ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LES ILLES BALEARS

PROJECTE DE FINAL DE CARRERA

Estudi:

Enginyeria Tècnica de Telecomunicacions.
Especialitat Telemàtica

Títol:

Infraestructura d'injecció
de fallades per a Ethernet amb
funcions específiques per a FTT

Alumne: Andreu Adrover Mas

Directors : Alberto Ballesteros Varela
Julián Proenza Arenas

Data: Novembre de 2015

Fem constar que el projecte de final de carrera titulat *Infraestructura d'injecció de fallades per a Ethernet amb funcions especíques per a FTT*, ha estat realitzat, sota la direcció de Alberto Ballesteros Varela i Julián Proenza Arenas, per Andreu Adrover Mas. Així mateix declaram que el projecte està finalitzat i preparat per a la seva presentació pública.

Palma, Novembre de 2015

Signat: Andreu Adrover Mas
Projectista

Signat: Alberto Ballesteros Varela
Co-director del projecte final de carrera

Signat: Julián Proenza Arenas
Co-director del projecte final de carrera

Abstract

En la actualitat *Ethernet* és l'estàndard de comunicació per a xarxes locals més utilitzat en entorns domèstics i d'oficina, però no ha estat fins els darrers anys que s'ha començat a utilitzar en entorns industrials. Això es deu a que *Ethernet* no proporciona serveis que són necessaris en aquests entorns: predictibilitat, fiabilitat i adaptabilitat. El paradigma de comunicacions FTT és una solució que proporciona garanties de temps real i permet el canvi dinàmic de les característiques del tràfic. HaRTES és una implementació *switched Ethernet* d'FTT que defineix una arquitectura on un *switch* fet a mida gestiona les comunicacions dels nodes connectats. No obstant, HaRTES té una reduïda capacitat per tolerar les fallades que limita la seva fiabilitat.

El projecte d'investigació FT4FTT de la Universitat de les Illes Balears té com a objectiu dissenyar, avaluar, validar, implementar i verificar mecanismes de tolerància a fallades per a HaRTES. La fase de verificació ha de permetre comprovar que els mecanismes de tolerància a fallades estan dissenyats i implementats correctament i, a més, que combinats els uns amb els altres funcionen de manera correcta. Una de les tècniques més utilitzades per a la verificació i validació de mecanismes de tolerància a fallades és la injecció de fallades.

Aquest projecte de final de carrera consisteix en el disseny, implementació, verificació i validació d'una infraestructura d'injecció de fallades per a *Ethernet* que permetrà comprovar que els mecanismes de tolerància a fallades desenvolupats dins del projecte FT4FTT funcionen correctament. Aquesta infraestructura permet provocar errors en les trames *Ethernet* que viatgin per qualsevol dels enllaços del *switch*, i en els missatges FTT que aquelles contenen. A més, permet monitoritzar, registrar, processar i analitzar el tràfic resultant, per tal de conèixer la resposta del sistema en front als errors provocats.

Índex

1 Introducció....1
1.1 Entorn i motivació del projecte....1
1.2 Objectius del projecte....3
1.3 Tasques realitzades....4
1.4 Estructura del document....5

Part I. Fonaments i treballs anteriors 7

2 Fonaments de Flexible Time-Triggered(FTT)....9
2.1 Elementary Cycle (EC)....9
2.2 Missatges FTT....11

3 Hard Real-Time Ethernet Switching(HaRTES)....12

4 Tolerància a fallades per a HaRTES....15
4.1 The Slave Elementary Cycle Synchronization Mechanism (SECSM)....15
4.2 Flexible Time-Triggered Replicated Star for Ethernet (FTTRS)....17

5 Descripció del prototipus HaRTES....19

6 Injecció de fallades....20
6.1 Arquitectura general d'un entorn d'injecció de fallades....20
6.2 Tipus d'injectors de fallades....21

Part II. Disseny 23

7 Llistat de requisits....25
7.1 Requisits funcionals....25
7.2 Requisits no funcionals....25

8 Arquitectura de la infraestructura d'injecció de fallades....26
8.1 Estructura d'un mòdul injector de fallades....27
8.1.1 Interfície Ethernet virtual (TAP)....28
8.1.2 Mòdul d'Intercepció....28
8.1.3 Mòdul d'Anàlisi....28
Funcions per trames *Ethernet* en general....28
Funcions específiques per FTT (FTT_TYPE 0x8ff0)....28
8.1.4 Mòdul de Modificació....29
Tipus d'errors que es poden provocar....29
8.1.5 Mòdul de Transmissió....30
8.1.6 Fitxers de trames capturades....30
Dispositiu de captura de tràfic netAnalyzer B500G-RE....30
8.2 Prototipus HaRTES amb la infraestructura d'injecció de fallades....32

9 Descripció detallada del llenguatge de configuració....33
9.1 Arguments que identifiquen el dispositiu on s'executa l'injector....33
9.2 Arguments de configuració d'un mòdul injector de fallades....34

Part III. Implementació 39

10 Infraestructura d'injecció de fallades....41
10.1 Mòdul de control....41
10.2 Mòdul d'Injecció de fallades i captura de dades (MIFCD)....42

10.2.1 Estructura del programa principal....43
10.2.2 Configuració dels MIFCDs....43
10.2.3 Intercepció i transmissió de trames....43
10.2.4 Anàlisi i injecció de fallades....44
10.2.5 Fitxer final de trames rebudes i enviades....45
10.3 Mòdul de processament i anàlisi dades (MPAD)....45

Part IV. Tests i experimentació 47

11 Test del comportament de la infraestructura d'injecció de fallades....49
11.1 Proves de retard introduït per l'injector....49
11.1.1 Mesura sense injector de fallades....50
11.1.2 Mesura amb injector de fallades en mode captura....51
11.1.3 Mesura amb injector de fallades en mode injecció....52
Experiment amb trames SDM....52
Experiment amb trames TM....53
11.1.4 Conclusions....54
11.2 Mesura de *l'offset* de la sortida del primer TM del HaRTES cap a 2 esclaus....55
11.2.1 Mesura de l'*offset* sense injector de fallades....56
11.2.2 Mesura de l'*offset* amb injector de fallades en mode captura....57
11.2.3 Mesura de l'*offset* amb injector de fallades en mode injecció....58
11.2.4 Conclusions....59

12 Exemples d'ús de la infraestructura d'injecció de fallades en casos reals....60
12.1 Mesura de l'*offset* de dos esclaus en finalitzar la finestra de TM....60
12.1.1 Mesura de l'*offset* sense l'injector de fallades....61
12.1.2 Mesura de l'*offset* amb injecció de tots els casos de pèrdua de rèplica de TM....62
12.2 Avaluació d'aturada de components d'una xarxa FTTRS....64
12.2.1 Comprovació de la tolerància a la caiguda d'un *switch*....66
12.2.2 Comprovació de la tolerància del tall d'un enllaç de forma permanent....66
12.3 Avaluació d'omissió de rèpliques de TM....69
12.4 Conclusions....70

Part V. Conclusions 71

13 Resum....73

14 Opinió Personal....77

15 Publicació dels experiments....79

Part VI. Apèndixs 81

A. Guia ràpida d'utilització de la infraestructura d'injecció de fallades....83

B. Exemples d'arxius de configuració....84
B.1. Exemple de *lock*....84
B.2. Exemple de *drop* amb rang de valors....85
B.3. Exemple de *crash*....85
B.4. Exemple de *allpossiblecases*....86
B.5. Exemple de delay amb 2 condicions d'injecció....87

Bibliografia....89

1 Introducció

En aquest capítol introductori, primer es comenta quin és l'entorn i les motivacions d'aquest projecte, tot seguit s'enumeren els objectius que es volen assolir durant la seva realització, després s'expliquen les tasques que s'han completat per tal d'aconseguir aquests objectius i finalment es descriu l'estructura d'aquesta memòria fent una breu introducció de totes les parts que la conformen.

1.1 Entorn i motivació del projecte

Actualment, *Ethernet* és l'estàndard de comunicació per a xarxes locals més estès en l'entorn domèstic i d'oficines. Els principals avantatges que presenta són la seva **elevada amplada de banda**, el **baix preu** dels seus components i la **compatibilitat amb altres estàndards** de comunicació. Per això es considera interessant utilitzar *Ethernet* en altres sistemes més propis d'entorns industrials [11].

Un exemple d'aquests sistemes són els sistemes encastats distribuïts (DES, per les seves sigles en anglès), on un conjunt de dispositius interconnectats, amb *hardware* de propòsit específic, combinen la informació que ells mateixos recullen amb la informació procedent d'altres nodes de la xarxa per controlar un determinat sistema. Alguns exemples de DES són: el sistema de control d'un avió o d'un automòbil, les cadenes de muntatge automàtiques de les fàbriques i els sistemes domòtics.

Els entorns industrials tenen requisits addicionals de **temps real** i de **garantia de funcionament** (fiabilitat, disponibilitat i/o seguretat). A més, no és estrany que aquests entorns siguin **dinàmics**, és a dir, entorns on no es poden determinar les condicions d'operació per avançat. Els DES desplegats en entorns amb requisits de temps real, amb garantia de funcionament i dinàmics han respondre en un temps acotat, no s'han d'avariar i, a més, han de poder reconfigurar-se si hi ha canvis a l'entorn.

Malauradament, *Ethernet* no proporciona aquests serveis. Per aprofitar els avantatges d'*Ethernet*, complint amb els requisits dels nous entorns, s'han desenvolupat nous protocols i estàndards basats en *Ethernet* que intenten resoldre aquestes mancances. Un exemple és el paradigma *Flexible Time-Triggered* (FTT) que assegura una resposta en temps real i, a més, permet el canvi dinàmic de les característiques del tràfic.

Hard Real-Time Ethernet Switching (HaRTES) és una implementació d'FTT basada en un *switched Ethernet* fet a mida que implementa els aspectes fonamentals del paradigma. HaRTES té una capacitat limitada per tolerar les fallades i, per tant, no és adequat per implementar sistemes amb requisits d'alta garantia de funcionament.

El projecte FT4FTT [6] de la Universitat de les Illes Balears té com a objectiu dissenyar, avaluar, validar, implementar i verificar mecanismes de tolerància a fallades per a HaRTES, amb el fi d'aconseguir una infraestructura que doni suport a aplicacions de control que siguin distribuïdes, adaptatives i fiables. La fase de verificació és especialment important, ja que permet comprovar que els mecanismes de tolerància a fallades estan dissenyats i implementats correctament i, a més, que combinats els uns amb els altres funcionen de manera correcta.

Una de les tècniques més utilitzades per a la verificació i validació de mecanismes de tolerància a fallades és la **injecció de fallades**. És important remarcar que una **fallada** és un defecte en el disseny o en l'operació del sistema, que no té per què manifestar-se. En canvi, un **error** és la manifestació d'una fallada i es percep com un valor incorrecte de l'estat del sistema. En aquest sentit, encara que el terme encunyat és injecció de fallades, la tasca que normalment duen a terme els injectors de fallades és la de provocar errors en el sistema i observar com aquest respon davant cadascun d'ells. Per tant, encara que en aquest document s'utilitzarà la terminologia estàndard, sempre que s'utilitzi el terme injecció de fallades o injector de fallades realment ens referim a errors.

Gràcies a la proliferació de variants de l'estàndard *Ethernet* en els entorns industrials, en els darrers anys s'han desenvolupat alguns injectors de fallades per avaluar el comportament de sistemes basat en *Ethernet*. No obstant això, es tracta d'injectors dissenyats per avaluar sistemes molt concrets, és a dir, no tenen generalitat. A més, no existeix cap injector que contengui funcionalitats específiques per a sistemes basats en paradigma FTT.

Aquest projecte de final de carrera està emmarcat dins del projecte FT4FTT, més concretament dins la part de verificació, i té com a objectiu el **disseny, implementació, verificació i validació d'una infraestructura d'injecció de fallades** per a *Ethernet* que permeti comprovar que els mecanismes de tolerància a fallades desenvolupats dins del projecte FT4FTT funcionen correctament.

El punt de partida per desenvolupar aquesta nova infraestructura és sfiCAN [8], una infraestructura d'injecció de fallades desenvolupada en el context del projecte CANbids per a una topologia en estrella i basada en protocol de comunicacions CAN. El punt central de la infraestructura d'injecció de fallades sfiCAN és un *hub* CAN que, a més d'interconnectar tots els nodes d'una manera fiable, permet provocar errors en els diferents enllaços de manera controlada i centralitzada.

Les funcionalitats principals de sfiCAN és duen a terme mitjançant mòduls lògics distribuïts anomenats *Network Configurable Components* (NCCs). Com es pot veure a la figura 1, els experiments es controlen mitjançant el *Fault-Injection Management Station* (FIMS) que és un programa que s'executa dins d'un PC connectat al *hub*. El *hub* conté un *Centralized Fault Injector* (CFI), que s'encarrega de provocar els errors, i un *Hub Logger* (HL), que s'encarrega de registrar el tràfic transmès i rebut des del punt de vista del *hub*. Finalment, el tràfic transmès i rebut per cada node és registrat pel *Node Logger* (NL) associat a l'esmentat node.

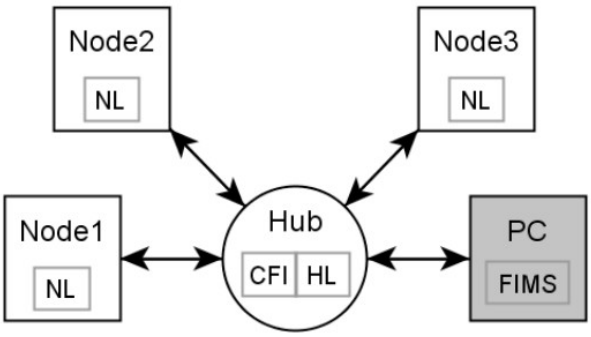


Figura 1: Arquitectura de sfiCAN

1.2 Objectius del projecte

Tenint en compte l'esmentat fins ara, aquest projecte consisteix en el disseny, implementació, verificació i validació d'una infraestructura automatitzada d'injecció de fallades per a *Ethernet* commutat que permeti la verificació dels mecanismes de tolerància a fallades desenvolupats en el projecte FT4FTT.

Aquesta infraestructura ha de permetre injectar diferents tipus de fallades, tant en el domini del valor com en el domini del temps, en trames concretes, a qualsevol dels enllaços del *switch* HaRTES. A més, ha de permetre monitoritzar i registrar el comportament dels dispositius de la xarxa durant les injeccions. Això s'aconseguirà emmagatzemant tot el tràfic que viatgi per la xarxa. Per a aconseguir una elevada precisió temporal en aquest procés de monitorització es proposa la utilització de l'analitzador de xarxa comercial *netAnalyzer* [7], disponible en el laboratori.

Com a punt de partida per a aquest projecte es prendrà sfiCAN [8], que és una infraestructura d'injecció de fallades desenvolupada en el context del projecte CANbids per una topologia en estrella basada en el protocol de comunicacions CAN.

Els objectius que es volen assolir en aquest projecte es poden especificar de la manera següent:

- Adaptar el disseny de la infraestructura de fallades sfiCAN a les característiques de HaRTES. A les tasques de monitorització on sigui necessari una elevada precisió temporal s'ha d'utilitzar l'analitzador de xarxa *netAnalyzer.*
- Implementar la infraestructura resultant sobre el prototipus HaRTES desenvolupat en el projecte FT4FTT.
- Verificar que el funcionament de la infraestructura implementada respon al disseny.
- Validar la infraestructura utilitzant-la per realitzar campanyes d'injecció de fallades en el prototipus desenvolupat en el projecte FT4FTT. D'aquesta manera s'ha de comprovar que el disseny de la infraestructura resol els problemes pels quals va ser concebut.

1.3 Tasques realitzades

En aquesta secció es descriuen les tasques que s'han dut a terme per tal de completar la infraestructura d'injecció de fallades.

Familiarització amb el prototipus HaRTES i l'analitzador de xarxa *netAnalyzer*.

La primera tasca va consistir a muntar el prototipus HaRTES amb dos nodes, el qual havia de servir com a xarxa model on injectar les fallades amb la nostra infraestructura. A més es varen realitzar una sèrie de mesures amb l'analitzador de xarxa amb la intenció de familiaritzar-se amb el funcionament del dispositiu i a la vegada obtenir valors de referència pels experiments posteriors.

Disseny del mòdul bàsic d'injecció de fallades.

Una vegada agafades les mesures de referència del prototipus, es varen sospesar les diferents opcions d'injectar fallades a la xarxa i es va decidir implementar un mòdul independent d'injecció per cada interfície de la xarxa.

Construcció del mòdul bàsic d'injecció de fallades.

El mòdul d'injecció bàsic es va implementar utilitzant C. Aquest, era capaç d'interceptar el tràfic d'una interfície *Ethernet* i provocar l'error de tots els possibles casos d'omissió de trames seguint unes determinades condicions abans d'enviar-lo al seu destinatari. Es va validar el funcionament del mòdul creant una petita aplicació de xarxa que enviava trames i se li va aplicar un mòdul d'injecció. Després, amb l'analitzador de trames *netAnalyzer,* es va comprovar que el tràfic resultant era l'esperat.

Disseny d'un llenguatge de configuració del mòdul de fallades

Per facilitar la configuració del mòdul d'injecció de fallades, es va dissenyar un llenguatge simple que permetés confeccionar un arxiu de configuració per part de l'usuari. En aquest arxiu s'especifiquen les condicions d'injecció de fallada, el tipus d'error que es vol provocar i la interfície associada al mòdul.

Addició de mòdul de logging al mòdul d'injecció.

Ja que el *netAnalyzer* només permet capturar el tràfic de dos enllaços a la vegada, es va afegir un mòdul de *logging* intern en cada mòdul d'injecció. Aquest, genera un fitxer amb les trames enviades. Així, s'obté informació addicional en experiments on hi hagi més de dos enllaços.

Recreació de l'experiment [5] utilitzant la infraestructura d'injecció

Una vegada validada la infraestructura d'injecció es va repetir l'experiment anterior però utilitzant l'analitzador per mesurar el temps de recepció de les trames en els esclaus i així recrear l'experiment descrit en l'article [5] amb la infraestructura d'injecció. Per tal de mesurar aquest temps, es varen haver de fer petites modificacions en el codi dels esclaus del prototipus.

Millora de la infraestructura

Per tal de millorar la infraestructura d'injecció, es varen afegir nous tipus d'errors provocables. També es va dotar a la infraestructura de funcions d'automatització, per tal que en haver capturat una quantitat de tràfic concret l'experiment finalitzes automàticament. Finalment, es va afegir un mòdul de control per facilitar la configuració dels mòduls injectors, la recuperació dels fitxers de captura de trames i la finalització centralitzada dels experiments per ordre de l'usuari.

Validació de la infraestructura d'injecció

Finalment es va utilitzar la infraestructura d'injecció desenvolupada per avaluar la tolerància a fallades del prototipus del projecte FT4FTT implementat en forma d'estrella replicada, FTTRS. Amb la infraestructura es va simular la fallada dels elements del prototipus, el tall de fins a tres enllaços simultanis i l'omissió de rèpliques de TM.

1.4 Estructura del document

La resta del document està dividit en sis parts:

A la primera part es descriu breument el treball anterior que conforma les bases en les quals es fonamenta aquest projecte i s'expliquen els conceptes necessaris per entendre el treball realitzant. Es fa una breu introducció al paradigma FTT i la seva implementació sobre *Ethernet*, una explicació sobre alguns mecanismes de tolerància de fallades, es descriu el prototipus HaRTES utilitzat durant els experiments que formen part d'aquest projecte i finalment s'introdueixen el concepte d'injecció de fallades i les diferents tècniques que existeixen.

La segona part, descriu el disseny de la nostra infraestructura d'injecció de fallades, els requisits que ha de complir, l'arquitectura de tots els mòduls que conformen la infraestructura i una explicació del llenguatge utilitzat per a la configuració dels mòduls injectors.

A la tercera part, s'explica com s'han implementat els diferents mòduls introduïts en la part de disseny.

La quarta part, correspon a la de test i experimentació on es descriuen els diferents experiments duts a terme per verificar i validar la nostra infraestructura d'injecció.

La cinquena part és la de conclusions. Consta d'un resum dels aspectes més importants del projecte, una opinió personal i finalment es comenta la publicació d'un article a un congrés internacional que descriu alguns experiments realitzats en aquest projecte.

La sisena està formada per 2 apèndixs. El primer és una guia ràpida d'utilització de la infraestructura d'injecció de fallades i el segon és un grup d'exemples d'arxius de configuració que provoquen diferents errors.

Finalment, a l'última pàgina, es troba la bibliografia referenciada al llarg d'aquesta memòria.

Part I. Fonaments i treballs anteriors

2 Fonaments de Flexible Time-Triggered(FTT)

Com hem comentat a la introducció, el paradigma FTT és un esquema de comunicació que possibilita l'intercanvi de dades en temps real de manera flexible per a sistemes encastats distribuïts (DES, per les seves sigles en anglès). Per això implementa diverses funcionalitats situades a la segona, tercera i quarta capa del model OSI. FTT s'ha de desplegar juntament amb un protocol de xarxa com a mínim de capa física. Alguns exemples d'implementació són: FTT-CAN, que defineix com es pot utilitzar FTT sobre el *Controller Area Network* (CAN), *FTT-Ethernet*, que és FTT sobre Ethernet en una topologia de bus o HaRTES *(Hard Real Time Ethernet Switching)* que és FTT sobre topologia d'estrella amb *switch* fet a mida.

A la figura 2 es pot veure l'arquitectura general de FTT. Aquesta arquitectura està composta per conjunt de nodes anomenats **esclaus**, interconnectats pel canal de comunicació, que implementen les funcionalitats dels nodes del sistema encastat. A més, consta d'un node especial anomenat **mestre** que coordina i administra la comunicació.

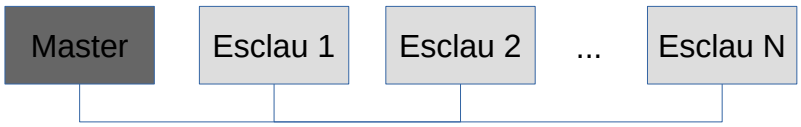


Figura 2: Arquitectura FTT.

2.1 Elementary Cycle (EC)

El mestre organitza la comunicació en ranures de durada fixa anomenades *Elementary Cycles* (**EC**s). Això fa possible que els dissenyadors d'aplicacions per DES puguin mapetjar les fases del bucle de control de les aplicacions a les comunicacions.

Com es veu a la figura 3, cada EC es divideix en dues finestres, la finestra síncrona i l'asíncrona. La primera s'utilitza per transmetre el tràfic periòdic, que en FTT s'anomena tràfic síncron, ja que es transmet sincronitzat amb l'EC, mentre que a la segona s'hi transmet el tràfic aperiòdic, que anomenarem tràfic asíncron, ja que la seva transmissió pot començar en qualsevol moment.

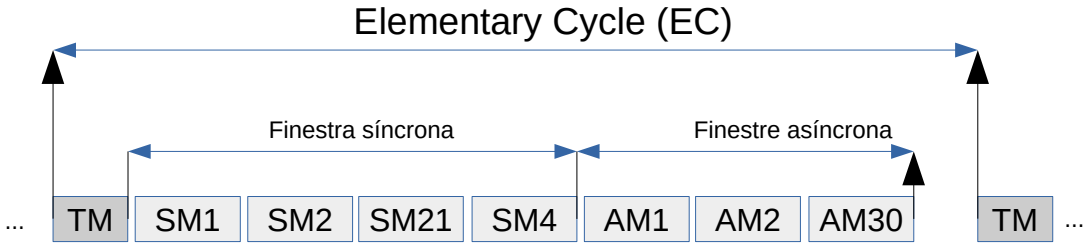


Figura 3: Estructura de l'Elementary Cycle (EC)

Cada EC s'inicia pel mestre seguint un esquema mestre-multiesclau, utilitzant l'anomenat ***Trigger Message* (TM)**. El propòsit d'aquest missatge és:

- Informar els esclaus que un nou EC ha començat. És a dir, sincronitza els esclaus.
- Transportar la planificació de l'EC per a la finestra síncrona. El que significa que específica els missatges que cada esclau ha de transmetre en la finestra síncrona de l'EC.

