Colaborare şi concurenţă în Internet

Mihai Budiu — mailto:mihaib+@cs.cmu.edu
http://www.cs.cmu.edu/~mihaib/

aprilie 2000

Subiect: slăbiciuni exploatabile ale protocolului TCP

Cunoştinţe necesare: cunoştinţe elementare despre reţele de calculatoare şi TCP/IP

Cuvinte cheie: fereastră, congestie, confirmare (acknowledgement)

Cuprins

Poate cel mai mare succes al ingineriei contemporane este Internetul. Acesta este cu siguranţă cel mai mare sistem creat vreodată de umanitate, care leagă sute de milioane de utilizatori de pe toate continentele. Admirabilă este proiectarea sa, care a permis sistemului să crească şi să funcţioneze excelent la o scară nemaiîntîlnită.

Unele dintre calităţile sale derivă cu certitudine din faptul că a fost proiectat iniţial ca o reţea militară, care este capabilă să supravieţuiască chiar unui atac nuclear, continuînd să funcţioneze chiar în prezenţa distrugerii unei porţiuni substanţiale din infrastructură. Dar deşi Internetul a fost proiectat pentru a rezista unor atacuri *dinafară*, nimeni nu a anticipat în design-ul original că multe probleme vor apărea *dinauntru*.

În acest articol voi prezenta trei astfel de slăbiciuni de concepţie, care permit unor calculatoare lipsite de scrupule să interfere în mod dramatic cu comportarea reţelei. Vom vedea apoi şi unele soluţii propuse pentru a contra-ataca astfel de acţiuni.

Acest text este bazat pe un articol al lui Stefan Savage, un excelent cercetător în domeniul reţelelor de calculatoare, care în curînd îşi va obţine doctoratul de la prestigioasa universitate Washington din Seattle.

1 Arhitectura Internetului

În PC Report au apărut o mulţime de articole despre arhitectura Internetului. Aceasta este conceptual simplă, dar destul de complicată pentru a umple o carte întreagă. De aceea, în această secţiune vom revizui doar unele dintre trăsăturile arhitecturale esenţiale care ne vor permite să înţelegem de ce apar unele probleme.

Pietrele de temelie ale Internetului sunt două protocoale[1] numite IP (Internet Protocol) şi TCP (Transport Control Protocol). IP este un protocol foarte simplu, care rulează pe toate calculatoarele care compun Internetul; IP primeşte pachete de date de la un calculator cu rugămintea de a fi livrate la o anumită destinaţie, şi încearcă să trimită pachetul acolo. Am vorbit despre IP cu mai multe ocazii (de pildă în PC Report din mai 1998, aprilie şi mai 1999; puteţi citi aceste articole din pagina mea de web).

În textul de faţă vom vorbi însă despre TCP. IP nu oferă nici un fel de garanţii asupra datelor trimise; TCP se foloseşte de IP pentru a trimite date în mod ne-fiabil, dar repară lipsurile lui IP: TCP oferă o transmisiune complet fiabilă a tuturor datelor transmise de la un capăt la altul.

2 Protocolul TCP

De fapt protocolul TCP rezolvă simultan nu mai puţin de *trei* probleme diferite:

Conexiuni: TCP construieşte *conexiuni virtuale* între doi participanţi la comunicaţie. Aceste conexiuni definesc durata comunicaţiei: pachete trimise între aceleaşi două calculatoare în conexiuni diferite nu sunt confundate între ele. De asemenea, conexiunile rezervă resurse pe calculatoarele care participă la comunicaţie, pentru a face procesarea datelor eficientă[2]. Managementul conexiunilor se face folosind nişte mesaje speciale şi un protocol simplu dar interesant numit "strîngerea de mînă triplă" (three-way handshake). În acest articol însă nu vom mai menţiona stabilirea şi desfacerea conexiunilor.

[1] Un protocol este un set de reguli care descrie cum participanţii la o discuţie (calculatoarele) conversează între ei pentru a se înţelege.

[2] În acelaşi fel în care majoritatea limbajelor de programare oferă funcţii speciale pentru a "deschide" fişiere şi apoi pentru a accesa conţinutul fişierelor, comunicaţia este separată într-o fază de stabilire a conexiunii şi una de transfer al datelor.

Fiabilitate: TCP transformă comunicaţia ne-fiabilă oferită de IP într-o comunicaţie a unor fluxuri de date (streams) 100% *fiabilă*. Metoda folosită este absolut clasică: detecţia pierderilor prin confirmări de la receptor (acknowledgements), şi retransmiterea datelor pierdute pe parcurs. Vom vedea mai multe exemple despre folosirea acestei scheme.

