Práctica 1 - Congestion Control

1. Execution descritpion

Executable name: code.py

Programming Language: Python

Necessary libraries: Matplotlib (graphics display)

Usage: code.py [-h] [-f F] [-fr FR] [-plot PLOT] [-type TYPE] [-printt PRINTT]

Optional arguments :

-h, –help show this help message and exit
-f F enter .tr file
-fr FR enter .rtt file (opcional)
-plot PLOT [cw] to plot congestion window, [to] to plot time out
-type TYPE [or_to] to plot original, [kar] to plot Karn/Patridge, [jak] to plot Jacobson/Karels, [or_cw] to plot original,[aimd] to plot AI/MD, [ss] to plot SlowStart, [all] to plot all
-printt PRINTT [or_to] to print original, [kar] to print Karn/Patridge , [jak] to print Jacobson/Karels, [aimd] to print AI/MD, [ss] to print SlowStart

2. Traces generated by NS

2.1. TimeOut

Original Algorithm: original.tr, original.rtt

Jacobson/Karels Algorithm: jacobson.tr, jacobson.rtt

Karn/Patridge Algorithm: karn.tr, karn.rtt

2.2. CongestionWindow

Original Algorithm: original.tr, original.rtt

AI/MD Algorithm, Jacobson/Karels Algorithm: additive.tr, additive.rtt

SlowStart Algorithm, Jacobson/Karels Algorithm: slow.tr, slow.rtt

3. Executions used

3.1. Plots TimeOut

TimeOut Plot Original Algorithm :

#python code.py -f original.tr -plot to -type or_to

TimeOut Plot Jacobson/Karels Algorithm :

#python code.py -f jacobson.tr -plot to -type jak

TimeOut Plot Karn/Patridge Algorithm :

#python code.py -f karn.tr -plot to -type kar

3.2. Plots CongestionWindow compared with .rtt files

CongestionWindow Plot Original Algorithm, Jacobson/Karels Algorithm :

#python code.py -f original.tr -fr original.rtt -plot cw -type or_cw

CongestionWindow Plot AI/MD Algorithm, Jacobson/Karels Algorithm :

#python code.py -f additive.tr -fr additive.rtt -plot cw -type aimd

CongestionWindow Plot SlowStart Algorithm, Jacobson/Karels Algorithm :

#python code.py -f slow.tr -fr slow.rtt -plot cw -type ss

3.3. Print values (Matplotlib library not required)

TimeOut Print Original Algorithm :

#python code.py -f original.tr -printt or_to

TimeOut Print Jacobson/Karels Algorithm :

#python code.py -f jacobson.tr -printt jak

TimeOut Print Karn/Patridge Algorithm :

#python code.py -f karn.tr -printt kar

CongestionWindow Print Original Algorithm, Jacobson/Karels Algorithm :

#python code.py -f original.tr -printt or_cw

CongestionWindow Print AI/MD Algorithm, Jacobson/Karels Algorithm :

#python code.py -f additive.tr -printt aimd

CongestionWindow Print SlowStart Algorithm, Jacobson/Karels Algorithm :

#python code.py -f slow.tr -printt ss

4. Plots

Les gràfiques dels TimeOut s'han realitzat en funció del número de seqüència com es mostra als apunts de FlowControl, les gràfiques de l'ns són en funció del temps per lo que no s'han pogut comparar.