El patró de comunicació definit per FTT per cada EC es mostra a la figura 4. Primer, el mestre marca l'inici de l'EC enviant el TM, que és rebut i descodificat pels esclaus. Llavors, durant la finestra síncrona, els esclaus intercanvien missatges síncrons seguint la planificació d'EC especificada en el TM. Aquests missatges contenen dades generades per les aplicacions que s'executen als esclaus. Després, durant la finestra asíncrona, els esclaus i el mestre, intercanvien missatges asíncrons, però ara sense cap planificació específica. Cal comentar que aquests missatges no són només utilitzats per transmetre dades de les aplicacions, sinó que també s'utilitzen per controlar la comunicació. Finalment, s'ha de tenir en compte que els esclaus es desincronitzen amb el pas del temps, el que pot provocar transmissions després de la finestra asíncrona. És per això que es reserva un temps de guarda al final de l'EC on els nodes més lents poden acabar les seves transmissions.

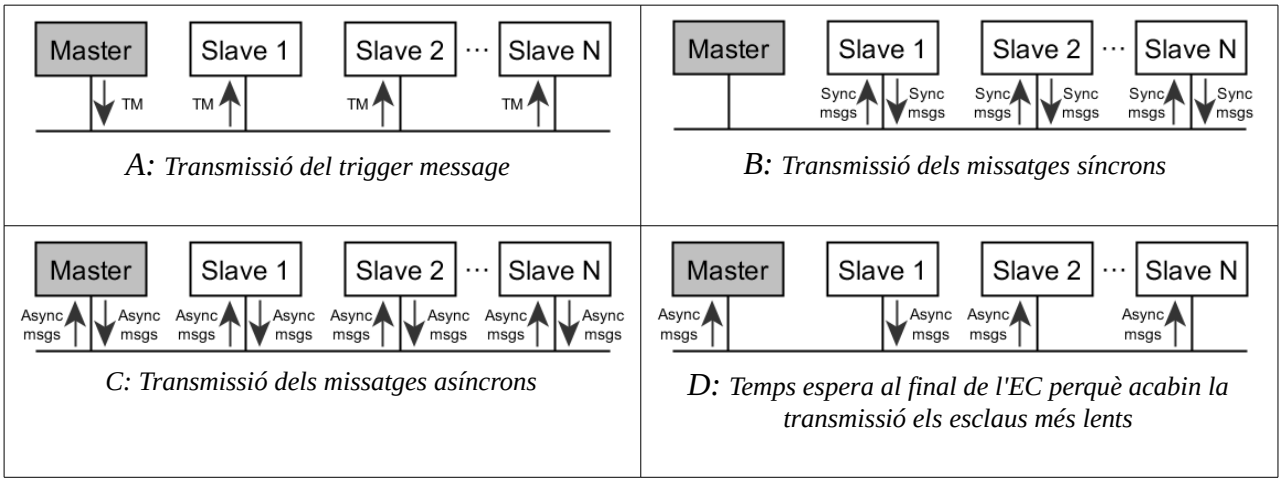


Figura 4: Esquema de comunicació FTT (basat en l'esquema de [3]).

Addicionalment, s'ha de comentar que la mida dels EC es pot modificar per adaptar-se als requisits del sistema. Així, amb un EC més llarg es poden enviar més dades, ja que disminuïm el pes de l'encapçalament, causat per la transmissió del TM, i el temps d'espera al final de cada EC. Però amb ECs més curts tenim una millor resposta del subsistema de comunicació, ja que la transmissió entre cicles és més curta.

2.2 Missatges FTT

A la figura 5 podem identificar els diferents tipus de missatges que es poden transmetre en una xarxa FTT. En una primera divisió es poden distingir dos grups: els missatges de dades i els missatges de control. Els missatges de dades són utilitzats pels esclaus per intercanviar dades síncrones i asíncrones d'aplicació. Els missatges de control, en canvi, són utilitzats tant pel mestre com pels esclaus i permeten controlar la comunicació. Aquest segon grup, a més del TM, inclou missatges asíncrons anomenats missatges *plug-and-play*, missatges *slave request* i missatges *master command*.

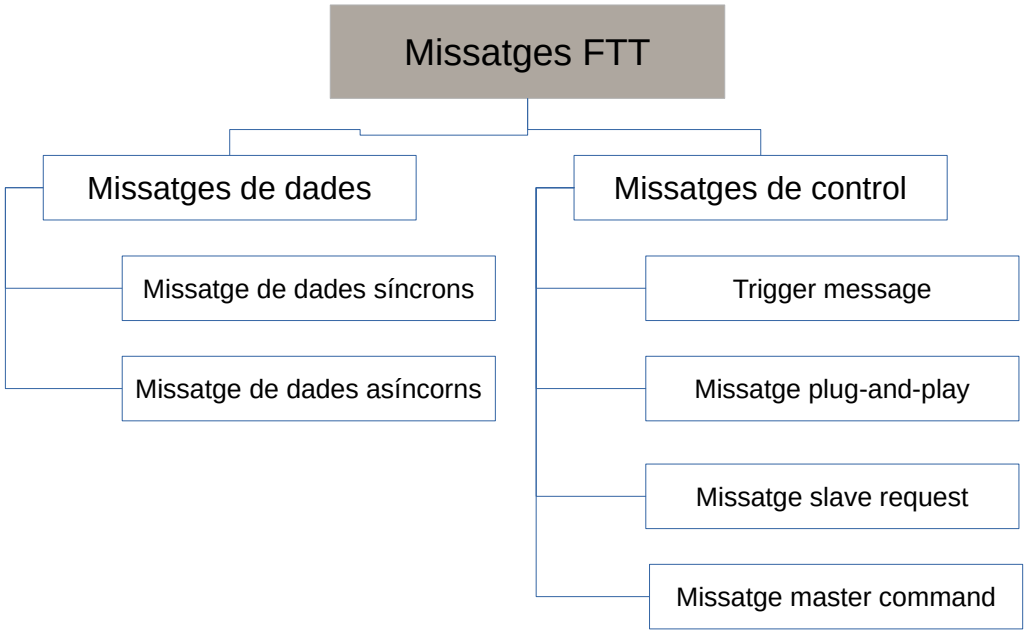


Figura 5: Tipus de missatges FTT.

Primer, els missatges *plug-and-play* s'utilitzen per implementar el mecanisme *plug-and-play*. Entrant més en detall, quan un esclau vol entrar a una xara FTT, ha de sol·licitar permís al mestre. El qual ha de respondre a la petició. Segon, els missatges *slave request* permeten als esclaus sol·licitar al mestre la modificació dels requisits de comunicació. Finalment, els missatges *master command* són utilitzats pel mestre per respondre a les sol·licituds de modificació de requisits de comunicació.

Per una descripció en profunditat d'aquests missatges recomanam la lectura de l'apèndix A de *"A description of the FTT-SE protocol" [3]* .

3 Hard Real-Time Ethernet Switching(HaRTES)

Gràcies a la seva gran amplada de banda i l'estès que està el seu ús, *Ethernet* és una opció interessant per implementar subsistemes de comunicació de DES. Tot i així, els requisits de temps real i fiabilitat d'aquests sistemes fan necessari una extensió del protocol. HaRTES és una implementació del paradigma de comunicació FTT sobre una xarxa *Ethernet* micro-segmentada, on el punt central és un *switch* fet a mida anomenat HaRTES *switch*. Aquest *switch* conté el mestre i s'encarrega de les funcions més rellevants d'FTT. Més concretament, s'encarrega de concedir accés a la xarxa. La comunicació entre el mestre i els esclaus segueix l'esquema FTT però aprofitant les avantatges que proporciona la topologia d'estrella respecte al bus.

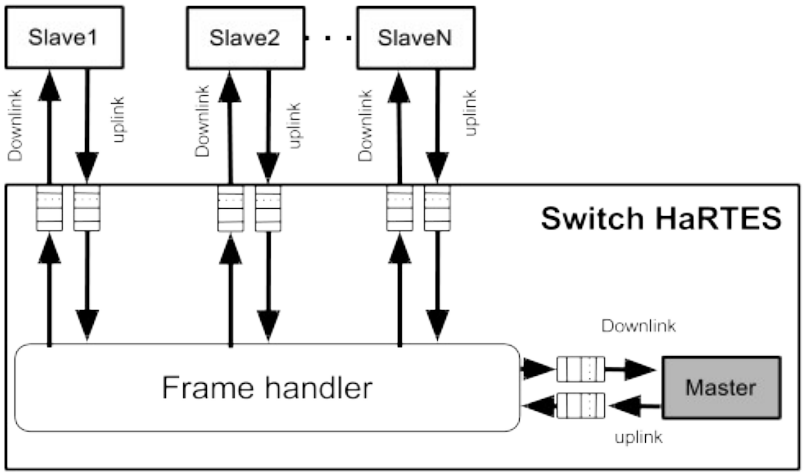


Figura 6: Topologia d'estrella amb switch HaRTES.

La característica més rellevant d'aquesta topologia, quan es compara amb la d'un bus, és l'existència d'enllaços dedicats per cada node i sentit de transmissió. A la figura 6 podem veure com cada node està connectat a la xarxa a través de dos enllaços diferents. Un l'utilitza per rebre (*downlink*) i l'altre per transmetre (*uplink*). Des del punt de vista del protocol, aquesta configuració aporta dos beneficis importants:

- Absència de col·lisions: Això facilita la implementació tant del mestre com dels esclaus. Pel que fa al mestre, les planificacions dels ECs es poden fer sense haver d'assegurar un accés al medi lliure de col·lisions. Pel que fa als esclaus, com que tenen enllaços dedicats, no tenen restriccions a l'hora de transmetre. És el *switch* qui s'encarrega d'organitzar tots els missatges al canal.
- Enviament paral·lel. Això dota cada esclau amb un enllaç de transmissió dedicat, el que pot utilitzar el planificador per augmentar el volum de treball total. Això es pot aconseguir construint una planificació que considera els missatges seguint camins disjunts. Que són camins que tenen diferent node de destinació i origen.

A la figura 7 es mostra un EC en els dos enllaços d'un esclau. Hi ha dos aspectes importants a remarcar en aquesta figura. El primer, l'existència d'un enllaç dedicat per a la transmissió i un altra per a la recepció fa possible a l'esclau transmetre missatges al mateix temps que en rep d'altres. El segon, la disposició de les finestres depèn dels missatges transmesos. És per això que la divisió en finestres que es fa de cada EC és diferent a cada canal.

Com veiem ala figura, mentre que a l'*uplink* ens trobam les finestres de FTT descrites a l'apartat 2.1 (finestres síncrona i asíncrona), al *downlink* tenim la finestra de guàrdia, on s'envia el TM, la finestra *time arroud* que equival al temps màxim que pot tardar un esclau a descodificar el TM i finalment tenim la finestra mixta on l'esclau rep els missatges síncrons i asíncrons indistintament dels altres nodes.

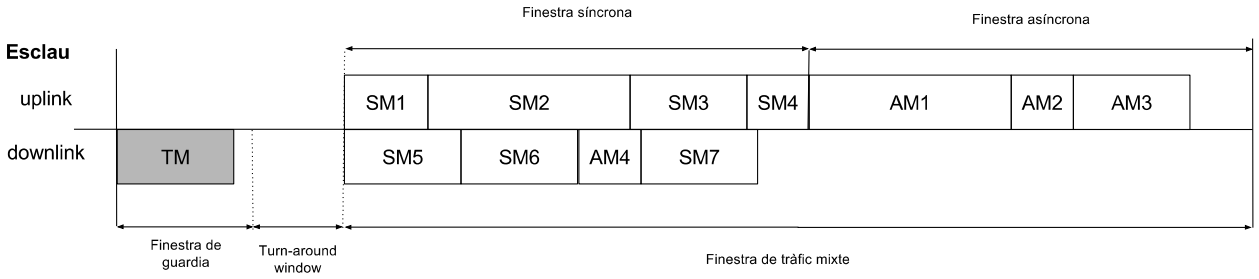


Figura 7: Tràfic als dos enllaços entre el mestre i un esclau.

A HaRTES, igual que a totes les implementacions de FTT sobre *Ethernet*, els missatges s'encapsulen en trames *Ethernet*. Més específicament, com es pot veure a la figura 8, les trames *Ethernet* tenen una capçalera, una secció de dades i un farciment. La capçalera conté les adreces MAC origen i destí i un tercer camp anomenat *ethertype*.

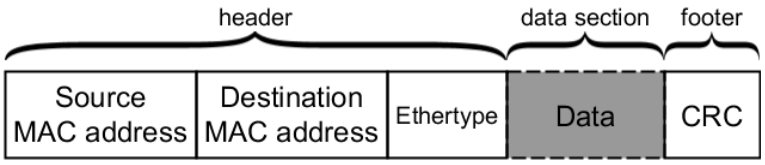


Figura 8: Camps d'una trama Ethernet.

En una xarxa *Ethernet* estàndard els camps MAC origen i destí s'utilitzen per encaminar les trames. Cada dispositiu de xarxa que segueix aquest estàndard rep una adreça única de 6 *Bytes* que es coneix com a adreça MAC. El camp MAC origen d'una trama sempre conté l'adreça MAC del node que ha generat la trama. En canvi, segons qui sigui el destinatari de la trama podem distingir tres tipus de trames amb un valor diferent del camp MAC destí.

El primer tipus de trama és ***l'unicast***. aquest s'utilitza quan la trama va dirigida a un sol node. Quan un node A de la xarxa vol enviar una trama un node B de la mateixa xarxa, assigna la seva adreça MAC al camp MAC origen i l'adreça MAC del node B al camp MAC destí i la transmet. El *switch* Ethernet comú quan rep aquesta trama llegeix el camp MAC destí i la transmet per la interfície on té

el Node B connectat. D'aquesta manera el Node B rep la trama i en llegir els camps origen i destí sap que la trama és per ell i qui li ha enviat. El segon tipus és el ***multicast*** on el destinatari son varis nodes. En aquest cas l'adreça destí comença per un codi especial 0x01-00-5E més els últims 23 bits que conformen les adreces *multicast* d'una capa superior. Finalment, tenim les trames de tipus ***broadcast*** on el destinatari de la trama són tots els nodes de la xarxa *Ethernet*. En aquest últim cas l'adreça MAC destí correspon a un codi especial FF:FF:FF:FF:FF:FF: en hexadecimal i el *switch* al rebre una trama amb aquest valor la transmet per totes les seves interfícies.

En el cas de HaRTES s'utilitzen els tres tipus de trames. No obstant, per adreçar les trames a una xarxa FTT s'utilitza el concepte de *stream*. Un *stream* és un canal virtual de comunicació entre nodes FTT en el que podem tenir múltiples receptors d'un missatge enviat per un únic node. Quan un node vol transmetre un missatge FTT a altres nodes el que fa, en comptes d'utilitzar l'adreça MAC, és indicar el *stream*. El mateix mòdul FTT s'encarrega d'utilitzar una adreça MAC *unicast*, *multicast* o *broadcast* depenent del nombre de receptors. El mestre és l'encarregat d'encaminar les trames i disposa de la llista de tots els *streams* actius i tots els esclaus que estan interessats a rebre els missatges de cada un d'ells. Cal comentar que els esclaus no són conscients de qui està associat a cada *stream*. Per ampliar la informació sobre els *streams* a FTT recomanem la lectura del capítol 5 de *[3]* .

Pel que fa al tercer camp esmentat de les trames *Ethernet*, el camp *ethertype* identifica quin tipus de missatge de capa superior es troba encapsulat dins la secció de dades. En el cas de trames on tenguin encapsulats missatges FTT, el valor de l'*ethertype* és 0x8ff0. La secció de dades conté el missatge FTT i, en casos on el missatge FTT superi la mida màxima de la secció de dades (1500 bytes) es divideix en diverses trames *Ethernet*.

Finalment, el farciment conté un codi de redundància cíclica (CRC) per les capçaleres i la secció de dades.

4 Tolerància a fallades per a HaRTES

Anteriorment hem parlat dels beneficis que pot proporcionar FTT en els DES, però cal fer notar que, a més de flexible, la comunicació en un sistema encastat distribuït crític ha de ser confiable. Per això s'ha de dotar a HaRTES de tolerància a fallades. En aquest apartat introduirem diferents tècniques desenvolupades en el context del projecte FT4FTT per tal d'obtenir una implementació de HaRTES amb tolerància a fallades.

4.1 The Slave Elementary Cycle Synchronization Mechanism (SECSM)

Com hem explicat al capítol anterior en FTT dividim el temps de comunicació en ECs. La manera que utilitza el mestre per marcar l'inici d'aquests ECs és el missatge TM. A més, aquest missatge és utilitzat pels esclaus per sincronitzar-se. Per tant, si volem aconseguir una elevada fiabilitat, és imprescindible garantir que aquests missatges arriben correctament als esclaus. Donat que el canal de comunicació és vulnerable a fallades transitòries que poden provocar la pèrdua de missatges, s'ha proposat com a part del projecte FT4FTT un mecanisme de replicació de missatges, de tal forma que el mestre envia cada TM diverses vegades a cada EC.

Els TMs s'envien durant la finestra de TM (TMW) que s'ha afegit a l'inici dels ECs. Durant la TMW el mestre només duu a terme l'enviament de TMs. Com ja hem comentat, la funció principal dels TMs és sincronitzar l'inici dels EC en els esclaus. En les implementacions que utilitzen múltiples rèpliques de TM els esclaus se sincronitzen utilitzant el mecanisme anomenat Slave Elementary Cycle Synchronization Mechanism (SECSM) [5]. Mitjançant aquest mecanisme cada esclau calcula quan finalitzarà la TMW i utilitza aquest càlcul per sincronitzar-se.

Perquè un esclau sigui capaç de calcular el final de la TMW s'han de complir 4 condicions:

1) L'esclau rep com a mínim una de les *k* rèpliques enviades pel mestre.
2) El mestre transmet les rèpliques del TM de forma que tots els esclaus que reben una determinada rèplica la reben al mateix temps.
3) El mestre transmet les rèpliques del TM esperant un temps entre transmissions anomenat τ, que és igual per a totes les rèpliques.
4) El desfasament que es produeix durant un EC entre els rellotges de dos esclaus que no fallen és menyspreable.

Per tal de poder implementar el SECSM s'han afegit 3 camps al TM respecte a l'utilitzat a FTT: un camp que indica el nombre de rèpliques del TM enviades pel mestre en cada EC (k), un camp que identifica l'ordre de cada rèplica (seq_no) i el temps entre l'arribada de rèpliques (τ).

Si un esclau rep almenys una rèplica de TM, pot calcular el final de la finestra de TM utilitzant la informació d'aquests 3 camps i el temps d'arribada de la rèplica rebuda de la següent manera:

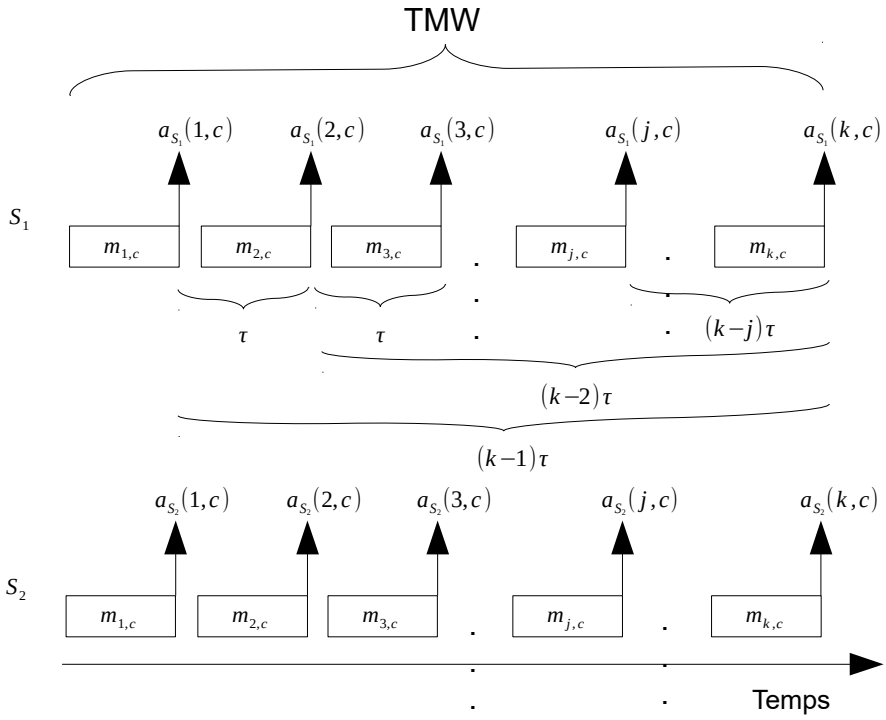


Figura 9: Temps esperat d'arribada de les k rèpliques de TM en els esclaus S1 i S2 [5].

Consideram *c* com el nombre de seqüència d'un determinat EC i *Sc* com el conjunt d'esclaus sense fallades a l'EC *c*. Prenem *k com* el nombre de rèpliques del TM i $m_{j,c}$ és la rèplica amb identificador *j* de l'EC *c*, on $j \in$N i $1 \leq j \leq k$. A més, $M_{s,c}$ representa les rèpliques de TM rebudes per l'esclau *s* durant l'EC *c*. Finalment $a_S(j,c)$ és el temps d'arribada esperat del missatge $m_{j,c} \in M_{s,c}$ a l'esclau $s \in Sc$. La Figura 8 mostra el temps esperat pels esclaus S_1 i S_2 per la recepció de les *k* rèpliques enviades a l'EC *c*.

Com es veu a la figura el temps esperat d'arribada de l'última rèplica $a_{S_1}(k,c)$ coincideix amb el final de la TMW. Per tant, si coneixem el moment d'arribada del missatge $m_{j,c}$, podem calcular el final de la TMW de la següent manera, $a_{S_1}(k,c)=a_{S_1}(j,c)+(k-j)\tau$ [5].

Donat que la condició (2) estableix que tots el*s* esclaus que reben una rèplica la reben al mateix temps, podem dir que $a_{S_1}(j,c)=a_{S_2}(j,c)$ i, per tant, ambdós esclaus determinaran que el final de la TMW ocorre al mateix moment. És a dir, els esclaus poden sincronitzar el final de la TMW mitjançant el SECSM, utilitzant la recepció dels TMs com a esdeveniment de sincronització.

4.2 Flexible Time-Triggered Replicated Star for Ethernet (FTTRS)

Com s'ha comentat anteriorment, l'objectiu del projecte FT4FTT és el d'utilitzar tolerància a fallades per obtenir un sistema de comunicacions altament fiable, però mantenint la flexibilitat del paradigma FTT. Perquè un sistema sigui confiable s'han d'eliminar els punts únics d'avaria, és a dir, aquests elements que, si s'avarien, provoquen una avaria global en el sistema.

Estudiant el cas de HaRTES es pot veure que un dels punts que pot provocar una fallada de la comunicació és una fallada del *switch*. Si aquest element falla, es produeix una fallada global del sistema. Per prevenir aquesta possibilitat en el projecte FT4FTT es proposa l'ús de l'arquitectura Flexible Time-Triggered Replicated Star for Ethernet (FTTRS) [4]. L'arquitectura FTTRS replica el *switch* HaRTES prevenint així davant la fallada tant del *switch* com del mestre.

Com es pot veure a la figura 10, per tal que els esclaus puguin utilitzar les rèpliques del *switch* es necessita un enllaç entre cada esclau i cada *switch* HaRTES. A més, per tolerar la fallada d'una de les rèpliques del *switch* s'ha d'assegurar que les rèpliques que continuen funcionant, continuen proporcionant un servei adequat, i que la transició a l'estat amb un nombre menor de rèpliques no interfereix en el correcte funcionament del sistema. Per això és necessari assegurar determinisme entre rèpliques que, normalment, requereix alguna forma de comunicació entre rèpliques. Per això a FTTRS s'introdueix un enllaç replicat que connecta entre elles cada rèplica de *switch* HaRTES.