Congestie: în fine, TCP are o funcţie foarte foarte importantă, anume *controlul congestiei*. Dacă sute de calculatoare încearcă să folosească un segment de reţea cu capacitate mică, TCP le permite să împartă în mod echitabil această resursă limitată.

În lipsa acestei funcţiuni a lui TCP, reţeaua ar intra rapid în ceea ce se numeşte *colaps de congestie*: calculatoarele ar trimite pachete mai repede decît reţeaua ar putea transporta, aceste pachete ar trebui să se piardă, ceea ce ar cauza re-transmiterea pachetelor pierdute, etc. Rezultatul ar fi întreruperea aproape completă a oricărei comunicaţii utile.

2.1 Specificaţia TCP

Protocoalele de funcţionare a Internetului sunt publicate într-o formă extrem de bizară: în nişte documente numite “Request For Comments”, sau RFC. Exista mai mult de 2000 de astfel de documente, dar nu toate sunt la fel de importante. Multe astfel de documente sunt de fapt vide (nu au fost niciodată publicate), unele sunt nişte glume sau poezioare, iar unele sunt documente foarte serioase. Toate sunt disponibile în formă electronică într-o mulţime de locuri de pe Internet.

Aceasta metodă de formalizare a arhitecturii stă într-un contrast dramatic faţă de metoda folosită de comitetele de standardizare internaţionale, care publică documente extrem de groase, revizuite pentru multă vreme, şi foarte “serioase”.

RFC-urile ilustrează evoluţia extrem de rapidă şi “impulsivă” a Internetului; multe RFC-uri sunt revizuite în mod repetat.

Specificaţia TCP/IP se întinde pe mai multe RFC-uri. Definiţia oficială a fost dată în 1981 în RFC 793. Corecţii şi lămuriri apar în RFC 1122. Din păcate alte detalii ale funcţionării sale se întind efectiv pe zeci de RFC-uri.

2.2 Date şi confirmări

TCP este un protocol simetric (full-duplex), în care ambele capete pot transmite date. E mai simplu însă să studiem problema ca şi cum unul dintre participanţi doar trimite date, iar celălalt doar le receptează.

Pentru a descrie comunicaţia vom folosi desene ca în figura 1.

TCP funcţionează cam ca scrisorile cu confirmare de primire de la poşta: poşta este un mecanism ne-fiabil de transmisiune, care pierde scrisori. De aceea trimiţătorul poate să ceară în mod explicit ca cealaltă parte să confirme primirea plicului. Cînd poştaşul livrează un plic, receptorul trebuie să semneze de primire. Apoi confirmarea de primire este trimisă înapoi tot prin poştă.

Pentru a obţine fiabilitate, emiţătorul aşteaptă ce aşteaptă, iar dacă nu primeşte nici o confirmare, mai trimite încă o dată scrisoarea. În clipa cînd a primit o confirmare, ştie că scrisoarea a ajuns la celălalt capăt.

Observaţi că lipsa recepţiei unei confirmări poate indica două lucruri:

- Ca în figura 2, datele se pot pierde.

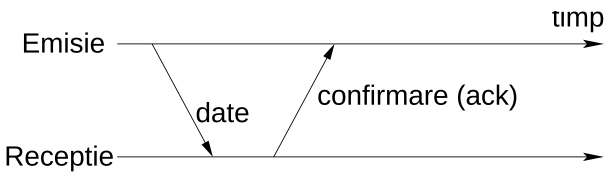

Figura 1: O diagramă care indică schimbul de mesaje între emiţător şi receptor. Timpul este simbolizat de săgeţile orizontale şi curge spre dreapta. Săgeţile oblice indică "traiectoria" pachetelor între emisie şi recepţie. Acest desen indică o conversaţie tipică, în care un pachet de date este trimis de emiţător şi o confirmare soseşte apoi de la receptor. Vom prescurta pachetele de confirmare cu abrevierea standard, ACK (acknowledgement).

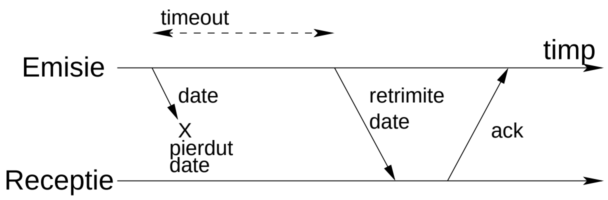

Figura 2: Dacă datele sunt pierdute de protocolul IP, protocolul TCP re-transmite după un anumit timp pachetul care nu a fost confirmat. În clipa în care primeşte confirmarea, ştie că pachetul a ajuns la destinaţie.

- Ca în figura 3, confirmarea se poate pierde.

