Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Aguado Arumi, Alberto; Robles Martinez, Sergi, dir. Desenvolupament d'un simulador de TCP. 2022. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/280741

under the terms of the  license

Desenvolupament d'un simulador de TCP

Alberto Aguado Arumí, 1457903

Resum—Aquest projecte es basa en la creació d'un simulador gràfic de TCP per mostrar el funcionament del protocol amb finalitats docents. Una eina gràfica ajuda en gran mesura a entendre funcionalment de TCP. Com a novetat envers altres simuladors, s'ha afegit al mateix temps enviaments automàtics amb retransmissions, probabilitat de pèrdua, dades disponibles als extrems i capacitat de "full-dúplex", o sigui gestió de dos fluxes de dades simultanis. Es detalla el procés d'anàlisi i disseny que ha permès obtenir el producte final. Primer, una llibreria amb la lògica completa de simulació, basada en l'especificació i reconamacions de la RFC9293 de TCP, que combina coneixements de Xarxes i alguns de congestió de Tecnologies Avançades d'Internet, i a més és fàcil d'importar i reutilitzar. En segona instància, una interfície web d'escriptori que a partir de les dades proporcionades per aquesta llibreria, es mostra el fluxe d'execució d'una simulació TCP altament configurable.

Paraules clau— Full-Dúplex, React, Redux, Retransmissió, Simulació Discreta, TCP, Typescript, Webpack.

Abstract— This project is based on the creation of a TCP graphic simulator to show the operation of the protocol for teaching purposes. A graphical tool greatly helps in functional understanding of TCP. As a novelty compared to other simulators, it has been added at the same time automatic sendings with retransmissions, probability of loss, data available at the ends and full-duplex capability, that is, management of two simultaneous data flows. The analysis and design process that allowed the final product to be obtained is detailed. First, a library with complete simulation logic, based on the TCP RFC9293 specification and renamings, which combines knowledge of "Xarxes" and a few congestion concepts from "Tecnologies Avançades d'Internet", and is also easy to import and reuse. In the second instance, a desktop web interface that, based on the data provided by this library, displays the execution flow of a highly configurable TCP simulation.

Keywords— Discrete Simulation, Full-Duplex, React, Redux, Retransmission, TCP, Typescript, Webpack.

1 Introducció

L'estudi dels protocols de xarxa requereix assolir un cert grau de comprensió per poder abstreure a nivell funcional la seva operativa. Tot i que protocols com TCP arriben a ser força complexos, l'objectiu que busquen solucionar no sol ser una idea complicada, per tant seria interessant facilitar la comprensió bàsica d'aquests protocols de xarxa.

Els protocols de xarxa estan compresos dins d'una pila de protocols, que operen a uns nivells concrets. Aquests nivells són independents entre sí, i permeten abstreure una connexió "punt-a-punt" entre cadascun dels nivells. El model OSI [1] presenta 7 nivells, però com que ens centrarem en el nivell de transport, podem parlar del model TCP/IP [2], que té 4 capes.

Com a principals protocols de xarxa a nivell de transport tenim UDP i TCP. El primer proporciona l'opció més senzilla, i generalment ràpida dels dos. El principal mecanisme de funcionament consisteix en transmetre les dades, sense cap tipus de mecanisme de control (excepte checksum). És a dir, no es fan re-enviaments.

En el cas de TCP, porta incorporat un mecanisme de control basat en retransmissions. TCP és un protocol relativament complex, i està completament descrit al RFC 9239 [3]. Originalment TCP va ser descrit al RFC 793, però ha quedat obsoleta. La descripció original, anotada amb canvis addicionals (generalment adreçats a millorar la resposta a la congestió), queda recollida a la RFC 9239.

Aquest especificació contempla totes les casuístiques possibles, i utilitza termes concrets per definir cadascun dels elements que comporten aquest protocol. Des del punt de vista educatiu, una comprensió completa del protocol requereix d'una gran quantitat de temps, i no és adequada per aprendre'l en un període curt. La realitat és que no és necessari una comprensió total del protocol per conèixer a nivell funcional quines operatives i processos principals es porten a terme. Amb una descripció d'alt nivell, sense entrar en gaires detalls tècnics de nomenclatures i casos especials, es pot assolir el coneixement necessari per poder integrar-lo com a coneixement, i col·locar-lo mentalment en el "lloc" corresponent de totes les piles de protocols d'Internet.

1.1 Objectius

Durant la docència de les assignatures de Xarxes i Tecnologies Avançades d'Internet (TAI) a la UAB, s'han identificat algunes dificultats a l'hora d'entendre algu-nes seccions sobre TCP. Alguns conceptes com "full-duplex", confirmacions acumulatives o establiments i tancaments de connexions són especialment difícils d'ensenyar a l'alumnat, en un període ajustat en quant a temps. Per tant, es fa evident la necessitat de disposar d'alguna alternativa que resolgui aquest problema.

Existeixen algunes solucions (comentat a la secció "Estat de l'art") que mostrin de forma gràfica el procés d'intercanvi de dades amb TCP, però utilitzen una lògica simpli-

- *E-mail de contacte: alberto.aguado3@gmail.com*
- *Menció realitzada: Tecnologies de la Informació*
- *Treball tutoritzat per: Sergi Robles Martinez (Departament d'Enginyeria de la Informació i de les Comunicacions*
- *Curs 2022/23*

ficada que només engloba un sub-conjunt del procés complet, i ensenyen aspectes molt concrets de l'implementació TCP. Es pretén obtenir un simulador que pugui agrupar de forma més completa el comportament del protocol TCP.

En aquest treball es proposa la creació d'una eina gràfica de simulació de TCP. Amb aquest simulador, es pretén facilitar el coneixement d'aquesta tecnologia, mostrant gràficament quins components interactuen, i que simuli seqüencialment el comportament esperat de TCP. Es mostraria quines dades i en quin ordre s'intercanvien dades els dos actors de la connexió, oferint la possibilitat de visualitzar també casos d'ús especials (pèrdues de paquets i retransmissions, establiment i tancament de connexió, entre altres).

L'objectiu principal, llavors, és aconseguir una eina de simulació gràfica del protocol TCP. Per tal d'aconseguir-ho, cal assolir una sèrie d'objectius bàsics:

1. Reunir el comportament descrit tant a l'especificació original de TCP, com l'explicada a classe (Xarxes, TAI), en forma de pseudocodi descriptiu i implementació
2. Crear una representació que combini aquests dos comportaments (l'especificació és una recoma-nació, formal. L'explicat a classe és una possible implementació)
3. Crear i implementar un disseny que satisfaci la representació anterior.
4. Finalment, obtenir una representació gràfica (en aquest cas) d'aquesta implementació, amb la qual es pugui interaccionar.

1.2 Metodologia

Pel desenvolupament d'aquest treball, inicialment s'ha optat per una metodologia en "Spiral", però finalment s'ha utilitzat una metodologia "Incremental".

L'objectiu inicial era assolir de forma iterativa un producte (inicialment un "Minimum Viable Product), on a cada iteració s'afegiria funcionalitat de TCP sobre l'anterior, fins aconseguir un simulador gràfic de TCP amb totes les seves funcionalitats. La realitat ha sigut que aquesta aproximació no és viable. Encara que les capes de xarxa tenen les seves responsabilitats i interaccions altament desacoplades, la recomanació d'implementació de TCP de la RFC9293 té un alt grau d'acoplament en tots els seus topics, i fa que una implementació en varies iteracions sigui no apropiada.

En primer lloc, s'ha fet un anàlisi inicial dels requeriments, i després s'ha construit la lògica de simulació. Una vegada completada, s'ha implementat l'interfície gràfica. Cadascuna d'aquestes tres tasques ha requerit de més d'una fase d'anàlisi i disseny (detallada a la secció 3).

