Tolérance aux pannes

Algorithme des généraux Byzantins

I. Introduction

Le problème des généraux Byzantins est une représentation abstraite d'une classe de programmes mettant en œuvre plusieurs intervenants comme des processeurs dans un ordinateur, des ordinateurs dans un réseau ou des robots dans une usine.

La tolérance aux pannes trouve ses origines dans les besoins militaires liés à la guerre froide : continuité des communications sur un réseau maillé (ARPA). L'explosion de l'informatique dans le monde industriel a soulevé d'autres questions comme la fiabilité des systèmes embarqués dans les avions et les fusées, la continuité de fonctionnement des outils de production ou la liaison des systèmes interconnectés.

Quant à la cryptographie, longtemps restée aux mains des militaires, elle pose le problème de la confidentialité des informations échangées sur un réseau comme Internet.

Je présenterai la problématique des généraux Byzantins (Cf. [GDP]), un algorithme de résolution du problème des généraux byzantins par « message oral », un algorithme par « message signé » et un algorithme de cryptographique (Système d'Elgamal).

Ce document est très largement inspiré de [GDP] et ne constitue qu'une présentation générale.

II. Un peu d'histoire

Cette partie est consacrée à positionner chronologiquement la problématique et ses solutions (Cf. [HTI])

A. Les réseaux et les télécommunications

1940 : première communication à distance entre machines à calculer
1957 : lancement du projet ARPA (Advanced Research Projects Agency) à des fins militaires
1960 : le premier réseau d' ordinateurs
1962 : premier réseau commercial
1965 : l' ARPA finance une étude sur les réseaux coopératifs d' ordinateurs à temps partagé
1967 : Symposium de l' ACM sules principes d' opération des réseaux (naissance officiel d'ARPANET)
1969 : ARPAnet (Advanced Research Projects Agency network), premier réseau informatique pour la recherche
1969 : premier noeud ARPANET
1970 : première utilisation du NCP (Network Control Protocol)
1973 : première connections internationales à l' ARPANET
1980 : INTERNET
1990 : naissance du World Wide Web (WWW)

B. La cryptographie

Début de l'écriture : invention de la cryptographie ?
Jusqu'en 1970 : quasiment utilisé par les militaires et les diplomates
1976 : Invention de la cryptographie à clé publique (W. Diffie et M.Hellmann)
1977 : Choix du standard DES (Data Encryption Standard)
1978 : Algorithme RSA (Rivest, Shamir, Adleman)
1985 : Algorithme Elgamal

III. Le problème des généraux byzantins

A. *Présentation générale*

Comme je l'ai introduit, le problème de composants défectueux dans un système informatique peut être exprimé de façon abstraite en terme de généraux de l'armée Byzantine qui campent autour d'une cité ennemie. A l'aide de messagers, ils doivent se mettre d'accord sur un plan de bataille commun, sinon la défaite est assurée.

Mais certains généraux, des traîtres, essayent de semer la confusion parmi les autres. Le problème est donc de trouver un algorithme pour s'assurer que les généraux loyaux arrivent tout de même à se mettre d'accord sur un plan de bataille (Cf. [GDP]). Ils doivent trouver un « consensus ».

B. *Analogie avec les systèmes informatiques*

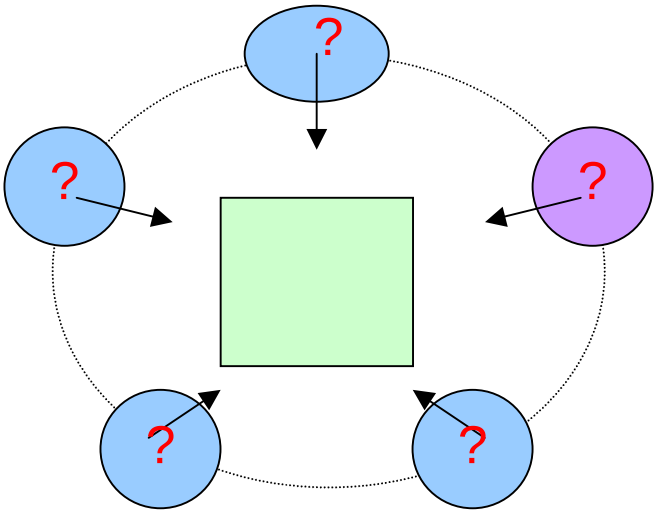
Dans cette description, un général est un composant informatique (un processeur ou un ordinateur), un messager est un canal de communication bidirectionnelle entre deux composants (Bus, câble ethernet, réseau téléphonique,…) et un traître est un composant défectueux.

