Implementació i validació de mecanismes per a l'intercanvi consistent d'informació entre nodes d'un sistema encastat distribuït basat en HaRTES

Maties Melià Galmés, Alberto Ballesteros
i
Julián Proenza Arenas
Departament de Ciències Matemàtiques i Informàtica
Universitat de les Illes Balears

Avui en dia i de cada vegada més s'està fent palesa la necessitat de poder utilitzar Sistemes Encastats Distribuïts (DES) en aplicacions crítiques que s'executen en entorns dinàmics, on aquests sistemes han de treballar de manera continua, ser flexibles o adaptables i altament fiables. El paradigma de comunicació *Flexible Time Triggered* (FTT) permet desenvolupar aquest tipus de sistemes proveint la flexibilitat necessària en les seves comunicacions. A més, FTT proveeix el suport necessari per complir amb els requisits de temps real. En canvi, FTT no proveeix els mecanismes necessaris per a la tolerància a fallades per tal de garantir una elevada fiabilitat en els sistemes.

Hard Real-Time Ethernet Switching (HaRTES) és una implementació d'FTT sobre *Switched Ethernet* utilitzada en el projecte FT4FTT com a base per construir un sistema altament fiable. En el projecte s'han d'aglutinat diferents mecanismes per a la tolerància a fallades arribant a dissenyar una topologia en forma d'estrella replicada per *Switched Ethernet*, que proporciona el grau de fiabilitat desitjat.

El present projecte consisteix en el desenvolupament de dues propostes que s'han fet al projecte FT4FTT per modificar HaRTES i així incrementar la seva tolerància a fallades. La primera proposta que s'ha implementat i verificat és de G. Rodriguez-Navas et al. i és un protocol anomenat *Total Order Publish/Subscribe* (TOPS), el qual millora HaRTES amb un servei de difusió fiable amb ordre total dels missatges síncrons (periòdics). La segona proposta que s'ha implementat i verificat, en aquest cas només parcialment, és de S. Derasevic et al.. Aquesta proposta es basa en l'anterior protocol, el qual modifica i amplia. Concretament, la proposta consisteix en un protocol de votació on nodes rèplica poden votar de manera consistent el resultat del còmput amb els missatges que s'intercanvien.

Paraules clau: Sistemes Encastats Distribuïts, Sistemes Adaptables, Garantia de Funcionament, Tolerància a Fallades, Xarxes de Comunicació per l'Automatització, Protocol Ethernet, Protocol FTT, Sistemes de Temps Real

1. INTRODUCCIÓ

Un sistema encastat (*embedded system*) és un sistema informàtic concebut amb el propòsit de controlar un altre sistema. La reducció dels costos en la fabricació de semiconductors ha estat decisiva per estendre aquesta tecnologia en un gran nombre de sectors com l'automoció, l'aviònica, la domòtica i les telecomunicacions, per esmentar només alguns que tenen gran importància. Encara més, en molts de sectors s'han convertit en elements essencials.

En general, els sistemes encastats solen haver d'assumir limitacions pel que es refereix a recursos energètics, pes i mida. A més, el seu disseny els ha de fer capaços de realitzar les seves tasques en un temps acotat, és a dir, són sistemes de temps real [H. Kopetz 1997], el que fa necessari l'ús de polítiques de planificació temporal específiques per garantir-ho [A. Burns, i A. Wellings 2009]. Encara més, aquests sistemes solen haver de ser altament fiables, ja que sovint controlen aplicacions crítiques. Per tal d'aconseguir aquest propòsit és necessari utilitzar diferents mecanismes de tolerància a fallades.

Freqüentment, els sistemes encastats es solen distribuir en diversos nodes que intercanvien informació mitjançant una xarxa de comunicacions, per això s'anomenen Sistemes Encastats Distribuïts, en anglès *Distributed Embedded Systems* (DES). Típicament, els entorns en els que aquests sistemes han de funcionar solen ser ben definits i coneguts, per tant, les condicions de funcionament són conegudes d'antuvi. Això ha afavorit el desenvolupament d'estratègies de planificació temporal estàtiques, sovint específiques per al problema en qüestió, per complir amb els requisits de temps real.

D'altra banda, actualment els DES es volen començar a introduir en entorns dinàmics on les condicions poden canviar en qualsevol moment de manera imprevisible. Per fer front a aquesta circumstància ha aparegut el concepte de *sistemes encastats adaptatius*. El principal repte d'aquests sistemes és que han de ser capaços de reaccionar als canvis en l'entorn sempre mantenint el seu rendiment en els nivells adequats. Per tant, aquests sistemes han de ser flexibles, de temps real i fiables en els diferents nivells de la seva arquitectura, el que inclou tant el sistema operatiu com la xarxa de comunicacions.

Per poder garantir flexibilitat i a la vegada la resposta en temps real en aquests sistemes és necessari que la xarxa doni suport tant a comunicacions disparades per esdeveniments o asíncrones (*event-triggered*) com a comunicacions periòdiques o síncrones (*time-triggered*) [H. Kopetz 1997] [L. Almeida, P. Pedreiras, i J. A. G. Fonseca 2002] i, a més, ha de proveir mecanismes per poder gestionar els canvis dinàmics en els requisits de les comunicacions. Però, de moment, no hi ha cap tecnologia de xarxa d'us generalitzat que tengui totes aquestes característiques. La majoria de les tecnologies de xarxa actuals només donen suport o bé a comunicacions disparades per esdeveniments o a disparades per temps. Actualment, FlexRay [FlexRay 2005] és una important tecnologia que dona suport als dos paradigmes de comunicació, però no permet gestionar els canvis en els requisits de les comunicacions.

Aquest treball ha estat recolzat pel projecte DPI2011-22992 atorgat pel *Ministerio de Economía y Competitividad* espanyol, amb fons FEDER.

Màster Universitari en Tecnologies de la Informació

Flexible Time Triggered (FTT) [L. Almeida, P. Pedreiras, i J. A. G. Fonseca 2002] és una proposta de caire acadèmic que combina mecanismes de comunicació que s'activen tant per esdeveniments com per temps. A més, FTT és flexible, ja que pot canviar els paràmetres de comunicació mentre està en funcionament per tal de complir amb les demandes canviants de trànsit sense que la resposta en temps real es vegi compromesa [P. Pedreiras, P.Gai, L. Almeida, i G. C. Buttazzo 2005]. Les comunicacions estan dirigides per un node central anomenat mestre (*master*), el qual implementa les polítiques de planificació temporal. La resta dels nodes que formen part de la xarxa s'anomenen esclaus (*slaves*) i són els que implementen les aplicacions habituals dels DES.

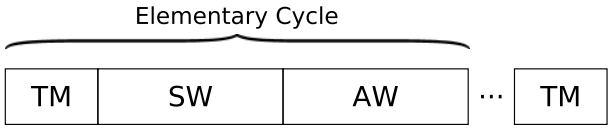

Fig. 1. Estructura de l'*Elementary Cycle*.

Per ser més precisos, FTT és un protocol de tres fases que divideix el temps de comunicació en ranures temporals de durada fixa anomenades *Elementary Cycles* (EC) (Figura 1). Cada EC és activat pel mestre enviant un missatge a tots els esclaus (*broadcast*) anomenat *Trigger Message* (TM), el qual transporta informació de control sobre quins missatges haurien de ser enviats i per qui (*EC-schedule*) en el present EC. La resta de l'EC es divideix en dues finestres temporals successives: la *synchronous window* (SW) on els esclaus transmeten els missatges periòdics i l'*asynchronous window* (AW) on transmeten els missatges aperiòdics.

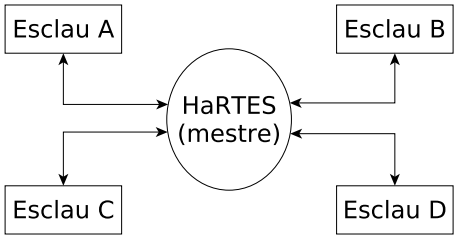

Fig. 2. Arquitectura HaRTES

El paradigma FTT ha estat adaptat a bastants tipus de xarxes. En un primer moment va ser desenvolupat per a *Controller Area Network* (CAN), però s'ha adaptat a *Ethernet* i *Switched Ethernet* [R. Marau, L. Almeida, i P. Pedreiras 2006], en els quals actualment hi ha un gran interès a causa del seu elevat ample de banda i del baix cost dels seus components. FTT sobre *Switched Ethernet* (FTT-SE) es compon de N nodes interconnectats per un commutador *Ethernet* i una implementació específica d'FTT-SE és *Hard Real-Time Ethernet Switch* (HaRTES) [R. G. V. dos Santos 2010], on el mestre està integrat en el commutador *Ethernet*, al qual cadascun dels esclaus està connectat gràcies a un enllaç *full-duplex* (Figura 2).

En general, a FTT els atributs per garantir una elevada fiabilitat s'han desenvolupat menys que els atributs per garantir la resposta en temps real, per això el present projecte consisteix en desenvolupar algunes de les propostes que s'han fet per proveir a FTT-SE de mecanismes per a la tolerància a fallades.

Respecte al desenvolupament realitzat, aquest s'ha dut a terme sobre HaRTES i forma part del projecte *Fault Tolerance for Flexible Time-Triggered Ethernet* (FT4FTT) [UIB 2015]. L'objectiu principal d'FT4FTT és desenvolupar els mecanismes de tolerància a fallades que siguin necessaris i integrar-los per aconseguir una infraestructura de comunicació altament fiable i flexible basada en FTT-SE.

Actualment, dins el marc del projecte FT4FTT ja s'han realitzat algunes millores en el commutador de HaRTES per poder facilitar la posterior tolerància a les fallades en els nodes de la xarxa. Les millores han consistit en desenvolupar alguns mecanismes per a la contenció d'errors per impedir la seva propagació des d'un node defectuós a la resta del sistema [A. Ballesteros, D. Gessner, J. Proenza, M. Barranco, i P. Pedreiras 2013].