4.1. TimeOut Plot Original Algorithm

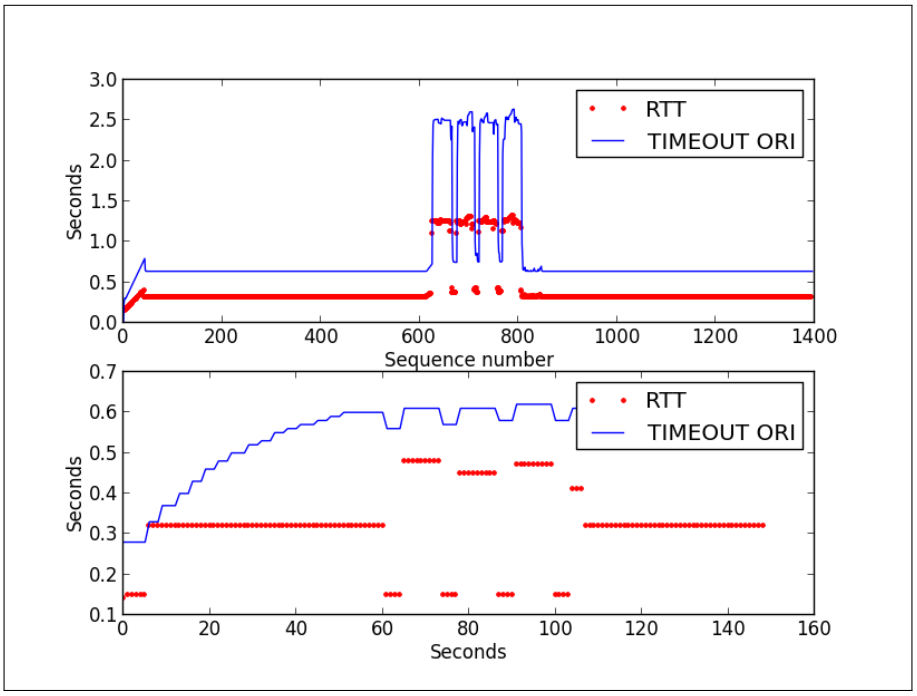

Figura 1: Exploits

4.2. TimeOut Plot Jacobson/Karels Algorithm

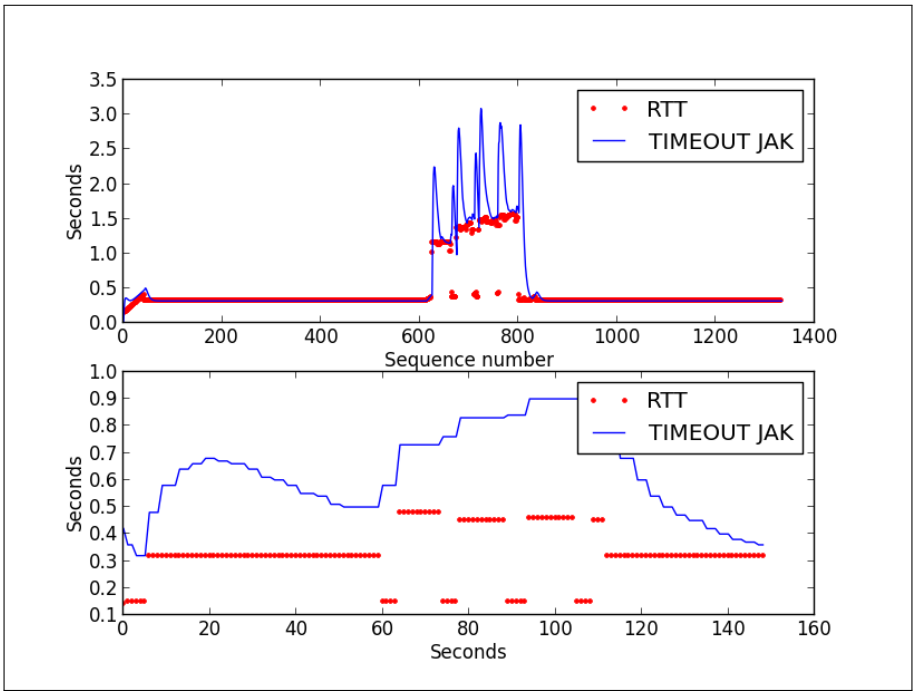

Figura 2: Exploits

4.3. TimeOut Plot Karn/Patridge Algorithm

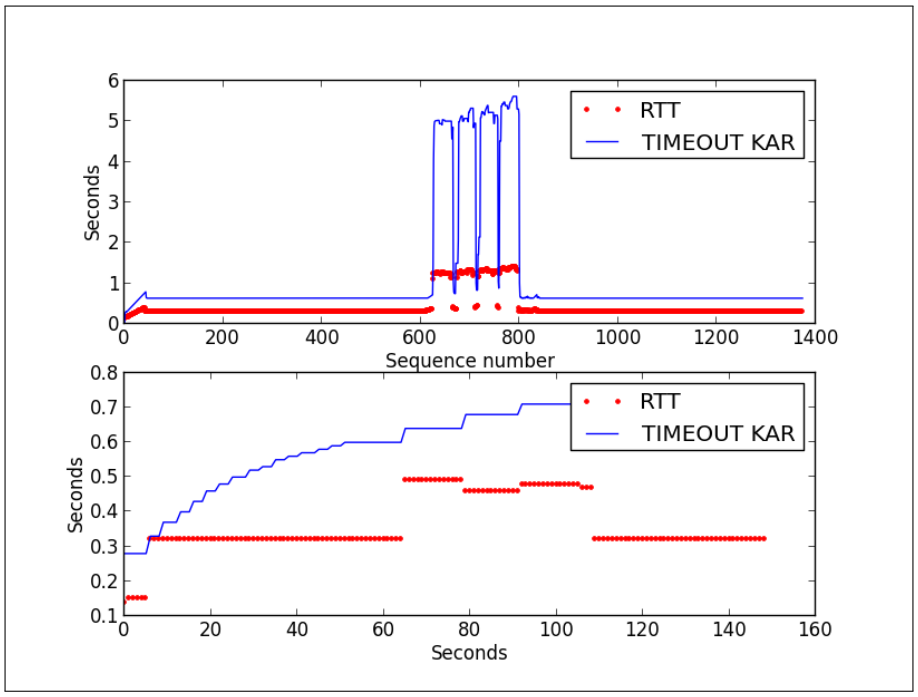

Figura 3: Exploits

4.4. CongestionWindow Plot Original Algorithm, Jacobson/Karels Algorithm

