eta nahi ditugun moduluak konfiguratu direla, garatzen ari garen programa konpilatzeko komandoa hau deitzearekin nahikoa da:

```
./waf
```

Bakoitzak nahiago duen IDE-a aukera dezake, hortaz, IDE bertatik edo komando lerrotik konpila daiteke.

Guk garatu dugun *script*-aren exekuzioa egin nahi badugu, *run* parametroaz baliatuko gara. Adibidez:

```
./waf --run scratch-simulator
       edo
./waf --run sfdandconsensus
```

A.2. Script-en sorrerak

Simulagailuak berak modulu ugari inplementatuak ditu. Simulazio bat garatzeko *Pyhton* edo C++ lengoaiatan oinarritutako scrippting teknika erabili daiteke. Script-ek ondorengo egitura edukiko dute:

- Nodoen eta sarearen antolaketa.
- Helbide emateak eta protokoloen instalazioa.
- Exekutatu nahi diren aplikazioen instalazioa eta simulagailuaren konfigurazioa.

A.2.1. Oinarrizko kontzeptu batzuk

Sare batean agertzen diren kontzeptuak, NS-3 simulagailuan nola erabiltzen diren azalduko da ondorengo lerroetan:

- **Node.** Internet munduan, sare batera konektatzen den dispositibo bati *host* bezala ezagutzen da. NS-3 internet simulagailu bat izateaz gain, ere sare simulagailu bat da. *Host* terminoa Internet eta honen protokoloei oso lotuta dagoenez beste termino baten beharra dago. Kontzeptu generikoago bat erabiltzeko, NS-3 simulagailuan grafoen teorian agertzen den *node* terminoa erabiliko da eta ondorioz oinarrizko konputazio dispositibo abstraktua *node* izango da. Abstrakzioa *Node* klasearen bidez adieraziko da C++ lengoaian. *Node*-a konputagailu bat bezala har daiteke, non protokoloak, txartelak eta aplikazioak instala daitezke.

- **Application.** Simulagailua ez da mundu erreal batean bezala sistema eragile batez baliatzen operazioak egiteko, beraz ez du sistema erreal batean bezala funtzionatuko. Aplikazioa ordenagailuan exekutatuko den softwarea da, non "mundu errealeko" simulazioa eginez nahi ditugun atazak exekutatuko daitezke. *Application*-ak *node*-etan exekutatzen dira.

  Ondorengo adibidean ikusiko den bezala, bezero/zerbitzari motako aplikazio bat simulatzeko *UdpEchoClientApplication* eta *UdpEchoServerApplication application* moduluak erabiliko dira. Hauek *application* klase abstraktuarekin adieraziko dira C++ lengoaian, non klase mota honetako programek simulazioan aktibitate batzuen sorrera egingo dute.

- **Channel.** Mundu errealean konputagailu bat sare batera konekta daiteke. Askotan, datuak igarotzen diren medioak *channel* bezala ezagutzen dira. NS-3 simulatutako munduan, *node* bat komunikazio kanala ordezkatzen duen objektu batera konektatzen da. *Channel* klasea abstrakturekin inplementatu da C++ lengoaian. *Channel* batean espezializazioa kable sinple bat edo ethernet switch bat izan daiteke. *Channel* espezializazio ugari inplementatuta ditu, hala nola, *CsmaChannel*, *PointToPointChannel* edo *WifiChannel*

- **NetDevice.** Konputagailua sare batera konektatzeko hardware dispositibo baten beharra dago, txartel periferikoa bezala ezagutzen dena. Txartel periferiko horrek funtzionalitate batzuk inplementatuak edukiko ditu, *NIC* edo *Network Interface Cards* deritzonak. Gaur egunen txartel hauek konputagailuek barneratuta dituzte. Dispositibo hauek, hardwarea kontrolatzen duen softwarerik gabe ez dute funtzionatzen. NS-3-n, *NetDevice* abstrakzioak dispositibo fisiko eta softwarea bezala jokatuko du. *NetDevice*-a *node*-an instalatuta egongo da, simulazioan *chanell*-en zehar beste *node*-ekin komunikatzeko. Mundu errealean bezala, *node* bat *channel* anitzetara konektatzen bada, *NetDevice* ugariak beharko ditu. C++-n errepresentazio abstraktua, *NetDevice* klasearen bidez izanen

da. *NetDevice*-k, *node* eta *channel*-eko konexioak maneiatzeko dituen metodoak eskainiko ditu. Hauek espezializatu daitezke *channel*-etan bezala. Adibidez, *CsmaNetDevice*, *PointToPointNetDevice* eta *WifiNetDevice*.