Efectivament, ser capaç de tolerar fallades en els nodes és fonamental per tal d'aconseguir una elevada fiabilitat en un DES. Un dels mètodes més coneguts per aconseguir aquesta tolerància és la replicació dels nodes; a la qual el mateix programa és executat en múltiples nodes, anomenats rèpliques, de forma que si una d'aquestes rèpliques falla, la votació entre els resultats obtinguts pel còmput de totes les rèpliques permet emmascarar la fallada. A FT4FTT s'utilitza replicació de nodes, en concret la que s'anomena replicació activa, que es caracteritza perquè les diferents rèpliques d'un node executen al mateix temps un codi idèntic (sense diversitat de disseny).

S'ha de tenir en compte que la utilització de replicació activa necessita ser gestionada adientment, ja que és imprescindible garantir la consistència, és a dir, tots els nodes rèplica que no hagin fallat han de generar els mateixos valors de sortida. Pot semblar trivial aconseguir que totes les rèpliques produeixin sortides consistents. Malauradament, aquest no és el cas. Aquest problema s'anomena *replica non-determinism* i es pot deure a vàries causes, com la limitació de l'abstracció del món real per part dels nodes (si cada rèplica té el seu propi sensor per mesurar un valor, aquest sensors poden donar valors lleugerament diferents fins i tot funcionant correctament), la impossibilitat d'arribar a un acord continu i perfecte entre ells (cadascun d'ells pot saber coses abans que els altres i és necessari un cert temps perquè la informació estigui disponible per a tots per igual, gràcies al intercanvi de missatges entre els nodes) i/o la falta de coordinació entre aquests [S. Poledna 1996].

A fi de donar solució al *replica non-determinism* s'han plantejat diferents mètodes com l'*external replica determinism enforcement* [S. Poledna 1996]. Aquest mètode està enfocat a combatre la limitació de l'abstracció del món real per part dels nodes i la impossibilitat d'arribar a un acord continu i perfecte entre el nodes. Una de les tècniques per combatre la impossibilitat d'arribar a un acord continu i perfecte entre els nodes és garantir la plena consistència (*full consistency*) quan les rèpliques intercanvien missatges, és a dir, quan una rèplica envia un missatge a les altres, o bé és rebut per totes o per cap.

Pel que fa al conjunt de nodes rèplica, aquest no és més que un grup de nodes, així anem a explicar a continuació com es fa la comunicació entre nodes a FTT-SE. Per a la comunicació en grup FTT-SE té un sistema bàsic de publicació/subscripció (*publish/subscribe*) de serveis amb un control d'admissió centralitzat. Això és, els nodes esclaus es relacionen entre ells mitjançant uns canals de comunicació virtuals (*streams*), el node que genera la informació notifica al mestre la creació d'un canal, que una vegada creat serà el que utilitzarà per transmetre els missatges. El mestre notifica a la resta d'esclaus la creació del nou canal. Els esclaus que ho desitgin es poden afegir al canal per rebre els missatges

notificant-ho al mestre. El mestre és l'encarregat de mantenir la informació dels grups de publicació/subscripció, on s'associa el canal de comunicació amb la direcció física dels esclaus i el seu rol, per això qualsevol canvi s'ha de notificar al mestre. Cal dir, FTT-SE només permet un *publisher* per canal de comunicació. Cada canal pot tenir varis *subscribers*.

Tot i amb això, el sistema de publicació/subscripció d'FTT-SE no és suficient per garantir la plena consistència (per exemple no es garanteix que no hi hagi nodes que no rebin un missatge que sí han rebut altres nodes), i per tal de respondre a això, s'ha proposat un nou protocol de consistència anomenat *Total Order Publish/Subscribe* (TOPS). Aquest garanteix tant la consistència com el compliment de l'ordre total (tots els nodes reben els missatges en el mateix ordre) en les transmissions de missatges síncrons en les comunicacions publicació/subscripció [G. Rodriguez-Navas, i J. Proenza 2013]. TOPS és el protocol que s'ha desenvolupat en el present projecte i aquest desenvolupament és el que es descriu principalment en la present documentació.

No obstant això, TOPS té el problema de que basta que un node no hagi rebut un missatge perquè tots els altres el rebutgin per tal de mantenir la consistència. En canvi a sistemes amb nodes replicats que voten els resultats obtinguts, és millor que una majoria de nodes hagin rebut un missatge i puguin votar amb ell, que no que tots els nodes el rebutgin perquè hi ha un node que no l'ha rebut. Per aquesta raó S. Derasevic et al. proposa un nou protocol anomenat *Consistent Replicated Voting* (CRV), que realitza alguns canvis a TOPS orientats a incrementar el nombre de missatges que els nodes poden considerar com a ben rebuts a l'hora de votar. Afegeix un sistema de retransmissions pels missatges. A més, la proposta elimina el requisit de la plena consistència de TOPS per a l'acceptació d'un missatge afegint un algorisme especialment dissenyat per missatges que s'intercanvien per votar damunt ells. Amb aquest algoritme les rèpliques poden determinar de manera consistent quins missatges poden ser acceptats i quines rèpliques poden votar amb aquests. L'algoritme s'anomena *Voting Set-Up Algorithm* (VSUA). Aquest es basa en que és suficient que la majoria de les rèpliques hagin intercanviat de manera consistent un número suficient de missatges [S. Derasevic, M. Barranco, i J. Proenza 2014]. En el present projecte el sistema de retransmissions no s'ha desenvolupat, però després d'haver estat desenvolupat TOPS, s'ha modificat per incloure l'VSUA i s'ha desenvolupat. Els canvis a TOPS s'han fet de manera que és possible seleccionar ja sigui TOPS o l'VSUA abans de compilar el *software* que s'executa sobre el mestre i els esclaus.

Finalment, tornant al context global del projecte FT4FTT, cal dir que el FT4FTT inclou més mecanismes per a la tolerància a fallades, dedicats a fer front a fallades en el commutador o el mestre, per exemple s'ha proposat el *flexible time-triggered replicated star* (FTTRS) [D. Gessner, J. Proenza, M. Barranco, i L. Almeida 2013] on el commutador HaRTES es replicat. En el present projecte això no té gran importància i no en detallarem el seu funcionament.

La present documentació està estructurada en cinc seccions. La primera és la present introducció on hem introduït les bases del present projecte i en que consistirà. La segona descriu en general les fases del projecte, la metodologia seguida per realitzar-lo i les tecnologies utilitzades. La tercera explica en més detalls el protocol TOPS i com s'ha desenvolupat pas a pas. La quarta secció es dedica a les modificacions de TOPS per incloure l'VSUA i com s'ha desenvolupat. Finalment, la darrera secció dóna una explicació sobre el possible treball futur i algunes conclusions.

2. FASES I METODOLOGIA

Com s'ha dit, aquest projecte ha consistit en el desenvolupament de dues propostes per augmentar la fiabilitat d'FTT-SE en la implementació especifica HaRTES. Principalment, la realització del projecte es pot dividir en quatre grans fases. La primera fase ha estat dur a terme l'estudi dels fonaments d'FTT-SE i familiaritzar-se amb el prototip disponible. La segona ha implicat l'anàlisi de la proposta TOPS i com ha de ser desenvolupada en el prototip i el propi desenvolupament. La tercera com la segona, ha estat dur a terme l'anàlisi de la proposta de S. Derasevic et al. i com es podria desenvolupar en el desenvolupament actual, després d'haver estat desenvolupat TOPS, ja que l'amplia i modifica en alguns aspectes, i el propi desenvolupament. Finalment, s'ha elaborat la present documentació.

Com era d'esperar, la segona i tercera fases són les més importants i són les que es descriuen en aquest document. Com s'ha dit abans, aquestes fases han implicat el desenvolupament d'un programari. De la mateixa manera que altres projectes, la realització ha implicat una especificació, una implementació i una validació.

En relació a l'especificació s'ha de dir que les dues propostes preexistents que varen prendre com a punts de partida són especificacions, però alguns aspectes no hi són tractats profundament i en ocasions s'han pres durant el present projecte algunes decisions de disseny, en relació a l'estructura dels missatges i les estructures de dades utilitzades, per tal d'abordar el desenvolupament de les propostes en el prototip. Les més importants d'aquestes decisions es descriuen en les seves corresponents seccions.

Amb referència a la implementació i validació, s'ha fet seguint una estratègia iterativa i incremental. On cada iteració està completament lligada a un fase dels protocols i s'han fet seguint el seu ordre natural, ja que en cada fase del protocol és necessari el treball previ. Això també es pot veure reflectit en cadascuna de les següents seccions i subseccions d'aquest document.

Concretament, la implementació s'ha fet sempre tenint en compte que aquest programari forma part d'un sistema de temps real crític. Les estructures de dades utilitzades són simples i limitades en espai per evitar el temps extra i incert que necessita el sistema operatiu quan s'utilitza memòria dinàmica. Una altra qüestió que s'ha tingut en compte és evitar l'ús d'algoritmes recursius que també utilitzen memòria dinàmica. Tot i això cal assenyalar una excepció, en la implementació de l'algoritme VSUA s'ha utilitzat memòria dinàmica en algunes estructures de dades, ja que la mida d'aquestes estructures varia amb molta freqüència i, a més, no són estructures massa grans.

Com s'ha dit abans, el desenvolupament d'aquest projecte s'ha fet seguint una estratègia iterativa i incremental. Això significa que cada tros de codi ha estat provat en cada fase de la implementació, sempre verificant a fons el contingut dels missatges intercanviats amb analitzadors de paquets i comprovant el contingut de les estructures de dades. Però només hi ha una subsecció dedicada a descriure la verificació duta a terme, després de l'explicació de tota la implementació de cada proposta. Cal dir que la verificació del desenvolupament de la proposta de S. Derasevic et al. només d'ha fet de manera parcial, quedant com a treball futur la seva verificació completa.