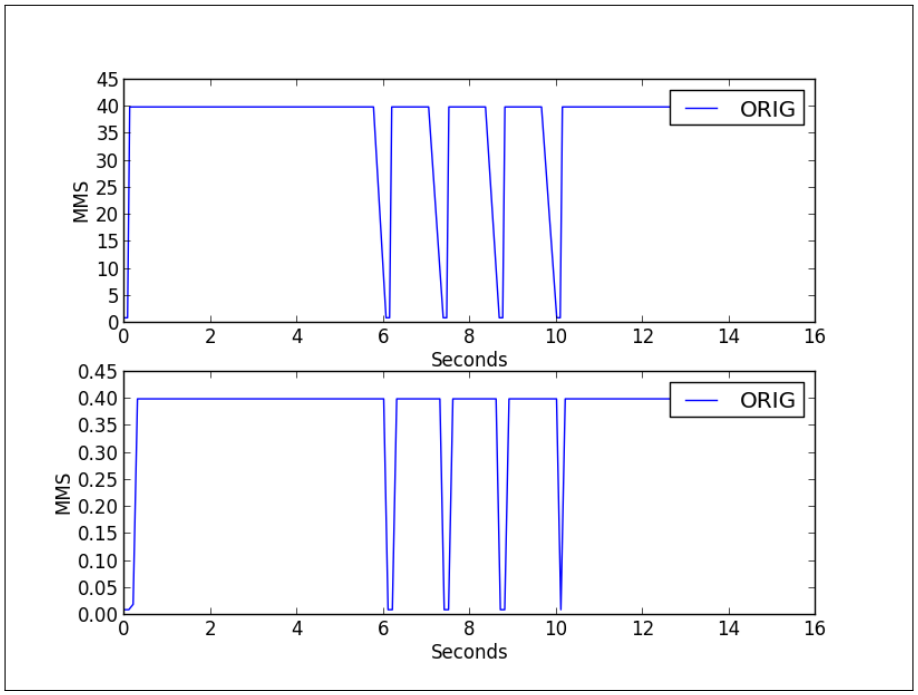

Figura 4: Superior generat a partir del .rtt de ns, inferior generat a partir del .tr

4.5. CongestionWindow Plot AI/MD Algorithm, Jacobson/Karels Algorithm

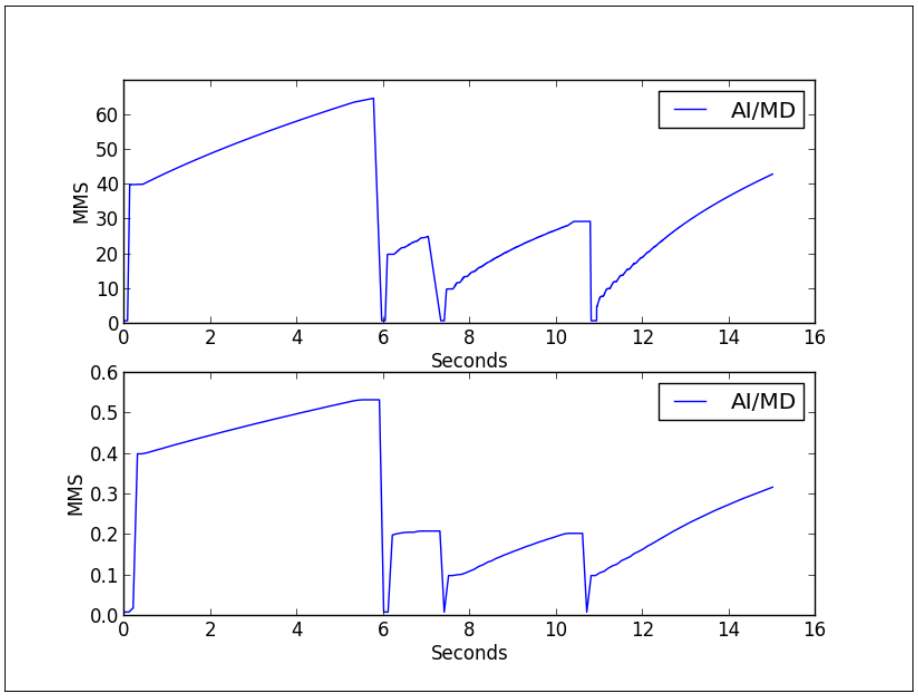

Figura 5: Superior generat a partir del .rtt de ns, inferior generat a partir del .tr