- **Topology Helpers.** Mundu errealean NIC-ez hornitutako konputagailuak aurkitzen dira, NS-3-n *NetDevice*-ak *node*-i lotutako zaizkie. Sare handiak simulatu nahi badira, *NetDevice*, *node* eta *channel*-en artean lotura ugari egin beharko dira. Lan astun hau errazago egiteko, simulagailuak *Helper* tresna eskaintzen du. Bere baiten funtzio hauek ditu besteak beste: *NetDevice*-ak eta *node*-ak lotu, *NetDevices*-ak *channel*-ak lotu edo helbideen asignazioa.

Sisteman trebetasuna lortzeko, lehen konpilatutako *example* moduluko adibideez baliatuko gara. Simulagailuaren direktorioaren barnean aurkitzen den *example* direktorioan daude. Simulagailua garatzen duen talde partaideek, simulagailuaren funtzionamendu egokia izateko jada inplementatuta dauden modulu eta programak ez moldatze gomendatzen dute. Honetarako *scratch* direktorioak eskaintzen da, simulagailu direktorioaren barnean.

scratch direktorioan bertan, *example* direktorioaren barnean dagoen *first.cc* adibidea kopia dezakegu, honen izena aldatuz gero konpilatzean errorerik ez emateko. Guk *script*-en bat sortu nahi badugu, *scratch* direktorioan inplementatuko dugu.

Behin oinarrizko kontzeptuak azalduta eta *scratch* direktorioan lan egin behar dela ikusita, simulagailuaren moduluak erabiltzeko eta modu sakonago batean aztertu nahi badira, ondorego estekatik pasa [NS-3-Tutorial-First.cc, 2014] eta *first.cc* adibidea sakonago aztertu. Beste adibide gehiago aztertu nahi badira, *first.cc* adibidearen ondoren beste batzuk eskura daude.

A.3. Moduluen sorrera

NS-3-n simulatzean simulatu nahi den aplikazioa inplementatua ez egotea gerta daiteke edo simulagailuak eskaintzen dituen tresnekin nahiko ez izatea gure simulazioa aurrera eramateko. Kasu hauetan gure eskaeretara moldatutako modulu baten sorrera egitea aukera dago. Gure nahien araberako aplikazioa sor daiteke. Behin sistema ongi instalatuta dagoela, ondorengo pausoak eman beharko dira modulu berri bat sortzeko:

A.3.1. Moduluen diseinuaren ezagutza

Modulu guztiak simulagailu erro direktorioko *src* direktorioan aurkitzen dira. Direktorioaren izena moduluaren berdina izango da. Adibidez, *spectrum* modulua hemen aurki daiteke:

```
/src/spectrum
```

Nahiz eta modulu guztietan direktorio guztiak ez agertu, modulu bakoitzaren direktorioak honako egitura edukiko du:

```
src/
└── module-name/
    ├── bindings/
    ├── doc/
    ├── examples/
    │   └── wscript
    ├── helper/
    ├── model/
    ├── test/
    │   └── examples-to-run.py
    └── wscript
```

A.3.2. Moduluen oinarrizko txantiloiaren sorrera

src direktorioan *Python* programa bat eskura dago moduluaren txantiloia sortzeko.

```
src/create-module.py
```

Modulu berriaren izena parametro bezala pasa behar zaio script-ari. Demagun gure kasurako *sfdandconsensus* dela, beraz:

```
src/create-module.py sfdandconsensus
```

sfdandconsensus direktorioaren barnean honako fitxategi eta direktorioak edukiko ditugu:

```
doc examples  helper  model  test  wscript
```