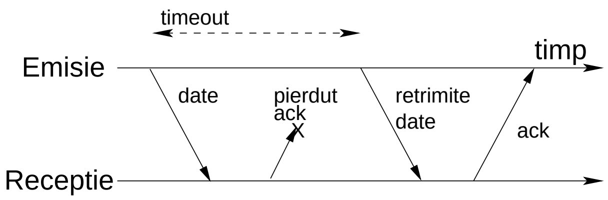

Figura 3: Dacă confirmarea este pierdută de protocolul IP, protocolul TCP re-transmite după un anumit timp pachetul care nu a fost confirmat. Retransmiţînd datele nu facem decît o oarecare risipă; dar pentru că emiţătorul nu poate distinge nicicum între pierderea datelor şi pierderea confirmării, nu are nici o altă alternativă.

2.3 Ferestre glisante

Schema cu confirmare este foarte eficace pentru a combate pierderea de pachete, dar este ineficace dacă pachetele călătoresc mult timp pînă la destinaţie[3]. Pentru că pînă la întoarcerea primei confirmări emiţătorul stă degeaba, o grămadă de resurse sunt irosite.

[3]De fapt ineficienţa este dată de mărimea care se numeşte *capacitatea reţelei*, care este produsul dintre lărgimea de bandă şi durata propagării unui pachet; cu cît această valoare este mai mare, cu atît ineficienţa unei scheme care aşteaptă o confirmare înainte de a trimite un pachet este mai mare. Dar acest subiect depăşeşte interesul acestui articol.

Din cauza asta, TCP încearcă sa trimită mai multe pachete la rînd, înainte de a primi confirmarea pentru fiecare din ele. Astfel, emiţătorul menţine o *fereastră*, care este lista pachetelor trimise dar încă ne-confirmate. De îndată ce primeşte confirmările, TCP avansează fereastra şi trimite pachete noi.

Modul în care se încrucişează pachetele de date şi confirmările este ilustrat de figura 4.

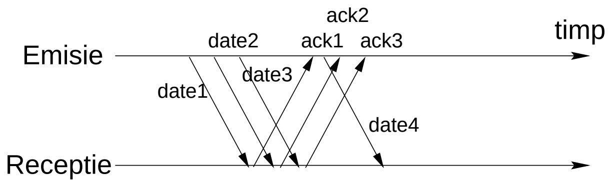

Figura 4: Pentru a nu irosi resursele reţelei, TCP de obicei trimite mai multe pachete înainte de a primi confirmarea primului. Această figură ilustrează o fereastră de emisie de dimensiune 3: TCP trimite trei pachete, şi trimite apoi pachete noi doar dacă primeşte confirmări pentru cele trimise.

2.4 Controlul congestiei

În fine, discutăm aici pe scurt despre modul în care TCP face faţă congestiei din reţea. Această secţiune este foarte importantă, pentru că toate cele trei atacuri pe care le discutăm în acest text manipulează slăbiciuni în specificarea şi implementarea acestei scheme.

Asumpţia de bază a lui TCP este că reţeaua de transmisiune este inerent fiabilă. TCP presupune că singura cauză de pierderi în reţea este congestia[4].

Ce este congestia? Să ne imaginăm un ruter[5] în reţea, care are trei interfeţe cu capacitatea de 1 megabit/secundă. Să ne imaginăm că pe două dintre interfeţe se află nişte trimiţători care emit cu 800 kbps, iar pe a treia se află receptorul datelor. Ei bine, suma celor două fluxuri de 800 kbps depăşeşte capacitatea legăturii spre receptor. Ca atare, pachetele care intră în ruter nu pot ieşi toate în timp util. Ruterul va trebui să stocheze pachetele în exces în memoria internă. Dar dacă această situaţie durează destul de mult, ruterul trebuie să consume cîte 600 kbps de memorie pentru a stoca aceste pachete în exces. După 30 de secunde de trafic neîntrerupt, e nevoie de peste 2M de RAM. Atunci cînd memoria ruterului este complet utilizată, ruterul nu mai are nimic de făcut decît să renunţe la a mai trimite pachetele spre ieşire (pur şi simplu sunt şterse din memorie). Aceasta este congestia, şi rezultatul ei, pierderea de pachete.

Deci TCP îşi imaginează că atunci cînd un pachet nu este confirmat în timp util, s-a pierdut fie pachetul fie confirmarea, deci reţeaua este congestionată. Ca atare, trimiţătorul imediat reduce rata de transmisiune, pentru a reduce numărul de pachete din reţea.

Dacă toţi transmiţătorii care detectează congestie reduc rata simultan, efectul este că ruterele din reţea primesc mai puţine date la intrare, şi au timp să golească pachetele stocate în memorie trimiţîndu-le la destinaţie. Pentru că marea majoritate a calculatoarelor din reţea folosesc protocolul TCP, acest comportament duce la dispariţia congestiei.