Sabent que la lògica no es pot implementar en periodes de temps separats i ben definits, podem concloure que la metodologia "Incremental" és més adequada. Incremental es basa en l'anàlisi, disseny, implementació i "testeig" incremental, i aquestes fases es poden solapar, fins arribar al producte final. Spiral d'altra banda necessita establir inicialment els potencials riscos, i les fases del cicle de desenvolupament no es poden solapar entre sí. Tot i ser una metodologia menys costosa, l'inexperiència en algunes de les tecnologies utilitzades provoca imprevistos en la planificació i incorpora riscos no identificats inicialment, pel que s'ha acabat encaminant cap a Incremental, ja que és més adequat quan s'utilitzen noves tecnologies, i hi ha objectius de risc alt, com la creació de la implementació de TCP.

En quant a planificació, seguint l'aproximació inicial de Spiral, es volia tenir un "Minimum Viable Product" a mitjans del semestre, per poder obtenir feedback d'usuaris de prova que puguessin interactuar amb aquest, de forma que s'incorporés desenvolupament de "back-end" i "front-end" en les dues meitats del semestre, acabant aproximadament el 5 de juny. Degut al canvi de metodologia i de planificació, l'estimació de data d'entrega hauria de ser la mateixa, programant primer el "back-end" i després el "front-end", però s'han trobat problemes inesperats a l'hora de connectar el "back-end" amb el "front-end", a més de realitzar una petita planificació o revisió durant maig, on la qual es debatia sobre opcions per que l'interfaç gràfica pogués donar més valor.

2 Estat de l'art

S'ha mencionat que ja existeixen simuladors gràfics de TCP, ja que per aquest projecte es podria haver agafat un punt de partida inicial que no fos desde zero (en el suposat que els drets de propietat intelectual ho permetessin). Aquests simuladors mostren certes facetes de TCP de forma gràfica, però no exhaustivament en quant a contingut o comportament del protocol.

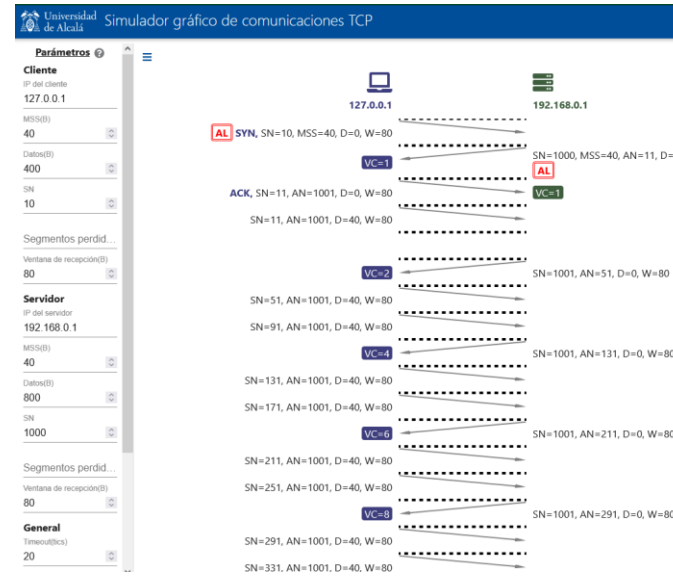


Fig. 1: Exemple de simulació determinista. Aquest simulador és notablement configurable, però careix de simulació en viu.

La motivació principal de desenvolupar aquest simulador envers d'altres, és la total configuració de paràmetres de TCP que, tot i mantenint limitades les opcions de configuració al simulador gràfic, permetria simular el comportament de gairebé qualsevol situació entre dos amfitrions TCP, centrant-nos en el caràcter funcional i no altres com el rendiment de xarxa, congestió, encaminaments...

Alguns simuladors representen el fluxe de dades entre els dos amfitrions. En el exemple de la fig. 1, ho fa de forma

determinista. Com que la simulació és determinista (i no es pot escollir la "seed" d'aleatoritat), s'ha d'escollir amb anterioritat quins segments es perden. En comptes de mostrar la simulació amb la progressió temporal, es calcula tota la simulació, i es mostra al final un resum amb els segments intercanviats.

El simulador que s'ha utilitzat prèviament a classe sí mostra temporalment com avancen les finestres, es propaguen els segments, i es confirmen les dades. El desavantatge principal d'aquest simulador és que no es pot conèixer el contingut dels segments que s'intercanvien (a més de ser una aplicació Flash, en des-ús).

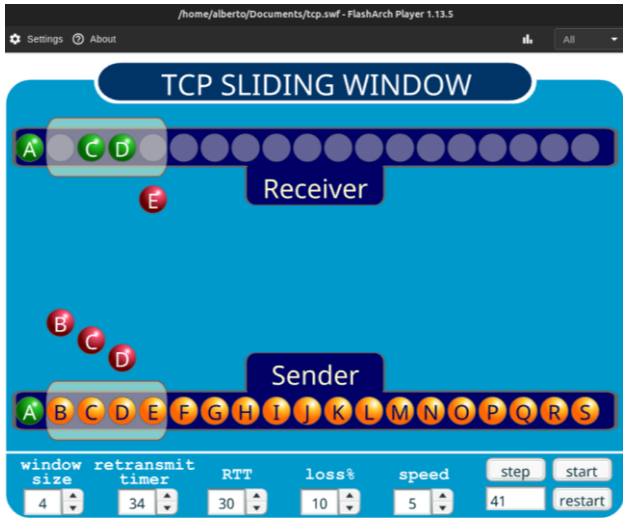


Fig. 2: Simulador de TCP utilitzat anteriorment a les classes de Xarxes. En destaca l'aspecte gràfic, que tot i ser simplista, transmet acuradament el funcionament del protocol.

Altres simuladors es focalitzen per exemple en el fluxe de les dades, més a nivell d'aplicació. Es recull el contingut dels buffers a mesura que llegeixen dades de l'aplicació i es reben al buffer destí. Aquest exemple es pot trobar a [4]. També hi ha un altre simulador [5] que sí disposa d'interactivitat: es pot seleccionar un segment i fer-li "drop". Es pot apreciar gràficament com viatgen les dades entre els dos amfitrions i com es reconeixen. El nivell de parametrització, però, és baix. A més, disposa d'una consola on s'indiquen els esdeveniments més importants de simulació.

Hi ha una gran varietat de simuladors, amb la qual es poden aprendre molts dels conceptes de TCP. Amb aquest simulador, però, es pretén que sigui simultàniament interactiable, es pugui investigar el contingut real dels segments que s'intercanvien, que sigui altament parametritzable, amb possibilitat de convertir-lo aleatori, que es puguin tenir dades disponibles als extrems, que es pugui enviar múltiples segments simultanis, i es puguin fer retransmissions, però sobretot que es pugui veure l'establiment i tancament de connexió (cap altre simulador trobat tracta els establiments i tancaments), i sigui full-dúplex (hi hagi un fluxe de dades bi-direccional, cada amfitrió pot fer d'emissor i de receptor al mateix temps). Llavors, es pretén crear una expansió d'altres simuladors, amb major completesa, que reflecteixi d'alguna manera gairebé tots els conceptes de la matèria de teoria de Xarxes.

3 ANÀLISI I DISSENY

El producte final és un simulador. Els simuladors poden dissenyar-se de forma que la simulació sigui de caràcter contínua o discreta. En les simulacions, es produeixen "events" que alteren l'execució de la simulació, i aquests es poden modelar moment a moment (contínua) si en tot moment es produeixen aquests events, ja que cadascun d'aquests altera la simulació i és necessari tornar a fer càlculs a cada pas. En el cas de la simulació a TCP, com que és un protocol extrem-a-extrem, la quantitat d'events a processar és petita, i llavors una aproximació discreta és més apropiada.