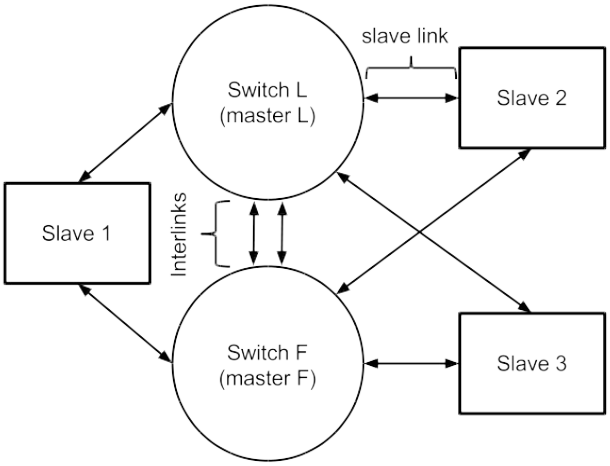


Figura 10: Diagrama de xarxa FTTRS amb dues rèpliques de Switch [4].

Per assegurar la sincronització dels ECs en les rèpliques de mestre, una d'elles es designa ***master leader*** i l'altre ***master follower***. El *master leader* utilitza el seu rellotge intern per sincronitzar la transmissió dels seus TMs. El *master follower* també utilitza el seu rellotge intern però, a més, utilitza els TMs que li arriben del *master leader* per calcular *l'offset* que hi pugui haver entre ell i el *leader* i corregeix aquest *offset* modificant el temps en què ha de transmetre els seus pròxims TMs [10].

Així, si decidim utilitzar 2 rèpliques de *switches*, l'esquema de l'arquitectura FTTRS serà el que es mostra a la figura 10. Dos *switches* HaRTES amb tots els esclaus connectats amb cada un d'ells i els dos *switches* estan connectats entre ells per diversos enllaços redundants anomenats *interlinks*. Els *interlinks* proporcionen determinisme de rèplica als 2 *switches*. No només això, els *interlinks* també s'utilitzen per retransmetre els missatges que un esclau envia a un *switch* a la seva rèplica augmentant la redundància de camins entre parells d'esclaus.

5 Descripció del prototipus HaRTES

Tot i que la infraestructura d'injecció de fallades pot simular errors en qualsevol xarxa amb capa d'enllaç que segueixi el protocol *Ethernet*, es varen incloure funcions específiques pensant en la implementació del protocol FTT sobre *Ethernet* amb topologia d'estrella que està duent a terme el grup del projecte *FT4FTT-Ethernet: Fault Tolerance mechanisms for adaptive distributed embedded systems based on FTT-Ethernet*. En aquest apartat es descriu el prototipus de xarxa que vàrem utilitzar per realitzar els diferents experiments.

Vàrem utilitzar una xarxa formada per un *switch* HaRTES connectat a un grup d'esclaus (entre un i tres depenent de les proves). Pel *switch* es va utilitzar un PC *multi-core* amb dos adaptadors *Ethernet* de quatre ports *Intel I350-T4*. En canvi pels esclaus es va utilitzar *hardware* per a dispositius encastats. Cada un d'ells s'executava en una *barebone Jetway JBC373F38-525-B* que té un processador *Intel Atom* i una targeta de xarxa de 4 interfícies.

Cada *switch* FTT inclou un mòdul mestre i un mòdul de *switch* amb fins a 8 ports *Ethernet*. El *switch* està totalment implementat en software i s'executa en l'espai d'usuari del sistema operatiu Linux. El seu codi està basat en una implementació del *switch* HaRTES. De la mateixa manera, esclaus estan formats per un mòdul FTT implementat per *software* que inclou una aplicació executada per a l'esclau i un *driver* FTT que permet enviar i rebre trames a través de la xarxa FTT. Cada esclau utilitza un port *Ethernet* per connectar-se al *switch* HaRTES amb un enllaç 100 Mbps.

A més, el *switch* HaRTES i els esclaus estan connectats a una estació d'instrumentació a través d'un *switch Ethernet*. Formant així, una xarxa secundària d'instrumentació utilitzada per iniciar, monitoritzar i aturar l'execució dels components software FTT de cada element del prototipus. Aquesta estació d'instrumentació és un altre *barebone Jetway JBC373F38-525-B* amb sistema operatiu *Ubuntu*. *A la* figura 11 es pot veure l'esquema descrit.

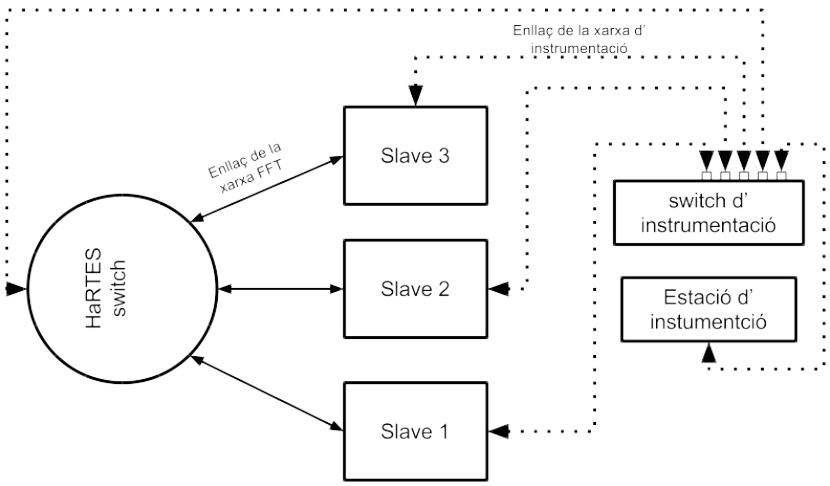


Figura 11: Esquema de xarxa FTT amb un *switch HaRTES* i 3 esclaus controlats per una interfície d'instrumentació.

6 Injecció de fallades

La injecció de fallades consisteix a crear artificialment erros en un sistema. El principal avantatge d'aquesta tècnica és la possibilitat de recrear un escenari específic de fallada a voluntat que, d'una altra manera, hauria tardat una gran quantitat de temps en ocorra. Per això un injector de fallades pot ser molt útil a l'hora d'analitzar el comportament d'un sistema d'avant un model de fallada concret. Entrant més en detall, permet avaluar la capacitat d'un sistema per detectar fallades aïllar-les, reconfigurar-les i recuperar-se'n.

6.1 Arquitectura general d'un entorn d'injecció de fallades

A la figura 12 es mostra l'arquitectura general d'un entorn d'injecció de fallades. Està dividit en dues parts, el sistema d'injecció de fallades i el sistema on s'injecten

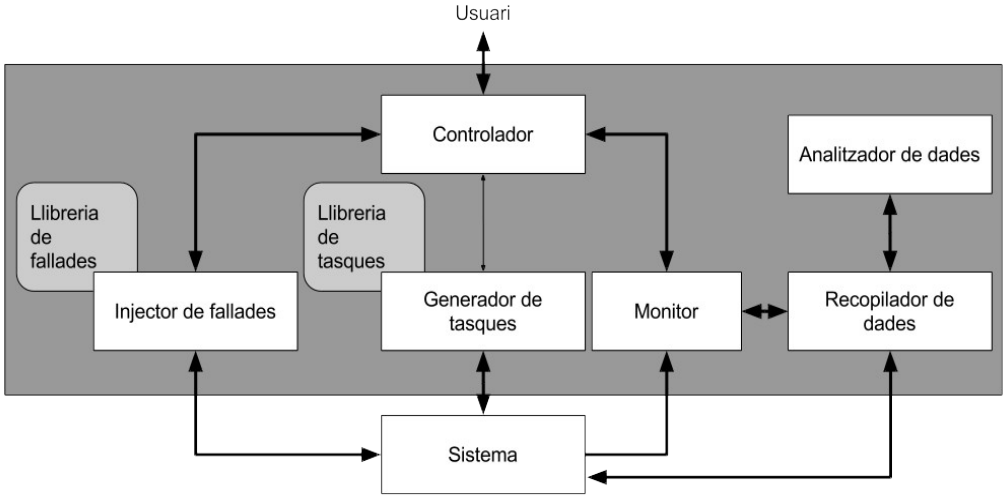


Figura 12: Arquitectura d'injecció de fallades (La figura està basada en una de Tsai and Iyer [1997]).

Com es veu a la figura, el sistema d'injecció de fallades es divideix en diversos mòduls: l'injector de fallades, el generador de tasques, el monitor, el recopilador de dades, l'analitzador de dades, i el controlador. Tots aquests mòduls treballen conjuntament, no només per injectar fallades, sinó també per capturar informació relacionada amb la resposta del sistema davant aquestes fallades.

Primer tenim el generador de tasques, que alimenta el sistema amb el grup de comandes que conformen les tasques ha d'executar. Per això, el generador de tasques utilitza la informació que li proporciona la llibreria de tasques, que especifica les tasques que s'han d'executar durant el test. Segon, el monitor rastreja l'execució de comandes i comunica al recopilador de dades quines dades capturar. És important remarcar que el recopilador de dades captura dades *online* en canvi l'analitzador de dades les processa i analitza *offline*. Tercer, l'injector de fallades és el responsable de provocar els erros en el sistema. Finalment, el controlador és l'encarregat de coordinar els diferents mòduls durant l'experiment. Més concretament, decideix quines tasques de la llibreria de tasques han de ser executades, quina fallada de la llibreria ha de ser injectada i quina informació s'ha de monitoritzar. L'usuari final s'encarrega de configurar el controlador.

6.2 Tipus d'injectors de fallades

Les tècniques d'injecció de fallades es poden classificar segons la maduresa de la implementació del sistema que es vol testejar o segons el lloc on es vol injectar les fallades.

Pel que fa al primer criteri de classificació podem distingir entre dos tipus de tècniques d'injecció:

- Tècniques basades en prototipus: Aquest tipus d'injecció de fallades opera sobre el prototipus existent d'un sistema. El principal benefici d'utilitzar aquestes tècniques és el realisme que tenen les dades obtingudes dels experiments. Però el problema que tenen és que, normalment, el cost d'obtenir aquestes dades és elevat si el prototipus està implementat en *hardware*.
- Tècniques basades en simulacions: En aquest tipus de tècniques la injecció de fallades es realitza en una abstracció del sistema, o el que és el mateix, en un model. El principal avantatge d'aquesta aproximació és que, com que el model i l'injector estan implementats en software, les dades d'un experiment són fàcils d'extreure directament dels valors que conformen l'estat del model. Però la qualitat d'aquestes dades depèn de la qualitat del model. El qual, a la vegada depèn dels seus paràmetres d'operació. Això és un desavantatge important, ja que, els valors d'aquests paràmetres solen ser difícils d'estimar.

Pel que fa al segon criteri de classificació, també tenim dos tipus de tècniques:

- Tècniques d'injecció de fallades hardware: Aquest tipus de tècniques utilitzen hardware addicional per provocar errors al hardware del prototipus o el sistema estudiat. Aquestes tècniques són les més adequades quan es vol injectar a un nivell baix o quan hi ha necessitat de resolució temporal. Però la construcció d'aquests tipus d'injectors sol ser cara i difícil.
- Tècniques d'injecció de fallades software: Aquest tipus d'injecció de fallades utilitza software per provocar errors d'alt nivell en el sistema. En contrast a la injecció de fallades per hardware, aquests erros s'injecten en les aplicacions o en el S.O mateix. La injecció mitjançant software és més flexible i barata que la injecció mitjançant hardware però el software addicional que s'executa en el sistema pot provocar pertorbacions no desitjades en les tasques reals.

Part II. Disseny

7 Llistat de requisits

Una vegada hem introduït tots els conceptes necessaris per entendre en quin entorn volem utilitzar la nostra infraestructura d'injecció de fallades, hem descrit a quin tipus de sistema de comunicació la volem aplicar, hem explicat els diferents mecanismes de tolerància de fallades que hi volem avaluar, hem presentat un prototipus d'aquest sistema i hem introduït els diferents tipus d'injectors de fallades que existeixen és el moment de començar a descriure el disseny de la nostra infraestructura d'injecció de fallades.

La primera passa en el disseny és definir quins requisits ha de complir, la infraestructura, per tal de poder complir amb els objectius d'aquest projecte. En aquest capítol enumerarem aquests requisits dividint-los amb funcionals i no funcionals.

7.1 Requisits funcionals

- L'usuari ha de ser capaç, executant un programa a cada component de la xarxa, d'especificar els escenaris de fallades que s'han d'injectar. Per això se l'ha de proveir d'un llenguatge simple perquè pugui crear els arxius de configuració específics a cada escenari.
- L'injector ha de ser capaç de reproduir escenaris que incloguin diversos errors simultanis.
- L'injector ha de ser capaç d'injectar fallades permanents i temporals.
- L'injector ha de ser capaç d'injectar fallades tant en els missatges que arriben a un element de la xarxa com en els que transmet aquest component.
- L'injector ha de ser capaç d'eliminar, modificar o retardar una trama. A més ha de ser capaç de simular la caiguda total d'un component de la xarxa o el bloqueig momentani d'aquest.
- L'injector ha de poder diferenciar entre diferents tipus de tràfic. *Ethernet* i FTT

7.2 Requisits no funcionals

- L'injector ha de ser transparent des del punt de vista de la xarxa.
- Ha de ser escalable, és a dir, ha de ser fàcil augmentar el nombre i el tipus d'errors que es provoquen a la xarxa
- Ha de ser capaç d'adaptar-se a diferent nombre d'elements de la xarxa
- L'injector ha d'estar format per mòduls independents i l'usuari s'ha de poder comunicar en tots ells a través de la interfície d'instrumentació

8 Arquitectura de la infraestructura d'injecció de fallades.

Una vegada hem establert el requisits que necessita la nostra infraestructura d'injecció de fallades hem de dissenyar la seva arquitectura per tal que els compleixi. En aquest capítol descrivim aquesta arquitectura, explicant per a quin tipus de sistemes està dissenyada, els diferents mòduls que la conformen i el procés que seguiran aquests mòduls per injectar les fallades en el sistemes on s'integrin.

La nostra infraestructura d'injecció de fallades està pensada per provocar errors en el tràfic d'un prototipus de comunicacions sobre *Ethernet*. Els errors que es provoquen són d'alt nivell i la injecció d'aquests, es realitza mitjançant software que s'executa, en els nodes del sistema, al mateix temps que l'aplicació de xarxa pròpia del prototipus. Aquest software d'injecció intercepta el tràfic i hi realitza canvis abans de transmetre'l si es donen unes determinades condicions.

Per tal de dependre el menys possible de la topologia de la xarxa i per poder recrear el màxim nombre d'escenaris d'error possible, hem dissenyat la infraestructura d'injecció de fallades de forma modular. La nostra infraestructura està formada per un conjunt de **mòduls injectors de fallades**, amb la seva configuració específica. Cada un d'ells s'executa en un node de la xarxa del prototipus. O fins i tot, es poden executar diversos mòduls injectors en un mateix node si aquest té més d'una interfície de xarxa. A més, volem que els mòduls injectors de fallades de la infraestructura realitzin la seva tasca de forma totalment transparent per l'aplicació de xarxa del node on s'executen.

Els mòduls injectors de la infraestructura d'injecció de fallades es comuniquen amb un **mòdul de control**, allotjat en una **estació d'instrumentació** (PC), a través d'una xarxa secundària que anomenam **xarxa d'instrumentació**. A la figura 13 podem veure un node de la xarxa executant una aplicació i el mòdul injector de forma separada. A més està connectat a la xarxa secundària d'instrumentació per comunicar-se amb l'estació d'instrumentació executant el mòdul de control.

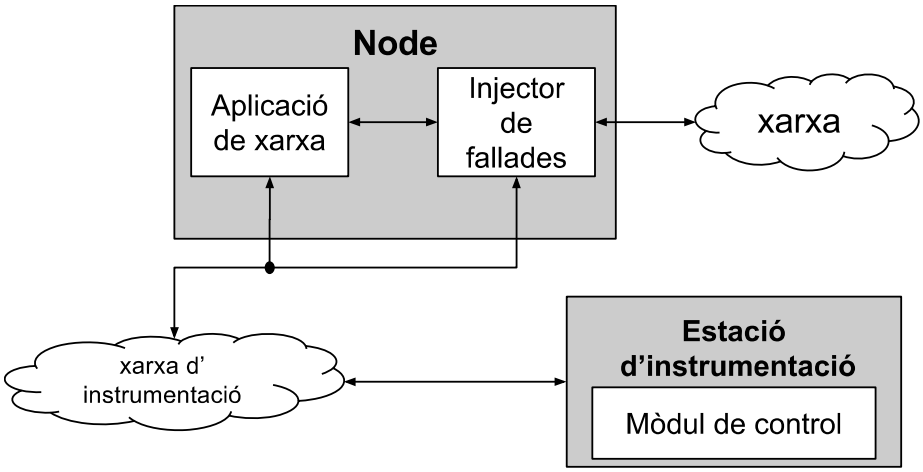


Figura 13: Mòdul injector de fallades d'un node de la xarxa

8.1 Estructura d'un mòdul injector de fallades

Podem diferenciar dos modes de funcionament del mòduls injectors: el **mode captura,** en el qual el mòdul injector no injecta fallades i es limita a capturar tot el tràfic que passa per ell per generar un fitxer de captures i el **mode injector**, en el qual el mòdul, a més de capturar el tràfic, té configurats un o diversos errors que ha de provocar al tràfic abans de transmetre'l.

En un node qualsevol executant una aplicació de xarxa, on executem un mòdul injector de fallades en mode injector, aquest intercepta el tràfic que transmet el node on s'està executant, l'examina i hi realitza les modificacions designades. Una vegada modificat el tràfic, el transmet a la xarxa. De la mateixa manera, intercepta tot el tràfic que arriba de la xarxa amb destí al node on s'executa, l'examina i el modifica abans de transmetre'l. En la figura 14 es mostra el diagrama intern d'un injector de fallades de la nostra infraestructura, on cada mòdul representa una part del procés d'injecció que realitza. Cal fer notar, que hem dissenyat els mòduls injectors pensant en els dos canals d'un enllaç *Ethernet*. De manera que tenen els mòduls interns duplicats, els que s'encarreguen del tràfic de l'***uplink*** els hem anomenat **tx** i els que s'encarreguen del tràfic del ***downlink*** els hem anomenat **rx**.

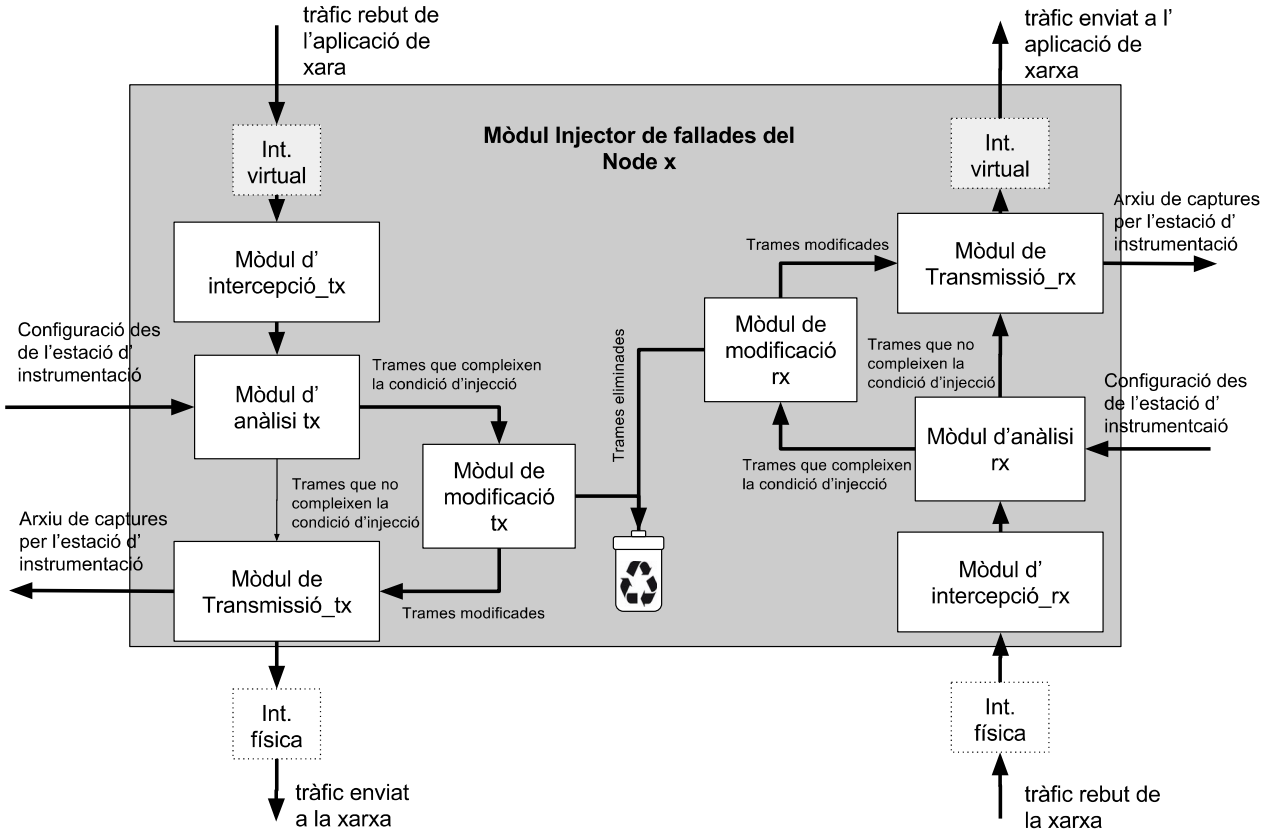


Figura 14:Diagrama intern del mòdul injector de fallades.

8.1.1 Interfície Ethernet virtual (TAP)

Perquè l'injector pugui interceptar el tràfic d'entrada i sortida del node, crea una interfície virtual. De manera que, l'aplicació de xarxa del node en comptes d'estar utilitzant la interfície *Ethernet* física està escoltant/enviant el tràfic a la interfície virtual creada per l'injector i és aquest el que rep/envia informació a través de la interfície física.

Aquest sistema permet separar completament el que és la infraestructura d'injecció de fallades de l'aplicació de xarxa que s'executa en un node. De manera, que la injecció s'executa de forma transparent des del punt de vista del node. Sense interfícies virtuals per tal d'obtenir les trames, hauríem hagut de modificar el codi de l'aplicació de xarxa.

8.1.2 Mòdul d'Intercepció

Com hem explicat abans el mòdul injector de fallades funciona com un pont entre l'aplicació de xarxa del dispositiu on s'executa i la xarxa. Per realitzar aquesta tasca, el mòdul d'intercepció, intercepta les trames que arriben tant a la interfície física, provinents de la xarxa, com les trames que arriben a la interfície virtual, provinents de l'aplicació de xarxa del dispositiu. Ambdós casos les trames interceptades s'han d'analitzar per determinar si compleixen les condicions d'injecció.

8.1.3 Mòdul d'Anàlisi

Com hem dit, quan el mòdul injector rep una trama l'ha d'analitzar per saber si compleix les condicions necessàries per provocar l'error configurat. Si no les compleix, el mòdul d'anàlisi, desvia la trama al mòdul de transmissió, si les compleix, l'envia al mòdul de modificació.

Com que volíem una infraestructura d'injecció per *Ethernet* però amb opcions específiques pel protocol FTT hem dissenyat el mòdul d'anàlisi amb dos grups de condicions: Condicions per trames *Ethernet* en general i condicions per trames FTT específicament.

En qualsevol cas el mòdul d'anàlisi pot agafar com a condició els operands, major, menor, igual, diferent i un rang de valors.

Funcions per trames Ethernet en general

En el cas d'una trama *Ethernet* general els camps que s'analitzen buscant una condició són:

- Adreça destí
- Adreça origen
- Ethernet type
- Un *byte* qualsevol de la trama

Funcions específiques per FTT (FTT_TYPE 0x8ff0)

Si la trama és, especifica del protocol FTT, a més de les condicions anteriors es pot utilitzar la majoria dels camps de les trames TM, ASM, SDM definits per FTT. Tots els camps disponibles estan descrits més endavant en el capítol 9.

8.1.4 Mòdul de Modificació

Si el mòdul d'anàlisi determina que la trama capturada compleix les condicions d'injecció, es procedeix a la injecció. Com s'explica als requisits, la nostra infraestructura d'injecció ha de ser capaç de reproduir diferents fallades de comunicació, basades en la modificació del tràfic *Ethernet*.