[4]Acest lucru nu este adevărat în anumite tipuri de reţele, cum ar fi cele radio; de aceea TCP nu funcţionează prea eficient în astfel de circumstanţe.

[5]Un ruter este un calculator intermediar în reţea, care serveşte drept una dintre legăturile dintre sursă şi destinaţie.

Cum reduce emiţătorul TCP rata de transmisiune? Într-un mod foarte simplu: reduce dimensiunea ferestrei, şi măreşte dimensiunea duratei de timeout. În felul acesta va face ca mai puţine pachete să fie injectate în reţea.

3 Colaborare şi concurenţă în Internet

Am văzut pînă acum care sunt mecanismele prin care TCP îşi îndeplineşte misiunile (unele dintre ele). Pentru a înţelege de ce aceste mecanisme nu sunt suficiente pentru orice scenariu, trebuie să aruncăm o privire asupra participanţilor la trafic din Internet şi asupra intereselor lor contradictorii.

3.1 Spectrul participanţilor

Într-o excelentă expunere despre arhitectura Internetului, Stefan Savage a prezentat gradaţia din figura 5.

Figura 5: Intenţiile participanţilor la comunicaţie în Internet variază între dorinţa de colaborare reciprocă pînă la intenţii distructive. Proiectanţii Internet-ului original au crezut că utilizatorii se vor plasa doar în regiunea "cooperării"; situaţia de astăzi este însă net diferită.

Cooperare: în continuare multe calculatoarelor din Internet cooperează pentru buna funcţionare a reţelei. De exemplu, toate ruterele comunică între ele informaţii despre structura reţelei, ceea ce permite datelor să circule între oricare două puncte, chiar dacă ele aparţin unor domenii administrative distincte.

Indiferenţă: probabil relaţia cea mai frecventă între două puncte terminale din Internet este cea de indiferenţă. De exemplu calculatorul de la `www.agora.ro` este indiferent la acţiunile mele, mai puţin dacă eu vreau sa obţin exact ceea ce el serveşte (pagini de web).

Între cooperare şi indiferenţă este o distincţie: de exemplu eu aş putea să doresc să măsor nişte parametri ai reţelei între mine şi calculatorul de la Agora, ca să ştiu de pildă unde se pierd pachete. Serverul de web `www.agora.ro` nu îmi va oferi însă nici un suport; nu are nici un avantaj din faptul că eu pot să măsor parametrii reţelei.

Concurenţă: pe de altă parte între mine şi oricare alt calculator care accesează acelaşi server de web este o relaţie de concurenţă: dacă serverul de web mi-ar consacra mie toate resursele, atunci aş primi un răspuns mult mai rapid şi mai prompt. (Nu e vorba de concurenţă în sens economic, ci de competiţie pentru resurse limitate ale reţelei.)

Duşmănie: în fine, după cum stau mărturie capetele de afiş din lunile trecute, Internetul permite atacuri pe scară largă împotriva calculatoarelor conectate, de la intruziuni, furt de informaţie (care ţin mai puţin de natura reţelei cît de implementarea sau administrarea defectuoasă a programelor), pînă la atacuri DOS (Denial of Service), care pot interzice accesul la anumite servicii (vedeţi de pildă PC Report din martie 2000).

3.2 Competiţie şi fair-play

În acest articol ne vom focaliza doar asupra uneia dintre aceste relaţii competitive: relaţia de concurenţă. În orice context în care mai multe entităţi concurează asupra unui număr redus de resurse, vom avea de a face cu competiţie. Pentru că în Internet avem de a face cu un număr uriaş de clienţi potenţiali, este virtual imposibil să avem resurse suficiente la dispoziţie pentru a satisface pe deplin pe oricine. Aşa că prezenţa competiţiei este absolut naturală. Atunci care e problema?

Problema este dacă nu cumva accesul la resurse nu este “democratic”. În mod ideal fiecare client ar trebui să obţină la fel de multe resurse cît ceilalţi[6].

Trebuie să fim deci pregătiţi pentru competiţie între feluriţii participanţi la trafic. Întrebarea este: asigură protocoalele din reţea o competiţie “onestă” între participanţi? Altfel spus, *impun* aceste protocoale un comportament echitabil? În mod normal ar trebui să fie imposibil să “păcălim” protocoalele în vreun fel. Serviciul preponderent ca importanţă — în termeni de trafic sau importanţă comercială — în Internet este serviciul de web, aşa că asupra lui ne vom concentra.

Într-un articol scris de Ion Stoica în colaborare cu autorul celui de faţă, în PC Report din mai 1999, am arătat cum un transmiţător care nu respectă semnalele de congestie (în acel caz un transmiţător care folosea protocolul UDP, despre care nu am vorbit în acest text), poate căpăta avantaje în detrimentul tuturor celorlalţi. Să zicem că un server refuză să asculte semnalele de congestie, şi transmite tot timpul la maxim. Celelalte servere, cinstite, vor reduce traficul, şi toată capacitatea reţelei va fi disponibilă pentru ticălos.