A.3.3. Fitxategien konpilazioa eta exekuzioa

Hurrengo pausoa moduluaren pertsonalizazioa izango da. Demagun gure modulu berriak, beste moduluen beharra behar dituela, adibidez, internet protokoloak, point-to-point... Hau egiteko, sortutako modulu berriaren direktorioan aurkitzen den *wscript* fitxategian ondorengo moldaketa egin beharko da:

Pertsonalizatu baino lehen honakoa aurkituko dugu:

```
def build(bld):
  module = bld.create_ns3_module('sfdandconsensus',['core'])
```

eta pertsonalizatu ondoren:

```
def build(bld):
  module = bld.create_ns3_module('sfdandconsensus',['internet', 'applications', 'netanim',
        'network', 'core', 'config-store', 'point-to-point', 'point-to-point-layout','csma'] )
```

Behin gure modulua eraikita dagoela, garatuko ditugun fitxategiak sortzen hasiko gara. Direktorioen izenek dioten bezala, *example* direktorioan modulua exekutatuko duen script nagusia edo nagusiak aurkituko dira. *Helper*-ean berriz, moduluaren aplikazioa errazago eta automatikoago instalatzeko programak aurkituko dira. *test* karpetan, egin den programa ondo funtzionatzen duen ala ez jakiteko egiten diren test programak aurkituko dira, hauen sorrera aukerazkoa izango da. Azken *model* direktorioan, guk garatutako moduluaren funtsa egongo da, proiektu honetan adibidez, *Consensus* algoritmoa, hutsegite detektorea eta beste klase batzuk.

Behin fitxategiak sortu eta garatu direla, konpilazioa eta hauen exekuzioa egiteko konfigurazio fitxategia berriz moldatu beharko da, *header* eta *source* fitxategiak erantsiz. Hau, moduluaren erro direktorioan aurkitzen den *wscript* fitxategian egingo da, aurreko konfigurazioa bezala. *module*, *helper* eta *test* direktorioek gordetzen dituzten fitxategiak sartu behar dira. Kasu honetan:

- **Source**

```
module.source = [
    'model/propose.cc',
    'model/routingSingleton.cc',
    'model/nodeInfoSingleton.cc',
    'model/statusChanger.cc',
    'helper/propose-helper.cc',
    'model/failureDetector.cc',
    'model/matrix.cc',
    'model/operators.cc',
    'model/tpHeader.cc',
    'helper/failureDetector-helper.cc',
    'model/bcHeader.cc'
    ]
```

- **Header**

```
headers = bld(features='ns3header')
headers.module = 'sfdandconsensus'
headers.source = [
```

```
        'model/propose.h',
        'model/routingSingleton.h',
        'model/nodeInfoSingleton.h',
        'model/statusChanger.h',
        'helper/propose-helper.h',
        'model/failureDetector.h',
        'model/matrix.h',
        'model/tpHeader.h',
        'helper/failureDetector-helper.h',
        'model/bcHeader.h'
        ]
```

Script nagusiaren konpilazioa ere beharrezkoa izango da eta horretarako konfigurazio batzuk egin behar dira. Kasu honetan, *example* direktorioaren barnean dagoen *wscript* fitxategiaren gainean. Honako emaitza lortuko genuke:

```
def build(bld):
  obj = bld.create_ns3_program('sfdandconsensus-example', ['sfdandconsensus'])
  obj.source = 'sfdandconsensus-example.cc'
```

create_ns3_program() programaren lehenbiziko parametroak sortutako script-aren izen berbera du. Bigarrenak, exekutatu nahi den programa zein moduluaz dependitzen duen adierazten du, gure kasuan *sfdandconsensus*.

Behin konfigurazio fitxategi guztiak pertsonalizatu direla, dena ongi dagoela bermatzeko simulagailua birkonpilatuko da:

```
./waf -d debug configure --enable-sudo --enable-examples  --enable-tests
./waf
./waf --run scratch-simulator
```

Gogoratu, behin *waf* fitxategiaren konfigurazioa burutu dela, konfigurazio komandoa berriz exekutatzea ez dela beharrezkoa. Konpilatzeko bigarren komandoarekin aski da eta nahi izanez gero, hirugarrenak komandoak, lehenengo konpilatu eta jarraian programa exekutatuko du.