4.6. CongestionWindow Plot SlowStart Algorithm, Jacobson/Karels Algorithm

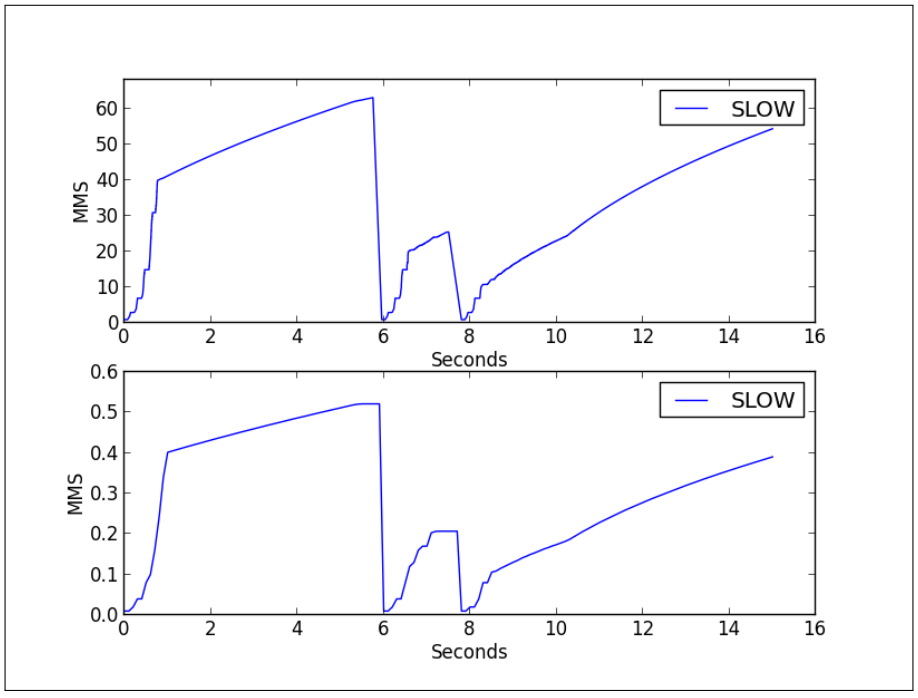

Figura 6: Superior generat a partir del .rtt de ns, inferior generat a partir del .tr

5. Script explaination

def read(data,filename): Emmagatzema la informació de l'arxiu .tr en un diccionari (data), amb el número de seqüència com a clau. A l'interior s'emmagatzema una llista amb totes les trames de la seqüència, amb un split de tots els camps.

def startfinishTwo(data): Emmagatzema a les llistes startctwo i finishctwo els punt d'inici i finalització de cada comunicació, marcades per l'inici de la transmissió i la recepció de l'ACK de confirmació de rebuda de la mateixa. En cas de no trobar ACK es marca el paquet, ja que voldrà dir que s'ha perdut i el temps del TimeOut s'ha exhaurit, manant una retransmissió del TCP.

def rtt_f(startctwo,finishctwo): Es calcula el RTT restant el temps d'arribada del ACK amb el temps en que s'envia el paquet TCP.

def rttestimated_f(rtt): Es calcula el RTTEstimat i el TimeOut mitjançant l'algorisme Original, i s'emmagatzemen a les llistes rttestimated i timeout corresponentment. A l'inici es realitza una inserció de 0, per a que la estimació es realitzi amb el rtt del paquet anterior i no el seu mateix.

def rttestimated_jak_f(rtt): Es calcula el RTTEstimat i el TimeOut mitjançant l'algorisme de Jacobson/Karels, i s'emmagatzemen a les llistes rttestimated_jak i timeout_jak corresponentment. A l'inici es realitza una inserció de 0, per a que la estimació es realitzi amb el rtt del paquet anterior i no el seu mateix.