Atîta vreme cît traficul “ticăloşilor” nu cauzează congestie (adică capacitatea reţelei este suficientă pentru a transporta acest trafic), ei vor putea transmite la capacitate maximă, iar participanţii oneşti nu vor transmite decît pe capacitatea rămasă. Dacă însă nu e suficientă capacitate nici pentru cei incorecţi, reţeaua ajunge imediat în colaps de congestie, şi nimeni nu mai poate transmite nimic.

Şi totuşi reţeaua funcţionează rezonabil. De ce oare? Ce împiedică comportamente anti-sociale pe scară largă?

3.3 Oportunitate şi motiv

Aparent am putea fi tentaţi să spunem: deşi protocoalele nu pot împiedica comportamente necinstite, totuşi în practică nu le putem folosi în mod ilegal, din cauza separaţiei între părţi. Iată argumentul:

Pentru a putea comite o faptă anti-socială trebuie să avem două elemente deodată: *motivul* şi *ocazia*. Să le analizăm pe fiecare în parte.

Motiv

Clientul este în directă competiţie cu ceilalţi clienţi; un client are deci toate motivele să obţină cît mai mult trafic de la un server, în pofida celorlalţi.

Serverul: putem spune acelaşi lucru: un server are toate intenţiile să transmită cît mai multe date cît mai rapid, pentru a-şi face proprii clienţi cît mai satisfăcuţi (în detrimentul clienţilor celorlalte servere, ale concurenţei).

[6] De fapt, mai cinstit ar fi ca fiecare să obţină după cum plăteşte, dar infrastructura Internetului contemporan nu permite o astfel de diferenţiere; majoritatea cercetării din reţele de calculatoare se desfăşoară în această direcţie: cum putem oferi servicii cu calitate diferită pentru clienţi diferiţi (Quality of Service, QoS).

Oportunitate

Client În traficul de web majoritatea datelor se merg de la server la client: clientul trimite o cerere de date, scurtă, iar serverul răspunde cu o cantitate mare de date. Clientul este un simplu "receptor". Aparent clientul nu poate face nimic pentru a căpăta mai multe date.

Server Serverul are toate intenţiile să trimită cît mai multe date posibil, în detrimentul celorlalte calculatoare care folosesc reţeaua: profitul pentru server este maxim cînd mai mulţi clienţi pot accesa informaţia. Aparent deci serverul poate ignora semnalele de reducere a congestiei din reţea, bazîndu-se pe faptul că celelalte servere vor micşora traficul, ceea ce îi va permite să transmită mai rapid.

În realitate lucrurile nu stau aşa: studiile arată că în cazul serverelor mari de web "gîtuitura" în Internet este de fapt foarte aproape de server. Resursa critică deci nu e partea din reţea comună cu alte servere, ci chiar "ieşirea" serverului la Internet. Mărind congestia deci serverul îşi dă singur cu ciocanul peste degete.

	Motiv	**Oportunitate**
Client	Da	Nu*
Server	Da	Nu

Tabelul 1: Oportunitate şi motivaţie pentru a "înşela" ceilalţi participanţi la trafic. Căsuţa marcată cu asterisc este de fapt incorect etichetată.

4 Trei atacuri asupra TCP

Iată deci că aparent cei care ar putea face rău (cei care trimit multe date, serverele) nu pot fi necinstiţi, pentru că-şi fac rău lor înşile, iar clienţii nu pot face rău, pentru că ei nu transmit multe date, ci doar recepţionează.

În realitate clienţii au la dispoziţie o unealtă foarte puternică cu care pot controla comportarea serverului, fără voia acestuia: confirmările.

Stefan Savage împreună cu colegii din grupul lui de cercetare au demonstrat acest lucru "pe viu": au luat un client Linux şi au făcut modificări minore în implementarea protocolului TCP (sub 100 de linii în total pentru toate atacurile prezentate). Apoi au demonstrat cum acest client, folosit pentru a extrage date de la mai multe site-uri de web foarte importante (deci care nu pot fi suspectate de colaborare), poate monopoliza reţeaua în detrimentul celorlalţi clienţi.

În cele ce urmează voi ilustra pe scurt trei modificări diferite şi impactul fiecăreia dintre ele. Datele experimentale sunt din articolul citat.