OHARRA:

Moduluaren kodea aurretik idatzita badago, lehenengo *src/create-module.py* komandoaren exekuzioarekin moduluaren sorrera egin behar da. Ondoren, gure fitxategiak dagozkien direktoriora kopiatu eta konfigurazio fitxategiak birkonfiguratu beharko dira. Behin dena konfiguratuta dagoela, goran aurkeztutako komandoak exekutatuz programaren konpilazioa eta exekuzioa egin daiteke.

A.4. Debugging

Sistema bat garatzen goazen heinean, programazio arazo aunitz aurki ditzakegu. Arazo hauek ebasteko *Logging* edo parametroen inprimatzea eta *debug* tresnak erabiltzen dira. Garaturiko aplikazioan zehar nahi ditugun *log*-ak edo mezuak inprima ditzakegu, parametroen balioa, erreferentzi puntuak... Hau oso baliagarria izango zaigu simulazioaren portaerak ezagutzeko.

Script edo moduluen klase guztietan *log*-a egiteko, garatutako *script* edo klasearen kodeketa hasieran izena eman behar zaio. Adibidez:

```
NS_LOG_COMPONENT_DEFINE ("sfdandconsensus");
```

Kasu honetan, sfdandconsensus programa exekutatzean bertan dauden *log* mezuak ikusi nahi baditugu, exekutatzean "sfdandconsensus" parametro sartu beharko da.

```
export NS_LOG="sfdandconsensus" && ./waf --run=sfdandconsensus
```

NS-3 simulagailuak klase aunitz ditu eta inplementatutako klase bakoitzean *logging*-erako definizio bat egiten da. Klase konkretu bateko *logging*-a egiteko eta honi atxikitutako izena ezagutzen ez badugu, ondorengo komandoaren bitartez etiketa lortuko dugu:

```
find . -name '*.cc' | xargs grep NS_LOG_COMPONENT_DEFINE
```

Zehaztasun gehiagoz bilatu nahi badugu, demagun *point-to-point* klaseko *logging* izena:

```
find . -name '*.cc' | xargs grep NS_LOG_COMPONENT_DEFINE | grep -i point
```

Logging egiteko maila desberdinak daude. Pantailaratzeko gure nahiaren arabera maila bat edo beste aukeratuko da. Ondorengo zazpi mailen artean aukera daiteke:

- NS_LOG_ERROR ⇒ Log error mezuak.
- NS_LOG_WARN ⇒ Log warning mezuak.
- NS_LOG_DEBUG ⇒ Log mezu arraroak, ad-hoc debugging mezuak.
- NS_LOG_INFO ⇒ Log programaren egoeraren informazioa.
- NS_LOG_FUNCTION ⇒ Log funtzio deientzat.
- NS_LOG_LOGIC ⇒ Log funtzioaren nondik norakoaren deskribapena.
- NS_LOG_ALL ⇒ Log mezu guztiak.

Gure aplikazioan zehar dauden *log*-ak inprimatzeko, ondorengo komandoa exekutatu beharko da.

Demagun, exekuzioan zehar sfdandconsensus aplikazioaren *log*-ean informazioa mezuak ikusi nahi ditugula, beraz:

```
export NS_LOG=sfdandconsensus=info && ./waf --run=sfdandconsensus
```

Maila desberdinak exekuta daitezke aldi berean:

```
export NS_LOG=sfdandconsensus=info|prefix_func && ./waf --run=sfdandconsensus
```

Programaren exekuzioko *log* guzti guztiak ikusi nahi baditugu, guk sortutako moduluarenak eta simulagailuak eskaintzen dituenak, *all_level* aukera dugu.

```
export NS_LOG=*=all_level && ./waf --run=sfdandconsensus > sfdandconsensus.out 2>&1
```

$*$-rekin *log* mezu mota guztiak aukeratuko dira eta *all_level* aukerarekin *logging* maila guztiak inprimatuko ditu. Mezuak ugari agertuko direla eta, irakurketa errazagoa izateko fitxategi batera birbideratuko da (> *sfdandconsensus.out 2>&1*).

