QNAP

Boulic Guillaume, Émeric Tosi

7 mars 2016

Sommaire

Chapitre 1

RTC : Réseau Téléphonique Commuté

1.1 Analyse sur un lien

1.1.1 Énoncé

Considérons un lien d'un réseau à commutation de circuits permettant de véhiculer de la voix téléphonique.

Chacune des connexions nécessite un débit de 64 Kb/s bi-directionnels. On peut multiplexer simultanément C appels téléphoniques sur ce lien.

Le nombre d'utilisateurs est suffisamment grand pour supposer que les arrivées des nouveaux appels suivent une loi de paramètre, les durées des appels sont supposées suivre une loi exponentielle de paramètre , (1$\hat{=}$ 3 min).

1.1.2 Déterminer à l'aide d'une simulation la probabilité de blocage d'appel pour une charge comprise entre 10 et 70 Erlangs

Blablabla

1.1.3 Faire varier la capacité C de telle sorte que la charge normalisée soit entre 0.5 et 1

Blablabla

1.1.4 Réaliser une étude préliminaire et théorique sur la probabilité de blocage en fonction de la charge et la capacité

Blablabla

$$P(\text{blocage}) = \frac{\frac{\rho^k}{k!}}{\sum_{i=0}^{k} \frac{\rho^i}{i!}}$$

avec ρ la charge en Erlang et k la capacité.

1.1.5 Comparer le taux de blocage expérimental au taux théorique

Blablabla

1.2 Analyse sur un réseau de trois commutateurs

1.2.1 Énoncé

Désormais, nous considérons le réseau composé des 3 nœuds suivant :

Les arrivées sont supposées Poissonniennes sur chacun des nœuds et le trafic se répartit équiprobablement entre les différents nœuds. Les durées des appels sont supposées exponentielles de même paramètre que dans la première partie (1-a). Nous ne considérons pas les appels locaux ni les appels qui n'aboutissent pas (absence).

1.2.2 Déterminer les probabilités de blocage dans le cas où l'on autorise le chemin de débordement en cas de saturation du chemin direct

1.2.3 Comparer avec les résultats de la partie (1-a) (on choisira donc des charges de trafic et des capacités de liens équivalentes)

1.2.4 Problèmes à très forte charge !

Une solution consiste à n'utiliser le chemin de débordement que lorsque celui-ci n'est pas très encombré (en dessous d'un certain seuil d'occupation sur chacun des liens). Cela revient donc à laisser une marge M aux appels directs.

Commenter ce choix

Simulation en prenant une marge comprise entre 1 et 3 par exemple

Chapitre 2

Commutation de paquets

2.1 Un commutateur de paquets

2.1.1 Énoncé

Nous cherchons à simuler un lien de sortie d'un commutateur de paquets.

L'arrivée des paquets est supposée suivre une loi exponentielle de paramètre λ. Nous positionnons une file en sortie du commutateur pour stocker les différents paquets. Les paquets ont une longueur exponentielle-ment distribuée de paramètre $\frac{1}{\nu} = 10000bits$. Le lien de sortie a un débit de 10 Mbit/s.

2.1.2 Calculer analytiquement le temps moyen de service $\frac{1}{\mu}$

$$\text{Temps moyen de service} = \frac{1}{\mu}$$

$$\iff \frac{1}{\nu} * \frac{1}{D} = 10^4 * \frac{1}{10^7} = \frac{1}{10^3} = 10^{-3} \text{ seconde}$$

2.1.3 Déterminer le nombre moyen de paquets dans la file et le temps moyen de réponse en fonction du taux d'arrivée pour différentes durées de simulation

$$\lambda = \rho * \mu$$

$$\text{Charge de trafic } \rho = \frac{\lambda}{\mu}$$

$$\text{Nombre moyen de client } \bar{N} = \frac{\rho}{(1-\rho)}$$

$$\text{Temps moyen de reponse } \bar{W} = \frac{1}{(\mu - \lambda)}$$

$$\bar{N} = \lambda * \bar{W}$$

ρ	0.1	0.5	0.9
λ	10^2	$5 * 10^2$	$9 * 10^2$
Nombre moyen de client	0.11111	1	9
Temps moyen de réponse	0.00111	0.002	0.01

2.1.4 Comparer le résultat de la mesure avec le résultat théorique

ICI COURBES ...