En el cas d'aquesta simulació, s'han identificat quins tipus d'event són rellevants per poder dur a terme la simulació de principi a fi, sense perdre cap dada important pel camí. Aquests events (descrits a la propera secció, també es poden trobar a l'apèndix A1) permeten desenvolupar la simulació, de forma discreta, encuant els events que sorgeixen de forma ordenada segons el temps d'execució previst, i reduïnt substancialment el requeriment computacional.

Com que el simulador necessita d'un suport gràfic per facilitar la labor didàctica, cal construïr una interfície gràfica que pugui mostrar com es realitza una simulació de TCP i mostrar per pantalla les dades a mesura que progressa. Per fer-ho, s'ha optat per separar la lògica de simulació, de la representació a l'interfície gràfica. Durant el desenvolupament, s'han trobat dades de la lògica que no són necessàries per l'interfície gràfica, i que simplifiquen en gran mesura el desenvolupament d'aquesta.

La lògica encapsula totes les peces necessàries per dur a terme la simulació, i obtenir en cada moment les dades que tindrien dos anfitrions que han establert una connexió de TCP, que intercanvien segments. Per aconseguir-ho, cal definir un model de dades i unes regles de simulació que puguin capturar tots aquest requeriments.

S'ha començat plantejant un algoritme o procediments que poguessin reproduïr el fluxe d'una connexió TCP, primer ignorant l'establiment i tancament de connexió. El format dels segments, el seu contingut, el reconeixement de les dades, les finestres d'emissió i recepció, i els "timeouts"són conceptes que queden clar a nivell teòric, però a mesura que es planteja la implementació, la quantitat de conceptes a aplicar i la diversitat de casos que poden ocórrer durant una simulació fan una implementació des de zero no viable. Tot i que pogués acabar sent funcional, el codi acabaria sent molt brut i amb poca abstracció i modularitat. La RFC9298 [3] presenta una certa estructuració conforme als termes i conceptes a tractar en determinades situacions, facilitant la creació de codi amb un objectiu més concret, i el seu índex és un exemple clar.

A partir d'aquest esquema de conceptes, s'ha definit quins aspectes es tracten en cada secció i de quina forma, escrivint pseudocodi genèric que permeti descriure el funcionament de conceptes com l'enviament i recepció de segments segons l'estat de connexió (màquina d'estats), la funcionalitat dels buffers, l'aplicació, el canal... A mesura

que s'anava completant, si es detectava alguna funcionalitat incompleta (p.ex: no rebre un segment segons el "forat" de buffer dins la finestra, en quin moment estimar el RTT...), es fa una re-evaluació del contingut fins aquell moment, i es retoquen les seccions que siguin necessàries, com si fos un model de "prototype".

El desenvolupament del "back-end" llavors, s'ha basat en transcriure el disseny amb pseudocodi inicial, amb algunes ampliacions que s'han trobat necessàries. Per recolzar i verificar el funcionament correcte, s'han creat tests unitaris que verifiquen el funcionament d'algunes classes, i un test "end-to-end" que assegura que dos amfitrions puguin intercanviar-se les dades després d'un número d'iteracions.

En quant al "front-end", durant la fase inicial s'ha fet un esquema bàsic per conèixer quins elements mostrar, i com distribuir-los en l'espai de la pantalla. Aquest esquema (fig. 3) està basat en els simuladors revisats a l'estat de l'art, sobretot en la disposició dels buffers i el trànsit en vertical dels segments.

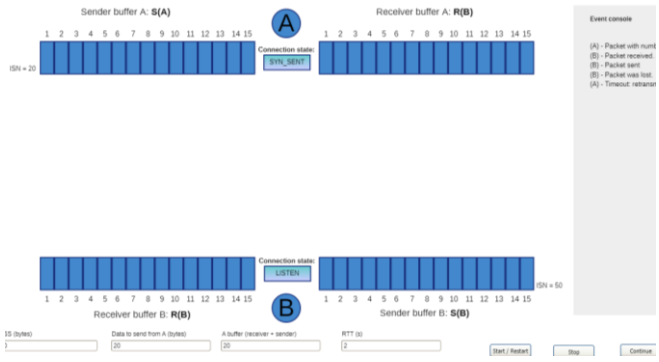


Fig. 3: Wireframe del plantejament inicial de l'interfície gràfica. Els segments d'A cap a B viatjarien al cantó esquerre, i de B a A al cantó dret.

Amb aquest wireframe però, no queda clar quins bytes es reconeixen quan es rep un segment, i quines dades s'han enviat. En una reunió amb professors del departament de Xarxes, es va arribar a la conclusió que es necessiten dos vistes: una gràfica que capturi el fluxe de les dades, i una taula amb un històric dels segments intercanviats, que capturi el caràcter temporal de la simulació. Aquesta taula fins i tot pot ser útil per alumnes de xarxes en alguns exercicis, ja que amb la configuració de l'enunciat poden replicar el plantejament de l'exercici.

4 IMPLEMENTACIÓ

Durant aquesta secció s'explica com es consolida la creació del simulador, una vegada s'han recollit els requisits funcionals principals. Es farà èmfasi primer en la creació de la lògica de simulació, altament fidel a una implementació TCP real en la que es es comuniquen dos amfitrions, i després de quina forma s'ha organitzat el codi en quant a l'interfície gràfica.

4.1. Lògica de simulació

Aquesta secció de codi existeix per facilitar tant el desenvolupament de la lògica com de l'interfície gràfica. Són dos components suficientment complexos com per justificar la seva separació, a més de resoldre dos propòsits clarament diferenciats. És una lògica escrita en Javascript 100% pur (amb compatibilitat de tipus per Typescript), que més tard serà utilitzada per l'interfície gràfica.

4.1.1. Simulació "Event-Driven"

La simulació, que és discreta, progressa amb l'execució d'uns events que representen certs instants d'una simulació real. Els 2 anfitrions i el canal tenen una cua d'events, que cadascú es pot gestionar ell mateix, i el simulador decideix fer una "enquesta" demanant el temps d'execució als 3 "actors", per saber quin executar i escollint el que vagi primer cronològicament, segons el temps d'execució previst que indiqui cadascun.

El simulador té els dos amfitrions i el canal, però també té el temps de simulació. Cada vegada que s'executi un event, el temps de simulació passa a ser el que indiqui el proper event (entre tots els events encuats). Però si tant els amfitrions com el canal es creen els seus events, necessiten saber quin és el temps de simulació. En lloc de mantenir els temporitzadors dels 3 actors, s'ha implementat el patró de "Observer" [6], de forma que aquests actors es subscriuen al temps d'execució del simulador ("Observed"), i cada vegada que aquest s'actualitzi, es notificará a tot subscriptor ("Observers").

Sabent que el model de simulació és discret i que podem calcular el temps d'execució de qualsevol event creat (el temps actual és conegut), cal "retardar" l'execució d'aquests events. Una primera solució ha sigut retardar el temps d'execució dels events amb "setTimeout", amb la que podem afegir un "callback" i el temps d'execució. En un context de "Javascript" pur és la solució més simple, però el principal problema és que llavors, executar una simulació requereix que es faci en temps real. Com s'explicarà més endavant, l'interfície s'encarregarà de donar aquest caràcter temporal al simulador.