En particular, la verificació de les dues propostes s'ha fet amb algunes aplicacions de capa superior basades en fils d'execució (*threads*) (Els nodes esclaus executen una aplicació amb un fil d'execució, on reben i envien missatges, o dos, un per a l'enviament i l'altre per a la recepció dels missatges). A banda d'això, per a la verificació de TOPS, també s'han realitzat proves on hem injectat a propòsit errors en la recepció dels missatges, per comprovar que la implementació funciona com cal també en presència de fallades.

Finalment cal afegir que la verificació de les dues propostes s'ha fet localment, això és, utilitzant un sol pc, el mateix en què s'ha

realitzat la implementació, usant un programari que crea un commutador virtual per interconnectar els processos que s'executarien en els diferents nodes. A més, en el cas de TOPS, s'ha utilitzat una plataforma *hardware* específica per provar el prototip que consisteix en alguns ordinadors de propòsit general interconnectats, on cada equip és un node mestre o esclau.

3. IMPLEMENTACIÓ I VALIDACIÓ DE TOPS

Després de la descripció en termes generals de les fases i metodologia seguida, aquesta secció se centra en l'explicació del desenvolupament de la proposta TOPS, i es divideix en vuit subseccions.

La primera subsecció està dedicada a explicar la proposta TOPS. En la segona s'explica l'estat inicial en què s'ha trobat el prototip HaRTES que hem utilitzat com a punt de partida i les diferències amb el protocol FTT. Aquesta subsecció vol centrar-se, en termes generals, sobre com HaRTES ha estat modificat per complir amb la proposta TOPS. Després d'això, les subseccions següents es reserven per explicar les diferents fases en què s'ha dividit la implementació de TOPS i finalment la seva validació.

3.1 Estudi de la proposta TOPS

El propòsit de TOPS és oferir un servei de comunicació per garantir *total order multicast/broadcast*, també conegut com *atomic broadcast* [X. Défago, A. Schiper, i P. Urbán. 2004], pels missatges síncrons d'FTT, és a dir, es pretén proporcionar plena consistència en els intercanvis de missatges síncrons. Això es pot aconseguir si es satisfan les següents propietats [X. Défago, A. Schiper, i P. Urbán. 2004]:

Validesa (*Validity*): Si un procés correcte difon un missatge *n*, llavors finalment lliura *n*.

Acord (*Agreement*): Si un procés correcte lliura un missatge *n*, llavors tots els processos correctes finalment lliuren *n*.

Integritat (*Integrity*): Per a qualsevol missatge *n*, cada procés lliura *n* com a màxim una vegada, i només si *n* va ser prèviament difós pel remitent (*sender*).

Ordre total (*Total order*): Si dos processos correctes *p* i *q* ambdós lliuren els missatges *n* i *n'*, llavors *p* lliura *n* abans de *n'* si i només si *q* lliura *n* abans de *n'*.

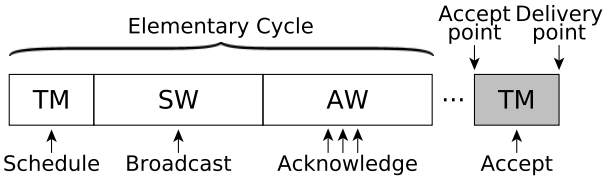

Fig. 3. Distribució de les fases de TOPS sobre l'estructura de l'EC.

Principalment, la proposta TOPS consisteix a modificar el protocol de comunicació de tres fases FTT que s'explica en la introducció d'aquest document. Amb aquesta proposta es vol aconseguir les propietats acord i integritat amb un protocol de confirmació executat en quatre fases. Les quatre fases s'efectuen entre la transmissió de dos TMs consecutius i garanteixen acord i integritat d'EC a EC. La figura 3 mostra com s'inclouen les fases de TOPS sobre l'estructura de l'EC.

Pel que fa a les altres propietats es pot dir que FTT garanteix un ordenament dels missatges que és consistent (igual per a tots els nodes) a través de la xarxa gràcies a l'*EC-schedule*. Si TOPS ofereix acord, és a dir, els esclaus es veuen obligats a lliurar els mateixos missatges rebuts, llavors han de lliurar els missatges en el mateix ordre en què es van planificar a l'*EC-schedule*. En aquest cas, acord implica ordre total. Finalment, la propietat de validesa pot aconseguir-se amb la retransmissió dels missatges quan es produeix un error. TOPS proposa algunes solucions per dur-ho a terme, però no s'han desenvolupat al present projecte.

La descripció de les fases de TOPS és la següent:

La primera fase, *Schedule phase*: El mestre difon als esclaus el TM transportant l'*EC-schedule* de l'EC actual. Rebut aquest missatge, tots els esclaus saben quins canals estan actius en aquest EC.

La segona fase, *Broadcast phase*: Cada *publisher* d'un missatge planificat el difon durant el transcurs de la SW mentre el mestre roman passiu i el commutador realitza "guardat i reenviat"(*store and forward*). Però els *subscribers* encara no lliuren els missatges rebuts a la capa superior, esperen a la darrera fase, ja que la transmissió de qualsevol dels missatges pot ser afectada per fallades en qualque enllaç del canal i ser inconsistent.

La tercera fase, *Acknowledge phase*: En aquesta fase cada *subscriber* envia al mestre una notificació, ja sigui positiva (ACK) o negativa (NACK), si han rebut el missatge planificat o no, respectivament. Això succeïx durant l'AW mentre el mestre segueix passiu i el commutador segueix realitzant "guardat i reenviat". Els esclaus que no són *subscribers* de cap dels missatges planificats no envien cap tipus de notificació al mestre. Al final d'aquesta fase el mestre sap si l'emissió d'un missatge ha estat consistent entre els seus *subscribers* o no, aquest instant s'anomena *Accept point*. En aquest moment el mestre construeix un vector anomenat *EC-Status vector* (EC-SV), en el qual indica quins missatges es poden lliurar a les capes superiors. Un missatge es pot lliurar si i només si tots els seus *subscribers* han transmès una notificació positiva.

La quarta fase, *Accept phase*: El mestre envia l'EC-SV en el pròxim TM. Ara el TM porta l'*EC-schedule* de l'actual EC i l'EC-SV amb informació de l'EC que tot just ha acabat. Una vegada el TM es rebut pels esclaus, amb la informació de l'EC-SV els esclaus lliuren o no els missatges de l'anterior EC a les capes superiors. Aquest instant s'anomena *Delivery point*.

3.2 Estudi del prototip HaRTES i les seves modificacions

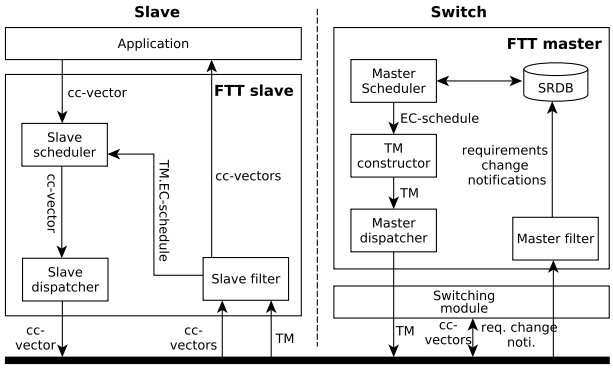

Fig. 4. Interior d'un esclau i del mestre HaRTES

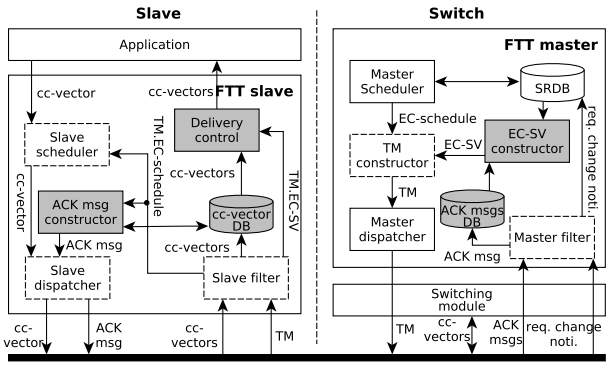

Fig. 5. Interior d'un esclau i del mestre HaRTES amb TOPS

La figura 4 mostra l'interior d'un esclau i del commutador HaRTES. A la figura els missatges difosos entre els diferents esclaus a través del commutador i que han de ser rebuts consistentment per tots els *subscribers* els hem anomenat cc-vectors. Això és terminologia N-Version Programming (NVP) [A. Avizienis 1985], terminologia típicament utilitzada a l'hora de descriure mecanismes per a la tolerància a fallades basats en replicació. En relació a l'esquema mostrat a la figura 4 cal dir que no és exhaustiu i només hi ha representades les parts amb les quals estem interessats:

El *master filter* i l'*slave filter* són els responsables de discriminar el tipus de missatge i d'extreure-ne la informació.

La *System's Requirements Database* (SRDB) conté tots els atributs dels canals de comunicació, que poden ser modificats pels esclaus, així com els paràmetres de funcionament del mestre.

El *master scheduler* és responsable de determinar el conjunt de missatges que han de ser transmesos en cada EC, depenent del contingut de l'SRDB, i construeix l'*EC-schedule*, el qual és enviat al *TM constructor* on el TM és muntat.

El *master dispatcher* i l'*slave dispatcher* són responsables de finalitzar el muntatge dels missatges i disparar la seva transmissió.

L'*slave scheduler* és responsable de planificar la transmissió dels missatges a l'esclau basant-se en l'*EC-schedule* rebut.

Finalment, el *switching module* del mestre és l'encarregat de les funcions pròpies del commutador *Ethernet*.

Per a plantejar la implementació de les fases de TOPS el codi de HaRTES ha estat inspeccionat. Quan ho hem fet ens hem adonat que a l'*slave dispatcher* la SW i l'AW no estan delimitades com a l'EC teòric del protocol FTT. Llavors, les finestres temporals s'han hagut de delimitar correctament. Aquest assumpte es discuteix en la subsecció 3.5 dedicada a la implementació de l'*acknowledge phase*.