2.1.5 Reprendre les deux questions dans le cas où les paquets ont une longueur constante (10000 bits)

Calculer analytiquement le temps moyen de service $\frac{1}{\mu}$

On obtient la même chose que précédemment :

$$\text{Temps moyen de service} = \frac{1}{\mu}$$

$$\iff \frac{1}{\nu} * \frac{1}{D} = 10^4 * \frac{1}{10^7} = \frac{1}{10^3} = 10^{-3} \text{ seconde}$$

Déterminer le nombre moyen de paquets dans la file et le temps moyen de réponse en fonction du taux d'arrivée pour différentes durées de simulation

voir pour trouver des formules, la progression doit être linéaire en fonction de ρ.

Analyse et conclusion

blablabla a voir ...

Conclusion

Too much bullshit here :P

Annexe A

Annexes

A.1 Exemple(s)

emphatique **gras** `machine à écrire` *incliné* Petites majuscules

The foundations of the rigorous study of *analysis* were laid in the nineteenth century, notably by the mathematicians Cauchy and Weierstrass. Central to the study of this subject are the formal definitions of *limits* and *continuity*.

Let D be a subset of $\mathbf{R}$ and let $f\colon D \to \mathbf{R}$ be a real-valued function on D. The function f is said to be *continuous* on D if, for all $\epsilon > 0$ and for all $x \in D$, there exists some $\delta > 0$ (which may depend on x) such that if $y \in D$ satisfies

$$|y - x| < \delta$$

then

$$|f(y) - f(x)| < \epsilon.$$

One may readily verify that if f and g are continuous functions on D then the functions $f+g$, $f-g$ and $f.g$ are continuous. If in addition g is everywhere non-zero then f/g is continuous.

A.2 Répétition

Implémentation de l'envoi pour le Code de Répétition

```
#!/usr/bin/perl

use strict;
use Physique::LinkUDP;
use Repetition;

my $link = P_open(@ARGV);
my $nbrRec = 0; # nbr caracteres recus
my $nbrErr = 0; # nbr d'erreurs
my $nbrErrD = 0; # nbr d'erreurs detectees

do{
my $carRec = P_recoitCar($link);

# on test la paritee de cette reception (on test si la reception
    est valide) :
   $nbrErrD+=1 unless (Repetition::verification($carRec)); # la
       reception ne respecte pas la parite

# on test la validite de ce caractere recu :
   $nbrErr+=1 if ( Repetition::decodage($carRec) ne 'o'); # le
       caractere n'est pas celui attendu

   $nbrRec += 1;
}while ($nbrRec < 100000);


print "\n";
print " -> nombre de receptions : ".$nbrRec."\n";
print " -> nombre de receptions supposees bonnes : ".($nbrRec-
    $nbrErrD)."\n";
print " -> nombre de vrais bons caracteres : ".($nbrRec-$nbrErr)."\
    n";
print " -> nombre d'erreurs au total : ".$nbrErr."\n";
print " -> nombre d'erreurs detectees : ".$nbrErrD."\n";
print " -> nombre d'erreurs non detectees : ".($nbrErr-$nbrErrD)."\
    n";
# attention a la division par "0" ! xD
print (" -> fiabilitee de l'envoi/reception : ".(100-100*$nbrErr/
    $nbrRec)."%\n") if($nbrRec != 0);
```

```
34 print (" -> fiabilitee de detection d'erreur : ".(100*$nbrErrD/
       $nbrErr)."%\n") if($nbrErr != 0);
35 print "\n";
36
37 P_close($link);
```

Implémentation de la réception pour le Code de Répétition

```
#!/usr/bin/perl

use strict;
use Physique::LinkUDP;
use Physique::Noise; # pour generer des problemes et erreurs de
    transmission
use Repetition;

my $car = 'o';
my $link = P_open(@ARGV);
my $nbrEnv = 0;
my $temp = Repetition::codage($car); # on ne l'encode qu'une seule
    fois puisque ce caractere ne change pas

print "\n -> envoi du caractere : ".$car." \n";

while($nbrEnv<100000)
{
   P_envoiCar($link,$temp);
   $nbrEnv+=1;
}

print "\n -> nombre de caracteres envoyes : ".$nbrEnv."\n\n";

P_close($link);
```