En el cas del "back-end", només cal saber en quin ordre s'executen aquests events. Per això, s'ha optat per utilitzar el patró de "Command", ja que volem retrassar l'execució d'una certa comanda o event. Per cada tipus d'event, només cal especificar el temps d'execució, i totes les dades que necessiti. Per exemple, la recepció d'un segment requereix conèixer el segment en qüestió, i l'amfitrió destí a qui arriva. A més, el temps d'execució serà el temps de simulació actual més RTT/2, sumant la variança del canal (σ_{canal}) si n'hi ha: $T_{ex} = T_{act} + RTT/2 + \sigma_{canal}$.

Començem per l'event de "insertar un segment al canal". Un canal que representa l'enllaç extrem a extrem entre els dos anfitrions. En el mateix moment que s'insereix un segment al canal, es calcula segons la probabilitat de pèrdua si el segment "s'ha perdut" en algun punt fins al destí. Com que la simulació no captura el comportament a capes inferiors de TCP, per la simulació de TCP no cal saber en quin moment exacte es perd d'aquest camí fictici.

També, en el moment de configurar els paràmetres de simulació, es crea quin és el temps que tarda un segment

en fer el camí d'anada i tornada, anomenat RTT. Estrictament parlant, els anfitrions mesuraran un RTT una mica més gran, ja que els amfitrions implementen el "Delayed Acknowledgement" [7]. Per tant, cal afegir aquest temps de "Delayed Acknowledgement", Ta (com a mínim es 1*Ta, però sí es rep un altre segment abans d'haver transcorregut Ta, es reinicia aquest temps d'espera Ta). L'altre amfitrió contesta amb un o més segments, i per cadascun d'aquests enviaments, s'afegeix un "temps de processament", Tp, que tot i ser molt petit, es pot tenir en compte per apropar la simulació al comportament real. D'aquesta manera, arribem a que l'estimació del RTT serà de SRTT > RTT + 1*Ta + 1*Tp (en cas mínim de rebre només 1 segment, un Ta, i de contestar amb 1 segment, Tp). Tot i això, si Ta i Tp es consideren molt menors que RTT, l'estimació s'apropa al RTT esperat.

Un altre aspecte a tenir en compte, és que el temps de propagació pel canal pot no ser sempre igual (depen de molts factors, com gestió de cues de segments, fenòmens físics que afectin la propagació, canvis de rutes...). Per representar aquest fenòmen, el canal pot afegir una variació al canal. Per aquest projecte, només s'ha implementat una variança amb distribució Gaussiana, però es pot afegir fàcilment altres distribucions de probabilitat.

Per simplificar, el canal és el responsable de guardar l'històric dels segments intercanviats, amb les dates de creació i d'arribada / pèrdua. Els segments que comencen el viatge, es considera que estan en moviment. Aquests segments en moviment, poden acabar al destinatari, o bé es perden. En qualsevol cas, el canal manté un registre de segments intercanviats.

Si el segment no s'ha perdut, s'encua un event de "rebre segment". Aquest event simplement executa la lògica de processar un segment rebut de l'altre amfitrió. Recordem que quan un amfitrió processa la rebuda d'un segment, s'encua un event de "respondre després del temps de guarda" (eliminant, si n'hi ha, el mateix event si l'amfitrió ja el tenia encuat). Segons l'estat de conexió en que es trobi, es modificarà l'interacció amb tots els components de l'amfitrió (buffers, proper estat de connexió, bloc de control, segment/s que s'envia/en...). En aquest moment, es crearan més events d'enviar segments pel canal. Aquest cicle es repetirà fins que els estats de connexió dictin que la connexió s'ha tancat.

En algunes ocasions, els amfitrions posaran en marxa un "temporitzador" de "timeout". Aquest timeout serveix per representar a la simulació les retransmissions que ocoreixen ocasionalment durant les connexions. Per tal de representar aquest "temporitzador", s'ha creat un event de "timeout", que si s'executa, l'amfitrió corresponent torna a enviar el segment (o segments) que li pertoquin. En el cas de que previ a començar el timeout s'hagin enviat dades, es retrocedeix cap enrere la finestra, treient els possibles bytes no reconeguts. En qualsevol cas, es realitza la retransmissió de forma normal.

La RFC6298 [8] ofereix consells sobre com gestionar el temporitzador de timeout. En el nostre cas, s'han seguit les recomanacions, tot i que amb canvis adicionals. L'implementació actual gestiona el temporitzador de la següent manera:

1. Quan la connexió es troba en un estat inicial (durant l'establiment de connexió) i envies un segment, comença/reinicia el temporitzador. Els estats inicials són CLOSED, LISTEN, SYN_SENT, SYN_RECEIVED.
2. Quan s'envia un segment amb dades dins del payload, en els estats ESTABLISHED, FIN_WAIT1, FIN_WAIT2 o CLOSE_WAIT.
3. Quan es rep un segment amb el que queden reconegudes totes les dades de la finestra, el temporitzador es para. Es pot fer la distinció gràcies a que els events de "timeout" activen un flag de "s'ha fet retranmsissió", que s'utilitza per no estimar el RTT amb aquella mesura.

Per finalitzar els timeouts, en aquest simulador no s'ha implementat el "back-off" del temporitzador de retransmissió, o sigui, multiplicar el temps d'espera amb cada retransmissió. En lloc d'això, s'obté el RTO = SRTT + max(clockGranularity, K * RTTvar). En cas que no s'hagi fet cap mesura, no es coneix SRTT, per tant, s'ha optat per utilitzar un timeout per defecte de 8 segons.

L'últim tipus d'event utilitzat és l'event de "tancar la connexió". Només l'utilitza l'anfitrió que envia l'últim segment a l'altre amfitrió. Després d'un temps d'espera, aquest amfitrió tanca la connexió, finalitzant la sessió de protocol establerta. En cas que, durant aquest temps d'espera, es rebi un segment, vol dir que l'últim segment s'ha perdut a l'anada. Si s'ha perdut durant l'anada, vol dir que l'altre anfitrió no ha rebut l'últim pas del tancament de connexió, pel que saltarà el seu timeout, ens contestarà, i podrem tornar a enviar l'últim missatge. D'aquesta manera, es respectaria el tancament "elegant" de connexió, on les dues parts la poden tancar amb seguretat.

Amb aquests events, el fluxe d'execució de la simulació pot progressar fins al final entre dues suposades aplicacions que es connecten amb TCP i intercanvien dades. No cal cap altre intervenció, excepte que una aplicació ha d'iniciar la connexió. Per això, el simulador diferencia els dos amfitrions: un amfitrió passiu, que rebrà la connexió, i un d'actiu que enviarà el primer segment. El passiu hauria d'estar en estat "LISTEN", mentre que l'actiu pot estar en "CLOSED" o "SYN_SENT" (en cas que s'hagi perdut el primer segment). Cal destacar que, l'amfitrió com a entitat té el mateix comportament, ja sigui el passiu o l'actiu: només es diferencien en l'estat inicial. La RFC original contempla el cas en que els dos amfitrions inicien la connexió simultàniament. Per simplificar l'implementació, s'ha optat per no contemplar-ho: l'amfitrió actiu sempre comença la sessió.

4.1.2. Flux de dades extrem a extrem

S'ha esmentat que éls amfitrions tenen una "aplicació" fictícia, que té el paper d'actuar com a font de bytes a transmetre. Les dades en aquest moment no es generen dinàmicament durant la simulació: queden establertes durant la configuració inicial. Els dos amfitrions poden començar la simulació amb dades a transmetre (tot i que el passiu ho fa opcionalment). Al acabar la simulació, les dades que tenia un amfitrió la tindrà l'altre, i viceversa. Aquest és un dels objectius: disposar de dades extrem-a-extrem, i un dels principis de la divisió de capes de xarxa (tot i que amb el

simulador s'ha fet més senzill, ja que no cal integrar la capa TCP amb les inferiors de xarxa).