En segon lloc, a HaRTES, com al protocol FTT, els esclaus lliuren els missatges immediatament tant prest com arriben a les capes superiors. Per tal de desenvolupar la segona fase de TOPS, s'ha construït una llibreria que inclou una estructura per emmagatzemar els missatges després de ser rebuts i mentre esperen a ser lliurats a l'aplicació. Pràcticament tota la implementació de la resta de les fases de TOPS en els esclaus depèn de la utilització d'aquesta llibreria, on s'ha anat modificant i implementant noves funcions a mesura que ha transcorregut el desenvolupament de TOPS.

Finalment, per a la implementació de la tercera i de la quarta fase de TOPS, cal definir un nou tipus de missatges per enviar les notificacions ACK i NACK al mestre. El qual les haurà de emmagatzemar, analitzar i construir l'EC-SV, que després inclourà en el següent TM. Això implica la modificació de l'estructura del TM i canviar la forma en què els esclaus el reben, ja que ara aquests l'hauran de comprovar per lliurar o no els missatges emmagatzemats en la *broadcast phase*.

Aquestes modificacions s'expliquen amb més detalls en les properes subseccions que tenen per títol la fase de TOPS implicada en cada cas. En la figura 5 es mostra com les modificacions afecten a l'esquema de HaRTES. Els rectangles blancs amb la vora puntejada representen components que s'han modificat parcialment i els rectangles grisos representen els nous components.

3.3 Implementació de l'*schedule phase*

A primera vista, per a la implementació d'aquesta fase no cal fer cap modificació a HaRTES, atès que la fase no es veu alterada pel protocol TOPS. Aquesta fase és igual en FTT que en TOPS i el procés de planificació dels missatges és el mateix. No obstant això, aquesta fase es veu afectada per la implementació de les últimes fases de TOPS, on és necessari fer més operacions que només el procés de planificació dels missatges.

A continuació s'exposa la implementació de la *broadcast phase* on comença la tasca d'implementació.

3.4 Implementació de la *broadcast phase*

Com s'ha dit abans, a HaRTES quan els missatges síncrons són rebuts pels esclaus són descodificats i lliurats immediatament a l'aplicació. En la implementació de TOPS s'ha impedit que els missatges síncrons siguin descodificats i immediatament lliurats a l'aplicació. Ara, així com els missatges van arribant són emmagatzemats en una nova estructura anomenada *MSG_TO_ACCEPT*, és a dir, la *cc-vector DB* de la figura 5. Els missatges no són descodificats, només s'emmagatzemen a l'espera de ser acceptats pel mestre i lliurats en l'*accept phase*. En l'*accept phase* ens aprofitarem de la funció original de HaRTES per lliurar els missatges a l'aplicació, és a dir, la mateixa funció que s'utilitzava anteriorment en aquest punt.

L'estructura MSG_TO_ACCEPT està inclosa en una llibreria que, com s'ha dit abans, s'utilitza en la resta de les fases que tenen lloc en l'esclau, on s'ha anat modificant al llarg de la implementació de TOPS les seves funcionalitats relacionades amb l'emmagatzemament, acceptació i lliurament dels missatges síncrons. A continuació es descriu la implementació de l'*aknowledge phase*.

3.5 Implementació de l'*aknowledge phase*

Abans de desenvolupar l'*aknowledge phase*, és necessari adequar la transmissió dels missatges síncrons i asíncrons a l'*slave dispatcher*. Al HaRTES original els missatges eren transmesos sense tenir en compte la durada de les finestres SW i AW, això significa que es podien transmetre tant missatges síncrons com asíncrons durant tot l'EC exceptuant durant el temps del TM. Això es feia utilitzant un sol bucle per transmetre tots els missatges, ja siguin síncrons o asíncrons.

Ara per implementar TOPS es necessita que els missatges síncrons siguin transmesos durant la seva finestra temporal, la SW, i que la durada d'aquesta sigui respectada explícitament. Per aconseguir això, el bucle original per a l'enviament dels missatges s'ha dividit en dos bucles. Un bucle per a l'enviament dels missatges síncrons i l'altre per a l'enviament dels asíncrons. Els dos bucles s'han col·locat un darrere l'altre, primer el bucle dels missatges síncrons seguit del bucle per a missatges asíncrons. A causa d'això s'han hagut de desenvolupar les funcions per recuperar els missatges síncrons i asíncrons per separat, ja que originalment només

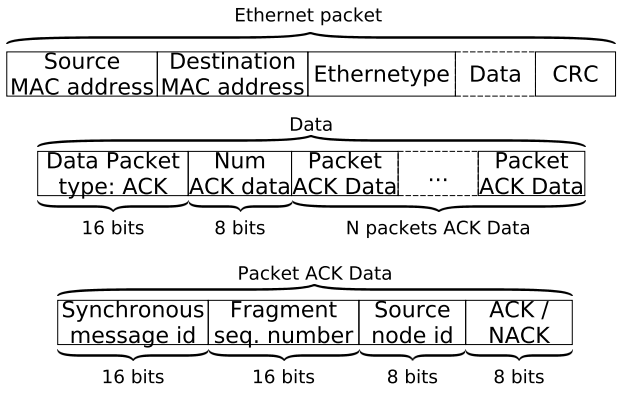

Fig. 6. Estructura dels missatges ACK

existia una funció per recuperar missatges, on no es distingia el tipus de missatge.

En realitat, aquestes modificacions no són suficients a fi que només els missatges síncrons siguin enviats durant la SW i es respecti de manera explícita la durada de la finestra, ja que normalment el temps necessari per transmetre els missatges síncrons és inferior a la durada establerta per a la finestra. Per tant, després del primer bucle per a l'enviament dels missatges síncrons s'ha afegit una espera per completar el temps restant de la SW (l'espera equivaldria al temps establert per la SW menys el temps que realment l'esclau ha necessitat per enviar els missatges síncrons). L'espera s'ha desenvolupat de dues maneres, mitjançant la interrupció del rellotge i espera activa. Després d'algunes proves s'ha pogut concloure que la interrupció del rellotge és la millor manera per desenvolupar l'espera, com ja es preveia, atès que l'espera activa és menys precisa que la interrupció del rellotge.

Una vegada els missatges síncrons només poden ser enviats durant la seva finestra temporal, la SW, l'*acknowledge phase* ha de ser col·locada immediatament després d'aquesta consumint part del temps dedicat a l'AW. La implementació de l'*acknowledge phase* consisteix a recórrer la llista de missatges que s'haurien d'haver rebut per l'esclau. Mentre es recorre aquesta llista es compara amb el contingut de l'estructura MSG_TO_ACCEPT, que són els missatges rebuts. Simultàniament, es va construint un missatge d'ACK que contendrà la llista de missatges rebuts i no-rebuts. A HaRTES la llista de missatges que s'haurien d'haver rebut per l'esclau està emmagatzemada en una estructura que pertany a l'*slave scheduler*, que s'omple amb la informació que porta l'*EC-schedule* (això és una mica diferent de la figura 5, ja que aquesta és una simplificació de la realitat). Cal remarcar que, construir un sol missatge ACK amb una llista de missatges rebuts i no-rebuts és diferent al que proposa TOPS, en aquest cada esclau transmet un ACK o NACK independent per missatge. A l'apartat 3.5.1 es descriu l'estructura d'aquest nou tipus de missatge.

Una vegada que s'ha definit el nou missatge ACK i l'*slave dispatcher* és capaç de transmetre-lo, s'ha d'habilitar la recepció d'aquests missatges al *master filter*. Al mestre la informació d'aquests missatges s'emmagatzema en una nova estructura anomenada db_acks, és a dir, l'*ACK msgs DB* de la figura 5. En aquesta nova estructura es guarda l'identificador de l'esclau que ha transmès el missatge ACK. A més, també s'hi guarda la llista dels missatges que ha rebut i no-rebut l'esclau transmissor de l'ACK, aquests missatges són identificats per l'identificador de l'esclau que els va enviar originalment. Això és exactament la informació que porta el missatge ACK. En el HaRTES original el mestre no necessitava conèixer l'identificador d'un esclau quan havia enviat un missatge. Ara aquesta qüestió és necessària per tal de poder guardar aquesta informació quan el mestre rep un ACK. Per això, s'ha hagut de construir una funció per obtenir l'identificador del node amb la seva adreça MAC, aquesta equivalència es pot trobar a l'SRDB.

Seguidament es descriu l'estructura dels missatges ACK i després es continua amb l'explicació de la implementació de l'*Accept point*

3.5.1 *Estructura dels missatges ACK.* La figura 6 mostra l'estructura d'un missatge ACK. En aquest tipus de missatges, el camp *Data* del paquet *ethernet* està compost per l'identificador del tipus de dades, en aquest cas ACK, el nombre de camps anomenats *Paket ACK Data*, que a la vegada és el nombre de missatges síncrons que s'haurien d'haver rebut per l'esclau, i una llista d'igual nombre de camps *Paket ACK Data*. A cada *Paket ACK Data* es identificat el missatge síncron amb el seu identificador, el número de fragment i l'identificador del node que l'hauria d'haver enviat. Finalment, a cada *Paket ACK Data* es indicat si aquest missatge s'ha rebut o no amb el caràcter 1 o 0, respectivament. (Com es pot veure, la presència del nombre de fragment en el camp *Paket ACK Data* implica que els missatges originals als quals els hi correspon l'ACK o NACK poden ser dividits amb fragments, per això cada fragment tendria el seu propi ACK o NACK per part de cada receptor. En el nostre cas seria suficient incloure només l'identificador del missatge síncron, ja que els missatges utilitzats no necessiten fragmentar-se, però el nombre de fragment s'ha inclòs per completar-lo).

Com ja s'ha esmentat, a la proposta TOPS cada esclau transmet un ACK o NACK per cada missatge síncron que havia de rebre, però en la implementació realitzada només s'envia un sol missatge que inclou la llista de tots els ACK-NACKs que ha d'enviar l'esclau. Això s'ha fet així, ja que té l'avantatge de que el temps per transmetre només un missatge amb tota la informació és menor que la transmissió de diversos missatges.