Tipus d'errors que es poden provocar

- ***Drop:*** Si l'error que es vol provocar és *drop*, la trama que complia la condició es descarta i no s'envia al mòdul de transmissió. Simulant una pèrdua de la trama o error d'omissió.
- ***Allpossiblecases:*** Per tal de comprovar que el mecanisme de replicació de TMs, implementat en els protocols FTT amb SECSM, explicat a l'apartat 4.1, fa que els nodes siguin capaços de començar l'*Elementary Cycle* a la vegada, tot i la pèrdua de tots els TMs menys 1, vàrem afegir un tipus especial d'error a l'injector. Aquest error provoca tots els casos possibles de pèrdua de TM en un grup d'esclaus consecutivament. Començant per provar tots els casos possibles d'error per la interfície que identificada com a interfície 0, després tots els casos possibles de la interfície identificada com a 1, i així successivament. De forma que la prova és escalable segons el nombre d'interfícies que tinguem. Així la durada d'aquesta injecció varia amb el nombre d'enllaços de la xarxa i amb el nombre de rèpliques que s'envien durant la finestra de TM. El nombre de ECs que es triga a acabar aquesta injecció és igual a $((2^k-1)^s)$, on "k" és el nombre de rèpliques i "s" és el nombre d'enllaços.

Per posar un exemple. Si tenim una xarxa amb un màster i un esclau amb 2 rèpliques de TM, hem de provar $((2^2-1)^1)$ = 3 possibilitats: que es perdi la primera rèplica, que es perdi la segona o que no se'n perdi cap. Per tant, només haurem d'esperar tres *Elementary Cycles* per veure tots els casos possibles. Si en canvi tenim un mestre amb dos esclaus i 2 rèpliques tenim dos enllaços hem de provar $((2^2-1)^2)$ = 9 possibilitats. Seguint l'ordre de la taula 1 on tm_{ij} representa la rèplica "i" de TM de l'enllaç *j*.

EC	tm_{00}	tm_{01}	tm_{10}	tm_{11}
0	Arriba	Arriba	Arriba	Arriba
1	Es perd	Arriba	Arriba	Arriba
2	Arriba	Es perd	Arriba	Arriba
3	Arriba	Arriba	Es perd	Arriba
4	Es perd	Arriba	Es perd	Arriba
5	Arriba	Es perd	Es perd	Arriba
6	Arriba	Arriba	Arriba	Es perd
7	Es perd	Arriba	Arriba	Es perd
8	Arriba	Es perd	Arriba	Es perd

Taula 1: Casos possibles que ha de provar l'error allpossiblecases amb un prototipus amb dos esclaus i 2 rèpliques de TM per EC.

- ***Lock:*** L'error *lock* simula un tall temporal de comunicació entre dos nodes. El procediment que segueix és, descartar tots els paquets que arriben al dispositiu, des del moment que es compleix

la condició d'inici, fins que es rep una trama que compleix la condició de final.

- ***Change:*** Aquest error permet modificar un camp o un byte concret de la trama que compleixi la condició d'injecció abans de passar-la al mòdul de transmissió.

- ***Delay:*** Amb aquest error simulam un retard. Donada una trama i un temps en microsegons l'injector emmagatzema la trama el temps indicat abans transmetre-la. Mentre segueix transmetent les altres trames normalment

- ***Crash:*** L'error *crash* simula la caiguda irrecuperable d'un dispositiu de la xarxa. Si una determinada trama compleix la condició de *crash*, el mòdul d'injecció del dispositiu deixa d'enviar i rebre trames i passa a la fase de generació del fitxer de captures de final de l'experiment.

8.1.5 Mòdul de Transmissió

Tant en els casos en els quals el mòdul d'anàlisi determina que no s'ha de provocar cap error en una trama que arriba al mòdul injector de fallades, com en els casos que el mòdul de modificació provoca un error de *change* o *delay*, el mòdul de transmissió envia les trames, intactes o resultants, per la interfície de sortida. Si la trama prové de la xarxa, la transmet per la interfície virtual i si la trama prové de l'aplicació de xarxa, la transmet per la interfície física. En el, cas de trames que compleixin la condició d'injecció d'un tipus d'error diferent de *change* o *dealy* el mòdul de transmissió no ha de fer res, ja que aquests altres tipus d'error duen associat l'omissió de la trama.

8.1.6 Fitxers de trames capturades

Ja que en els requisits hem determinat que la nostra infraestructura d'injecció de fallades necessita un mòdul de captura de dades per possibilitar la seva posterior anàlisi, totes les trames enviades pel mòdul de transmissió s'emmagatzemen en un fitxer. Així, al final de l'experiment tenim un **fitxer de captures** per cada mòdul injector de fallades, el qual conté totes les trames que passen pels dos mòduls de transmissió (rx o tx) i una marca temporal associada a cada una d'aquestes trames.

Dispositiu de captura de tràfic netAnalyzer B500G-RE

Per poder capturar tràfic de la forma més acurada possible s'ha afegir al disseny un analitzador de tràfic que és capaç de capturar les trames obtenint una marca temporal amb una exactitud de fins a 10 ns.

Aquest analitzador disposa de dues interfícies anàlogues amb dos ports cada una. Per tal de capturar el tràfic d'un enllaç mestre-esclau s'ha de connectar un port *Ethernet* del *switch* HaRTES al primer port del capturador i el port *Ethernet* de l'esclau al segon de forma que el *netAnalyzer* quedi en "serie" entre els dos elements de la xarxa. Ja que el *netAnalyzer B500G-RE* disposa de dues interfícies (TAP A i TAP B) podem capturar el tràfic de fins a dos enllaços simultàniament. A la figura 15 es mostra quin és l'esquema de connexions que s'ha de seguir per capturar dos enllaços.

A més de les dues interfícies de captura l'analitzador s'ha de connectar amb un PC amb S.O. Windows a través d'un enllaç *Ethernet* de 1000Mbps on s'executa el software propi de l'analitzador.

Aquesta aplicació, s'utilitza per iniciar/finalitzar la captura i per anar emmagatzemant les trames al disc dur del PC. Una vegada finalitzada la captura el software genera un arxiu .pcap que es pot analitzar amb *Wireshark* o altres analitzadors.

Cal fer notar que, a més de l'augment en la resolució que proporciona el capturador, utilitzant aquesta configuració també aconseguim que les marques temporals de les trames capturades en els dos enllaços provinguin del mateix rellotge (el del *netAnalyzer*). Això no passa en el cas dels fitxers creats pels injectors de fallades. Ja que les marques temporals que contenen aquests fitxers s'agafen a partir del rellotge intern de la màquina on s'executa cada mòdul injector de fallades.

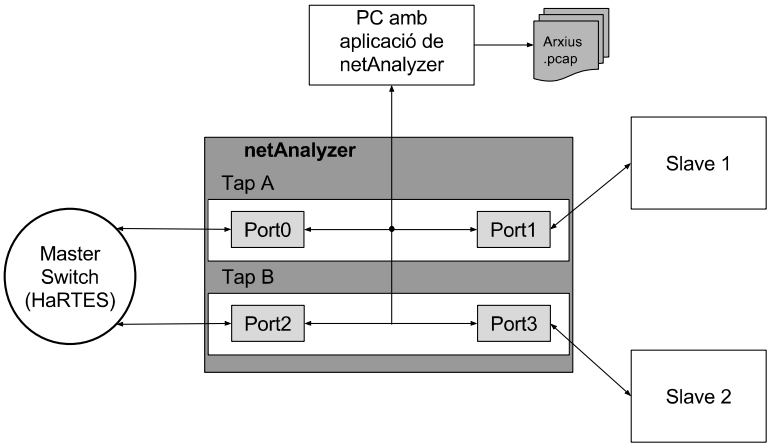


Figura 15: Esquema de connexions dispositiu de captura B500G-RE entre 2 enllaços mestre-esclau

8.2 Prototipus HaRTES amb la infraestructura d'injecció de fallades

Una vegada tenim el disseny de la nostra infraestructura d'injecció de fallades acabat per una xarxa *Ethernet* en general, concretam com quedaria integrada al prototipus explicat a l'apartat 5.

Com es pot veure a la figura 16, tenim 6 mòduls injectors controlats des de l'estació d'instrumentació. Tres mòduls al *switch* HaRTES, un per cada enllaç mestre-esclau, i un mòdul a cada esclau. Per facilitar la configuració i aprofitant que el PC tenia més capacitat de processament que els *barbones* vàrem decidir centralitzar la injecció de fallades en els mòduls injectors de fallades del *switch* HaRTES. Deixant així, els mòduls dels esclaus en mode captura, amb l'objectiu d'obtenir el tràfic de cada enllaç des del punt de vista dels esclaus. També vàrem afegir al prototipus el capturador *netAnalyzer* per tal de capturar trames en dos dels enllaços de la xarxa FTT com hem explicat a l'apartat anterior.

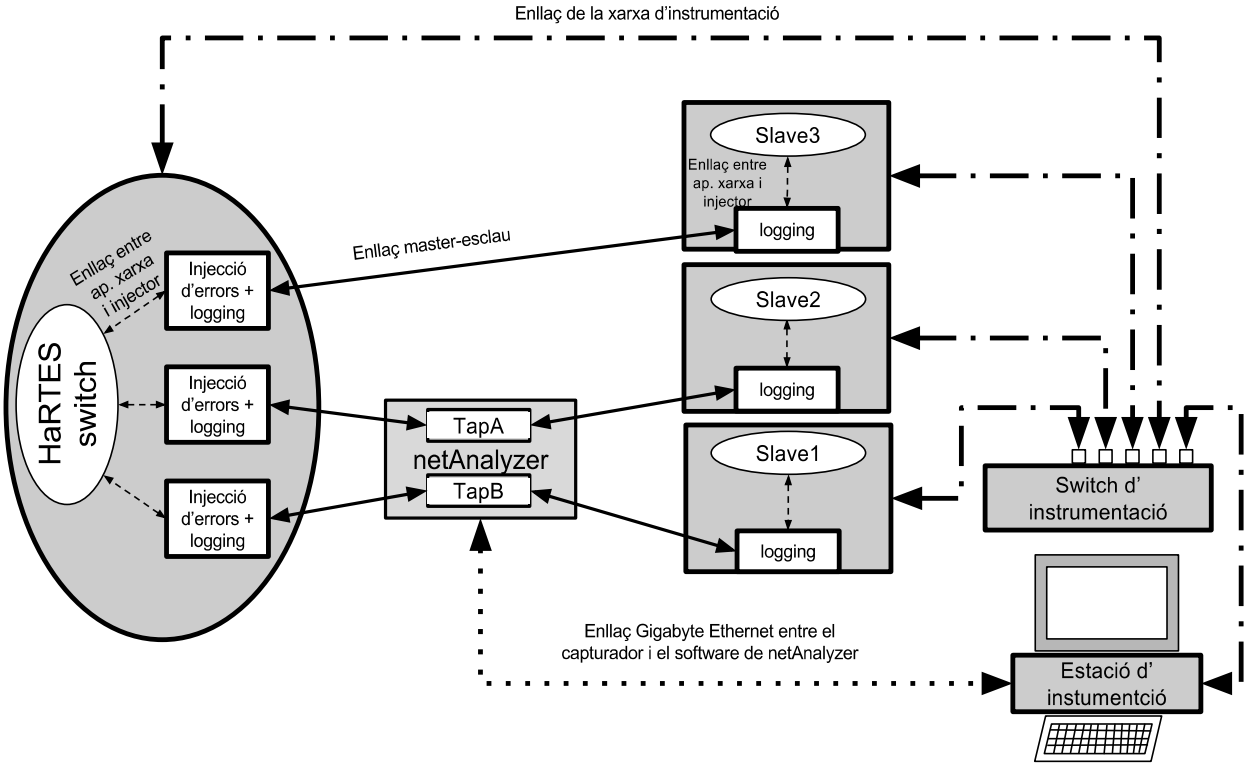


Figura 16: Prototipus HaRTES amb un mòdul de l'injector a cada interfície del switch en mode injecció i un mòdul de l'injector en cada esclau en mode captura i el netAnalyzer capturant el tràfic dos enllaços mestre-esclau.

9 Descripció detallada del llenguatge de configuració

En l'apartat de requisits hem comentat que l'usuari ha de ser capaç de crear escenaris de fallada diversos i configurar-los d'alguna manera en els mòduls injectors de fallades. Per això havíem de definir un llenguatge que permetés crear un arxiu de configuració que especificàs el comportament de cada injector que s'executàs durant un experiment.

Vàrem decidir utilitzar un llenguatge d'etiquetes basat en l'estructura d'un document XML. Tenim una jerarquia d'elements on cada element és una característica que necessita ser configurada. Els elements poden contenir valors directament o altres elements. A més alguns elements també poden tenir atributs. En aquest capítol s'explica la sintaxis que han de complir aquests fitxers de configuració que utilitzen els mòduls d'injecció de fallades. Explicant tots els elements permesos i enumerant els atributs de cada un d'aquests elements.

Per tant, en els arxius de configuració tenen un element arrel que conté tots les característiques. Aquest element és un contenidor amb el nom *<args></args>*. Perquè conté tots els arguments que necessita l'injector quan s'inicia per funcionar.

La configuració d'un injector de fallades de la nostra infraestructura d'injecció pot constar de 2 elements principals: El primer conté la informació que identifica el dispositiu on s'executa el mòdul injector de la infraestructura i les interfícies de les quals s'encarrega aquest mòdul i el segon que determina l'error que es vol provocar definint el tipus d'error, les condicions d'injecció...

9.1 Arguments que identifiquen el dispositiu on s'executa l'injector

L'element arrel d'aquest grup s'anomena *<device></device>*. Aquest i els 4 elements que conté són obligatoris i donen a l'injector la informació necessària per iniciar la seva execució en mode captura.

- Identificador del dispositiu: <iddevice></<idevice> que determina el nom que rep el dispositiu. Principalment aquest element s'utilitza per donar nom a l'arxiu de captura final.
- Interfície física: *<phyif></phyif>* que identifica el nom de la interfície física del dispositiu on aquest injector actuarà
- Interfície virtual: *<vif></vif>* que identifica el nom de la interfície virtual que ha de crear l'injector.
- Interfície d'instrumentació: *<instif></instif>* que identifica el nom de la interfície d'instrumentació del dispositiu on s'executa l'injector per on l'usuari controla l'injector.

L'estructura d'un document amb aquests elements seria la següent:

```
<args>
        <device>
                <iddevice>"nom del dispositiu"</device>
                <phif>"nom de la interfície física"</phif>
                <vif>"nom de la interfície virtual"</vif>
                <instif>"nom de la interfície d'instrumentació"</instif>
        </device>
</args>
```

Llista 1: Sintaxi dels arguments relacionats amb el dispositiu.

9.2 Arguments de configuració d'un mòdul injector de fallades

Una vegada definits els arguments necessaris per identificar el mòdul injector dins la infraestructura si volem que l'injector en mode injecció, hem de configurar els errors que volem que provoqui.

Ens interessava que un injector pogués provocar múltiples errors a la vegada i a més que ho pogués fer en els dos sentits de comunicació. Per això vàrem definir 2 elements arrel dins aquest grup. Un element que conté tots els arguments necessaris per provocar un error en el tràfic que transmet la interfície física *<tx_error></tx_error>* i un element que conté tots els arguments necessaris per provocar un error en el tràfic que rep aquesta mateixa interfície *<rx_error></rx_error>*. Aquests 2 elements poden contenir els mateixos elements. L'única diferència entre ells és el sentit del tràfic on es provoca l'error.

Si volem configurar el mòdul injector perquè injecti múltiples fallades només hem d'afegir tants d'elements *<rx_error>* o *<tx_error>* com vulguem. Com hem dit anteriorment si en un arxiu de configuració no hi ha cap element *rx_error* o *tx_error* el mòdul injector es troba en mode captura.

Un elements *rx_error* o *tx_error* com a mínim han de contenir l'element que determina el tipus d'error al qual fan referència *<type></type>*. Com a valor contingut dins *<type>* tenim 6 possibilitats, que són els tipus d'error descrits a l'apartat 8.1.4, *allpossiblecases, drop, lock, change, delay* i *crash*.

Després s'ha d'especificar el tipus de trama la qual es vol utilitzar com a condició d'injecció amb l'element <message_type> *</message_type>*. Els valors admesos en aquest element són "ETH" per trames *Ethernet* en general i per missatges específics de FTT tenim les opcions de "TM" per *trigger message*, "SDM" per missatges de dades síncrons i "ADM" per missatges de dades asíncrons.

Cal fer notar que el tipus de missatge no té sentit en el tipus d'error *allpossiblecases* perquè aquest error està dissenyat exclusivament per missatges "TM". En canvi hi ha un element específic per aquest tipus d'error que és *<idslave></idslave>*. El valor per aquest element ha de ser un enter que representa l'identificador de l'enllaç.

Per tots els errors menys *allpossiblecases* necessitarem un altre element que especifiqui la condició que ha de complir el tipus de missatge especificat a *<message_type>* perquè el mòdul d'anàlisi l'envii al mòdul de modificació. Aquest element és *<check></check>*. L'element *<check>* pot contenir dos tipus de condicions: el valor d'un camp específic del tipus de missatge o el valor d'un byte qualsevol de la trama.

En el cas que tinguem el valor d'un camp, com a atribut a l'element tenim la condició que ha de complir el camp. Pot ser igual al valor de l'element (E), diferent del valor de l'element(D), menor(L) , major(G), o dins un rang de valors donat(R). Sintaxi com a l'exemple de la llista 2

```
<nom del camp [atribut="E | D | L | G" | ( R"/ side="S | F")]>
        "valor de camp"
</nom del camp>
```

Llista 2: Valor d'un camp com a condició d'injecció.

Per a missatges *Ethernet* en general els noms del camp que s'admeten són: "eth_dest" per la MAC destí, "eth_src" per la MAC origen, "eth_type" per tipus de missatge *Ethernet*.

Per a missatges FTT:

- Si el missatge és un TM: "rlh_numTMs", "rlh_Tmseq", "rlh_interarrivel", "rlh_seq_no", "EC_time", "NSM", "NAM", "NSA".
- Si el missatge és un missatge de dades síncron o asíncron: "id","seq_no","flag_s","frag_seqno","source_nodeID"

En el cas que sigui un byte, l'element ha de contenir el "valor del byte" que s'ha de complir i un altre element que indica la posició del byte dins la trama. Un exemple de la sintaxis en la llista 3

```
<byte >
        "valor del byte"
        <pos_byte>"posició dins la trama"</pos_byte>
</byte>
```

Llista 3: Valor d'un byte de la trama com a condició.

És interessant notar que l'injector permet la utilització de diversos elements *<check>* dins un mateix element error i els interpreta com a diverses condicions a complir a la vegada.

A la llista 4 veiem un exemple que equivaldria a una condició=condició 1 *AND* condició 2. En el cas de voler una condició=condició 1 *OR* condició 2 hauríem d'utilitzar 2 elements error amb condicions diferents però injectant el mateix tipus d'error en els dos casos. Com en l'exemple de la llista 5.

```
.....
<check> condició 1 </check>
<check> condició 2 </check>
...
```

Llista 4: Exemple de condició1 AND condició2.

```
.....
<rx_error>
...
        <check> condició 1 </check>
...
</rx_error>
...
<rx_error>
...
        <check> condició 2 </check>
...
</rx_error>
...
```

Llista 5: Exemple de sintaxi de múltiples errors.

En el cas d'errors tipus "change" a més de l'element *<check>* necessitam un altre element per especificar quin camp o quin byte volem modificar a les trames que compleixin la condició del *<check>*. Aquest element és el *<change></change>*. Dins el contenidor change ens trobam les mateixes opcions que pel contenidor *<check>* la diferència és que aquests valors en comptes d'interpretar-se com una condició s'interpreten com el nou valor que ha de contenir la trama una vegada modificada. A la llista 6 tenim un exemple de sintaxi d'un error en recepció de tipus *change*.

```
.....
<rx_error>
        <type>change</type>
        <message_type>TM</message_type>
        <check> condició </check>
        <change> modificació </change>
</rx_error>
...
```

Llista 6: Tipus d'error change a missatges trigger message

Per altra banda, si l'error és de tipus "delay" en comptes de l'element *<change>* s'ha d'utilitzar l'element *<time_delay>* on el valor contingut s'interpreta com el nombre de microsegons que s'han de retardar les trames que compleixin la condició de *<check>*. Tenim un exemple a la llista 7.

```
.....
<rx_error>
        <type>delay</type>
        <message_type>TM</message_type>
        <check> condició </check>
        <time_delay> microsegons de retard </time_delay>
</rx_error>
...
```

Llista 7: Tipus d'error delay a missatges trigger message.

Per acabar, si l'error és de tipus "lock" amb l'element *<check>* començam a eliminar tot el tràfic rebut i s'ha d'introduir un nou element que indiqui la condició de desbloqueig. Aquest element és *<unlock>*. L'element *<unlock>* conté un altre element, *<name field>*, que és idèntic al *name field* contingut a l'element *<check>*. Tenim un exemple de sintaxis a la llista 8.

```
.....
<rx_error>
        <type>lock</type>
        <message_type>TM</message_type>
        <check> condició </check>
        <unlock> condició </unlock>
</rx_error>
...
```

Llista 8: Tipus d'error lock a missatges TM.

En la llista 9 es mostra el BNF de la sintaxi que tindria un arxiu de configuració amb tots els arguments possibles que un injector en el moment d'inicialitzar-lo reconeix. A l'apèndix B hi ha exemples d'arxius de configuració per escenaris de fallada concrets.

Configuracio.xml ::= **<args>** device {tx_error} {rx_error} **</args>**

device ::= **<device>** id_device physical_interface virtual_interface intrumentation_interface**</device>**

* *id_device* ::=**<iddevice>**device name**</iddevice>**
* *physical_interface* ::=**<phyif>**physical interface name **</phyif>**
* *virtual interface* ::=**<vif>**virtual interface name**</vif>**
* *intrumentation interface* ::=**<instif>**intrumentation interface name**</instif>**

tx_error ::= **<tx_error>** type_error type_msg [node_id] [{cond}] [{mods}] [unlock/error lock] **</tx_error>**

rx_error ::= **<rx_error>**type_error type_msg [node_id] [{cond}] [{mods}] **</rx_error>**

* type_error ::=**<type>** "*allpossiblecases*" | "drop" | "change" | "crash" | "delay" **</type>**
    ** *allpossiblecases* ::= all the possibilities of losing all the TMs but one for EC
    ** drop ::= we dismiss a frame
    ** lock ::= we dismiss all the messages of all types between check and unlock conditions.
    ** change ::= we change a field or a byte of the frame
    ** delay ::= we delay a frame X us
    ** crash ::= we drop all frames since this moment until the end of the experiment

* type_msg ::=**<message_type>** "TM" | "SDM" | "ADM" | "ASM" | "ETH" **</message_type>**
    ** TM::=FTT trigger message
    ** SDM::=FTT synchronous data message
    ** ADM::=FTT asyncronous data message
    ** ETH::=General *Ethernet* frame

* node_identifier ::= **<idslave>** 0|1|2|3|4 **</idslave>** (only for *allpossiblecases* error)

* cond ::= **<check>** field | byte **</check>**
    ** field ::= **<name field** [operator="E|D|L|G" | ("R" / side="S|F")]**>** value field **</name field>**
        *** E ::= equal; D ::= different
        *** L ::= lower; G ::= greater;
        *** R ::= range of values; S ::= Start of values range; F::=final of values range.
        *** name field ::=
        ("**TM**" / rlh_numTMs | rlh_TMseq | rlh_interarrival| rlh_seq_no | EC_time | NSM | NAM | NSA) |
        ("**SDM**" | "**ADM**" / id | seq_no | flag_s | frag_seqno | source_nodeID) |
        ("**ETH**" / eth_dest | eth_src | eth_type)
    ** byte ::= **<byte>** value byte **</byte>** **<pos_byte>** index byte **</pos_byte>**

* mods::=(**<change>** field to change | byte **</change>**) | (**<time_delay>** us delay **</time_delay>**)
    ** field to change ::= **<name field>** value field **</name field>**

* unlock::=**<unlock>** field check for unlocking **</unlock>**
    ** field to check for unlocking :: =**<name unlock field>** value field **</name unlock field>**
        *** name field ::= ("**TM**" / rlh_seq_no)

Llista 9: Sintaxi del fitxer de configuració dels mòduls injectors de fallades

Part III. Implementació

10 Infraestructura d'injecció de fallades.

En aquest capítol s'explica la implementació de la infraestructura d'injecció de fallades que s'ha dut a terme a partir del disseny explicat anteriorment. Com es pot veure a la figura 17 la infraestructura de fallades està formada per diferents mòduls. Primer, el **mòdul de control (MC)** permet a l'usuari interactuar amb la infraestructura d'injecció. Després tenim diversos **mòduls d'injecció de fallades i captura de dades (MIFCD)** cadascun dels quals es correspon amb el disseny d'un dels mòduls d'injecció de fallades explicat en l'apartat 8.1. És a dir, són els encarregats d'injectar les fallades a cada interfície de la xarxa prototipus i generar el fitxer de captures. Finalment, tenim un **mòdul de processament i anàlisi de dades (MPAD)** que s'encarrega de processar els diferents fitxers de captures i extreure'n la informació.