Per tal d'intercanviar les dades, cal primer que una part d'aquestes vagi cap a un "buffer" d'enviament. Les dades de l'aplicació s'obtenen amb una política "FIFO" (First In, First Out). S'agafa la quantitat de dades que pot cabre al buffer, i s'escriu cada "byte" en cada posició del buffer. Aquest procés pretén fer un símil a la primitiva d'aplicació "os.write()". En aquest moment, ja estan preparades per enviar-se al destí (tot i que primer cal establir la connexió).

Per arribar al destí, els buffers utilitzen la finestra lliscant (fig. 4) de TCP establerta a la RFC original. S'ha fet servir la mateixa nomenclatura que l'especificada a la RFC

The following diagrams may help to relate some of these variables to the sequence space.

```
                1          2          3          4
           ----------|----------|----------|----------
                  SND.UNA    SND.NXT    SND.UNA
                                        +SND.WND
```

1 - old sequence numbers that have been acknowledged
2 - sequence numbers of unacknowledged data
3 - sequence numbers allowed for new data transmission
4 - future sequence numbers that are not yet allowed

Figure 3: Send Sequence Space

The send window is the portion of the sequence space labeled 3 in Figure 3.

```
                    1          2          3
               ----------|----------|----------
                      RCV.NXT    RCV.NXT
                                +RCV.WND
```

1 - old sequence numbers that have been acknowledged
2 - sequence numbers allowed for new reception
3 - future sequence numbers that are not yet allowed

Figure 4: Receive Sequence Space

The receive window is the portion of the sequence space labeled 2 in Figure 4.

Fig. 4: secció de la RFC9293, que explica amb nomenclatura estandaritzada l'estructura de les finestres d'emissió i recepció, respectivament.

per les variables de les finestres d'enviament i rebuda. Les dades es carreguen, i inicialment SND.UNA és igual a SND.NXT. El moment en que s'enviin dades, SND.NXT avança tantes posicions com "bytes" s'hagin enviat. El destinatari les reconeixerà, i respondrà fins a quin SND.UNA es pot avançar, amb el camp de segment de número d'ACK. D'aquesta manera es pot moure la finestra lliscant. El contingut del buffer es estàtic (no rep dades fins que totes hagin sigut reconegudes), per tant, en aquesta implementació, quan totes les dades són reconegudes al buffer, la finestra d'enviament de dades es torna zero, i inmediatament l'aplicació carrega més dades al buffer d'enviament, augmentant la finestra al tamany original.

Per la banda del receptor també es disposa d'una finestra lliscant, però en aquest cas és de recepció. Després de l'establiment de connexió i haver intercanviat informació sobre números de seqüència i finestres, s'espera la recepció de segments que coincideixin amb l'interval esperat. De forma breu, l'interval esperat pel número de seqüència escau entre RCV.NXT i RCX.NXT + RCV.WND, o sigui, dins de la finestra de recepció. D'aquesta forma no cal rebre el següent segment en ordre seqüencial: si el número de seqüència cau dins de la finestra de recepció, s'accepta.

En el cas que el segment no sigui acceptable, la RFC9293 [3] recomana a la secció 3.10.7.4 que es respongui amb un segment que tingui populat el camp d'ACK (número i bit de control activat), d'aquesta forma es dóna la oportunitat a corregir el número de seqüència i sigui acceptable a la recepció. En el cas de la simulació aquest comportament no és necessari, ja que cadascun dels amfitrions actuen independentment i adherint-se al protocol TCP, però en cas de poder-se modificar el contingut dels segments per exemple, aquest cas d'ús queda contemplat.

En el cas que algún dels segments es perdi, existeix la funcionalitat de reenviaments quan s'esdevingui un "timeout". Si es dona el cas, es retrocedeix SND.NXT cap a SND.UNA, i es torna a enviar les dades que pertoquin del buffer. La finestra de recepció no pateix canvis. Però, què succeeix si per exemple, es reben dos (1 i 3) dels tres segments (1, 2, 3) que s'han enviat? En aquesta implementació, la finestra de recepció avança fins la primera posició del buffer que tingui buida, o sigui fins la posició on s'espera rebre el segment número 2. Això implica, que tot i haver rebut el segment 3, es tornaran a enviar els segments 2 i 3 (i possiblement algún més), augmentant innecessàriament la congestió de la xarxa una mica. Es pot evitar gràcies a la recomanació de la RFC 2018 [9], que inclou una opció TCP de "Selective ACK", SACK. Permet informar sobre els segments que s'han rebut correctament, podent-se deduïr quins "forats" de dades té l'altre amfitrió, i enviant només els segments d'aquests forats. L'opció de SACK no s'ha implementat al simulador, ja que només és una recomanació del protocol i no aporta gaire contingut didàctic.

Finalment, a mesura que les dades es van rebent, el buffer de recepció es va omplint. Una vegada s'ha omplert completament, l'aplicació de destí de l'amfitrió executa una primitiva de lectura, "os.read()", de forma que es buida el contingut del buffer i avança la finestra de recepció. Aquest procés es repeteix fins que l'amfitrió hagi intercanviat totes les dades que volia enviar. La RFC9298 no especifica que s'hagi d'anunciar el tancament de connexió una vegada s'han intercanviat les dades, però per tal de finalitzar automàticament la simulació, s'ha optat per aquest tancament en funció de les dades a enviar (que no només han de ser enviades, sinó enviades i confirmades).

Tenint en compte que els amfitrions passiu i actiu tenen el mateix comportament, i que els dos poden intercanviar dades, el procés complet de transferir les dades d'un extrem a l'altre es pot fer simultàniament en el sentit contrari. Com que els segments tenen els camps de número de seqüència (SEQ), número de reconeixement (ACK), finestra anunciada (WIN) i bits de control pertinents, quan es rep un segment i es prepara la resposta per reconèixer les dades, el protocol TCP pot aprofitar per afegir dades al payload del segment i enviar-les amb el número de seqüència rellevant. És a dir, durant la recepció i enviament de segments, pot haver flux bi-direccional de les dades. Per això, TCP es considera full-dúplex.

4.1.3. NPM

D'alguna manera, el codi del back-end s'ha de poder utilitzar al front-end. Existeixen dos aproximacions per aconseguir-ho, sense "copiar - enganxar" el codi del back-end al front-end: publicar el package al registre de NPM [10] (Node Package Manager), i utilitzar una "Project Reference".

S'ha optat per publicar el package al registre de NPM. Aquest registre és d'accés públic, però pot ser d'accés privat amb un pla de pagament. Com que el codi no és d'ús privatiu, s'ha publicat al registre públic, de forma que qualsevol projecte pot descarregar-lo i utilitzar-lo. D'aquesta manera, es pot instalar desde el projecte del front-end, i en cas que sigui necessari, desplegar noves versions amb canvis si és necessari, i utilitzar-los canviant la versió importada. És l'opció més directa i més convenient.

L'altra opció és utilitzar "Project Reference". Consisteix en tenir una còpia del back-end emmagatzemat en memòria local, i especificar la ruta fins aquest projecte. D'aquesta manera es pot estalviar publicar el package a NPM, però com que no és codi privat, al final s'ha descartat aquesta opció.