A continuació l'explicació de la implementació de l'*accept point* i l'*acceptance phase*.

3.6 Implementació de l'*accept point* i l'*acceptance phase*

Quan arriba l'*accept point* l'AW ha acabat i un nou EC està començant. Tota la informació que s'ha transmès en els missatges ACK està emmagatzemada a la db_acks del mestre i és el seu torn per analitzar-la. La db_acks emmagatzema la llista d'identificadors dels diferents missatges síncrons amb el seu número de fragment i l'identificador de l'esclau que els produí. Cada missatge síncron identificat a la db_acks té associades dues llistes. Una és la llista dels identificadors dels esclaus que han enviat un ACK positiu i l'altra és la llista dels que l'han enviat negatiu.

Amb tota la informació estructurada de la manera exposada abans, durant l'*acceptance phase* l'*EC-SV constructor* recorre l'*EC-schedule* de l'EC anterior, mentre cerca cada missatge a la db_acks i consulta a l'SRDB el nombre de *subscribers* de cada missatge. Si el missatge es trobat a la db_acks i el número d'esclaus que han enviat un ACK positiu és igual al número de *subscribers* trobat a l'SRDB (i el nombre d'esclaus que han enviat NACK és zero), el *EC-SV constructor* assenyala a l'EC-SV que el missatge pot ser lliurat pels esclaus a l'aplicació. En cas contrari, assenyala que el missatge no pot ser lliurat. Cal afegir que no és necessari en aquest punt comprovar que els esclaus que han enviat els ACK o NACK són certament *subscribers* dels missatges als quals han enviat confirmació, ja que a l'*aknowledge phase* el mestre a l'hora de recollir

la informació dels missatges ACK s'haurà assegurat de que això sigui així. En cas de que un esclau hagi enviat un ACK o NACK d'un missatge del qual no és *subscriber* aquest no s'haurà tingut en compte i s'haurà notificat un error.

A el pròxim apartat de la subsecció s'explica l'estructura de l'EC-SV i on està inclòs dins el TM.

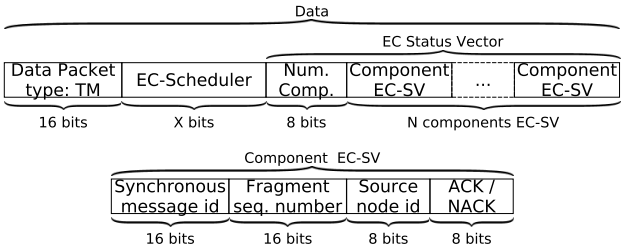

Fig. 7. Estructura dels missatges TM amb l'EC-SV

3.6.1 *Estructura de l'EC-Status vector.* La figura 7 mostra l'estructura d'un TM, que inclou l'EC-SV. Com es pot veure, l'estructura de l'EC-SV és la mateixa d'un missatge ACK inclosa en un altre missatge. No obstant això, en aquest cas el significat dels camps ACK/NACK no és el mateix. Ara, aquests camps signifiquen que el missatge síncron, identificat al component del vector, pot set lliurat (1) o no (0) per l'esclau cap a l'aplicació.

Una vegada l'EC-SV ha estat construït per *EC-SV constructor* és lliurat al *TM constructor*, que inclou el vector immediatament després de l'*EC-schedule*. Finalment, el missatge és entregat al *master dispatcher* per ser difós als esclaus, que examinaran el contingut de l'EC-SV en el *delivery point*, que s'explica en la següent subsecció.

3.7 Implementació del *delivery point*

A l'*slave filter*, una vegada que el contingut de l'*EC-schedule* s'ha descodificat i emmagatzemant, s'inicia el *delivery point*. La seva implementació consisteix a recórrer el contingut de l'EC-SV cercant els missatges a l'estructura MSG_TO_ACCEPT. Si l'esclau és *subscriber* d'un missatge inclòs a l'EC-SV i està marcat com que es pot lliurar, l'esclau cerca aquest missatge a MSG_TO_ACCEPT. Si el missatge es troba a l'estructura, es lliura a l'aplicació utilitzant les mateixes funcions que anteriorment ja s'utilitzaven a HaRTES. Si el missatge no es troba a l'estructura és un error.

Com s'ha dit, el sistema de retransmissions, que s'hauria d'activar quan un missatge síncron no s'ha rebut per un esclau, no s'ha desenvolupat. Amb la finalitat d'impedir que s'excedeixi la capacitat de l'estructura MSG_TO_ACCEPT, els missatges després del *delivery point* s'eliminen quan han passat un nombre d'ECs. Aquest nombre es pot configurar abans de compilar el *software* en una variable anomenada *MAX_AGE*, que està a la llibreria de l'estructura MSG_TO_ACCEPT. L'acumulació de missatges a MSG_TO_ACCEPT només pot succeir quan a l'EC-SV es senyalitza un missatge com que no pot ser lliurat a l'aplicació, ja que alguns esclaus no l'han rebut. Però el present esclau sí ha rebut el missatge i no el pot lliurar.

Una vegada s'ha explicada la implementació de TOPS, en la següent subsecció s'explica com s'ha dut a terme la seva validació.

3.8 Validació de TOPS

Com s'exposa a la secció 2, la implementació i validació del present projecte s'ha fet seguint una estratègia iterativa i incremental, en aquest cas, significa que cada fase desenvolupada s'ha anat validant a mesura que s'anava programant. Les validacions s'han dut a terme tant a nivell local amb l'equip amb què s'ha desenvolupat el projecte, com utilitzant una plataforma *hardware*, específicament muntada per provar el prototipus, formada per ordinadors de propòsit general interconnectats.

Per dur a terme les proves s'han construït diferents aplicacions d'alt nivell per executar en els esclaus, on s'intercanvien missatges entre elles. Per validar el correcte funcionament de TOPS s'ha imprimint per consola el contingut de les estructures de dades construïdes i la informació que s'intercanvien les aplicacions. A més, amb l'analitzador de paquets *Wireshark* [Wireshark Foundation 2015] s'han capturat els paquets d'informació intercanviats, i així esbrinar si el seu contingut és el correcte.

Per a les proves a nivell local s'ha utilitzat l'aplicació *Vde_switch* [Virtualsquare Team 2015] capaç de crear un commutador virtual amb diferents interfícies de xarxa. En aquestes interfícies virtuals s'han interconnectat els processos que s'executen en el mestre i els esclaus. A HaRTES, com ja s'ha dit anteriorment, el *software* del commutador ja està inclòs en el mestre, però és necessària la presència d'interfícies de xarxa físiques o, en aquest cas, virtuals per poder-se comunicar amb els esclaus.

Una vegada fetes les proves a nivell local, la plataforma *hardware* utilitzada per provar el prototipus està formada per: un PC amb un processador multinucli i dues targetes *Ethernet* Intel 350-T4 de quatre ports, que realitzarà les funcions de commutador, i varis *barebones* Jetway JBC373F38-525-B, que seran els esclaus, això és un *hardware* específic per a sistemes encastats de xarxa que tenen un processador Atom i quatre targetes *Ethernet* estàndards (NICs) d'una interfície cada una.

Pel que fa al *software*, sobre els esclaus i el commutador s'executa un sistema operatiu (SO) GNU/Linux. El commutador sobre el SO executa el *switching module* i el mestre HaRTES. Anàlogament, els esclaus executen l'esclau HaRTES i les aplicacions d'alt nivell per a proves.

Una vegada feta tota la implementació de TOPS i fetes les pertinents proves parcials per poder comprovar el seu funcionament, s'ha realitzat una prova final on s'avalua el comportament de TOPS. En aquesta prova s'avalua tant quan els esclaus reben els missatges com quan els esclaus no els reben. A causa de l'absència de fallades d'aquest caire s'han hagut de provocar a propòsit.

La prova es realitza amb tres esclaus i un mestre, on un esclau és *producer* i els altres dos són *subscribers* del mateix canal de comunicació del *producer*. L'aplicació desenvolupada per a l'esclau *producer* es limita a indefinidament comptar fins a 255 i tornar a començar la compta. A cada iteració el *producer* envia al canal de comunicació el valor de la compta. Els altres dos esclaus, els *subscribers*, executen la mateixa aplicació, que consisteix en imprimir per consola el valor rebut pel canal de comunicació.

Com s'ha dit, amb aquesta prova pretenem veure si es respecta el comportament de TOPS. Si un dels esclaus *subscriber* no rep un dels missatges i l'altre sí, el que rebi el missatge no l'ha de lliurar a l'aplicació. Per altra banda, només si els dos *subscribers* reben el missatge llavors els dos l'han de lliurar. Si això es compleix llavors es respectarà la consistència plena. Per poder provocar la pèrdua de missatges, com s'ha dit, s'ha hagut de fer a propòsit, de tal manera que el que es fa és no guardar el missatge rebut a l'estructura MSG_TO_ACCEPT.

Concretament, s'ha realitzat un test que es basa en que cada aplicació que s'executa en els esclaus *subscribers* té un identificador, exactament 2 i 3, mentre el *producer* té l'1. Aquest test s'ubica a

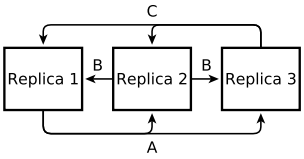

Fig. 8. Intercanvi de missatges entre 3 nodes

	A	B	C
Replica 1	T/F	T/F	T/F
Replica 2	T/F	T/F	T/F
Replica 3	T/F	T/F	T/F

Fig. 9. Contingut *MS-vector* amb 3 nodes

l'*slave filter* i pretén provar tots els casos possibles. Aquest es repeteix cada quatre ECs de la següent manera: Al primer EC els dos esclaus *subscriber*s guarden el missatge rebut a MSG_TO_ACCEPT. Al segon EC el *subscriber* que té l'identificador 2 guarda el missatge a l'estructura i el que té l'identificador 3 no. Al tercer EC es fa el mateix que a l'anterior, però ara és el 3 el que guarda el missatge i el 2 no. Finalment, al quart EC cap dels dos *subscriber*s guarda el missatge.