def rttestimated_kar_f(rtt): Es calcula el RTTEstimat i el TimeOut mitjançant l'algorisme de Karn/Patridge, i s'emmagatzemen a les llistes rttestimated_kar i timeout_kra corresponentment. A l'inici es realitza una inserció de 0, per a que la estimació es realitzi amb el rtt del paquet anterior i no el seu mateix. Quan s'executen els algorismes en funció de la detecció dels TimeOut marcats a la funció startfinishTwo. També es te en compte la retransmissió de paquets amb el diccionari retransmsegments de la funció retransmited_pack.

def retransmited_pack(): Es comprova si s'ha introduït el mateix paquet dos cops dins del mateix número de seqüència, el que significaria que el paquet s'ha tornat a enviar al haver-se perdut.

def oversteppedTimeout(): Detecta els TimeOut, els punts on el RTT és més elevat que el TimeOut. S'hagués pogut detectar els TimeOut mitjançant els flags de les traces amb l'activació del bit A, però s'ha volgut realitzar d'aquesta manera per intentar aproximar-se més al càlcul que es realitza en l'entorn real.

def readcont(datacont,filename): Es torna a parcejar el arxiu .tr ara en l'ordre d'arribada de les traces, per poder abordar el càlcul de la Finestra de Congestió.

def congestionWindow_o(): Es calcula la Finestra de Congestió amb l'algorisme Orgiginal, mitjançant la consulta a la llista overstepped on es troben els TimeOut, s'aplica l'algorisme en funció de l'arribada d'un paquet ACK o la execució d'un TimeOut. S'insereix el valor del resultat a la llista cwnd_o i el seu temps d'execució a la llista cwnd_o_time.

def congestionWindow_aimd(): Es calcula la Finestra de Congestió amb l'algorisme d'Increment Aditiu/Decrement Multiplicatiu, mitjançant la consulta a la llista overstepped on es troben els TimeOut, s'aplica l'algorisme en funció de l'arribada d'un paquet ACK o la execució d'un TimeOut. S'insereix el valor del resultat a la llista cwnd_aimd i el seu temps d'execució a la llista cwnd_aimd_time.

def congestionWindow_ss(): Es calcula la Finestra de Congestió amb l'algorisme de SlowStart, mitjançant la consulta a la llista overstepped on es troben els TimeOut, s'aplica l'algorisme en funció de l'arribada d'un paquet ACK o la execució d'un TimeOut. S'insereix el valor del resultat a la llista cwnd_ss i el seu temps d'execució a la llista cwnd_ss_time.

def plottimeout(rtt,timeout,timeout_kar,timeout_jak,rttestimated): Dibuixa el gràfic dels TimeOut dels algorismes. Si s'insereix l'arxiu .rrt a comparar mitjançant els argument, realitza una doble gràfica per visualitzar la comparació entre els resultats calculats a la pràctica i els obtinguts per l'ns.

def plotcongestionw(cwnd_o,cwnd_o_time,cwnd_aimd,cwnd_aimd_time,cwnd_ss,cwnd_ss_time,cw_rtt_fil
Dibuixa el gràfic de les Finestre de Congestió dels algorismes. Si s'insereix l'arxiu .rrt a comparar mitjançant els argument, realitza una doble gràfica per visualitzar la comparació entre els resultats calculats a la pràctica i els obtinguts per l'ns.

def print_to(listt): Per printar el valor dels TimeOut quan es requereixi mitjançant l'argument -printt.

def print_cw(cwnd,cwndt): Per printar el valor de les Finestres de Congestió quan es requereixi mitjançant l'argument -printt.

def parseRttFiles(cw_rtt_file,t_rtt_file,rtt_rtt_file,to_rtt_file,filename): Parseja els arxius .rtt amb els resultats de l'ns per realitzar la comparació amb els resultats obtinguts.

if __name__ == "__main__" :

- Mitjançant la llibreria argparse s'estableixen i tot els arguments que s'utilitzaran.
- Menú d'ajuda en cas de no introduir arxiu .tr per analitzar.
- Si s'introdueix arxiu per comparar es procedeix a parcejar-lo.
- No es carrega la llibreria Matplotlib si no s'inicia l'argument -plot, per poder executar les impressions dels valor sense tenir que instal·lar la llibreria.
- Execució de totes les funcions per la realització del càlculs dels TimeOut i les Finestres de Congestió.

6. Conclusions

Si haguéssim de tornar a realitzar la pràctica, realitzaríem una simulació del protocol TCP des del punt de vista de l'emissor i del receptor, en comptes de realitzar el parcejament dels arxius, ja que a pesar de la dificultat inicial els càlculs posteriors es realitzarien més fàcilment.