B ERANSKINA

Terminologia

Proiektua dokumentatzeko garaian, batzuetan jatorrizko lanean agertzen diren terminoak euskaratzea zaila da. Eranskin honetan euskaraturiko hitzak agertuko dira, jatorrizko eta itzulpen terminoak. Ondorengo hiru mailetan sailkatu dira

Jatorrizko hizkuntzan mantendutakoak : B.1 taulan jatorrizko hizkuntzan mantendutako diren terminoak daude.

host	crash	send-omission	receive-omission
general-omission	Validity	Agreement	Termination
Byzantine	SmartCard	Timing attack	Buffering attack
TrustedPals	InConnected	OutConnected	Connectivity
Propose	Consensus	script	well-connected
peer-to-peer	Self-stabilitation	broadcast	Integrity
heartbeat	Reliable-broadcast	InConnectivity	InConnectedness

B.1 Taula: Ingelesezko terminoak

Euskaraturiko terminoak : B.2 taulan euskaratutako terminologia agertzen da.

Kasu bereziak : dokumentazioan zehar bi hizkuntzetan erabiltzen diren terminoen kasu berezi batzuk badira.

- *Omission - Omisioa.* Unean tratatzen ari den ingurunearen arabera bata edo bestea erabiliko da. Omisioen kontzeptu orokorra erabiltzen bada, euskaratutako terminoaz idatziko da. Berriz, omisio konkretu bateri erreferentzia egiten bazaio, adibidez *send-omission*, ingelesa erabiliko da.

Ingelesez	Euskaraz
Trusted system	Sistema fidagarria
Untrusted system	Sistema ez fidagarria
Uniform Consensus	Adostasun uniformea
Omission	Omisio
Faulty	Okerra
Failure detector	Hutsegite detektatzailea
Fault-tolerance	Hutsegite-tolerantzia

B.2 Taula: Euskarazko terminoak

- *Consensus - Adostasuna.* Printzipioz beti euskaratuko den termino bat da, baina salbuespenak badira. *Consensus* algoritmoari erreferentzia egiten bazaio ingeles terminoa erabiliko da, algoritmoaren izen propioa delako.
- *Failure detector - Hutsegite detektatzailea. Consensus*-en kasu berdina da. Algoritmoari erreferentzia egiten zaionean ingeles terminoa erabiliko da, euskarazkoa bestela.

C ERANSKINA

Kontsultatutako erreferentziak

Network Simulator 3-ren azterketa burutzeko eta galdera askoren erantzunak lortzeko eskuera eduki ditudan estekak:

- NS-3 Simulagailuaren web orri ofiziala [NS-3-Consortium, 2014]
- Tutoriala eta erabilpen gida [NS-3-Tutorial, 2014]
- Klaseen eta moduluen dokumentazioa [NS-3-Doxygen, 2014]
- NS-3 erabiltzaileen forua [NS-3-Forum, 2014]

PlanetLab-en informazioa eskuratzeko:

- PlanetLab web orri ofiziala [PlanetLab, 2014]

OMNet++en eta INET-en informazioa eskuratzeko:

- OMNet++ web orri ofiziala [OMNet++, 2014]
- INET web orri ofiziala [INET, 2014]

OPNET-en informazioa eskuratzeko:

- OPNET web orri ofiziala [OPNET, 2014]
- OPNET wikipedian [OPNETWiki, 2014]

GTNets-en informazioa eskuratzeko:

- GTNets web orri ofiziala [GTNetS, 2014]

GST edo Self-Stabilitation kontzeptuak ulertzeko erabilitako informazioa:

- Self-Stabilitation wikipediako informazioa [Self-Stabilitation, 2014]
- “Self-stabilizing systems in spite of distributed control” Dijkstra, Edsger W. [Dijkstra, 2014]

Reliable-Broadcast informazio gehigarria:

- Reliable-Broadcast wikipediako informazioa [Reliable-Broadcast, 2014]
- Azalpen Gardenkiak [Alvisi, 2014]

Garapena egiteko erreferentziak:

- C++ lengoaiarako dokumentazioa [C++, 2014]
- C++ lengoaiako artikuluak [Code-Project, 2014]
- Edozein garapen kontsultarako forua [Stackoverflow, 2014]
- Tex inguruko kontsultarako forua [Stackexchange, 2014]
- LaTeX wikibook [Wikibook, 2014]