Un vegada realitzat aquest test podem dir que ha estat satisfactori, ja que els esclaus es comporten de la manera esperada. Els *subscriber*s lliuren la informació a l'aplicació quan toca. A més, s'han capturat els paquets que es transmeten entre els esclaus i el mestre i el contingut és el correcte.

Una vegada explicat com s'ha dut a terme la validació de TOPS podem seguir amb l'explicació del desenvolupament de la proposta de S. Derasevic et al., el protocol CRV.

4. IMPLEMENTACIÓ I VALIDACIÓ DEL CRV

A la present secció es descriu la implementació i validació del protocol CRV. desenvolupat sobre l'anterior desenvolupament descrit. Aquesta secció està estructurada en quatre subseccions.

A la primera subsecció es descriu en que consisteix el protocol CRV. A la segona s'expliquen els canvis que s'han de dur a terme sobre el prototipus després d'haver implementat TOPS. A la tercera es detalla la implementació duta a terme. Finalment, a la darrera subsecció s'explica com s'ha verificat el protocol CRV.

4.1 Estudi del protocol CRV

Com s'exposa a la secció 1 la proposta de S. Derasevic et al. consisteix a modificar i ampliar el protocol TOPS amb l'objectiu de solucionar el problema del *replica non-determinism*, fins i tot amb presència de fallades a la xarxa o als nodes. Problema ocasionat a l'hora d'utilitzar replicació activa. Concretament, els nodes rèplica realitzen execucions parcials del mateix *software* en paral·lel, això són segments. Per cada segment els nodes rèplica obtenen una sortida, també anomenada cc-vector (a partir d'ara en endavant els missatges síncrons seran anomenats així), que cada rèplica intercanvia amb la resta per poder votar i així obtenir un cc-vector de consens, amb què poder continuar executant el següent segment.

A la proposta es determinen alguns temes referents al sistema de retransmissions per quan algun cc-vector es perd. A banda d'això, proposa l'algoritme VSUA que elimina el requisit de la plena consistència de TOPS. Com ja s'ha dit, el sistema de retransmissions no s'ha desenvolupat en el present projecte, així que aquest tema no serà detallat en la present documentació.

Respecte a les altres propostes que es fan en el protocol CRV, en aquest es proposa la substitució de l'EC-SV de TOPS per un nou vector anomenat *Messages Status vector* (MS-Vector). En aquest vector s'inclou informació referent a quins nodes rèplica han enviat el seu cc-vector i quins nodes rèplica han confirmat cada cc-vector. Llavors el vector es tractat a cada esclau al *delivery point* amb l'algoritme VSUA. Amb aquest algoritme els esclaus decideixen quins cc-vectors poden ser utilitzats per votar i quins nodes rèplica ho poden fer amb aquests cc-vectors. Finalment, lliuren la decisió presa a l'aplicació. A la proposta el contingut del vector està explicat a mode d'exemple.

A l'exemple proposa un sistema format per tres esclaus on cada un és *subscriber* dels altres dos i a la vegada *producer*, de tal manera cada node rèplica envia un cc-vector als altres dos, com a la figura 8. El contingut de l'MS-Vector seria com el que es pot veure a la figura 9. En aquest vector el mestre posa vertader (T) a cada fila si ha rebut el cc-vector del corresponent node rèplica *producer* i si aquest mateix node ha rebut el cc-vector dels altres nodes rèplica dels quals és *subscriber*, això significa que l'esclau ha enviat una notificació positiva per cada cc-vector en concret.

Pel que fa l'algoritme VSUA, el seu objectiu es garantir que la votació de l'actual segment es realitzi amb la majoria de cc-vectors possibles i a la vegada que aquesta votació es realitzi en la majoria de nodes rèplica, per així tenir el màxim d'entrades, cc-vectors, per al pròxim segment. Sent $majoria = \lfloor X/2 \rfloor + 1$, on X és el nombre de cc-vectors o nodes rèplica.

L'algoritme VSUA és un algoritme genèric que serveix per a qualsevol nombre de nodes rèplica. L'algoritme amb l'MS-Vector genera totes les possibles combinacions per poder votar i ho fa de la següent manera. Primer determina el nombre màxim de cc-vectors útils amb què cada node rèplica vota. Això és, si un cc-vector no està marcat com a vertader en la casella del seu *producer*, no pot ser que els *subscribers* hagin enviat una notificació positiva a la recepció d'aquest cc-vector. Per això, l'algoritme marca a fals les caselles dels esclaus *subscribers* d'aquest cc-vector. Una vegada VSUA té aquests cc-vectors, considerats com a majoria, genera totes les possibles combinacions de nodes rèplica que han notificat positivament la recepció d'aquests cc-vectors. Després l'VSUA escull la millor combinació, és a dir, la que implica la majoria de nodes rèplica. Si no es troba la majoria de nodes rèplica per aquesta majoria de cc-vectors, el nombre de cc-vectors a considerar decreix en 1. El procés es repeteix fins a trobar la solució o el nombre de cc-vectors ja no pot ser considerat com a majoria, això és $M < \lfloor S/2 \rfloor + 1$, sent M el nombre de cc-vectors i S el nombre de nodes rèplica.

L'algoritme VSUA a la proposta ha estat simulat amb JAVA i aquest troba la millor solució, és a dir, la que implica el major nombre de nodes rèplica i cc-vectors. Aquest ho fa prioritzant el nombre de cc-vectors davant el nombre de nodes rèplica. Gràcies a això, segons la proposta, en algunes situacions l'algoritme aconsegueix permetre una fallada més de les que podrien ser suportades amb X nodes rèplica, però el nombre de fallades que normalment suporta són $X - majoria$.

A continuació s'exposen les modificacions que s'han de dur a terme sobre l'actual prototipus, el prototipus HaRTES amb la implementació de TOPS ja realitzada, per tal d'implementar la present proposta.

4.2 Estudi de les modificacions de HaRTES amb TOPS

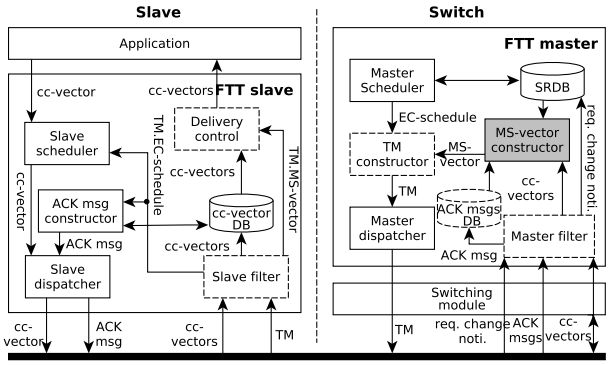

Fig. 10. Interior d'un esclau i del mestre HaRTES amb VSUA

A la subsecció 4.1 una de les primeres qüestions que s'exposa és que l'EC-SV ha de ser substituït per l'MS-Vector. A més, s'explica que aquest vector inclourà informació referent a quins nodes rèplica han enviat el seu cc-vector i quins nodes rèplica han confirmat cada cc-vector. Això significa que el mestre no solament ha de saber quins nodes rèplica *subscribers* han rebut els cc-vectors (ja garantit per TOPS) sinó que també ha de saber quins nodes rèplica *producers* han enviat els seus cc-vectors. Com podem veure a les figures 4 i 5, els cc-vectors en el mestre només arriben al *switching module*. Això fa necessari fer que aquests cc-vectors siguin lliurats al *master filter*. A més, per emmagatzemar la informació de l'arribada dels cc-vectors es modificarà la base de dades d'ACKs per incloure-la.

Respecte a l'MS-Vector, com hem dit, ha de substituir a l'EC-SV, llavors serà necessari substituir l'*EC-SV constructor,* el qual ja no serà necessari, pel constructor d'aquest nou vector, l'*MS-Vector constructor*. A més, s'haurà de modificar el *TM constructor* per tal d'incloure l'MS-Vector en el TM.

Una altra qüestió important que implica el canvi de l'EC-SV per l'MS-Vector és que s'haurà de modificar l'*slave filter* per tal de descodificar l'MS-Vector i passar-lo al *delivery control*. A més, el *delivery control* ja no serà tan senzill com a TOPS, ja que els esclaus hauran d'executar l'algoritme VSUA per tal de decidir si lliurar o no els cc-vectors a l'aplicació.

A la figura 10 podem veure com queden sobre l'esquema les modificacions acabades d'exposar. Ben igual que en les anteriors figures, els rectangles blancs amb la vora puntejada representen components que s'han modificat parcialment i els rectangles grisos representen els nous components, en aquest cas respecte a 5.

Seguidament s'exposa en més detall la implementació realitzada.

4.3 Implementació del protocol del CRV

En primer lloc, s'ha habilitat al *master filter* la recepció dels missatges síncrons, o cc-vectors. Això és tan senzill gràcies a que el mestre rep tots els cc-vectors encara que no vagin dirigits a ell, ja que el commutador *Ethernet* està inclòs en el propi mestre, però els cc-vectors no passen a les capes superiors i és necessari habilitar-ho.

En segon lloc, s'ha modificat l'estructura db_acks del mestre per indicar que un *producer* ha enviat un cc-vector. Com s'exposa a la subsecció 3.6, l'estructura db_acks és una llista d'identificadors de cc-vectors, on cada entrada té associada dues llistes d'identificadors de nodes *subscribers*. Una de les llistes té els identificadors dels esclaus que sí han notificat que han rebut el cc-vector i l'altra llista té els identificadors dels que han notificat que no l'han rebut. Llavors, a cada entrada de la llista de cc-vectors s'ha associat un nou camp. Camp que el mestre posa a vertader quan rep el cc-vector del *producer*. Per fer-ho, s'ha desenvolupat una funció que es crida des del *master filter* quan el mestre rep un cc-vector. La funció cerca a la db_acks l'identificador del cc-vector, si el troba, és indicat com a enviat pel *producer*. En cas contrari, el mestre afegeix una nova entrada a la db_acks, també indicant que el cc-vector ha estat enviat pel *producer*.