4.2 Interfície gràfica: Front-end

L'altra peça del simulador és l'interfície gràfica. Aquesta interfície utilitzarà només un sub-conjunt de les funcionalitats del back-end necessari per manifestar visualment el comportament del simulador i facilitar l'interacció amb aquest. React [11] s'encarrega de mostrar i actualitzar tots els components (nomenclatura de React), mentre que Redux [12] converteix les dades del back-end en altres compatibles amb React, i les emmagatzema de forma centralitzada, simplificant l'accés i modificacions. En primer lloc, el back-end exposa només els mètodes i propietats que siguin necessaris de la simulació. Tot i no existir el concepte de "public" o "private" a Javascript, amb Typescript sí. Aquesta funcionalitat existeix en temps de compilació: quan el codi sigui transpilat a Javascript, deixaran d'existir aquests cualificadors, el que implica que es pot compilar codi que accedeixi a variables privades, encara que el compilador mostri errors. Això, però, serveix de guía pel desenvolupador: el tipat fort de les variables i modificadors d'accés (entre altres) serveixen per trobar possibles errors que amb Javascript normal no serien detectables fins que s'executi, forçant a evitar defectes que siguin difícils de detectar després. D'aquesta manera, es pot mostrar els mètodes i propietats que es desitgin, a mode de "facade" [6], com mètodes per carregar configuració de simulació, executar el següent event... i propietats tals com els amfitrions passiu i actiu, i el canal entre aquests.

Les propietats exposades de la simulació s'actualitzen amb l'execució dels events, de forma que s'actualitzaran dinàmicament segons l'especificació de l'implementació TCP i la configuració de simulació utilitzada. Aquestes propietats seran consumides per Redux. Comença convertint les dades del back-end, que tenen dades d'interès i altres específiques de la simulació que no són rellevants. Són convertides a dades "serialitzables", un requisit del magatzem de Redux, escollint només les dades rellevants i en formats simples (enters, strings, arrays sí... classes, dates i altres no són serialitzables), o transformant-les si és necessari, com convertir dates a segons (enters).

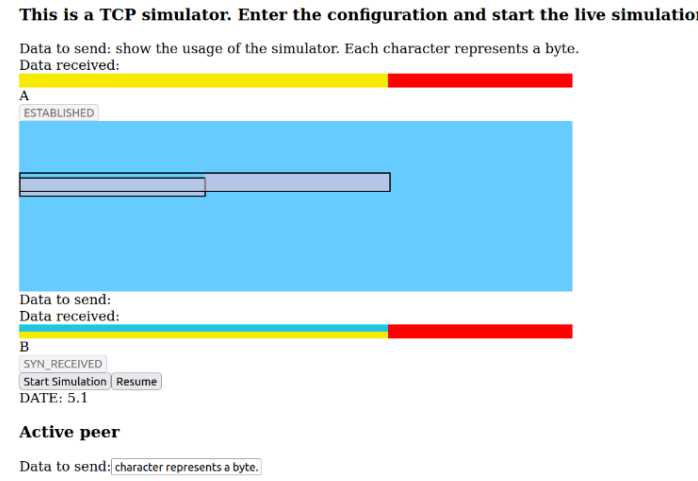


Fig. 5: Captura del simulador. Hi ha 2 segments en trànsit. Per cadascun dels amfitrions, es correspon de forma gràfica el contingut del payload i ack dels seus propis buffers d'enviament i de rebuda, respectivament.

Per fer-ho, en una configuració de Redux s'exposen "accions". Les accions tenen un payload (opcional) i un tipus. Aquest tipus d'acció s'utilitzarà en tots els "reducers" per identificar quin comportament realitzar segons el tipus especificat. L'altra peça gran de Redux són les "store" i els "reducer". Cada store emmagatzema la peça d'estat que es desitgi. Qualsevol component pot accedir a aquest estat, però no el pot modificar directament. Per fer-ho, ha de despatxar una acció amb un payload (opcional). Quan es rep una acció, s'executen els "reducers". Són funcions pures, que utilitzen l'estat anterior i (opcionalment) unes dades d'entrada (payload), i retornen l'estat següent: interaccionen directament amb les stores. L'idea és que desde qualsevol lloc de l'aplicació, es poden "despatxar" accions, i es respectaria l'ordre d'execució d'aquestes: amb cada acció, es té actualitzat l'estat que s'ha modificat anteriorment. Aquesta configuració de Redux permet agafar les dades del back-end de simulació (payload) i despatxar una acció (actualitzar interfície gràfica), que quan arribi al reducer del simulador, modificarà la store amb les dades del nou event executat, fent el procés de transformació de dades necessari.

Una vegada es disposa de les dades en format simple, aquestes es poden utilitzar directament en els components del simulador. Només cal marcar quin component utilitzarà Redux, i tant aquest component com tots els seus fills podran accedir a l'estat i actualitzar-lo despatxant accions. El component que utilitza Redux és el del simulador, ja que la resta de components no necessiten accés a Redux.

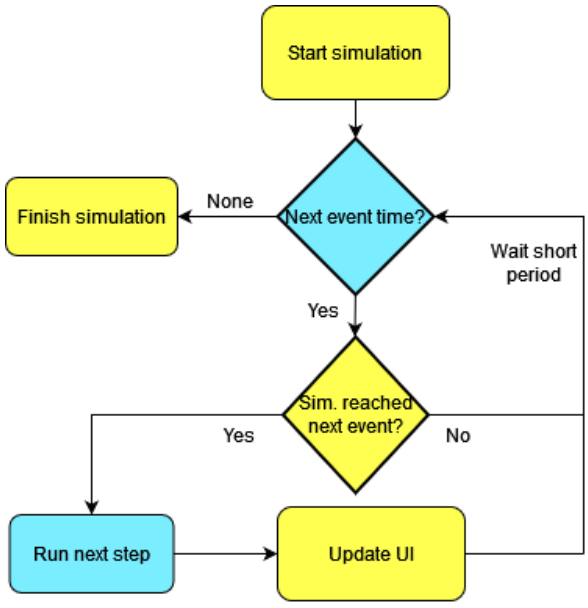


Fig. 6: Fluxe automàtic d'execució de l'interfície gràfica. Gràcies als events ordenats per temps creats per la lògica (representada en blau), l'interfície gràfica (groc) pot recollir les dades i mostrar-les en temps real.

Quan es carreguin les dades d'una configuració i comenci una simulació, el component arrel (del simulador) manté com a estat el temps de simulació, si la simulació està en execució o parada, i una instància de l'objecte "Simulation" que fa d'interfície amb tota la lògica. Aquest component, amb aquesta informació, dirigeix la progressió de la simulació, que fins aquest moment, pot progressar només endavant en el temps: no hi ha programada funcionalitat de marxa enrere. Quan no quedin més events a executar, la simulació finalitza. Aquest fluxe general d'execució queda recollit a la fig. 6.

A mesura que progressa la simulació, qualsevol segment que o bé s'hagi entregat o bé s'hagi perdut, anirà apareixent a la secció de l'històric de segments, ordenats segons el temps de simulació en que s'hagin creat, de forma que al acabar, hi hagi un registre dels segments intercanviats, ordenats per temps de creació, i mostri les dades més rellevants d'aquests.

Tot el codi del "back-end" i "front-end" està escrit en Typescript amb sintaxi de EcmaScript [13]. Aquesta té sentències que són (de moment) incompatibles amb navegadors, principalment els "import". La conversió de Javascript a Typescript és senzilla, ja que amb qualsevol compilador es pot "transpilar" el codi. De fet, qualsevol codi de Typescript s'executa als navegadors amb la versió transpilada en Javascript. La sintaxi de EcmaScript es pot fer executable en navegadors amb eines com Webpack [14]. La configuració sol ser senzilla, i facilita la compatibilitat de bases de codi Javascript amb navegadors (sobretot per codi "server-side").