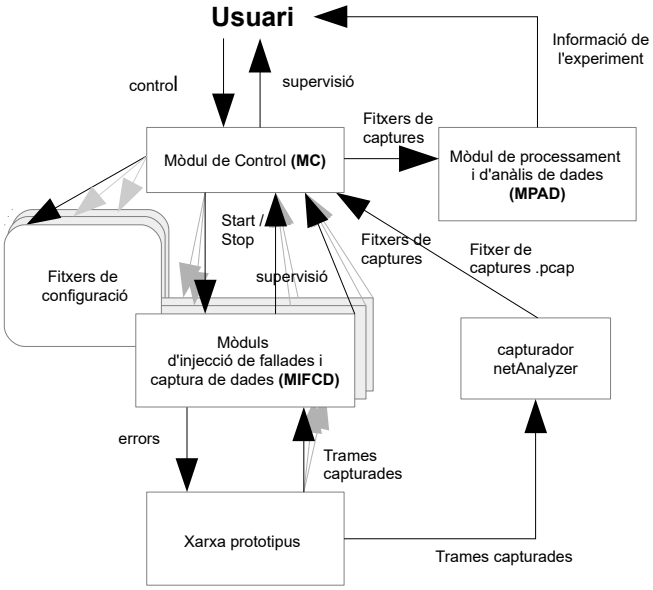


Figura 17: Implementació de la infraestructura d'injecció de fallades.

10.1 Mòdul de control

Com hem dit, el mòdul de control (MC) permet a l'usuari interaccionar amb la infraestructura d'injecció. En la nostra implementació aquest mòdul es troba allotjat a l'estació d'instrumentació i es comunica amb els altres mòduls utilitzant la xarxa d'instrumentació. Per a cada experiment l'usuari ha de realitzar diverses tasques des de l'MC:

- Abans d'iniciar l'experiment, l'usuari crea els arxius de configuració de cada MIFCD i amb l'MC els envia a l'element de xarxa on s'han d'executar. Després inicia els MIFCD i monitoritza la seva execució. Una vegada l'experiment ha finalitzat, l'usuari utilitza l'MC per recuperar els fitxers de captura que ha generat cada MIFCD a l'element on s'ha executat. Per realitzar aquest conjunt de tasques l'MC utilitza *secure shell* (ssh). L'usuari accedeix remotament a cada element de la xarxa a través de la interfície d'instrumentació.
- Per altra banda, l'usuari pot donar l'ordre de final d'experiment a tots els MIFCDs utilitzant

el mòdul de control. Per realitzar aquesta tasca, s'ha implementat un programa en C que envia una trama *broadcast Ethernet* especial als MIFCD, també a través de la xarxa d'instrumentació. Quan aquests reben la trama comencen el procés d'aturada. Aquest programa s'anomena *control/control.c* i s'executa a l'estació d'instrumentació.

- A més, el mòdul de control es comunica amb el capturador *netAnalyzer* utilitzant el software propietari d'aquest dispositiu a través d'una connexió *GigabitEthernet*. Aquest software propietari, com ja hem comentat, s'utilitza per començar la captura i per exportar-la a un fitxer de captures del tipus .pcap.

10.2 Mòdul d'Injecció de fallades i captura de dades (MIFCD)

Com hem explicat abans, existeix un MIFCD per a cada interfície de cada dispositiu del prototipus on es vulguin injectar fallades o realitzar captura de dades. Entrant més en detall, a cada interfície s'executa una instància de l'aplicació MIFCD, implementada a l'arxiu ***mifcd.c*** que es mostra a la figura 18. El comportament de cada instància ve determinat pel fitxer de configuració proporcionat per l'MC. L'aplicació MIFCD està implementada amb el llenguatge de programació C i compilat amb el compilador *GNU Compiler Colection* (gcc).

En aquest apartat s'explica com estan implementades les funcionalitats de cadascun dels MIFCDs.

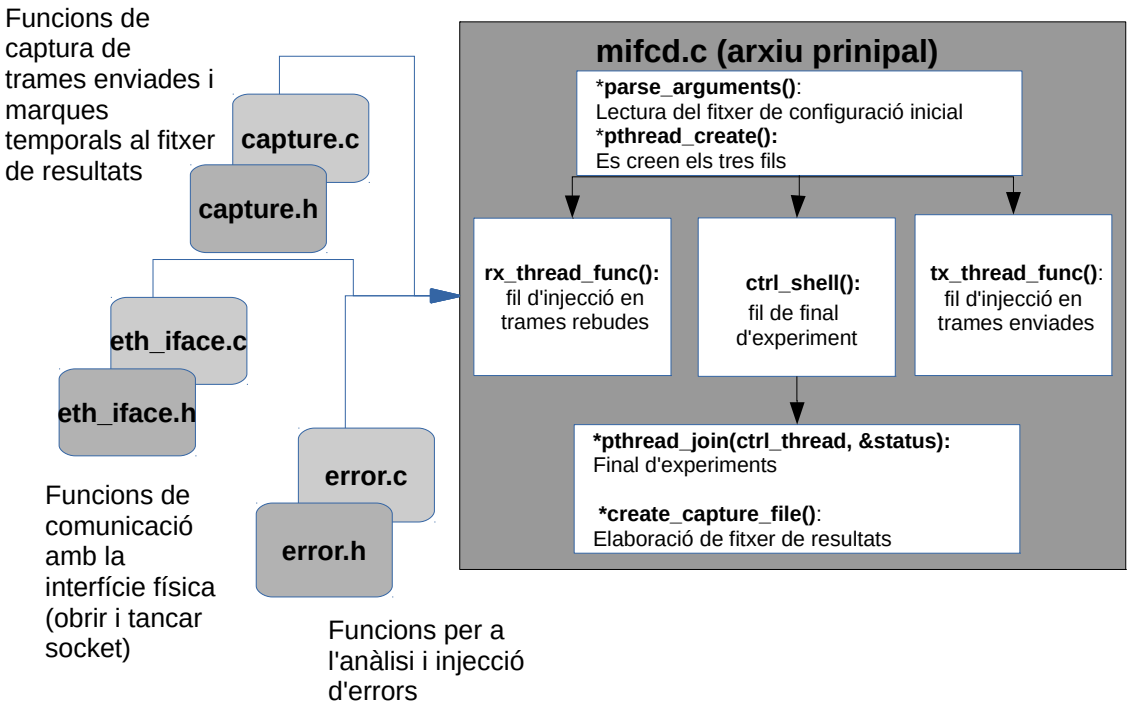


Figura 18: Diagrama dels arxius i funcions més importants que implementen un mòdul d'injecció de fallades i captures de dades

10.2.1 Estructura del programa principal

Una vegada el controlador inicia els MIFCDs, s'extreu la informació dels fitxers de configuració. Després s'estableixen els canals de comunicació amb les interfícies virtual i física. Tot seguit, per tal de millorar el rendiment de l'aplicació, es divideix l'execució del programa principal en tres fils:

- El primer, el qual s'encarrega d'interceptar, analitzar, modificar i capturar les trames provinents de la interfície física, s'anomena ***rx_thread***.
- El segon, el qual s'encarrega d'interceptar, analitzar, modificar i capturar les trames provinents de la interfície virtual, s'anomena ***tx_thread***.
- El tercer, què permet al controlador finalitzar l'execució de l'MIFCD, s'anomena ***ctrl_thread***. L'usuari pot iniciar el final d'execució a través de l'MC quan vulgui o pot esperar que els MIFCDs s'aturin automàticament perquè s'arriba al nombre màxim de trames que es volien capturar.

Una vegada s'arriba el final de l'experiment, cada MIFCD genera el fitxer de captures que utilitzarà l'MPAD per processar, analitzar i presenta la informació a l'usuari.

Per implementar els fils hem utilitzat la llibreria *pthread.h.* Les funcions específiques dels tres fils estan definides a l'arxiu principal anomenat *mifcd.c* i s'anomenen *rx_thread_func(), tx_thread_func()* i *ctrl_shell()* respectivament.

10.2.2 Configuració dels MIFCDs

Com hem explicat, per configurar cada MIFCD s'utilitza un fitxer de configuració el qual es passa com a paràmetre en el moment que el mòdul de control inicia el mòdul. Aquest arxiu s'ha de crear seguint la sintaxi especificada al capítol 9. El programa principal de cada MIFCD utilitza les llibreries *libxml/parser.h* i *libxml/tree.h* per extreure la informació del seu fitxer de configuració. Per guardar la informació estreta es defineix l'estructura *t_tx_thread_args*. S'utilitzen dues estructures d'aquest tipus: una pel fil *tx_thread* i una altra pel fil *rx_tread. El seu nom és tx_thread_args* i *rx_thread_args* respectivament. La funció encarregada d'extreure la informació de l'arxiu de configuració s'anomena *parse_arguments().* Les funcions i estructures anteriors estan definides directament a l'arxiu principal mifcd.*c.*

10.2.3 Intercepció i transmissió de trames.

Com s'explica al capítol 8.1 cada MIFCD intercepta, amb el **mòdul d'intercepció,** el tràfic provinent de l'aplicació de xarxa, realitza les tasques que té configurades i envia, amb el **mòdul de transmissió**, el tràfic resultant a la xarxa o viceversa.

Per implementar-ho utilitza una interfície virtual com a interfície de comunicació amb l'aplicació de xarxa del dispositiu on s'executa i una interfície física per comunicar-se amb la xarxa prototipus.

Per crear la interfície *Ethernet* virtual hem utilitzat el mòdul de *Linux TUN/TAP* a través de la llibreria *linux/if_tun.h* que permet crear interfícies virtuals de capa d'enllaç (TAP) i de xarxa (TUN). Una vegada creada aquesta interfície virtual s'utilitza exactament igual que qualsevol interfície

Ethernet del dispositiu.

Amb la funció *open_sckt()*, definida a l'arxiu eth_iface.c, l'MIFCD obre un *socket* a la interfície física per poder enviar i rebre trames de la xara prototipus. D'altra banda, la instrucció *tun_alloc()*, que es troba en l'arxiu principal de l'injector, permet crear la interfície virtual (TAP) per poder enviar i rebre trames de l'aplicació de xarxa [1].

10.2.4 Anàlisi i injecció de fallades

Cada MIFCD és l'encarregat d'injectar fallades a la interfície que té associada. Cada vegada que intercepta una trama, analitza si la trama compleix les condicions d'injecció. Si les compleix, provoca l'error i si no, la transmet directament. En el capítol de disseny vàrem anomenar el mòdul encarregat de realitzar aquesta anàlisi de trames **mòdul d'anàlisi** i l'encarregat de realitzar aquesta injecció **mòdul de modificació**. En aquest apartat s'explica quines funcions s'utilitzen per implementar aquests dos mòduls.

Les funcions que s'utilitzen per comprovar si una trama compleix les condicions d'injecció i, si cal, realitzar la injecció de fallades es troben definides a l'arxiu ***/libs/errors.c***. La funció, on s'analitza si les trames compleixen la condició d'injecció, s'anomena *check_fields()*. Una vegada el mòdul d'anàlisi determina que la trama compleix la condició d'injecció, el mòdul de modificació executa la funció associada al tipus d'error. *crash_machine()* per l'error de ***crash***, *delay_packet()* per l'error de ***delay***. *change_fields()* per l'error ***change*** i *filter_TM()* per l'error *allpossiblecases*. Pel cas d'error ***drop*** i ***lock*** no hi ha una funció específica definida dins error.c, ja que, en els dos casos, per injectar la fallada només és necessari no transmetre la trama si la funció *check_fields()* determina que és una trama que compleix la condició d'injecció.

Per guardar les condicions d'injecció, el tipus d'error i els camps o bytes a què afecta un determinat error definim l'estructura ***t_error*** a l'arxiu ***/libs/error.h***. En cada una de les estructures t_tx_thread_args introduïdes en l'apartat 10.2.2 declaram tres vectors d'aquesta estructura: la primera per guardar les condicions d'injecció dels diferents "*error[MAX_NUM_ERRORS]*", la segona per guardar les modificacions que s'han de fer a les trames "*error_changes[MAX_NUM_ERRORS]*", i la tercera que només s'utilitza quan la condició d'injecció és un rang "*error_range[MAX_NUM_ERRORS]*" (en aquest cas a la primer s'hi guarda el valor de condició més baix i a la tercera el valor de la condició més alt).

Quan alguna d'aquestes funcions fa referència a un camp determinat d'una trama, sigui Ethernet en general o FTT en particular, s'utilitza l'arxiu ***ftt-global.h*** on es defineixen les capçaleres de totes les trames que utilitza el prototipus HaRTES explicat al capítol 5. Cal remarcar que el codi d'aquest arxiu no és part d'aquest projecte sinó que està agafat directament del codi del prototipus.

10.2.5 Fitxer final de trames rebudes i enviades

Cada vegada que un MIFCD transmet una trama, per la interfície virtual o la física, la guarda en una estructura anomenada *frames* del tipus *t_frame[]* definida a l'arxiu ***capture.h*** una vegada l'experiment finalitza aquest vector d'estructures es guarda en un fitxer de captures, que com hem explicat l'usuari recupera del dispositiu de xarxa mitjançant l'MC. El nom de l'arxiu resultant és "*Cap_frames_[identificador del node]_[identificador de la interfície física].txt*"

El cada estructura del vector *frames* està format per 4 camps: la trama, la longitud de la trama, si la trama arriba per la interfície virtual o la física i la marca temporal de captura. La marca temporal s'obté del rellotge intern de la màquina on s'executa l'MIFCD mitjançant la funció *clock_gettime()* de la llibreria ***time.h*** i es guarda en una estructura *timespec*. Les funcions per realitzar les tasques de *logging* es troben definides a l'arxiu ***/lib/capture.c***

10.3 Mòdul de processament i anàlisi dades (MPAD)

Com ja hem comentat, una vegada finalitzat un experiment cada MIFCDs genera un fitxer de captures i és necessari tractar-los. Segons l'experiment cal ajuntar-los, o extreure'n les dades necessàries per elaborar una gràfica dels resultats o estimar estadísticament algun paràmetre de l'injector o del sistema que s'està provant. Per aquest propòsit em utilitzant diversos scripts de bash on majoritàriament hem utilitzant llenguatge *GAWK*. *GAWK* és la versió utilitzada a *GNU Linux* de l'*AWK* el qual és un llenguatge de programació de *UNIX* amb funcions específiques per manipular dades i generar informes de resultats. En aquests scripts també hi ha altres comandes de Linux com *cat, grep, parse, ...*

A més dels fitxers de captura provinents dels MIFCDs, si s'ha utilitzat el *netAnalyzer* , l'MPAD ha de processar i analitzar el fitxer de captures .pcap que genera el software d'aquest capturador. En aquests scripts a més de les eines comentades al paràgraf anterior vàrem utilitzar l'analitzador de tràfic de *Tshark*. El Tshark és un analitzador de protocols de xarxa de línia de comandes que permet, entre altres coses extreure informació dels arxius .pcap. Cal fer notar que només vàrem utilitzar el *Tshark* per extreure els camps de trames capturades que ens interessaven dels arxius .pcap per poder-los processar en les instruccions posteriors del script. En cap cas vàrem utilitzar les funcions d'analitzador o capturador de tràfic que inclou aquesta eina.

L'usuari executa, l'MPAD directament a l'estació d'instrumentació, que és on amb l'MC, ha recollit tots els fitxers de captura que necessiten els scripts de l'MPAD.

Part IV. Tests i experimentació

11 Test del comportament de la infraestructura d'injecció de fallades

Una vegada tenim la implementació de la nostra infraestructura, necessitam obtenir dades sobre l'impacte que produeix la seva introducció en un sistema que es vol avaluar. Aquestes dades serviran com a referència a l'hora d'analitzar les dades extretes en qualsevol experiment on s'utilitzi la nostra infraestructura i així poder diferenciar si un resultat és causat per la injecció de fallades o a causa, directament, per l'execució dels mòduls d'injecció dins els sistema que s'avalua.

En aquest capítol es descriuen els experiments que hem realitzar per a mesurar l'impacte que provoca el nostre injector en el prototipus d'una xarxa. Més concretament, mesuram quin és el retard i el *jitter* que introdueixen l'MIFDs quan es despleguen al prototipus.

Les mesures s'han duit a terme en dos escenaris diferents. En el primer s'avalua l'impacte de l'injector a una sola interfície, usant un programa de transmissió fet a mida. En el segon, dos MIFCDs es despleguen juntament amb el mestre per tal d'avaluar l'impacte amb dues interfícies.

11.1 Proves de retard introduït per l'injector.

El primer que volíem saber era el retard i el *jitter* que s'introduïa pel fet que les trames hagin de viatjar a través dels MIFCDs. Per fer aquesta prova, hem implementat una aplicació anomenada *SentTest* que envia una trama Ethernet cada 1 ms a una interfície virtual. Aquesta interfície es connecta a un MIFCD, que processa la trama i la transmet per una interfície física.

Tal com es mostra a la figura 19, cadascuna d'aquestes aplicacions realitza diverses tasques de manera seqüencial. En aquest sentit, la sortida de cadascuna d'aquestes tasques defineix un instant de temps rellevant: T1 és és l'instant en que l'aplicació transmet la trama per la interfície virtual, T2 és l'instant en què l'MIFCD acaba de rebre la trama, T3 és l'instant en que l'MIFCD acaba de processar la trama i T4 és l'instant en que l'MIFCD transmet la trama per la interfície física.

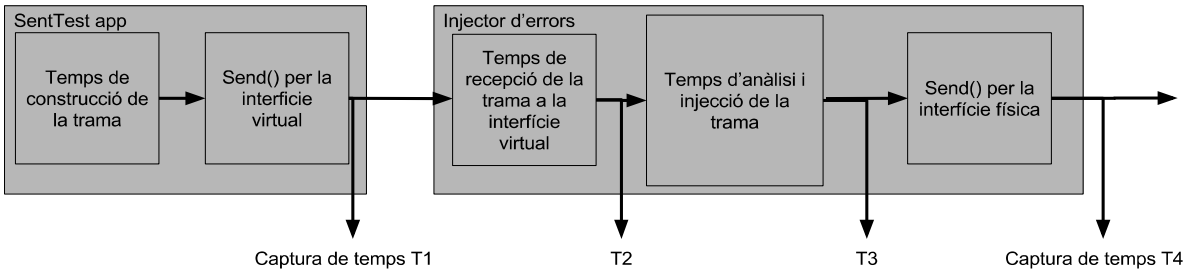


Figura 19: Diagrama temporal de la creació i l'enviament d'una trama per una interfície.

En el primer experiment utilitzam l'aplicació *SentTest*, en absència de l'MIFCD, i així tenir una referència del determinisme del sistema. Seguidament, en els dos experiments posteriors, mesuram el temps que transcorre des que l'aplicació *SentTest* transmet una trama (T1) fins que aquesta és transmesa per l'MIFCD a través de la interfície física (T4). Aquest experiment es realitza amb, i sense, regles d'injecció, per tal d'avaluar el pes que té la lògica necessària per detectar les trames que han de ser injectades i per realitzar les accions corresponents.

11.1.1 Mesura sense injector de fallades

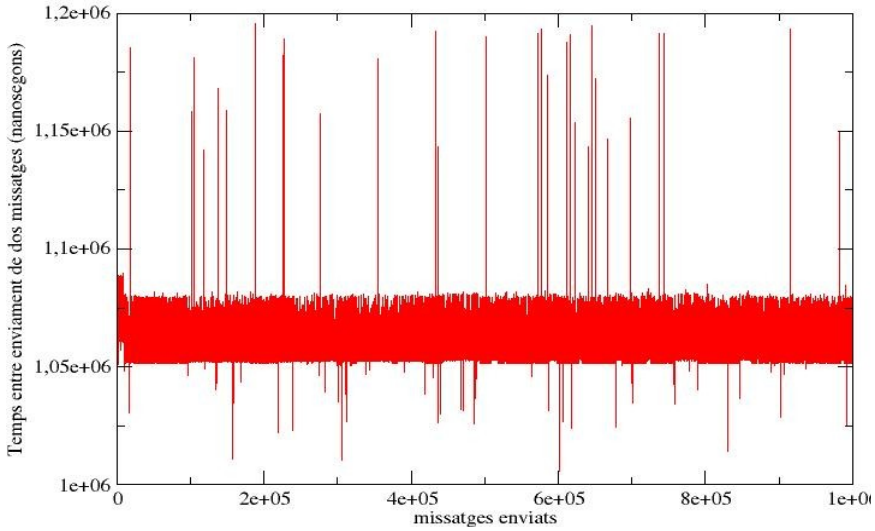
Com hem explicat, en aquest primer pas vàrem decidir fer un experiment inicial per tal de mesurar el determinisme del sistema i així tenir una referència amb la qual comparar els resultats dels experiments posteriors. En aquest experiment capturam l'instant de temps en què cada trama és transmesa, és a dir, capturam T1 per a totes les trames. Per tant, no desplegam cap MIFCD.

En un entorn ideal la diferència de temps entre dues captures consecutives seria 1 ms, que és el temps que l'aplicació *SentTest* deixa entre transmissió i transmissió. No obstant això, a causa de l'indeterminisme del sistema operatiu, sempre hi ha una variabilitat en la resposta dels programes. Per tant, aquest temps entre transmissions no serà exactament 1 ms. El que ens interessa és determinar el valor en temps d'aquesta variabilitat, o *jitter.*

L'experiment s'ha duit a terme per a 1 milió de trames i els resultats es mostren a continuació. A la taula 2 podem veure que el temps entre missatges mitjà és d'1,07 ms amb una desviació de 3,1 µs. Aquest temps addicional, és el temps que necessita l'aplicació per generar i enviar cada missatge. No obstant això, si observam el màxim, podem veure que l'aplicació pot arribar a necessitar 0,2 ms més. A la gràfica 1 mostram tots els temps entre parells de missatges enviats consecutivament i podem observar que, tot i que, en la majoria dels casos el temps entre missatges s'acosta a la mitjana, en algunes ocasions trobam pics de retard.

Mitjana (ms)	Desviació estàndard (µs)	Màxim (ms)
1,07	3,1	1,2

Taula 2: Anàlisi estadístic del temps entre la transmissió de dos missatges a l'aplicació SendTest.



Gràfica 1: Temps entre l'enviament de dos missatges de l'aplicació SendTest.

11.1.2 Mesura amb injector de fallades en mode captura

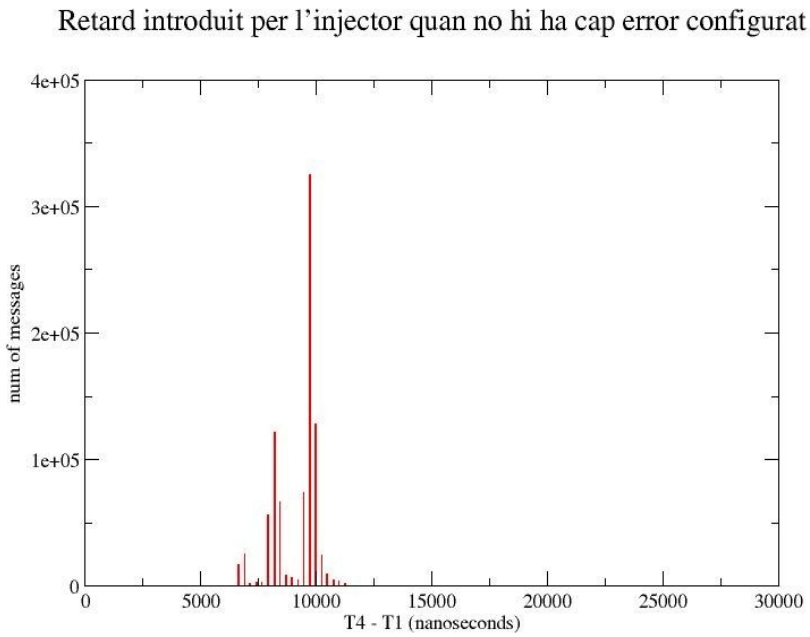
En aquest experiment mesuram el retard introduït per l'MIFCD quan no hi ha regles d'injecció. És a dir, tenim en compte el temps que necessita l'aplicació SentTest per transmetre la trama a través de la interfície virtual, el temps que tarda l'MIFCD a rebre-la i el temps que necessita per transmetre-la a través de la interfície física.