En tercer lloc, una vegada el mestre ja pot tenir tota la informació necessària per construir l'MS-Vector es pot desenvolupar l'*MS-Vector constructor* i substituir la inclusió en el TM de l'EC-SV pel nou vector. L'estructura de l'MS-Vector està detallada en el següent apartat de la present subsecció i es pot veure en la figura 11. Com avançament dir que el vector està format per tantes entrades com nombre de nodes rèplica participants en la comunicació multiplicat pel nombre de canals de comunicació que utilitzen. Tan nombre de rèpliques com canals ha de ser el mateix (Nº entrades MS-Vector $= N \times N$ sent N el nombre de nodes rèplica o el nombre de canals). A cada entrada el vector inclou l'identificador del cc-vector, és a dir, l'identificador del canal, i l'identificador del node que l'hauria d'haver enviat o rebut. Respecte a la seva construcció s'ha de dir que aquesta és similar a la de l'EC-SV, però amb certes peculiaritats, atès que el vector ha de ser tractat com una matriu per l'algoritme VSUA dels esclaus. Per això s'estableix un ordre entre les entrades del vector i així no tenir problemes a l'hora de tractar-lo.

Ben igual que en la construcció de l'EC-SV, per construir l'MS-Vector és necessari recorre l'*EC-schedule* de l'EC anterior mentre es cerca cada identificador dels cc-vectors a la db_acks. En aquest cas l'*EC-schedule* ha estat prèviament ordenat per identificador de cc-vector. Si l'identificador del cc-vector es troba a la db_acks, en el vector s'inclou una nova entrada indicant si el cc-vector ha estat o no enviat pel *producer*, segons el contingut de l'estructura db_acks. Després es recorre la llista de *subscribers* del cc-vector en qüestió obtinguda consultant l'SRDB. Mentre es recorre aquesta llista es va afegint una nova entrada a l'MS-Vector per *subscriber*. A cada entrada s'indica si els esclaus han enviat confirmació o no de la recepció del cc-vector, segons el contingut de la db_acks. En cas contrari, si l'identificador del cc-vector no es troba a la db_acks, a l'MS-Vector s'inclouen tantes entrades com *producer* i *subscribers*. A cada entrada s'indica que el cc-vector no ha estat ni enviat ni rebut per cap. Finalment, el vector és ordenat per identificador d'esclau. A més, si el nombre d'entrades del vector resultant és diferent al nombre de canals utilitzats multiplicat pel nombre de nodes rèplica, ho notifica com error.

En quart lloc, per quan el TM arriba l'*slave filter*, s'ha de substituir la separació de l'EC-SV del TM per la separació de l'MS-Vector. Fet això, l'MS-Vector és lliurat a l'algoritme VSUA, que es troba en una nova llibreria anomenada *S_FT4FTT_voting* que forma part del mòdul *delivery control*. A més de l'VSUA a la llibreria podem trobar funcions per a la interpretació de l'MS-Vector. Aquestes són utilitzades abans de començar l'algoritme VSUA pròpiament dit. A més d'això, hi podem trobar també funcions per calcular les combinacions sense repetició dels cc-vectors, les quals utilitza l'VSUA. Això s'ha desenvolupat de manera recursiva i iterativa i es pot escollir abans de compilar el *software* quin algoritme per calcular les combinacions es vol utilitzar. Cal dir que per desenvolupar les funcions que obtenen les combinacions ha estat necessari

afegir noves operacions en una llibreria de funcions matemàtiques, que anteriorment només formava part del mestre i s'ha convertit en global, del mestre i l'esclau.

Respecte a l'algoritme VSUA cal dir, una vegada ha finalitzat retorna 1 si ha trobat una combinació de nodes i cc-vectors satisfactòria i 0 en cas contrari. A més, l'algoritme deixa emmagatzemada la solució en una estructura. Per això s'han desenvolupat diferents funcions que consulten aquesta estructura. Una funció que respon afirmativament si el node rèplica en qüestió pot participar en la votació i una altra que retorna els identificadors dels cc-vectors escollits per a la votació.

Una altra qüestió que cal tenir en consideració és, l'algoritme VSUA no utilitza el nombre de fragment que identifica els cc-vectors. Això fa que per al seu correcte funcionament és necessari que tots els canals de comunicació tenguin un sol fragment. A més, si no fos així, es podrien construir MS-Vectors inadequats per a l'algoritme VSUA, amb identificadors de cc-vectors repetits.

Finalment, una vegada l'esclau ha executat l'VSUA, utilitzant les funcions presents a *S_FT4FTT_voting* aquest pot saber si pot participar en la votació del segment i obtenir els identificadors dels cc-vectors a utilitzar. Identificadors que utilitza per cercar els cc-vectors a l'estructura MSG_TO_ACCEPT, si els troba, els lliura a l'aplicació. En cas contrari seria un error. Per cercar els cc-vectors a l'estructura MSG_TO_ACCEPT s'ha hagut de desenvolupar una nova funció, a causa de la no consideració del nombre de fragment per part de l'VSUA, ja que no es té el nombre de fragment en el resultat de l'algoritme VSUA.

Com s'ha dit, en el següent apartat de la present subsecció es detalla l'estructura de l'MS-vector.

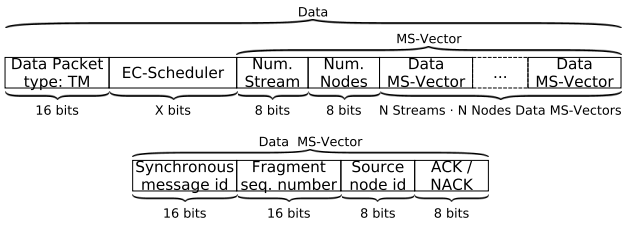

Fig. 11. Estructura dels missatges TM amb l'MS-Vector

4.3.1 *Estructura de l'MS-Vector*. Com s'ha dit, a la figura 11 trobem representada l'estructura del TM amb l'MS-Vector inclòs després de l'*EC-schedule*, que ara substitueix a l'EC-SV. Aquest es composa de dos camps. Un on hi ha el nombre de canals de comunicació i l'altre el nombre de nodes rèplica. A més de $N \times N$ components, anomenats *Data MS-Vector*. Sent N el nombre de canals de comunicació o el nombre de nodes. Com ja s'ha dit, aquest vector tendrà la forma adequada si el nombre de canals i nodes rèplica és igual.

Com es pot veure, cada *Data MS-Vector* té la mateixa forma que els components que formen l'EC-SV (figura 7), però ara el camp ACK/NACK indica que el node ha enviat (1) o no (0) el cc-vector, o que el node l'ha rebut (1) o no (0), segons si el node és *producer* o *subscriber* del canal de comunicació. Gràcies al doble ordenament que es realitza quan es construeix el vector, primer per canal i després per node, els N primers *Data MS-Vectors* corresponen a la informació referida als cc-vectors corresponents a un node, els $N+1$ fins a $2N$ correspon a un node diferent, els $2N+1$ fins a $3N$ a un altre, i així successivament. Això fa que el vector tingui la forma de matriu desitjada per ser tractada en l'algoritme VSUA.

Certament, vist aquest ordenament del vector i vist que cada *Data MS-Vector* té l'identificador del node i del canal de comunicació, un pot pensar que molta d'aquesta informació és redundant. S'ha fet així per ser més fàcil a l'hora construir-lo i tenir més coherència amb la resta d'estructures de missatge plantejades en el present projecte, per això també s'ha inclòs el nombre de fragment, encara que l'algoritme VSUA no el tingui en compte.

A continuació s'exposa com s'ha dut a terme la validació de l'algoritme de votació.

4.4 Validació parcial de la implementació del protocol CRV

A diferència de la validació de TOPS, en el present projecte només s'ha fet una validació parcial de la implementació de la proposta de S. Derasevic et al.. Aquesta validació només s'ha fet de manera local utilitzant les mateixes eines utilitzades per validar TOPS exposades en la subsecció 3.8. Concretament, S'ha validat el correcte contingut de l'estructura db_acks i s'ha constat que els TM amb l'MS-Vector eren construïts i intercanviats correctament. A més, s'han validat els dos algoritmes per a la generació de les combinacions sense repetició dels cc-vectors, el iteratiu i recursiu, i les funcions matemàtiques auxiliars. La validació d'aquests dos algoritmes i funcions auxiliars s'ha fet de manera aïllada a la resta del sistema, utilitzant valors numèrics que simularien ser el contingut de l'MS-Vector.

Així doncs, la validació de la implementació de l'algoritme VSUA pròpiament dit no s'hauria dut a terme. En realitat ha estat feta per A. Ballesteros et al. una vegada han desenvolupat el sistema de retransmissions per quan algun cc-vector es perd proposat en el protocol CRV i una vegada han integrat la nostra implementació amb la resta del sistema. L'explicació de la validació es pot consultar en el document *First Experimental Evaluation of the Consistent Replicated Voting in the Hard Real-time Ethernet Switching architecture* presentat en la vintena conferència ETFA de l'IEEE [S. Derasevic, M. Melià, A. Ballesteros, M. Barranco, i J. Proenza 2015].

5. CONCLUSIONS I TREBALL FUTUR

En el present document hem explicat com s'ha duit a terme la implementació i validació de dues propostes de mecanismes per a l'intercanvi consistent d'informació en un DES, les quals s'havien fet dins el projecte FT4FTT per modificar un sistema preexistent anomenat HaRTES i així incrementar la seva tolerància a fallades. Aquestes dues propostes són dos protocols anomenats TOPS i CRV.