5 Conclusions

Amb aquest projecte es volia aconseguir una eina de simulació gràfica que ajudés en la docència de Xarxes durant l'explicació del protocol de TCP. Hi ha conceptes com "full-duplex", confirmacions acumulatives o establiment i tancament de connexió que tot i ser trivials, la combinació d'aquests fa que el protocol sencer siguir difícil d'entendre. És per això que una eina gràfica és una ajuda molt adequada, ja que és capaç de visualitzar el funcionament d'aquests conceptes, i a més ho fa amb exemples reals, que complementen la teoria d'aquesta.

La flexibilitat de les tecnologies web permeten fer una gran varietat de desenvolupaments, ja siguin web o de servidor. Frameworks com React ajuden a crear interfícies gràfiques amb gran facilitat, i eines com Webpack faciliten la compatibilitat amb navegadors. Tot i trobar-se l'eina gràfica en un estat primitiu, es pot expandir fàcilment amb la configuració d'arquitectura de codi actual.

Com a eina gràfica, es disposa de l'informació completa d'un intercanvi de dades amb TCP: finestres lliscants, full-dúplex, dades disponibles als extrems, retransmissions, informació de control als segments, i fins i tot algunes mesures anti-congestió. La funcionalitat de l'històric de segments, es pot fer servir per donar a la simulació gràfica una dimensió temporal de l'intercanvi de dades TCP (tant d'intercanvi de dades, com d'establiment i tancament), i també complementar alguns dels exercicis de TCP a classe de Xarxes, adaptant-se a l'enunciat del problema gràcies a la seva alta configurabilitat.

D'altra banda, aquest package de lògica de simulació TCP incorpora de principi a fi totes les funcionalitats del protocol. Aquest package es pot utilitzar en el seu estat actual com a instrument de simulació, ja que incorpora tot el necessari. Es pot modificar, i afegir funcionalitats de protocol adicionals, però en l'estat actual es pot fer servir per executar simulacions senceres, i pot utilitzar-se per investigar el comportament del protocol. Cal destacar que està escrit completament en Javascript pla (i amb opció d'importar-se en projectes Typescript), sense cap dependència externa, el que fa que sigui molt lleuger.

5.1. Línies futures de treball

Tot i haver consignat una eina de simulació de TCP gràfica, hi ha alguns punts de millora que es podrien adreçar com a continuació. Alguns són de caràcter funcional, com l'adició d'optimitzacions TCP o noves interaccions amb la UI, i altres de caràcter cosmètic.

Visualment, caldria fer una revisió a alt nivell de l'interfaç, en la qual s'assesoraria l'experiència d'usuari i l'usabilitat. Alguns punts a millorar actualment en són l'estilat dels components, la posició d'aquests en quant a agrupament lògic de seccions (separar-les millor en "simulació en viu", configuració, controls de simulació i històric) i fer més intuitiu l'ús dels controls.

De cara a l'interfaç, en l'aspecte funcional, es pot expandir afegint la possibilitat de modificar el contingut d'algun dels segments en trànsit, per exemple modificar el número de seqüència comportaria el reenviament d'un segment on

al camp d'ACK rebem el número de seqüència que s'espera, ja que en el comportament del simulador està contemplat, però sense cap intervenció aquesta situació no es produïria. Un altre exemple seria encendre el bit de "RST", i que es reflecteixi en el tancament immediat de la connexió.

Un aspecte a mediar entre "front-end" i "back-end" és la separació de responsabilitats entre aquests. Per començar, s'hauria d'especificar quines dades necessita o pot necessitar el front-end. Ha d'especialitzar-se en mostrar acuradament el funcionament de TCP, però no vol dir que necessiti conèixer per exemple les constants de l'algoritme de Karn, o la probabilitat de pèrdua del canal.

També es fa necessari definir una interfície universal que obligui a qualsevol back-end a complir les necessitats del simulador (executar següent pas, començar la simulació i carregar una configuració principalment). D'aquesta manera, el "back-end" només s'ha de preocupar d'implementar correctament aquests mètodes i el farien compatible amb el "front-end". Això s'ha dificultat durant el desenvolupament del projecte degut al risc inicial d'aconseguir cohesionar una versió completament funcional. Ara que està completada, seria un bon moment per plantejar aquestes idees, ja que és més fàcil prendre aquestes decisions tenint en compte el resultats, que no desde zero, sense tenir situat el gran abast de conceptes que té TCP.

Per últim, a nivell de "back-end" i en vista dels resultats obtinguts, podria ser interessant implementar un "logger" que reculli els esdeveniments més clau de simulació. Aquests no servirien per fer "debugging", sino per ajudar a l'usuari a entendre millor detalls que no siguin capturats amb l'interfaç, com per exemple "segment rebutjat degut a que s'esperava un ACK", o "escrivint X bytes de l'aplicació al buffer". S'hauria de plantejar quins "logs" serien necessaris, perquè alguns com "segment enviat..." serien més efectius visualment que amb una entrada al "logger".

6 BIBLIOGRAFIA

[1] The ISO model of Open Systems Interconnection (OSI). (1986). Communications Standards. https://doi.org/10.1016/b978-0-08-034092-0.50024-7

[2] Kale, C. (1991, January). RFC-1180: A TCP/IP Tutorial. " RFC editor. Retrieved March 14, 2023, from https://www.rfc-editor.org/rfc/rfc1180.txt

[3] Eddy, W. (2022, August 18). RFC FT-IETF-TCPM-rfc793bis: Transmission control protocol (TCP). IETF Datatracker. Retrieved March 2, 2023, from https://datatracker.ietf.org/doc/html/rfc9293

[4] TCP Flow control animation. Flow control. (n.d.). https://www2.tkn.tu-berlin.de/teaching/rn/animations/flow/

[5] Go-Back-N Protocol. Go-back protocol. (n.d.). https://computerscience.unicam.it/marcantoni/reti/applet/GoBackProtocol/goback.html

[6] The catalog of Design Patterns. Refactoring.Guru. (n.d.). https://refactoring.guru/design-patterns/catalog

[7] Braden, R. T. (1989, October 1). RFC 1122: Requirements for internet hosts - communication layers. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc1122

[8] Sargent, M., Chu, J., Paxson, Dr. V., & Allman, M. (2011, June 22). RFC 6298: Computing TCP's retransmission timer. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc6298

[9] Floyd, S., Mahdavi, J., Mathis, M., & Romanow, Dr. A. (1996, October 1). RFC 2018: TCP Selective Acknowledgment Options. IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc2018

[10] NPM. npm. (n.d.). https://www.npmjs.com/

[11] React. (n.d.). https://react.dev/

[12] API reference. Redux. (2022, April 17). https://redux.js.org/api/api-reference Kale, C. (1991, January).