4.1 Diviziunea confirmărilor

În TCP datele transmise formează un flux (stream). Fiecare pachet trimite o parte din date, şi indică poziţionarea lor în flux. De pildă, primul pachet ar putea trimite datele de la 0 la 1000, al doilea de la 1001 la 2000, etc. Confirmările pe de altă parte sunt *cumulative*: o confirmare care spune "3001" înseamnă: "am primit toate datele între 0 şi 3000; aştept date începînd de la 3001 în continuare". În felul acesta, dacă o confirmare pentru un pachet se pierde (de exemplu cea pentru pachetul 1001–2000), confirmările ulterioare o pot subsuma, reducînd traficul necesar.

Primul atac e simplu: cînd receptorul primeşte un pachet, să zicem cu datele 0–1000, el va trimite în loc de o confirmare, trei: una pentru 333, una pentru 667, una pentru 1000. Atacul este ilustrat de figura 6.

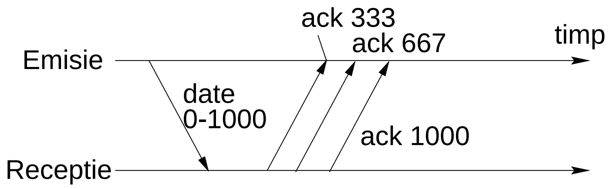

Figura 6: Atacul prin diviziunea confirmărilor. Receptorul primeşte un singur pachet, dar confirmă mai multe bucăţele.

E clar că în felul acesta protocolul rămîne corect. Care e problema?

Problema este în modul în care emiţătorul trebuie să reacţioneze la astfel de mesaje: dacă citim în RFC 2581 vom vedea un paragraf care spune următoarele:

> ...cînd un pachet de confirmare este primit, trimiţătorul măreşte fereastra cu SMSS (Sender Maximum Segment Size)[7].

Nu ne interesează prea tare cît e valoarea lui SMSS (această valoare este stabilită între cele două părţi cînd se stabileşte conexiunea); important e că, trimiţînd mai multe confirmări, receptorul poate mări fereastra emiţătorului aproape în mod arbitrar! Dacă alege să confirme fiecare octet, poate mări fereastra de mii de ori după un singur pachet primit! Figura 7 arată ce se întîmplă după aceea.

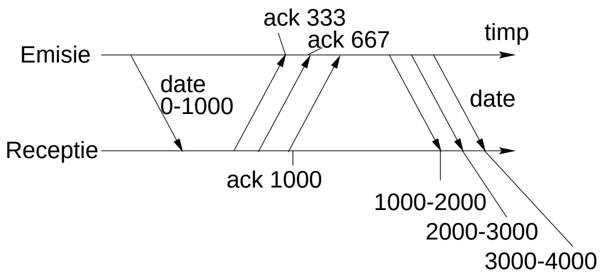

Figura 7: Rezultatul atacului prin diviziunea confirmării. Emiţătorul creşte fereastra cu o cantitate mare (SMSS) pentru fiecare confirmare primită, trimiţînd o rafală de date imediat după primirea confirmărilor.

Asta înseamnă că emiţătorul (serverul) va trimite apoi o rafală de pachete de date. Observaţi că pachetele sunt trimise unul după altul, înainte de a da o şansă reţelei să trimită semnale de

[7] De fapt comportamentul protocolului este uşor diferit în timpul iniţierii şi după detectarea congestiei, dar calitativ rezultatul este acelaşi.

congestie. În mod normal fereastra emiţătorului creşte treptat pînă atinge o valoare de echilibru, care nu produce congestie (această creştere se numeşte “slow start”). Graficul din figura 8 arată că receptorul poate “suge” de la sursă aproape instantaneu o cantitate foarte mare de date.