TOPS millora HaRTES amb un servei de difusió fiable amb ordre total dels missatges síncrons. Gràcies a la planificació dels missatges síncrons per part del mestre HaRTES i al servei de publicació/subscripció amb control d'admissió centralitzat, l'ordre d'aquests missatges ja estava garantit però no la consistència plena, la qual aporta TOPS amb un nou protocol de confirmació de 4 fases basat en els serveis que ja aportava HaRTES. Això significa que amb el desenvolupament de TOPS sobre HaRTES ara tenim un servei de difusió fiable per als missatges síncrons entre els esclaus, de tal manera que les aplicacions que s'executen sobre els esclaus reben consistentment cada missatge que es difós si cap esclau ha fallat o bé no el reben, ja que és suficient que un esclau perdi un missatge perquè aquest sigui rebutjat pels altres (els que no han fallat) i a més tots els esclaus receptors reben aquests missatges en el mateix ordre.

La implementació de TOPS ha consistit en modificar alguns mòduls de HaRTES i en la construcció d'alguns mòduls nous. Concretament, a l'esclau s'han modificat els mòduls *slave filter*, *slave scheduler* i *slave dispatcher*; i s'han construït els mòduls *cc-vector DB* (també anomenat MSG_TO_ACCEPT), *ACK msg constructor* i *delivery control*. Al mestre s'han modificat els mòduls *master filter* i *TM constructor*; i s'han construït els mòduls *ACK msgs DB* (també anomenat db_acks) i *EC-SV constructor*. S'ha de destacar que la implementació de TOPS ha implicat la definició d'un nou tipus de missatge que envien els esclaus al mestre, l'ACK, que conté una llista de notificacions positives i negatives a la recepció dels missatges síncrons que havien de ser rebuts per part dels esclaus que eren *subscribers* d'aquest missatges. Això difereix de com s'havia plantejat a TOPS, al qual es proposava enviar un missatge ACK o NACK independent com a resposta a cada missatge síncron rebut o no rebut, respectivament. A més, la implementació de TOPS també ha implicat la modificació del TM on s'ha inclòs l'EC-SV, en el qual el mestre indica quins missatges síncrons els esclaus poden lliurar a l'aplicació.

La validació de TOPS ha consistit en realitzar diferents proves parcials a nivell local (utilitzant l'equip amb què s'ha realitzat la implementació) a mesura que s'ha anat implementant el protocol. Aquestes proves han consistit en la programació d'aplicacions pels esclaus que s'intercanvien missatges i en la interconnexió dels processos que s'executen sobre els esclaus i el mestre mitjançant un commutador virtual. Utilitzant això s'ha pogut comprovar el contingut dels missatges intercanviats mitjançant analitzadors de paquets i el contingut de les estructures de dades imprimint el seu contingut per consola. A més, s'ha realitzat una prova final utilitzant una plataforma *hardware* formada per diferents equips i muntada a propòsit per provar el prototipus. En aquesta prova s'ha realitzat un test on s'han injectat errors a propòsit per tal de comprovar el comportament de TOPS quan els esclaus perden els missatges. Llavors, s'ha pogut validar que TOPS es comporta com s'esperava, només quan tots els esclaus *subscribers* reben un missatge síncron, aquest es lliurat a l'aplicació per cada *subscriber*.

Una vegada vista la implementació i validació de TOPS, hem vist com ha estat implementat el protocol CRV. Aquest modifica TOPS i aporta l'algoritme VSUA per als esclaus. Aquest algoritme s'ha concebut vist que la manera en què TOPS aconsegueix la plena consistència (quan una rèplica envia un missatge a les altres, o bé és rebut per totes o per cap) és massa restrictiva a l'hora d'utilitzar replicació activa on els nodes voten els resultats obtinguts, ja que amb TOPS és suficient que un esclau no confirmi la recepció del seu missatge, encara que tots els altres l'hagin rebut correctament, perquè els que l'han rebut correctament no el puguin lliurar a l'aplicació. L'algoritme VSUA es basa en que és suficient que la majoria de nodes rèplica hagin intercanviat consistentment un nombre suficient de missatges, perquè puguin votar amb ells, fent que no sigui necessària la plena consistència. Això fa que s'aprofiti millor la redundància aportada per la utilització de replicació activa.

La implementació del protocol CRV ha implicat la modificació de la implementació de TOPS alterant alguns mòduls i substituintne d'altres. Concretament, a l'esclau s'han hagut de modificar els mòduls *slave filter* i *delivery control*. D'aquest darrer mòdul s'ha de destacar que és el que conté l'algoritme VSUA. Al mestre s'ha modificat els mòduls *master filter* on s'ha habilitat la recepció dels missatges (cc-vectors) que no van dirigits al mestre, *ACK msgs DB* i *TM constructor*; i s'ha substituït l'anterior mòdul *EC-SV constructor* de TOPS pel mòdul *MS-Vector constructor*. Cal dir que les modificacions i substitucions d'aquests mòduls han implicat la construcció de l'MS-Vector on el mestre inclou informació referent a quins nodes rèplica han enviat el seu cc-vector i quins nodes rèplica han confirmat cada cc-vector. Aquest MS-Vector és el que els esclaus utilitzen com entrada a l'algoritme VSUA per decidir quins nodes rèplica poden votar i amb quins cc-vectors ho han de fer.

La validació del protocol CRV només s'ha fet parcialment i a nivell local. En aquesta validació s'ha comprovat el correcte contingut de les estructures i que els missatges intercanviats estaven ben construïts. A més d'això, s'han validat els algoritmes per a la generació de combinacions de cc-vectors que utilitza l'algoritme VSUA. Finalment, la validació completa ha estat realitzada per A. Ballesteros et al. i en [S. Derasevic, M. Melià, A. Ballesteros, M. Barranco, i J. Proenza 2015] es pot veure com la implementació realitzada compleix els seus objectius.

Com a treball futur, en el marc del projecte FT4FTT continuaran treballant per avaluar el protocol TOPS, incloent una verificació formal mitjançant la tècnica de *model checking*, la qual permetrà comprovar de manera exhaustiva tots els possibles comportaments del model del sistema, i així poder verificar que en el nostre cas hem contemplat la totalitat dels casos. A més, es realitzarà l'anàlisi de fiabilitat de TOPS que consisteix en construir models estocàstics del sistema que permeten quantificar el nivell de fiabilitat aconseguit en el disseny. Com també es continuarà avaluant de manera exhaustiva i experimental el desenvolupament de la proposta de S. Derasevic et al. amb més rèpliques. A més de construir també un model formal per a la seva validació per *model checking*. Finalment dir, en el projecte FT4FTT continuaran treballant amb l'objectiu de construir un sistema complet altament fiable basat en FTT-SE.

AGRAÏMENTS

A família i amics pel seu suport incondicional.

REFERÈNCIES

A. AVIZIENIS. 1985. The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions on Software Engineering 11*, 12, 1491–1501.

A. BALLESTEROS, D. GESSNER, J. PROENZA, M. BARRANCO, I P. PEDREIRAS. 2013. Towards preventing error propagation in a real-time ethernet switch. In *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Cagliari, Italy, 1 – 4.

A. BURNS, I A. WELLINGS. 2009. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, Fourth ed. Addison-Wesley Educational Publishers Inc, USA.

D. GESSNER, J. PROENZA, M. BARRANCO, I L. ALMEIDA. 2013. Towards a Flexible Time-Triggered replicated star for ethernet. In *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Cagliari, Italy, 1 – 4.

FLEXRAY. 2005. FlexRay Communications System - Protocol Specification, Version 2.1.

G. RODRIGUEZ-NAVAS, I J. PROENZA. 2013. A proposal for flexible, real-time and consistent multicast in FTT/HaRTES switched ethernet. In *18th IEEE Conf. on Emerging Technologies & Factory Automation (ETFA)*. IEEE, Cagliari, Italy, 1 – 4.

H. KOPETZ. 1997. *Design Principles for Distributed Embedded Applications*, First ed. Kluwer Academic Publishers, Norwell, MA, USA.

L. ALMEIDA, P. PEDREIRAS, I J. A. G. FONSECA. 2002. The FTT-CAN protocol: why and how. *Industrial Electronics, IEEE Transactions 49*, 6, 1189 – 1201.

P. PEDREIRAS, P.GAI, L. ALMEIDA, I G. C. BUTTAZZO. 2005. FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *Industrial Informatics, IEEE Transactions 1*, 3, 162 – 172.

R. G. V. DOS SANTOS. 2010. Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications. Ph.D. thesis, Universidade de Aveiro.

R. MARAU, L. ALMEIDA, I P. PEDREIRAS. 2006. Enhancing real-time communication over COTS Ethernet switches. In *2006 IEEE International Workshop on Factory Communication Systems*. IEEE, Torino, Italy, 295 – 302.

S. DERASEVIC, M. BARRANCO, I J. PROENZA. 2014. Appropriate consistent replicated voting for increased reliability in a node replication scheme over FTT. In *2014 IEEE Emerging Technologies & Factory Automation (ETFA)*. IEEE, Barcelona, 1 – 4.

S. DERASEVIC, M. MELIÀ, A. BALLESTEROS, M. BARRANCO, I J. PROENZA. 2015. First Experimental Evaluation of the Consistent Replicated Voting in the Hard Real-time Ethernet Switching architecture. In *2015 IEEE Emerging Technologies & Factory Automation (ETFA)*. IEEE, Luxembourg.

S. POLEDNA. 1996. *Fault-tolerant real-time systems: The problem of replica determinism.*, First ed. Springer US.

UIB. 2015. Fault Tolerance for Flexible Time-Triggered project page. [Online]. http://srv.uib.es/ft4ftt/.

VIRTUALSQUARE TEAM. 2015. Virtual Distributed Ethernet. [Online]. http://vde.sourceforge.net/.

WIRESHARK FOUNDATION. 2015. Wireshark. [Online]. https://www.wireshark.org/.

X. DÉFAGO, A. SCHIPER, I P. URBÁN. 2004. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Computing Surveys 36*, 4, 372–421.