Els resultats obtinguts es mostren a continuació. La taula 3 resumeix els resultats estadístics. Més concretament, podem observar que usar una interfície virtual i processar la trama a una altra aplicació afegeix un retard de 8,57 µs amb una desviació de 0,6 µs. Tenint en compte els requisits temporals dels prototipus, són uns valors acceptables. No obstant això, podem veure que el màxim retard és un valor bastant elevat. Això indica un *jitter* important que ha de ser estudiat més en profunditat.

És per això que a la gràfica 2 mostram l'histograma dels valors registrats. Entrant més en detall, podem observar que més del 30% de les mostres obtingudes (300.000 trames d'un milió) tenen un valor de 10 µs. A més a més, la gran majoria es troben entre 6 i 11 µs. Valors fora d'aquest rang són minoritaris. Com veiem el màxim en aquest experiment també es desvia molt de la mitjana, això és perquè arrosseguem el problema dels pics que ja hem comentat en l'experiment anterior i que associam al mateix sistema on s'executa l'MIFCD més que al mòdul.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
8,57	0,6	27,9

Taula 3: Anàlisis estadístic del retard introduït per l'MIFCD.



Gràfica 2: Histograma del retard introduït per l'MIFCD sense regles d'injecció.

11.1.3 Mesura amb injector de fallades en mode injecció

En aquest experiment vàrem configurar l'injector de fallades perquè introduís errors per tal d'afegir el temps d'anàlisi i injecció al retard calculat a l'experiment anterior. Per això vàrem repetir l'experiment anterior però amb l'injector de fallades en mode injecció. Concretament el vàrem configurar perquè provoqués l'error *allpossiblecases*. Vàrem escollir aquest cas en particular perquè, a causa de la seva implementació, suposam que és el que introdueix més retard i així tenim la mesura del pitjor cas.

Per tal de diferenciar com influeix l'MIFCD quan el mòdul d'anàlisi decideix si es compleix la condició d'injecció o no, vàrem fer dues proves. A la primera vàrem fer que *SendTest* enviés trames del tipus SDM què no tenen relació amb el tipus d'error escollit, així, el mòdul d'anàlisi les enviarà directament al mòdul de transmissió sense passar pel mòdul de modificació. A la segona vàrem enviar trames TM de forma que totes les trames serien detectades com a susceptibles a injecció. Així és més evident el retard introduït pel mòdul d'anàlisi i pel mòdul de modificació.

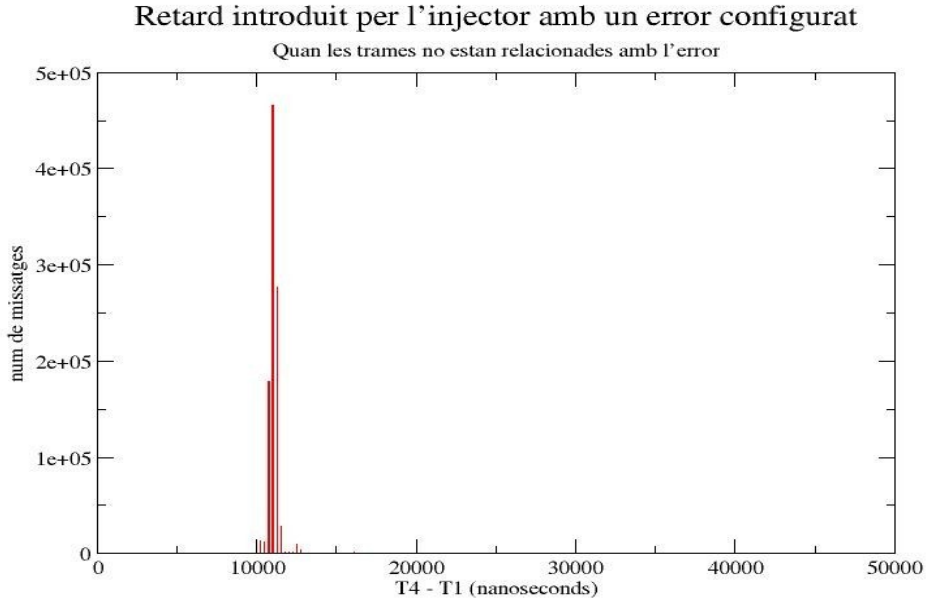
Experiment amb trames SDM

Els resultats obtinguts durant la prova transmetent trames que no complien la condició d'injecció es mostren a continuació. La taula 4 resumeix els resultats estadístics. Més concretament, podem observar que l'MIFCD afegeix un retard a la transmissió de trames d'11,11 µs amb una desviació típica d'1,49 µs. Com calia esperar, l'MIFCD introdueix més retard en la transmissió quan analitza totes les trames que intercepta que en el cas estudiat anteriorment on només les capturava. Però l'augment continua sent acceptable tenint en compta els requisits temporals del prototipus.

Com en el cas anterior, el temps màxim és molt superior a la mitjana el que indica un *jitter* important. A la gràfica 3 podem veure l'histograma de tots els valors capturats. De la mateixa manera que en l'experiment anterior veiem que la majoria de valors registrats estan propers a la mitjana, concretament tenim més d'un 45% dels (450.000 trames de milió) tenen un valor d'11 µs i la gran majoria es troba entre 10 i 13 µs. De manera que la gran diferència entre el màxim i la mitjana es pot tornar a atribuir als pics que ja vàrem detectar al primer experiment i que associam al sistema on s'executa l'MIFCD.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
11,11	1,49	131,07

Taula 4: Anàlisi estadístic del retard introduït per l'MIFCD
quan la trama analitzada no compleix la condició d'injecció.



Gràfica 3: Retard introduït per un MIFCD en trames que no compleixen les condicions d'injecció.

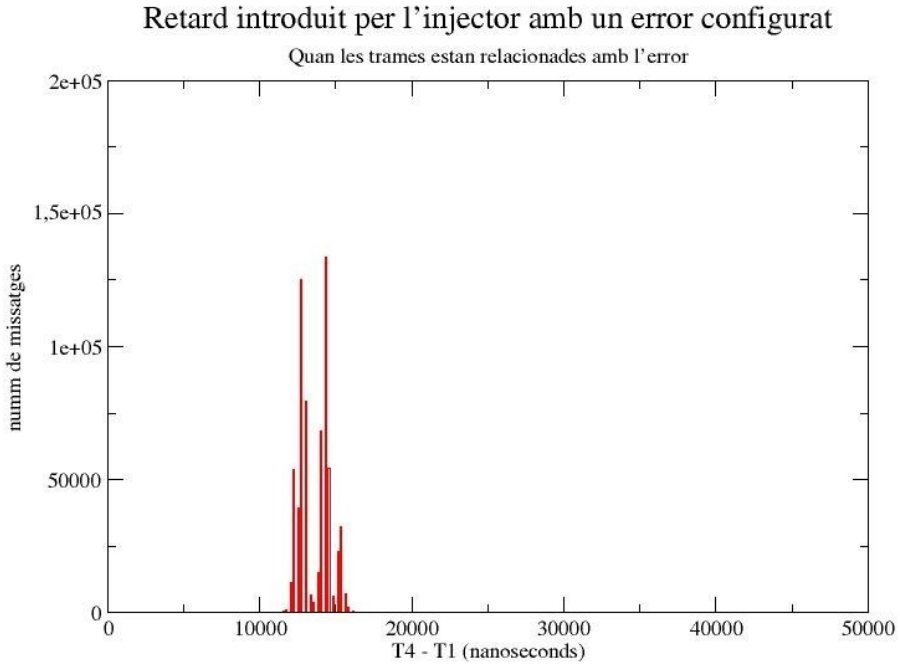
Experiment amb trames TM

Per acabar, vàrem realitzar la prova enviant trames TM que complien la condició d'injecció. Els resultats obtinguts es mostren a continuació. La taula 5 resumeix els resultats estadístics. Més concretament, podem observar que l'MIFCD afegeix un retard a la transmissió de trames de 13,67 µs amb una desviació típica de 1,07 µs. Com calia esperar l'MIFCD introdueix més retard en la transmissió quan ha d'injectar la fallada abans de transmetre les trames. Però l'augment continua sent acceptable tenint en compta els requisits temporals del prototipus.

Com en el cas anterior, el temps màxim és molt superior a la mitjana el que indica un *jitter* important. A la gràfica 3 podem veure l'histograma de tots els valors capturats. A més, podem observar que en la majoria de trames l'MIFCD introdueix un retard d'entre 12 µs i 15 µs i que només en alguns casos puntuals el retard és molt superior.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
13,67	1,07	41,73

Taula 5: Anàlisi estadístic del retard introduït per l'MIFCD quan la trama analitzada compleix la condició d'injecció.



Gràfica 4: Retard introduït per un MIFCD en trames que compleixen les condicions d'injecció.

11.1.4 Conclusions

Comparant els histogrames i les taules dels 3 experiments veiem, com esperàvem, que a mesura que l'MIFCD ha de realitzar més tasques, el retard que incorpora en la transmissió també augmenta. Concretament un mòdul d'injecció afegeix un retard de mitjana, en cada trama, d'entre 8,5 i 14 µs, depenent de si està, en mode captura o en mode injecció. Aquests valors són tolerables segons els requisits del prototipus.

Cal comentar que el retard màxim és a causa de l'execució de l'injector sobre un PC i té més relació amb les aturades degudes a tasques pròpies del S.O., que no amb el temps d'execució de la nostra aplicació. El que provoca una desviació estàndard tan diferent depenent de l'experiment. Això s'ha de tenir en compte a l'hora d'interpretar el resultat dels experiments posteriors. A més s'han de configurar els paràmetres de transmissió del prototipus de manera que aquests pics no influeixin en els experiments.

Aquest fet és degut a la implementació software del sistema i a que aquesta encara no ha estat optimitzada per sistemes del temps real. Es preveu que els tècnics encarregats del prototipus del projecte FT4FTT milloraran la resposta de temps real per a la implementació final. Això hauria de millorar el resultats relacionats amb el *jitter* mostrats en aquest capítol.

11.2 Mesura de *l'offset* de la sortida del primer TM del HaRTES cap a 2 esclaus

Seguint amb els experiments destinats a mesurar l'impacte dels mòduls d'injecció, executam dos MIFCDs dins el *switch*, que ens serviran per a provocar l'omissió de TMs transmesos a dos esclaus connectats al *switch*. En un entorn ideal els dos esclaus rebrien el mateix TM al mateix instant, és a dir, l'*offset* entre l'enviament de TMs seria 0. No obstant això, per culpa de la implementació software del prototipus, hi ha desviacions de temps a l'hora de transmetre el mateix missatge a través de diverses interfícies.

Seguint la lògica de l'experiment anterior, vàrem dividir l'experiment en tres parts: En el primer vàrem capturar el tràfic a la sortida del *switch* sense executar l'injector. En el segon vàrem tornar a capturar el tràfic a la sortida del *switch* però aquest cop, executant l'injector en mode captura. Finalment, en el tercer vàrem tornar a capturar el tràfic però aquest cop executant l'injector en mode injecció. Així, agafant el primer experiment com a referència, podem veure com afecta cada mòdul de l'injector a la xarxa.

A diferència dels experiments del capítol anterior, ara és necessari registrar els instants de temps en dos dispositius diferents, els esclaus. No obstant això, els seus rellotges funcionen de manera independent i, per tant, no podem relacionar les marques de temps obtingudes. És per això que hem fet el muntatge que es mostra a la figura 20. Més concretament, entre el *switch* i els esclaus col·locam el *netAnalyzer*, que ens serveix per registrar l'instant de sortida dels TMs. A partir d'aquestes dades podem calculam l'*offset* mitjançant un post-procés.

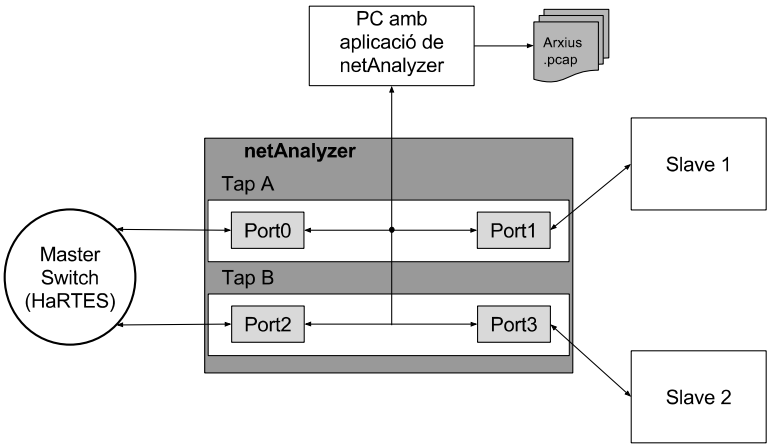


Figura 20: Esquema de connexions dispositiu de captura B500G-RE entre 2 enllaços mestre-esclau

Des del punt de vista de la transmissió de TMs, se segueix l'esquema descrit al capítol 4.1. El *switch* HaRTES envia a cada esclau *k* rèpliques de TM per tal de sincronitzar l'inici de cada EC. En aquests experiments definim l'*offset* en la transmissió de TMs com la diferència entre l'instant en què s'envia la primera rèplica de TM d'un EC a l'esclau 1 i l'instant en què s'envia la primera rèplica a l'esclau 2.

Per a un EC concret:

$$Offset\ de\ TM_k\ l_1 l_2 = \left| Ttm_{kl_1} - Ttm_{kl_2} \right|;$$

on és l'offset de transmissió de la rèplica k de TM entre els enllaços mestre-esclau, és l'instant en què es captura la rèplica de TM k a l'enllaç mestre-esclau 1 i és l'instant en què es captura de la rèplica k de TM a l'enllaç màster esclau 2.

11.2.1 Mesura de l'*offset* sense injector de fallades

En el primer experiment vàrem configurar el *switch* HaRTES perquè comuniqués amb dos esclaus enviant 4 rèpliques TMs per sincronitzar ECs de 10ms. Vàrem capturar el tràfic de 100000 ECs de cada un dels dos enllaços mestre-esclau.

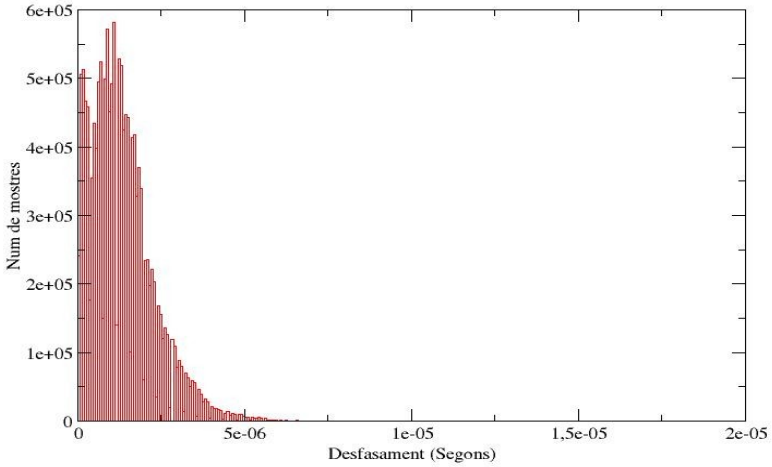
Una vegada acabada la captura vàrem processar les dades obtingudes del fitxer .pcap amb un script per tal de determinar l'offset en la transmissió de TMs de cada EC, així com alguns valors interessants que es mostren a continuació . A la taula 6 tenim els resultats de l'anàlisi estadístic realitzat per l'MPAD. Com veiem hi ha una *offset* de 2,93 µs de mitjana entre l'enviament dels dos missatges. El que sembla un valor acceptable segons els paràmetres de transmissió configurats. Però la desviació estàndard de 38,63 µs i, sobretot, el valor de 1597,59 µs semblen, a priori elevats i s'han d'estudiar a fons.

$$Offset\ de\ TM_0\ l_1 l_2 = \left| Ttm_{0l_1} - Ttm_{0l_2} \right|;$$

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
2,93	38,63	1596,59

Taula 6: Anàlisi estadístic de l'offset de la primera rèplica enviada per l'enllaç 1 i l'enllaç 2 sense injector de fallades.

Per aquest motiu, vàrem elaborar la gràfica 5. En aquest histograma podem observar els *offset*s calculats per cada EC agrupats segons el seu valor. Com veiem la gran majoria de rèpliques s'envien amb un *offset* menor a 3 µs i només en casos aïllats s'arriba al màxim calculat. Aquests pics es deuen a la implementació software del *switch* HaRTES.



Gràfica 5: Histograma de l'offset entre la primera rèplica de missatge TM de dos enllaços mestre-esclau

11.2.2 Mesura de l'*offset* amb injector de fallades en mode captura

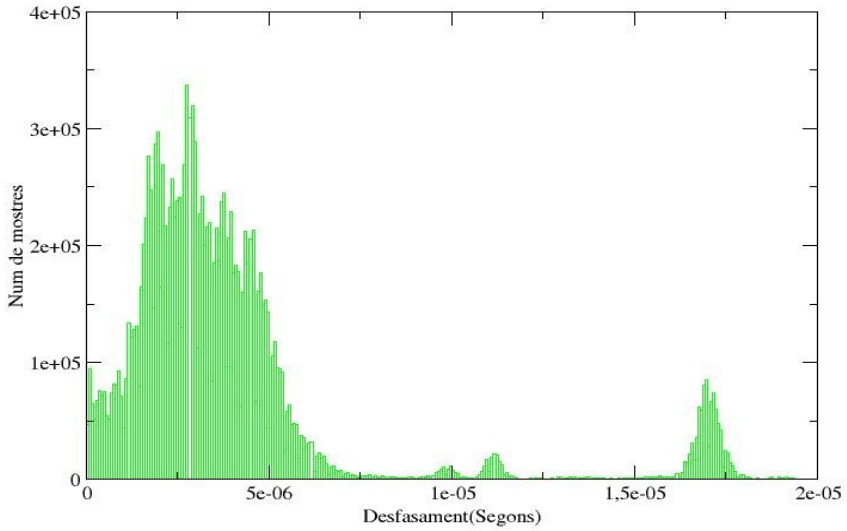
Partint de les dades obtingudes a l'apartat 11.2.1 com a referència, vàrem afegir al prototipus la infraestructura d'injecció de fallades. Concretament vàrem executar dos MIFCDs al *switch* un per cada enllaç mestre-esclau. En aquest experiment no vàrem configurar cap tipus d'error a cap dels dos injectors, deixant-los només en mode captura. Per així, poder veure com afecte l'injector al comportament del prototipus sense injecció de fallades.

Com en el cas anterior vàrem capturar 100.000 ECs, de 10ms cada un, amb 4 rèpliques de TM. Igualment, vàrem aplicar el mateix script de l'MPAD per extreure, processar i analitzar les dades, dels arxius .pcap generats pel software del capturador de tràfic.

A la taula 7 tenim els valors extrets de l'anàlisi de les trames capturades pel *netAnalyzer.* Concretament podem veure que l'*offset* mitjà entre les dues primeres rèpliques de TM a un EC és de 5,47 µs amb una desviació estàndard de 54,54 µs. Si comparam aquesta dada amb l'experiment anterior, ens indica que el procés d'intercepció de trames i la seva transmissió per a la interfície de l'MIFCD incorpora un *offset* addicional de 2,54 µs. Com veiem el màxim no varia significativament del cas anterior, ja que es deu a l'excussió del S.O. del PC on tenim el *switch* més que al funcionament del propi *switch.* Així i tot s'ha de comentar que, observant l'histograma de la gràfica 6, veiem que el nombre d'ECs on l'offset és superior a la mitjana augmenta considerablement quan s'afegeix la infraestructura d'injecció al switch sobre un 5% dels ECs tenen un *offset* d'entre 15 i 20 µs. No hem trobat un patró que expliqui l'augment en aquest 5% de casos.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
5,47	54,54	1599,7

Taula 7: Anàlisi estadístic de l'offset de la primera rèplica enviada per l'enllaç 1 i l'enllaç 2 amb mòduls injectors en mode captura de dades.



Gràfica 6: Histograma de l'offset entre rèplica de missatge TM de dos enllaços mestre-esclau amb un injector en mode captura per cada enllaç executant-se al switch.

11.2.3 Mesura de l'*offset* amb injector de fallades en mode injecció

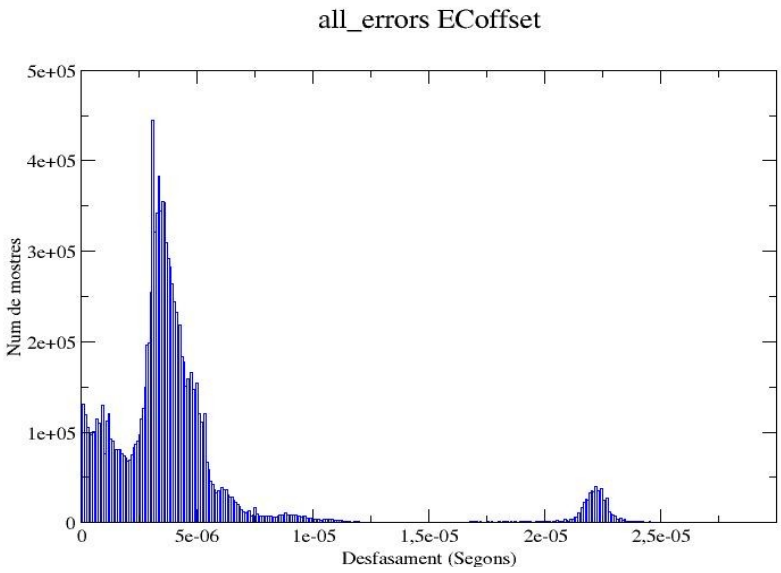
Per acabar vàrem tornar a realitzar la captura de 100.000 ECs d'1 ms amb 4 rèpliques i dos injectors executant-se al *switch*. No obstant, aquesta vegada configuram l'error *allpossiblecases*. Una vegada finalitzat l'experiment vàrem tractar els arxius .pcap generats pel *netAnalyzer* per obtenir les dades que es veuen en l'histograma de la gràfica 7.

Cal comentar que en aquest tercer experiment no en tots els ECs es podien comparar els temps de captura de la primera rèplica de TM dels dos esclaus, ja que el tipus d'error provocat fa que en alguns ECs aquesta rèplica es perdi. Per això vàrem modificar lleugerament el script de tractament de dades. En aquest sentit, en cada EC vàrem comparar el temps de captura de la primera rèplica que s'hagués transmès pels dos enllaços. En el cas que en un EC, els dos esclaus, no haguessin transmès cap mateixa rèplica les captures referents a aquest EC es rebutjaven.

A la taula 8 es poden veure els resultats calculats a partir de l'anàlisi de les captures realitzades pel *netAnalyzer* en les dues interfícies del *switch* HaRTES. La mitjana de *offset* entre les rèpliques no varia considerablement, ja que en aquest experiment és de 5,25 µs amb una desviació estàndard de 29,28 µs. Comparant els resultats amb l'experiment de referència sense MIFCDs, ens dóna una augment de l'offset mitjà causat per l'MIFCD de 2,32 µs. Com sempre, el màxim de *offset*s es manté molt per sobre de la mitjana però en pics molt puntals com reflecteix l'histograma de la gràfica 7. A l'histograma també s'observa, que la majoria de ECs tenen un *offset* entre rèpliques d'entre 3 i 5 µs però hi ha un grup d'un 5% que té un *offset* molt major, d'entre 20 i 25 µs.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
5,25	29,28	1588,99

Taula 8: Anàlisi estadístic de l'offset de la primera rèplica enviada per l'enllaç 1 i l'enllaç 2 amb mòduls injectors provocant l'omissió de TMs.