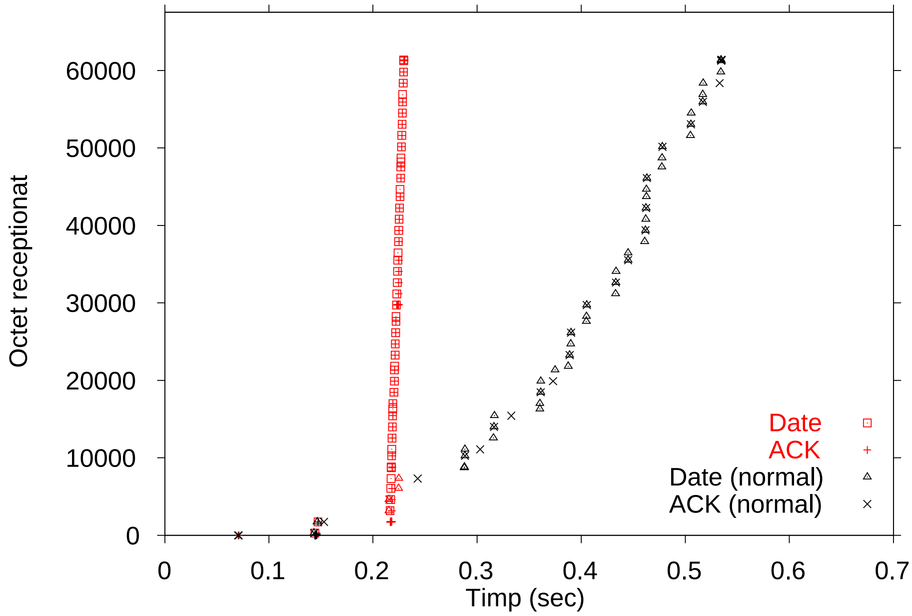

Figura 8: Rata de transfer măsurată la client în Internet pentru două scheme: TCP normal şi TCP modificat cu diviziunea confirmării. TCP-ul care trişează obţine o rată de transfer uriaşă, în detrimentul celorlalte calculatoare din reţea.

4.2 Confirmări duplicate

A doua schemă este şi mai simplă şi se bazează pe aceeaşi slăbiciune din specificaţia TCP. În loc să trimită mai multe pachete de confirmare diferite, receptorul trimite în mod repetat un singur pachet de confirmare. Receptorul poate folosi chiar pachetul de confirmare pentru octetul 1 (ca şi cum n-ar fi primit încă nimic).

Figura 9 arată cum se comportă emiţătorul. Performanţa măsurată în Internet (şi nu într-o reţea de laborator!) este aceeaşi ca pentru schema cu diviziune.

Această schemă este şi mai perfidă decît cea precedentă; pentru cea precedentă serverul ar putea să devină suspicios, pentru că sunt confirmate doar fragmente de pachet[8]; din punct de vedere al serverului un scenariu în care vin mai multe confirmări identice este perfect plauzibil (protocolul IP nu promite că nu duplică pachete).

4.3 Confirmări anticipate

În fine, ultimul atac este mai riscant, pentru că confirmă date care încă nu au fost primite. Ideea

[8]Reţeaua are dreptul să fragmenteze pachetele în bucăţi, dar acest lucru se întîmplă foarte rar, pentru că adesea părţile care comunică aleg o mărime de pachet suficient de mică pentru a nu fi nevoie să fie vreodată fragmentată; motivul este o performanţă mai bună.

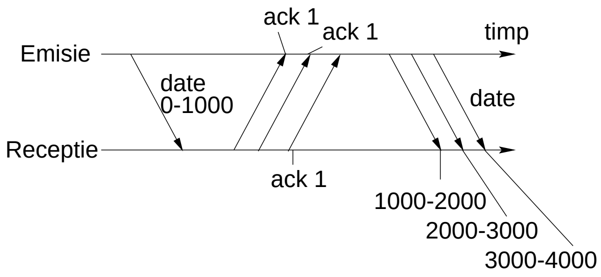

Figura 9: Rezultatul atacului prin duplicarea confirmărilor. Receptorul trimite mai multe confirmări identice; fiecare măreşte mult fereastra emiţătorului, care emite o apoi multe date.

este că confirmările şi datele se încrucişează pe parcurs, şi serverul are impresia că datele au ajuns mult mai repede. Figura 10 arată cum funcţionează acest atac.

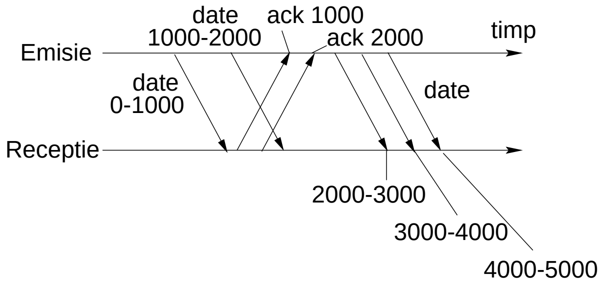

Figura 10: Rezultatul atacului prin anticiparea confirmărilor. Receptorul trimite confirmări chiar pentru pachetele care nu au fost primite; aceste pachete imediat confirmate dau impresia că receptorul este foarte "aproape", şi ca atare cauzează creşterea rapidă a ferestrei.

Desigur, în cazul unei pierderi reale de date, receptorul are probleme, pentru că emiţătorul nu va re-trimite datele deja confirmate. Dar, aşa cum arată cercetătorii de la U Washington, protocolul HTTP, prin care clientul comunică cu serverul de web[9] permite clientului să re-ceară date de la server în mod selectiv. Deci pierderile la nivel TCP pot fi compensate de un client modificat, prin nivele superioare de corecţie a erorilor.

Performanţele schemei cu anticipare de confirmări sunt mai puţin spectaculoase, dar oricum, mult superioare celei a unui client normal (figura 11).