Implémentation de la vérification pour le Code de Répétition

```
use strict;
use warnings;

package Repetition;

our $nbrRepetition = 2;

sub codage {
    @_ == 1 or die;
    my ($code) = @_;
    $code = $code x $nbrRepetition;
    return $code;
}

sub verification {
    @_ == 1 or die;
    my ($code) = @_;
    my $chaine1 = substr($code,0,length($code)/$nbrRepetition);
    my $chaine2 = substr($code,length($code)/$nbrRepetition);
    return $chaine1 eq $chaine2;
}

sub decodage {
    @_ == 1 or die;
    my ($code) = @_;
    return substr($code,length($code)/$nbrRepetition);
}

1;
```

A.3 VRC

Implémentation de l'envoi pour le VRC

```
1  #!/usr/bin/perl
2
3  use strict;
4  use Physique::LinkUDP;
5  use Parity;
6
7  sub verification {
8      my $car = ord($_[0]);
9
10 # retour du resultat du test de la parite de ce caractere :
11     return not Parity::parity($car);
12 }
13
14 sub decodage {
15     my $car = ord($_[0]);
16
17 # ou retourner le caractere vide si il y a erreur :
18     return '' unless ( verification(chr($car)) );
19
20 # retourner le decodage du caractere par decalage a droite si la
       paritee est respectee :
21     return chr( $car >> 1);
22 }
23
24
25
26
27 my $link = P_open(@ARGV);
28 my $nbrRec = 0; # nbr caracteres recus
29 my $nbrErr = 0; # nbr d'erreurs
30 my $nbrErrD = 0; # nbr d'erreurs detectees
31
32 do{
33 my $carRec = P_recoitCar($link);
34
35 # on test la paritee de cette reception (on test si la reception
       est valide) :
36    $nbrErrD+=1 unless (verification($carRec)); # la reception ne
          respecte pas la parite
37
```

```
38 # on test la validite de ce caractere recu :
39    $nbrErr+=1 if (decodage($carRec) ne 'o'); # le caractere n'est
          pas celui attendu
40
41    $nbrRec += 1;
42 }while ($nbrRec < 100000);
43
44
45 print "\n";
46 print " -> nombre de receptions : ".$nbrRec."\n";
47 print " -> nombre de receptions supposees bonnes : ".($nbrRec-
       $nbrErrD)."\n";
48 print " -> nombre de vrais bons caracteres : ".($nbrRec-$nbrErr)."\
       n";
49 print " -> nombre d'erreurs au total : ".$nbrErr."\n";
50 print " -> nombre d'erreurs detectees : ".$nbrErrD."\n";
51 print " -> nombre d'erreurs non detectees : ".($nbrErr-$nbrErrD)."\
       n";
52 # attention a la division par "0" ! xD
53 print (" -> fiabilitee de l'envoi/reception : ".(100-100*$nbrErr/
       $nbrRec)."%\n") if($nbrRec != 0);
54 print (" -> fiabilitee de detection d'erreur : ".(100*$nbrErrD/
       $nbrErr)."%\n") if($nbrErr != 0);
55 print "\n";
56
57 P_close($link);
```

Implémentation de la réception pour le VRC

```
#!/usr/bin/perl

use strict;
use Physique::LinkUDP;
use Physique::Noise; # pour generer des problemes et erreurs de
    transmission
use Parity;

sub codage {
   my ($car) = @_;

# ajout d'un zero a droite de la valeur binaire du caractere (
    decalage a gauche) :
   $car = ord($car) << 1;

# application de la paritee si ce caractere "impair", on inverse le
     dernier bit qui devient donc un "1" :
   $car = $car ^ 1 if Parity::parity($car);

# on retourne le caractere et non pas sa valeur binaire :
   return chr($car);
}



my $car = 'o';
my $link = P_open(@ARGV);
my $nbrEnv = 0;
my $temp = codage($car); # on ne l'encode qu'une seule fois puisque
     ce caractere ne change pas

print "\n -> envoi du caractere : ".$car." \n";

while($nbrEnv<100000)
{
   P_envoiCar($link,$temp);
   $nbrEnv+=1;
}

print "\n -> nombre de caracteres envoyes : ".$nbrEnv."\n\n";

P_close($link);
```

Implémentation de la vérification VRC

```
use strict;
use warnings;

package Parity;

sub parity {
    @_ == 1 or die;
    my ($code) = @_;
    my $parity = 0;
    while($code != 0) {
    $parity ^= 1 if ($code & 1);
    $code >>= 1;
    }
    return $parity;
}

1;
```