Gràfica 7: Histograma l'offset entre rèplica de missatge TM de dos enllaços mestre-esclau amb un injector en mode injecció per cada enllaç executant-se al switch. Tipus d'error allpossiblecases

11.2.4 Conclusions

De manera similar als experiments del capítol 11.1, podem notar que el *jitter* del sistema sense els mòduls de injecció de fallades i captura de dades és significant degut a la implementació software dels components. Tenint això en compte i observant les dades obtingudes, podem concloure que aquests mòduls introdueixen un retard que por ser considerat menyspreable (un poc més de 2 us de mitja). De la mateixa manera, el *jitter* que s'introdueix és també menyspreable.

12 Exemples d'ús de la infraestructura d'injecció de fallades en casos reals

En aquest capítol es descriuen diversos experiments duits a terme amb la infraestructura de fallades, destinats a avaluar experimentalment alguns del mecanismes de tolerància a fallades implementats en el prototipus d'FT4FTT

12.1 Mesura de l'*offset* de dos esclaus en finalitzar la finestra de TM

Com s'ha explicat al capítol 4.1, un dels mecanismes de tolerància a fallades desenvolupat dins el projecte FT4FTT assegura que els esclaus sempre puguin rebre el TM, encara que ocorrin errors transitoris durant la finestra de TM. Això s'aconsegueix enviant *k* rèpliques de TM. No obstant això, el TM també serveix per indicar als esclaus on ha de començar la finestra síncrona. Es a dir, serveix com a mecanisme de sincronització.

En aquests experiments vàrem utilitzar la infraestructura d'injecció de fallades per mesurar el nivell de sincronització que s'aconsegueix amb aquest mecanisme, en presència d'errors transitoris a la finestra de TM. Més concretament mesuram l'*offset* en l'inici de la finestra síncrona de dos eslaus si es produeix, com a màxim, la pèrdua de totes les rèpliques de TM menys una.

El muntatge utilitzat en aquests experiments és el que es descriu al capítol 5, el qual està format per 1 *switch* HaRTES, 2 esclaus connectats aquest *switch* i una estació d'instrumentació connectada a tots aquests dispositius des d'on l'usuari executa l'inici de l'experiment.

Per tal de determinar l'instant en què els esclaus consideren que la finestra asíncrona comença és necessari extreure alguna marca temporal de cadascun dels dos esclaus i comparar-les. No obstant això, no podem comprar marques temporals obtingudes de dos rellotges diferents. Per tant, en aquest experiment usarem el *netAnalyzer* que, a més de fer possible la captura de tràfic de dues interfícies amb el mateix dispositiu, i per tant amb el mateix rellotge, ens proporciona mesures molt més exactes que les que es poden prendre per software. El muntatge final es mostra a la figura 21.

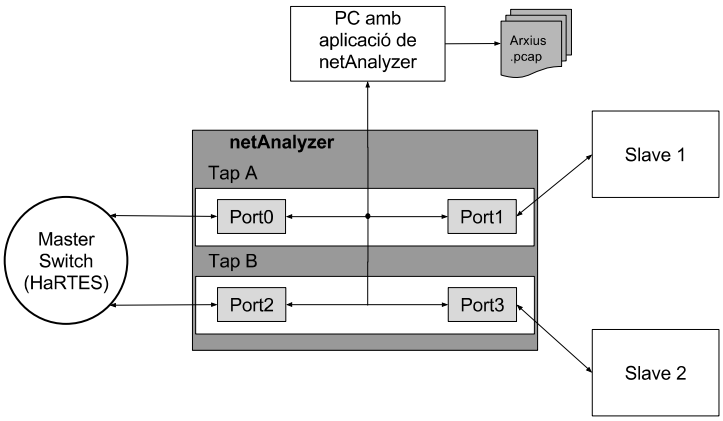


Figura 21: Esquema de connexions dispositiu de captura B500G-RE entre 2 enllaços mestre-esclau.

Per dur a terme aquests experiments, necessitàvem que els esclaus comuniquessin al *netAnalyzer* el moment exacte en el qual la finestra de TM finalitzava. Això ho vàrem implementar fent una modificació en el codi dels esclaus perquè enviessin un missatge al *netAnalyzer* en aquest instant de temps. Així, el *netAnalyzer*, en rebre aquest missatge, el captura juntament amb una marca temporal. Comparant el temps de recepció d'aquest missatge, des de l'esclau 1 i des de l'esclau 2, podem calcular l'*offset* que existeix entre l'instant en el qual cada esclau considera que la finestra síncrona ha començat.

A continuació descrivim els dos experiments que hem duit a terme.

12.1.1 Mesura de l'*offset* sense l'injector de fallades

En aquest primer experiment calculam l'*offset* entre el final de la finestra de TM, dels dos esclaus, sense provocar errors. Això ens proporciona dades de referència per després poder discutir els resultats obtinguts en el següent experiment, en el qual sí que es provoquen errors.

La captura de dades es va fer amb el *netAnalyzer.* Per tant, no va ser necessari executar cap MIFCD en cap element de la xarxa. En aquest sentit, l'*offset* mesurat té en compta l'*offset* derivat de la implementació software del prototipus *HaRTES* en les diferents màquines, com l'offset provocat per la transmissió del missatge de notificació d'inici de la finestra síncrona.

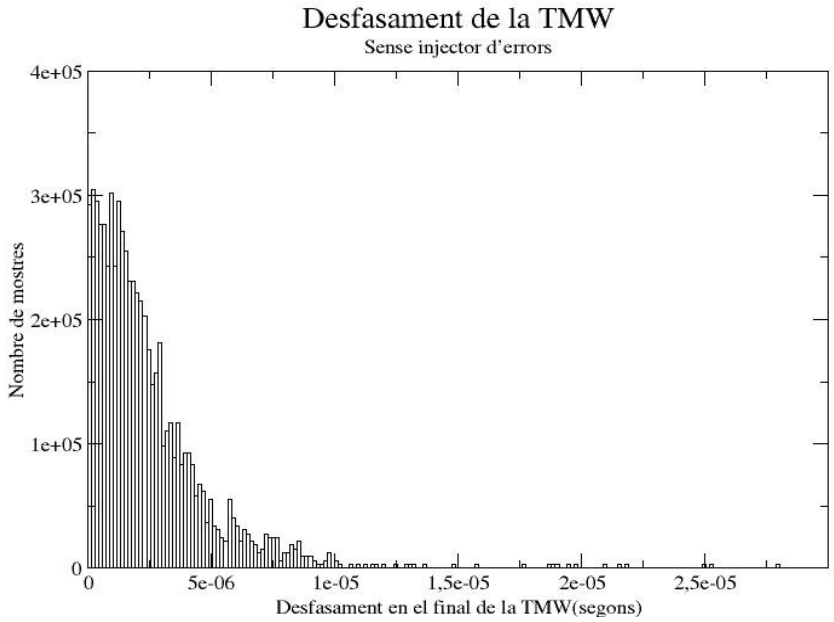
En aquest experiment vàrem capturar la marca temporal d'inici d'EC en cada esclau durant 100 mil cicles amb una longitud d'EC de 10 ms. Després vàrem utilitzar un script de l'MPAD per processar i analitzar el fitxer de captures del *netAnalyzer*

Els resultats obtinguts es mostren a continuació. La taula 9. resumeix els resultats estadístics on podem observar que l'*offset* mitjà entre el final de la *finestra de TM* dels dos esclaus, és de 2,49 µs amb una desviació estàndard de 3 µs. Són uns resultats bastant bons tenint en compta la mitjana suposa una desviació entre esclaus d'un 0,0249%. Pel que fa al valor màxim, encara que és 10 vegades superior al valor mitjà, és un valor que no és excessivament gran i, per tant, acceptable.

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
2,49	3	28,06

Taula 9: Anàlisi de l'offset entre el final de la finestra de TM en els dos esclaus sense la infraestructura d'injecció en el prototipus.

Finalment, a la gràfica 8 podem observar els *offset*s calculats agrupats per temps, on veiem que, tot i que el màxim *offset* calculat és de 28,06 µs, la gran majoria dels valors està per sota dels 3 µs.



Gràfica 8: Histograma de l'offset entre el final de la finestra de TM en els dos esclaus sense la infraestructura d'injecció en el prototipus.

12.1.2 Mesura de l'*offset* amb injecció de tots els casos de pèrdua de rèplica de TM

El següent pas va ser repetir l'experiment anterior injectant errors per veure com afectaven el mecanisme de sincronització. Amb aquest objectiu, i utilitzant l'esquema de xarxa de la prova anterior com a punt de partida, vàrem desplegar dos MIFCDs al *switch HaRTES*, un per cada enllaç mestre-esclau, i els vàrem configurar per injectar l'error *allpossiblecases*. Així vàrem veure com es comporten els esclaus en cada escenari possible de pèrdua de rèplica de TMs sempre que almenys els arribi una rèplica.

En aquest experiment, amb el *netAnalyzer*, vàrem registrar l'inici de la finestra síncrona de cada esclau durant 100 mil ECs, cadascun amb una longitud de 10 ms. Després vàrem utilitzar el mateix script de l'MPAD que en l'experiment anterior per processar i analitzar el fitxer de captures.

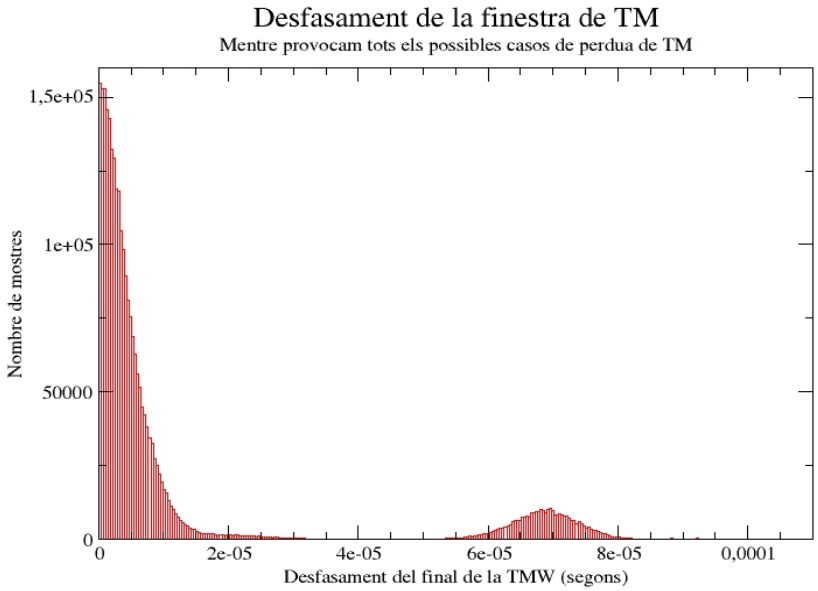
Els resultats obtinguts es mostren a continuació. La taula 10 resumeix els resultats estadístics. Més concretament, podem observar que l'*offset* mitjà entre el final de la *finestra de TM* dels dos esclaus, és de 12,89 µs amb una desviació estàndard de 23µs. Comparat amb el cas anterior veim que tenim un augment de 10,4 µs de *offset* per culpa de l'omissió de rèpliques de TM i al *offset* produït per executar els MIFCDs en el *switch HaRTES.*

Mitjana (µs)	Desviació estàndard (µs)	Màxim (µs)
12,89	23	109,12

Taula 10: Anàlisi de l'offset entre el final de la *finestra de TM* en els dos esclaus provocant l'error de tots els possibles casos d'omissió de TM.

També podem comparar aquest *offset* amb el calculat a l'apartat 11.2.3, on hem determinat que l'offset mitjà que provocava l'execució, al switch HaRTES, de dos MIFCDs provocant l'error *allpossiblecases* era de 2,23 µs. Si descomptam aquest *offset* atribuïble a l'injector, podem concloure que l'omissió de rèpliques de TM fa que el final de les finestres de TM, dels dos esclaus, es tingui un offset de 8,13 µs de mitjana.

Pel que fa a la desviació estàndard i al màxim, a la gràfica 9, podem observar els *offsets* calculats agrupats per temps, on veiem que, tot i que, el màxim offset calculat és de 109,12 µs la gran majoria dels valors està per sota dels 2 µs.



Gràfica 9: Histograma de l'offset entre el final de la finestra de TM en els dos esclaus provocant l'error de tots els possibles casos d'omissió de TM.

12.2 Avaluació d'aturada de components d'una xarxa FTTRS

En aquest experiment utilitzam l'injector d'errors per mostrar que la proposta d'arquitectura FTTRS del projecte FTT4FTT, comentat a l'apartat 4.2, pot tolerar tant la caiguda d'un enllaç qualsevol de la xarxa com diverses combinacions de caiguda de múltiples enllaços de la xarxa, a la vegada.

Com es pot veure a la figura 22, el prototipus utilitzat per fer l'experiment consta de dos *switches* i tres esclaus, tots ells connectats a una estació d'instrumentació. Per poder simular la caiguda de qualsevol enllaç de la xarxa FTT es desplega un MIFCD a cadascuna de les interfícies dels *switches*, tal com es mostra a la figura 23.

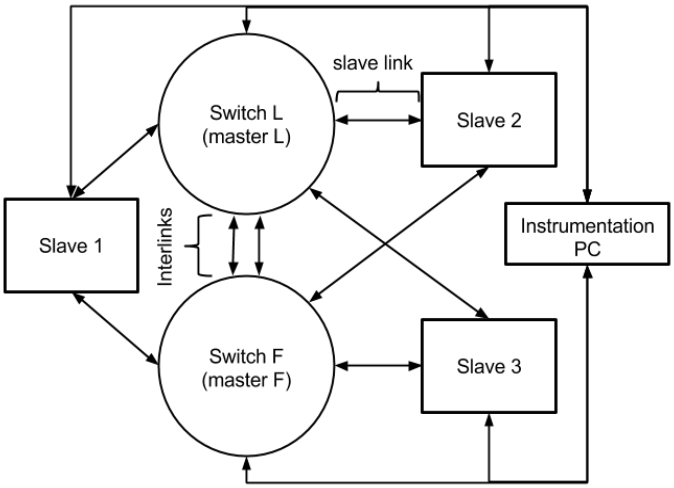


Figura 22: Prototipus FTTRS amb 2 switches HaRTES i 3 esclaus.

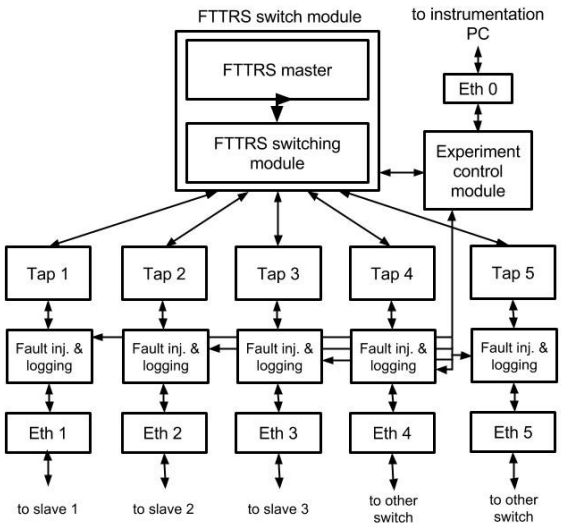


Figura 23: Switch HaRTES amb 5 interfícies amb un MIFCD cada una.

De la mateixa manera, com es mostra a la figura 24, els esclaus tenen diversos components específics d'FTTRS: un mòdul esclau FTTRS que inclou l'aplicació de xarxa i un *driver* FTTRS que permeti a l'aplicació enviar i transmetre trames a través de la xarxa FTTRS. Cada esclau també compta amb dos ports *Ethernet*, un per cada *switch*. Addicionalment, desplegam un MIFCD per a cada port *Ethernet* amb el fi de capturar totes les trames transmeses i rebudes per l'esclau. En cap experiment es varen utilitzar aquests mòduls per injectar errors, ja que vàrem determinar que el millor lloc per simular els errors d'aquest experiment era a la banda del *switch*.

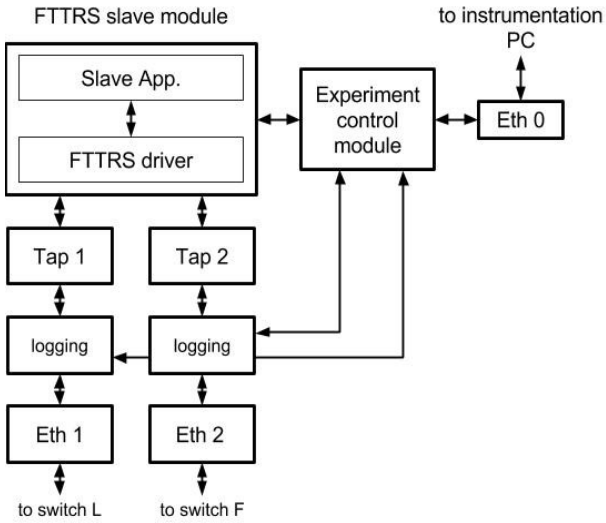


Figura 24: Esclau FTTRS amb 2 interfícies i 2 MIFCDs amb mode captura

A més vàrem connectar tots els dispositius de la xarxa a través d'una interfície d'instrumentació a l'estació d'instrumentació per tal d'inicialitzar els elements de la xarxa, de configurar la injecció i recuperar els fitxers de captura que es varen realitzar al final de cada experiment.

L'aplicació que s'executava als esclaus és una simplificació del control de replicació. Tots els esclaus realitzen la mateixa tasca de manera sincronitzada i després es posen d'acord amb el resultat. Es pot dividir la tasca que realitza cada esclau en tres fases. A la primera fase cada esclau genera un valor. Aquest valor podria provenir d'un sensor però per simplificar l'aplicació prové d'un comptador. A la segona fase els tres esclaus s'intercanvien el valor. Finalment, en l'última fase els esclaus executen un algoritme de votació per acordar un valor. Cada fase es realitza en un EC diferent, el que significa que el màster programa l'intercanvi de valors entre esclaus cada 3 ECs.

Duran l'experiment vàrem injectar errors en els components de la xarxa per comprovar que cada esclau, tot i els errors era capaç de fer arribar el seu valor del seu comptador als altres dos. Si n'eren capaços, vàrem considerar que la fallada havia estat tolerada correctament.

12.2.1 Comprovació de la tolerància a la caiguda d'un *switch*

En aquest apartat vàrem configurar els mòduls d'injecció primer d'un *switch* i després de l'altre perquè simulassin un error de *crash*. En els dos casos vàrem revisar els fitxers de captura creats pel mòdul de captura dels esclaus i vàrem comprovar que, malgrat la caiguda d'un dels dos *switches*, els tres esclaus estaven d'acord amb el nombre ECs transcorreguts durant l'experiment. A més els esclaus sempre varen ser capaços de transmetre el valor del seu comptador als altres. Per tant vàrem determinar el sistema era capaç de tolerar el *crash* d'un dels dos switches.

12.2.2 Comprovació de la tolerància del tall d'un enllaç de forma permanent

L'objectiu d'aquest apartat era comprovar que l'arquitectura de xarxa FTTRS és capaç de tolerar el tall dels enllaços màster - esclau o dels *interlinks*. Per comprovar-ho vàrem configurar els diferents mòduls d'injecció que estàvem executant als dos *switche*s perquè simulassin un tall de tots els enllaços. Utilitzant l'opció d'injecció tipus *lock* del nostre injector vàrem poder automatitzar la prova de tots els casos possibles de tall d'enllaç. Vàrem configurar talls de només 20 ECs esgraonats per accelerar l'experiment, ja que un tall superior a 3 ECs, si el sistema no tolera la fallada, ja provoca que l'esclau no pugui intercanviar el seu valor amb els altres.

Al nostre prototipus amb dos *switches* i tres esclaus, tenim dos enllaços per esclau (un per *switch*) i dos *interlinks* que interconnecten els *switches.* Per tant tenim 8 enllaços en total.

- En la primera prova vàrem simular la caiguda de només un enllaç a la vegada per tant només vàrem haver de provar 8 possibles casos. La configuració dels mòduls d'injecció va ser molt simple, ja que cada un només va haver de simular un error durant 20 ECs. Els talls es varen fer segons indica la taula 11. On l_{ij} representa l'enllaç entre l'esclau o el *switch* "i" i el *switch* "j". En tots els casos l'error va ser tolerat com s'esperava.

Enllaços tallats	Rang d'ECs utilitzats com a condició de tall
l_{1L}	70-89
l_{1F}	90-109
l_{2L}	110-129
l_{2F}	130-149
l_{3L}	150-169
l_{3F}	170-189
l_{LF_1}	190-209
l_{LF_2}	210-229

Taula 11: Seqüència d'ECs on es simula el tall de cada enllaç.

- En la segona prova vàrem simular el tall de dos enllaços a la vegada. En aquest cas teníem $\binom{8}{2}=28$ casos possibles de dos talls simultanis. Però sabíem que el sistema no estava dissenyat per tolerar 4 d'aquests casos, els quals corresponen al tall dels dos enllaços que arriben a un esclau o el tall dels dos *interlinks*. Per tant només vàrem configurar els mòduls d'injecció perquè simulassin 24 dels casos possibles. Els ECs escollits com a condició de *lock* estan a la taula 12. Com en el cas anterior, tots els erros simulats varen ser tolerats i els esclaus varen ser capaços d'enviar el seu valor als altres i varen estar d'acord amb el nombre ECs transcorreguts durant l'experiment.

Enllaços tallats	**Rang d'ECs utilitzats com a condició de tall**	**Enllaços tallats**	**Rang d'ECs utilitzats com a condició de tall**
l_{1L} , l_{2L}	70-89	l_{2L} , l_{3L}	310-329
l_{1L} , l_{2F}	90-109	l_{2L} , l_{3F}	330-349
l_{1L} , l_{3L}	110-129	l_{2L} , l_{LF_1}	350-369
l_{1L} , l_{3F}	130-149	l_{2L} , l_{LF_2}	370-389
l_{1L} , l_{LF_1}	150-169	l_{2F} , l_{3L}	390-409
l_{1L} , l_{LF_2}	170-189	l_{2F} , l_{3F}	410-429
l_{1F} , l_{2L}	190-209	l_{2F} , l_{LF_1}	430-449
l_{1F} , l_{2F}	210-229	l_{2F} , l_{LF_2}	450-469
l_{1F} , l_{3L}	230-249	l_{3L} , l_{LF_1}	470-489
l_{1F} , l_{3F}	250-269	l_{3L} , l_{LF_2}	490-509
l_{1F} , l_{LF_1}	270-289	l_{3F} , l_{LF_1}	510-529
l_{1F} , l_{LF_2}	290-309	l_{3F} , l_{LF_2}	530-549

Taula 12: Seqüència d'ECs on es simula el tall dels parells d'enllaços.

Finalment vàrem provar talls de 3 enllaços simultanis. Hi ha $\binom{8}{3}=56$ casos possibles però només n'hi ha 24 que segons el disseny del sistema haurien de ser tolerats. Per tant vàrem provar aquests 24. A la taula 13 tenim els ECs escollits com a condició de *lock* per cada mòdul injector. Gràcies als camins redundants de la topologia FTTRS les 24 fallades varen ser tolerades com esperàvem.

Enllaços tallats	**Rang d'ECs utilitzats com a condició de tall**	**Enllaços tallats**	**Rang d'ECs utilitzats com a condició de tall**
l_{1L} , l_{2L} , l_{3L}	70-89	l_{1F} , l_{2F} , l_{3L}	310-329
l_{1L} , l_{2L} , l_{3F}	90-109	l_{1F} , l_{2F} , l_{3F}	330-349
l_{1L} , l_{2L} , l_{LF_1}	110-129	l_{1F} , l_{2F} , l_{LF_1}	350-369
l_{1L} , l_{2L} , l_{LF_2}	130-149	l_{1F} , l_{2F} , l_{LF_2}	370-389
l_{1L} , l_{2F} , l_{3L}	150-169	l_{2L} , l_{3L} , l_{LF_1}	390-409
l_{1L} , l_{2F} , l_{3F}	170-189	l_{2L} , l_{3L} , l_{LF_2}	410-429
l_{1L} , l_{2F} , l_{LF_1}	190-209	l_{2F} , l_{3L} , l_{LF_1}	430-449
l_{1L} , l_{2F} , l_{LF_2}	210-229	l_{2F} , l_{3L} , l_{LF_2}	450-469
l_{1F} , l_{2L} , l_{3L}	230-249	l_{2L} , l_{3F} , l_{LF_1}	470-489
l_{1F} , l_{2L} , l_{3F}	250-269	l_{2L} , l_{3F} , l_{LF_2}	490-509
l_{1F} , l_{2L} , l_{LF_1}	270-289	l_{2F} , l_{3F} , l_{LF_1}	510-529
l_{1F} , l_{2L} , l_{LF_2}	290-309	l_{2F} , l_{3F} , l_{LF_2}	530-549

Taula 13: Seqüència d'ECs on es simula el tall dels grups d'enllaços.