5 Soluţii

Partea cea mai neplăcută din toată povestea asta este că la ora actuală *nu există nici un fel de soluţii practice pentru a împiedica astfel de comportamente*. Lucrările lui Savage şi a echipei sale propun mai multe soluţii, dar toate necesită schimbări în infrastructura Internetului, care schimbări sunt foarte greu de făcut şi mai ales implementat pe scară largă.

Voi menţiona doar pe scurt natura propusă a soluţiei; pentru detalii cei interesaţi pot vedea articolul original. Ideea ar fi ca emiţătorul să poată "verifica" fiecare confirmare venită de la

[9]HTTP foloseşte TCP pentru a transmite date, HTTP indică *cum* arată cererile şi răspunsurile.

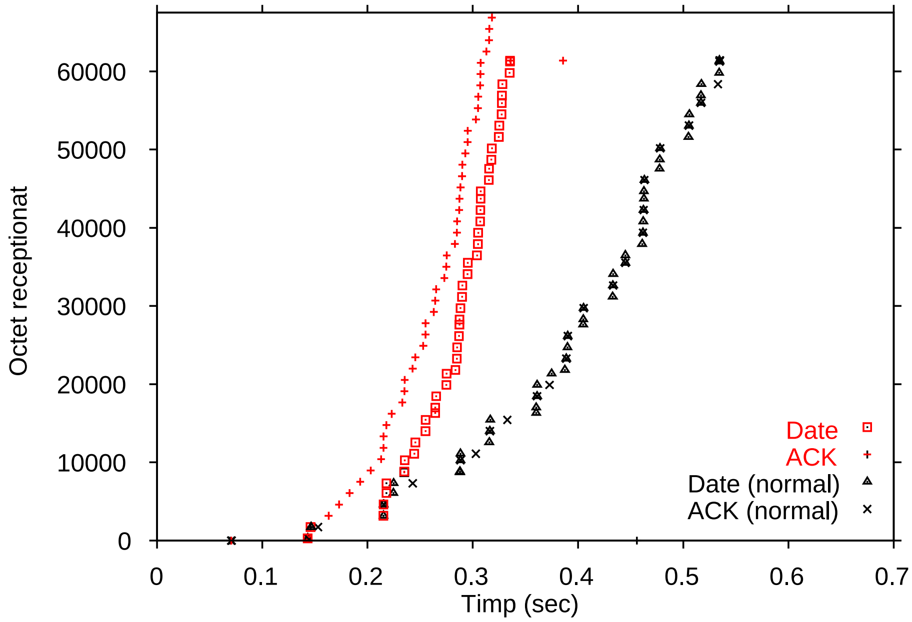

Figura 11: Performanţă comparată măsurată în Internet a unui client normal şi a unui client care foloseşte schema cu confirmări anticipate.

receptor, garantînd că acesta nu încalcă regulile. Pentru aceasta, emiţătorul va pune în fiecare pachet un număr aleator diferit. Receptorul va trebui să includă în confirmare numărul cu pricina. Aceasta va preveni atacuri cu confirmări anticipate.

De asemenea, emiţătorul va ţine minte confirmările primite, şi nu va creşte fereastra pentru confirmări duplicate sau divizate.

6 Alte surse de informaţie

O listă a tuturor RFC-urilor, care are şi o interfaţă de cautare, este prezentă la `http://www.cis.ohio-state.edu/hypertext/information/rfc.html`.

RFC-ul de bază care descrie funcţionarea TCP pentru prevenirea congestiei este RFC 2581 `http://www.cis.ohio-state.edu/htbin/rfc/rfc2581.html`.

Pagina de web a lui Stefan Savage este la `http://www.cs.washington.edu/homes/savage`.

Articolul lui Savage care a inspirat acest text este "TCP Congestion Control with a Misbehaving Receiver", *Stefan Savage, Neal Cardwell, David Wetherall and Tom Anderson*, ACM Computer Communications Review, pp. 71-78, v 29, no 5, October, 1999, `http://www.cs.washington.edu/homes/savage/papers/CCR99.ps`.

Despre comportarea necinstită în Internet, efectele ei, şi soluţii, vedeţi articolul "Scalabilitatea în reţele de comunicaţii de date", Ion Stoica şi Mihai Budiu, din PC Report mai 1999, de asemenea la `http://www.cs.cmu.edu/~mihaib/articles/csfq.ps.gz`.

Pentru o descriere amplă şi plăcută a funcţionării Internetului vedeţi de pildă cartea "Internetworking with TCP/IP, volume I, Principles, Protocols and Architecture", *Douglas Comer*, Prentice Hall 1995.

Un ghid interesant despre RFC-uri este la
`http://www.netbook.cs.purdue.edu/othrpags/page24.htm`. Acesta este de fapt o anexă a cărţii mai sus-citate, care însă din păcate este uşor învechită (de exemplu RFC 2581 despre congestie nu este menţionat de loc).