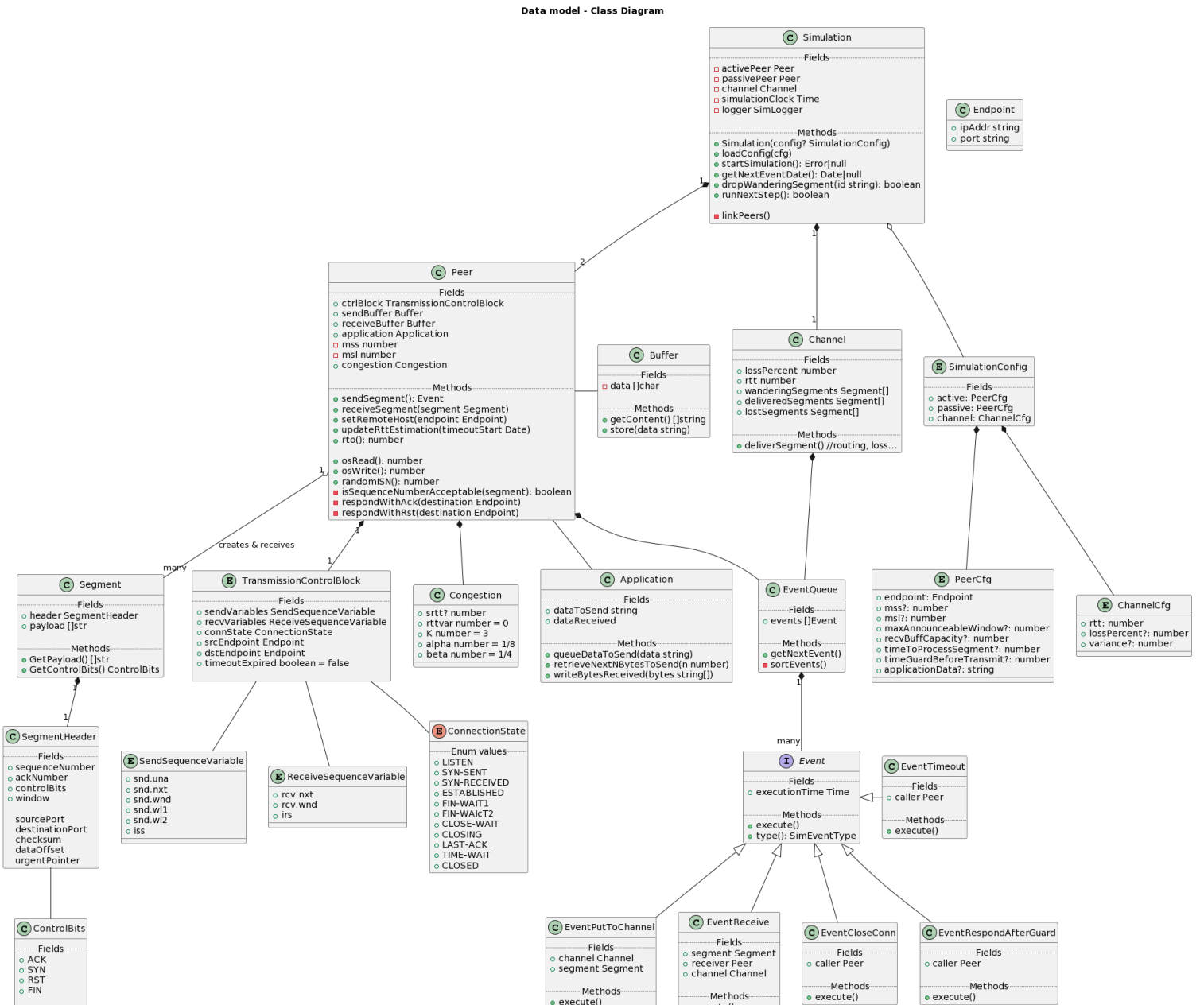
[13] MozDevNet. (n.d.). ECMAScript - Glosario de mdn web docs. Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN. https://developer.mozilla.org/es/docs/Glossary/ECMAScript

[14] Concepts. webpack. (n.d.). https://webpack.js.org/concepts/ Quick start. • React. (n.d.). Retrieved March 5, 2023, from https://beta.reactjs.org/learn

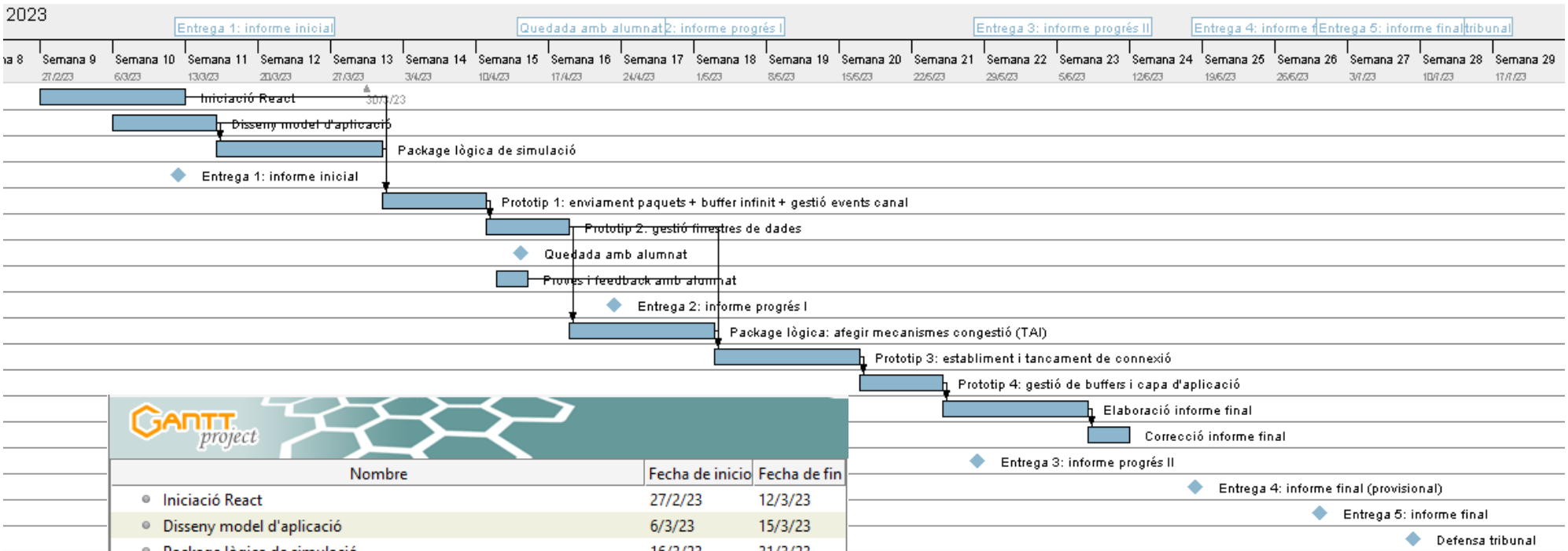
[15] Okonta, A. O. (2023). React.js design patterns: Learn how to build scalable react apps with ease (English edition). BPB PUBLICATIONS.

APÈNDIX

A1.DIAGRAMA DETALLAT UML DE CLASSES DEL BACK-END



A2. DIAGRAMA DE GANTT DURANT LA PLANIFICACIÓ INICIAL



Nombre	Fecha de inicio	Fecha de fin
Iniciació React	27/2/23	12/3/23
Disseny model d'aplicació	6/3/23	15/3/23
Package lògica de simulació	16/3/23	31/3/23
Entrega 1: informe inicial	12/3/23	12/3/23
Prototip 1: enviament paquets + buffer infinit + gestió events canal	1/4/23	10/4/23
Prototip 2: gestió finestres de dades	11/4/23	18/4/23
Quedada amb alumnat	14/4/23	14/4/23
Proves i feedback amb alumnat	12/4/23	14/4/23
Entrega 2: informe progrés I	23/4/23	23/4/23
Package lògica: afegir mecanismes congestió (TAI)	19/4/23	2/5/23
Prototip 3: establiment i tancament de connexió	3/5/23	16/5/23
Prototip 4: gestió de buffers i capa d'aplicació	17/5/23	24/5/23
Elaboració informe final	25/5/23	7/6/23
Correcció informe final	8/6/23	11/6/23
Entrega 3: informe progrés II	28/5/23	28/5/23
Entrega 4: informe final (provisional)	18/6/23	18/6/23
Entrega 5: informe final	30/6/23	30/6/23
Defensa tribunal	9/7/23	9/7/23