12.3 Avaluació d'omissió de rèpliques de TM

Com hem explicat abans un element crític pel correcte funcionament dels sistemes basats en FTT són els *Trigger Messages* (TM). En la implementació de FTT del prototipus utilitzant en els nostres experiments es repliquen els missatges TMs per tolerar la seva pèrdua. D'aquesta manera cada TM es retransmet K vegades per cada màster a l'inici de cada EC. Ja que els TMs en FTTRS també s'intercanvien entre els *switches* i després els passen als esclaus un *switch* transmet el seu TM k vegades i els TM de l'altre *switch* unes altres k vegades. Això significa que cada EC 2k rèpliques de TM es transmeten per cada enllaç cap a un esclau. Mentre 1 de les 2k rèpliques arribi a cada esclau, tots els esclaus el procés de sincronització d'EC es funcionarà correctament i els esclaus estaran d'acord amb el nombre d'ECs transcorreguts durant un experiment.

Per tant, les 2k rèpliques d'un enllaç donat es poden corrompre o ometre s $2^{2k}-1$ i almenys una rèplica arribarà al seu destí. Si tenim s esclaus amb dos enllaços cada un, tenim $((2^{2k}-1)^{2s})$ possibilitats de corrompre les rèpliques de TM en un EC de manera que arribi almenys una rèplica a cada esclau. En canvi, si un esclau només té un enllaç, llavors hi ha un total de $((2^{2k}-1)^{s})$ possibilitats. Totes aquestes possibilitats haurien de ser tolerades per FTTRS.

En un principi volíem agafar el prototipus descrit al capítol anterior i aplicar l'error *allpossiblecases* per fer aquesta comprovació perquè l'experiment durés un temps raonable vàrem decidir dividir l'experiment en dues parts variant el prototipus per reduir els casos possibles.

- A la primera prova vàrem utilitzar els prototipus de l'apartat anterior però només utilitzant 2 esclaus en comptes de tres. Vàrem agafar un k=2 per tant necessitàvem provar $((2^{4}-1)^{4})$ = 50625 possibilitats.
- A la segona prova amb k=3, 3 esclaus i dos *switches* però els esclaus només estaven enllaçats amb un dels dos *switches* per tant teníem $((2^{6}-1)^{3})$ = 250047 possibilitats diferents.

Per simular aquests dos experiments vàrem utilitzar l'injector amb l'opció *allpossiblecases* el que fa aquesta opció de l'injector és provar un de tots els possibles casos a cada EC de forma consecutiva. Per tant el primer experiment va durar 50625 ECs i el segon 250047 ECs. Revisant els fitxers de captura vàrem comprovar que en els dos casos els esclaus estaven d'acord en el nombre d'ECs transcorreguts durant l'experiment i que havien transmès el valor del seu comptador als altres esclaus de la forma prevista. Per tant varen comprovar que el prototipus FTTRS tolerava tots els escenaris de fallada simulats.

12.4 Conclusions

Com hem pogut comprovar en aquests darrers experiments, el disseny modular de la infraestructura permet adaptar-se a diferents escenaris fàcilment. Més concretament, hem passat d'una topologia en estrella amb dos nodes a una topologia en estrella replicada amb tres nodes simplement desplegant més MIFCDs dins dels dispositius.

A més, disposar de l'analitzador de xarxa *netAnalyzer* en la infraestructura fa possible registrar el tràfic de qualsevol enllaç amb una precisió molt elevada. Addicionalment, també permet registrar el tràfic de dos enllaços de manera simultànea.

Part V. Conclusions

13 Resum

Aquest document presenta el disseny, implementació, verificació i validació d'una infraestructura d'injecció de fallades per a xarxes *Ethernet* amb funcionalitats específiques per a l'avaluació de sistemes basats en HaRTES.

Una de les característiques més rellevants d'aquesta infraestructura és que les seves funcionalitats estan implementades mitjançant diversos mòduls software desplegats pels diferents dispositius de la xarxa. Aquest disseny proporciona nombrosos avantatges. Primer, la infraestructura es pot fer servir per injectar fallades en qualsevol xarxa *Ethernet* on els seus dispositius funcionin amb un sistema operatiu Linux. És important remarcar que, si bé els mòduls estan preparats per treballar amb trames *Ethernet*, aquests també inclouen funcionalitats específiques per treballar amb els missatges FTT encapsulats dins d'ells. Això facilita la definició de les configuracions d'injecció. A més, resultaria senzill modificar els mòduls perquè fossin capaços de treballar amb altres protocols de xarxa basats en Ethernet. Segon, els mòduls d'injecció de fallades i captura de dades són transparents, tant per a l'aplicació que hi ha per sobre com pel *hardware* que hi ha per sota. És a dir, ni l'aplicació ni el hardware han de ser modificats per tal de poder funcionar amb els mòduls. Finalment, tenir les funcionalitats encapsulades en mòduls facilita la seva modificació i, a més, facilita el desplegament quan hi ha nombrosos dispositius a la xarxa.

Com es descriu al capítol 10, hem definit tres tipus de mòduls. Primer, existeix el mòdul d'injecció de fallades i captura de dades, anomenat MIFCD. Normalment s'instal·len diversos mòduls d'aquest tipus al *switch* HaRTES, un per a cadascun dels enllaços. Cada MIFCD realitza les tasques descrites a continuació. Intercepta el tràfic que arriba a la interfície associada, ja sigui des de l'aplicació del node o des del *switch*. Una vegada interceptat, analitza si aquest tràfic compleix les condicions d'injecció i, si les compleix, du a terme les modificacions pertinents segons el tipus d'error configurat. Finalment, una vegada modificat el tràfic resultant, el transmet per la interfície, en el sentit que correspongui. Addicionalment, els MIFCDs també capturen el tràfic resultant amb la finalitat de registrar el comportament, tant del *switch* com dels nodes.

El segon mòdul que hem definit és un mòdul de processament i anàlisi de dades anomenat MPAD fa possible extreure informació rellevant a partir de les dades recollides durant l'execució dels experiments. L'MPAD pot utilitzar dues fonts d'informació. Per una banda, pot utilitzar els fitxers de captures propis dels MIFCDs abans esmentats. D'altra banda, en experiments on es necessiti un registre del tràfic amb gran resolució temporal o un registre sincronitzat en diferents enllaços, s'utilitza el producte comercial *netAnalyzer* que genera fitxers de captures en format .pcap. Els *scripts* utilitzats per processar aquests fitxers .pcap i extreure'n la informació rellevant també formen part del mòdul MPAD.

El darrer mòdul que hem definit és un mòdul de control anomenat MC permet a l'usuari interactuar amb la resta dels mòduls. Més concretament, permet configurar, iniciar i aturar l'execució dels diferents MIFCDs. A més, permet recuperar els fitxers de captura generats en aquests mòduls per tal de proporcionar a l'MPAD les dades que necessita per extreure els resultats.

Pel que fa a la gestió dels mòduls de la infraestructura d'injecció de fallades, existeix una xarxa secundària d'instrumentació que connecta tots els dispositius amb una estació d'instrumentació que executa l'MC i, a més conté l'MPAD.

Per facilitar a l'usuari la tasca d'especificar la configuració dels MIFCDs s'ha creat un llenguatge, descrit al capítol 9. Per a cada MIFCD es genera un arxiu que defineix les condicions que ha de complir una trama per ser susceptible a la injecció, així com el tipus d'error que s'ha de provocar a aquesta trama, en cas que hagi d'esser injectada. En un arxiu hi pot haver múltiples errors, cada un d'ells amb les seves condicions d'injecció específiques. Si un MIFCD no té cap error configurat, diem que es troba en mode captura, i es limita a capturar el tràfic que passa a través d'ell per tal de crear el fitxer de captures final abans esmentat. La possibilitat de configurar cada MIFCD per separat permet obtenir una infraestructura flexible, amb capacitat d'injectar múltiples fallades, amb un o diversos mòduls d'injecció i de manera coordinada.

Durant la realització d'aquest projecte vàrem dur a terme diversos tests per tal de determinar el correcte funcionament dels diferents mòduls i de les seves funcionalitats. Així mateix, també hem realitzat diversos experiments per tal de mesurar l'impacte en el sistema que implica usar la infraestructura. Més concretament, hem mesurat el retard addicional que s'introdueix en la comunicació per culpa del pas de les trames pels MIFCDs (veure capítol 11). La conclusió que podem extreure d'aquests darrers experiments és que els MIFCDs no introdueixen un retard que sigui significatiu pel funcionament correcte del prototipus. D'altra banda, el *jitter* mesurat sí podria ser significatiu en l'avaluació de sistemes de temps real. No obstant això, com s'ha aclarit al mateix capítol 11, les funcionalitats de xarxa del prototipus HaRTES encara no estan optimitzades per a sistemes de temps real i, per tant, és normal trobar *jitters* significatius en moments puntuals. Quan els tècnics de laboratori del projecte FT4FTT acabin d'implementar el prototipus s'espera que aquest *jitter* estigui més acotat i, per tant, que l'impacte en el sistema sigui menor.

Finalment, en el capítol 12 hem validat la infraestructura d'injecció, és a dir, hem provat la utilitat d'aquesta infraestructura en escenaris reals i complexos. Per a aquest propòsit vàrem realitzar diferents experiments sobre el prototipus HaRTES en els que vàrem verificar la correcta implementació dels mecanismes de tolerància a falles presentats als articles [5] i [4]. Primer, al capítol 12.1 hem descrit l'experiment presentat a [5], que consisteix a avaluar el mecanisme de sincronització dels esclaus en tots els casos possibles d'error en el TM. Segon, al capítol 12.2 avaluam la capacitat de l'arquitectura FTTRS per tolerar l'avaria de diferents combinacions d'enllaços i, inclús, d'un dels *switches*. És important remarcar que l'arquitectura d'FTTRS inclous dos *switches* interconnectats i diversos nodes que es connecten a ambdós *switches*. És a dir, aquests experiments servixen també per demostrar la capacitat de la infraestructura d'injecció de desplegar-se a xarxes més complexes. Finalment, al capítol 12.3 avaluam la capacitat de l'arquitectura FTTRS per tolerar totes les combinacions d'omissió de rèpliques del TM. Per això utilitzam el prototipus de l'experiment anterior i provocam els errors descrits als experiments del capítol 12.1.

Per concloure, cal dir que hem complit els objectius del projecte, especificats al capítol 1.2. Hem construït una infraestructura d'injecció de fallades modular per a *Ethernet* que permet injectar fallades en el prototipus HaRTES, així com registrar el seu comportament davant d'aquestes fallades. No obstant, durant la realització d'aquest treball han sorgit problemes i noves necessitats que s'han anat resolent. Per exemple, hem hagut de desenvolupar i descriure en detall els criteris de selecció de les trames susceptibles a ser injectades, així com els tipus de errors que es poden provocar. Una altre exemple és l'estudi que va ser necessari fer per decidir la millor manera en la que implementar els mòduls de la infraestructura per tal que aquests fossin transparents, tant per a l'aplicació com per al *hardware*, i perquè aquests no introdueixin un retard significatiu en la transmissió i recepció de trames.

14 Opinió Personal

La realització d'aquest projecte m'ha resultat molt interessant, ja que he tingut l'oportunitat d'estudiar un protocol de comunicacions en profunditat i veure com estava implementat el seu prototipus. Per tal d'entendre el codi d'aquesta implementació, he hagut d'adquirir nous coneixements de programació que he pogut utilitzar en la implementació de la nostra infraestructura d'injecció de fallades. Exemples d'ús d'aquests coneixements serien: la divisió del procés principal en diferents fils d'execució per millorar el rendiment o la utilització de semàfors per mantenir la consistència de les dades capturades.

També, m'ha agradat molt tenir l'oportunitat d'utilitzar l'analitzador de tràfic *netAnalyzer* en un prototipus de xarxa i poder provar les seves funcionalitats. A més, per processar i analitzar els fitxers de captures d'aquest analitzador he hagut d'aprendre a fer *scripts bash* que incorporen *gawk* i altres comandes del S.O. Estic segur que obtindré molt de rendiment a aquests tipus d'*scripts* per automatitzar tasques que fins ara feia manualment, per tant, tot i que era una tasca secundària del projecte, estic molt satisfet d'haver après a fer-los.

Finalment, crec que em serà de molta utilitat haver elaborat aquesta memòria, ja que m'ha ajudat a millorat les meves habilitats d'escriptura.

15 Publicació dels experiments

Cal destacar, que alguns dels experiments duts a terme durant l'execució d'aquests projecte, es varen utilitzar per elaborar un article de congrés del qual sóc coautor. Els altres coautors de l'article són David Gessner, Alberto Ballesteros i Julián Proenza.

Es tracta d'un article de *work-in-progress*, on s'avaluen, utilitzant la infraestructura d'injecció de fallades descrita en aquest projecte, alguns mecanismes de tolerància a fallades implementats al projecte de recerca FT4FTT. Aquest article va ser presentat en el congrés World Conference on Factory Communication Systems (WFCS) del 2015, que va tenir lloc a la Universitat de les Illes Balears en Maig de 2015. L'article és el següent:

- D. Gessner, A. Ballesteros, A. Adrover, J. Proenza. Experimental Evaluation of Network Component Crashes and Trigger Message Omissions in the Flexible Time-Triggered Replicated Star for *Ethernet*. In Proceedings of the 2015 IEEE World Conference on Factory Communication Systems (WFCS), Palma de Mallorca, Spain, 2015.

Part VI. Apèndixs

A. Guia ràpida d'utilització de la infraestructura d'injecció de fallades

Per tal d'utilitzar la nostra infraestructura d'injecció de fallades en un prototipus s'han de seguir les següents passes. Suposam un prototipus de dispositius amb S.O. Linux que, a més d'estar interconnectats a través d'una interfície Ethernet pròpia del prototipus, estan connectats entre ells i a una estació d'instrumentació (PC) per una xarxa secundària d'instrumentació. Ens comunicarem amb els nodes des de l'estació d'instrumentació, utilitzant *secure shell* (ssh).

1) S'ha de crear un arxiu de configuració .xml per a cada interfície *Ethernet,* de cada node de la xarxa, on es vulgui utilitzar un mòdul d'injecció de fallades i captura de dades (MIFCD). La sintaxi d'elaboració i les especificacions mínimes que ha de contenir estan explicades al capítol 9 d'aquesta memòria.

2) Des de l'estació d'instrumentació utilitzant una consola *ssh*, s'ha d'iniciar l'executable *mifcd* a cada node on es vulguin injectar fallades o capturar dades amb el nom de l'.xml creat a l'apartat anterior com a argument. En cas de que el node tingui més d'una interfície on es vulguin provocar errors, s'ha d'iniciar una instància del *mifcd,* amb el seu propi arxiu de configuració, per cada una de les interfícies. Cada instància que s'hagi iniciat, crearà una interfície Ethernet virtual al node, amb el nom que se li hagi especificat en l'arxiu de configuració.

3) S'han d'iniciar les aplicacions de xarxa pròpies del prototipus en els diferents nodes. Les aplicacions de xarxa han d'utilitzar com a interfície de comunicació, les interfícies virtuals creades en el pas 2.

4) Una vegada realitzats aquests passos l'experiment ja haurà començat. A l'estació d'instrumentació podrem seguir l'evolució de l'experiment a les consoles on hem iniciat cada MIFCD.

5) L'experiment pot finalitzar de dues maneres

a. De forma automàtica, esperant que els MIFCDs capturin tantes mostres com la variable END_CAPTURE, que es troba a la llibreria *capture.h,* indica.

b. Manualment, executant l'aplicació *control* des de l'estació d'instrumentació. Aquesta aplicació s'ha d'iniciar amb el nom de la interfície *Ethernet* que utilitza l'estació com a interfície d'instrumentació com a únic argument. Una vegada executat aquest programa esperarà i quan li passem un caràcter qualsevol per línia de comandes, enviarà l'ordre de final d'experiment a tots els MIFCDs de la infraestructura.

6) Quan el final d'experiment s'inicia en els MIFCDs, cada un d'ells genera un fitxer amb la informació que ha recollit durant tot l'experiment de nom "Cap_frames_<dev>_<if>.txt", on *dev* correspon al node on s'executa l'MIFCD i *if,* correspon al nom de la interfície física associada l'MIFCD.

B. Exemples d'arxius de configuració

En aquest apartat plantejarem alguns escenaris d'injecció de fallades i descriurem com s'hauria de crear l'arxiu de configuració corresponent. Tots els exemples suposaran que tenim un MIFCD associat a la interfície, anomenada ***slave1***, d'un *switch* HaRTES, anomenat ***sw1***, que correspon a un enllaç amb un esclau. Donam el nom de ***tap1*** a la interfície virtual que crearem i la interfície del *switch* que està connectada a la xarxa d'instrumentació és la ***eth0***.

B.1. Exemple de *lock*

Si volem simular un tall temporal d'aquest enllaç que provoca la pèrdua de 10 ECs, concretament els ECs amb identificador de seqüència del **10** al **19**, l'arxiu de configuració seria el que es mostra a la llista 10. En aquest exemple tots ometrien totes les trames que el *switch* hagués de transmetre cap a *slave1* del cicle 10 al 19.

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave1</phyif>
            <vif>tap1</vif>
            <instif>eth0</instif>
      </device>
      <tx_error>
            <type>lock</type>
            <message_type>TM</message_type>
            <check>
                  <rlh_seq_no>10</rlh_seq_no>
            </check>
            <unlock>
                  <rlh_seq_no>19</rlh_seq_no>
            </unlock>
      </tx_error>
</args>
```

Llista 10: Exemple de lock de l'EC 10 al 19

B.2. Exemple de *drop* amb rang de valors.

Si volem que s'ometin només un tipus de paquets determinat i no tot el tràfic utilitzam l'error drop. En l'exemple de la llista 11 s'ometen els missatges de tipus TM que l'***sw1*** transmet a l'***slave1*** amb el camp de seqüència d'EC és major o igual a **0x40** i menor o igual **0x50**.

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave1</phyif>
            <vif>tap1</vif>
            <instif>eth0</instif>
      </device>
      <tx_error>
            <type>drop</type>
            <message_type>TM</message_type>
            <check>
                  <rlh_seq_no operator="R" side="S">40</rlh_seq_no>
                  <rlh_seq_no operator="R" side="F">50</rlh_seq_no>
            </check>
      </tx_error>
</args>
```

Llista 11: exemple de drop amb rang 0x40 ≤ EC ≤ 0x50

B.3. Exemple de *crash.*

Si volem simular una caiguda irrecuperable de l'esclau connectat a la interfície ***slave1*** del *switch*, a partir de l'EC numero **0x46**, utilitzaríem l'arxiu mostrat a la llista 12.

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave1</phyif>
            <vif>tap1</vif>
            <instif>eth0</instif>
      </device>
      <tx_error>
            <type>crash</type>
            <message_type>TM</message_type>
            <check>
                  <rlh_seq_no>46</rlh_seq_no>
            </check>
      </tx_error>
</args>
```

Llista 12: Exemple de crash a partir de l'EC 46

B.4. Exemple de *allpossiblecases.*

Si volem simular tots els possibles casos d'omissió de TM entre dos esclaus connectats a les interfícies ***slave1*** i ***slave2*** del *switch*, necessitam dos MIFCDs executant-se, un associat a cada interfície. Identificam l'slave1 amb ***idslave=1*** i l'slave2 amb ***idslave=2***. La llista 13 mostra l'arxiu de configuració de l'MIFCD de la interfície *slave1* i la llista 14 mostra l' arxiu de configuració de l'MIFCD de la interfície *slave2*.

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave1</phyif>
            <vif>tap1</vif>
            <instif>eth0</instif>
      </device>
      <tx_error>
            <message_type>TM</message_type>
            <idslave>1</idslave>
      </tx_error>
</args>
```

Llista 13: Allpossiblecases de l'esclau 1

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave2</phyif>
            <vif>tap2</vif>
            <instif>eth0</instif>
      </device>
      <tx_error>
            <message_type>TM</message_type>
            <idslave>2</idslave>
      </tx_error>
</args>
```

Llista 14: Allpossiblecases de l'esclau 2

B.5. Exemple de delay amb 2 condicions d'injecció.

A l'arxiu de configuració de la llista 15 es provoca un retard de **500** microsegons a la rèplica **1** del TM a tots els ECs a partir del 10è. Com en tots els casos anteriors aquesta fallada s'injecta en la interfície ***slave1*** del *switch* ***sw1*** i la interfície d'instrumentació es la ***eth0.***

```
<args>
      <device>
            <iddevice>sw1</iddevice>
            <phyif>slave1</phyif>
            <vif>tap1</vif>
            <instif>eth0</instif>
      </device>

      <tx_error>
            <type>delay</type>
            <message_type>TM</message_type>
            <check>
                  <rlh_seq_no operator="G">9</rlh_seq_no>
                  <rlh_TMseq operator="E">1</rlh_TMseq>
            </check>
            <time_delay>500</time_delay>
      </tx_error>
</args>
```

Llista 15: Exemple de delay amb dues condicions d'injecció

Bibliografia

[1] Florian Thiel (2002). The Linux Kerner Archives. Universal TUN/TAP device driver. URL: https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

[2] P. Pedreiras and L. Almeida. The Flexible Time-Triggered (FTT) paradigm: An approach to QoS management in distributed real-time systems. In Proc. Int. Parallel and Distributed Processing Symposium. IEEE Comput. Soc, 2001.

[3] A. Ballesteros, J. Proenza. A description of the FTT-SE protocol. December, 2013. http://srv.uib.es/a-description-of-the-ftt-se-protocol/

[4] D. Gessner, J. Proenza, M. A. Barranco, L. Almeida.Towards a Flexible Time-Triggered Replicated Star for *Ethernet*. In Proceedings of the 18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2013), Cagliari, Italy, 2013.

[5] D. Gessner,I. Alvarez, A. Ballesteros, M. A. Barranco, J. Proenza. Towards an Experimental Assessment of the Slave Elementary Cycle Synchronization in the Flexible Time-Triggered Replicated Star for *Ethernet*. InProceedings of the 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014), Barcelona, Spain, 2014

[6] System, Robotics and Vision (SRV). FT4FTT - FT4FTT-Ethernet: Fault Tolerance mechanisms for adaptive distributed embedded systems based on FTT-Ethernet. URL: http://srv.uib.es/ft4ftt/.

[7] Hilscher. *Ethernet* Analyzer Real-Time *Ethernet*. URL: http://www.hilscher.com/en/products/product-groups/analysis-and-data-acquisition/*Ethernet*-analysis/nanl-b500g-re/.

[8] A. Ballesteros, "Construction of sfiCAN: a star-based fault-injection infrastructure for the Controller Area Network". Projecte de final de carrera de Ingenyieria Informàtica. Universitat de les Illes Balears. Juliol, 2012.

[9] D. Gessner, A. Ballesteros, A. Adrover, J. Proenza. Experimental Evaluation of Network Component Crashes and Trigger Message Omissions in the Flexible Time-Triggered Replicated Star for *Ethernet*. In Proceedings of the 2015 IEEE World Conference on Factory Communication Systems (WFCS), Palma de Mallorca, Spain, 2015

[10] D. Gessner, J. Proenza, M. A. Barranco. A Proposal for Master Replica Control in the Flexible Time-Triggered Replicated Star for Ethernet. In *Proceedings of the 10th IEEE International Workshop on Factory Communication Systems (WFCS 2014), Toulouse, France, 2014*

[11] B. M. Wilamowski and J. David Irwin. Industrial Communications Systems. CRC Press, 28 Feb 2011. ISBN 9781439802816.