Si un canal de communication entre deux composants est défectueux, nous pouvons considérer que le composant émetteur est un traître. De cette manière, nous pouvons simplifier le problème en raisonnant uniquement sur les composants.

Un plan d'action, dans le cadre d'une gestion de pannes, peut se représenter comme le résultat d'une fonction mathématique (***Ex*** : calcul de la trajectoire d'une fusée). L'évaluation de la fonction est réalisée par plusieurs calculateurs. Le plan d'action peut être le résultat à la majorité des calculateurs ou la moyenne des résultats obtenus. La victoire est le résultat attendu et la défaite est un mauvais résultat.

Remarque : Un composant envoie rarement de mauvaises informations. Elles sont souvent le fruit des perturbations pouvant exister sur une liaison entre composants (Perturbation sur une ligne téléphonique, rupture de communication, …). Il est, donc, fort probable que le composant se « crashe » ; dans ce cas, le composant n'envoie plus de messages.

C. *La problématique*



Imaginons que plusieurs divisions de l'armée Byzantine (en **bleu** et en **mauve**) campent autour de la cité ennemie (en **vert**), chacune d'entre elle étant dirigée par son propre général. La seule façon de communiquer dont ils disposent est l'utilisation de messagers (***trait en pointillé***). Après avoir observer l'ennemi, ils doivent se mettre d'accord sur un plan d'action commun. Le problème est que certains de ces généraux (en **mauve**) peuvent être des traîtres, qui tentent d'empêcher les généraux loyaux (en **bleu**) de se mettre d'accord.

Un algorithme des généraux Byzantins doit posséder les propriétés suivantes :

A. Tous les généraux loyaux se mettent d' accord sur le même plan d' action.
B. Un petit nombre de traître ne doit pas interférer dans le choix des généraux loyaux.

En effet, peu importe le choix des traîtres, les généraux loyaux doivent appliquer ce qui leur a été demandé. Ces derniers doivent se mettre d'accord sur un plan « raisonnable ».

Nous pouvons remarquer que la condition B est difficile à formaliser. Si le plan d'action est le résultat d'une évaluation booléenne (Attaque ou Retraite), comme une fonction de majorité, la condition B est formalisée avec un nombre de traîtres inférieur à la moitié de la totalité des généraux. Mais, si le plan d'action est le résultat d'une fonction mathématique, comme une moyenne, la condition B n'est pas évidente à formaliser.

Remarque : Il est bien évident que tous les généraux doivent posséder la même fonction d'évaluation (majorité ou moyenne), selon la condition A et cette fonction doit être « robuste » pour respecter la condition B (elle doit toujours donner le même résultat avec le même vecteur de valeurs).

D. Les conditions nécessaires et suffisantes

Chaque général observe l' ennemi et communique ces observations aux autres.

- Soit **n**, le nombre de généraux.
- Soit $\mathbf{v_i}$ l' information communiqué par le $i^{ème}$ général.
- Chaque général doit donc utiliser une méthode pour combiner les valeurs $v_1,..., v_n$ en un plan d' action unique. La condition A peut être obtenue si tous les généraux utilisent la même méthode pour combiner les informations, et la condition B peut être obtenue en utilisant une méthode "robuste".

Exemple :

Si la décision qui doit être prise est soit Attaque ou Retraite, alors v_i peut être la décision du général i et la décision finale peut être basée sur la majorité des décisions. Des traîtres peuvent modifier cette décision seulement si les généraux loyaux étaient divisés de manière égale quant à la décision à prendre, auquel cas aucune des décisions ne peut être considérée comme mauvaise.

Bien que cette approche puissent ne pas être la seule façon de satisfaire les conditions A et B, c' est la seule connue. Elle suppose qu' il existe une méthode qui permette aux généraux de communiquer leurs valeurs v_i aux autres. Une méthode évidente est, pour le $i^{ème}$ général, d 'envoyer v_i par messager aux autres généraux.

Malheureusement, cela ne fonctionne pas car pour que la condition A soit satisfaite, il faut que tous les généraux obtiennent le même ensemble de messages $v_1,..., v_n$, et un traître peut très bien envoyer des messages différents à chacun des autres généraux.

Pour que la condition A soit satisfaite, il faut que la condition suivante soit vraie :

1) Tous les généraux loyaux doivent obtenir les mêmes informations $v_1,..., v_n$.

Cette condition implique qu' un général loyal, ne va pas forcément utiliser la valeur v_i reçue de i, puisque si le $i^{ème}$ général est un traître, il a très bien pu envoyer des valeurs différentes à chacun. Cela signifie donc que, si l' on souhaite vérifier la condition 1 et qu' on ne fait pas attention, il est possible que les généraux utilisent une

valeur de v_i différente de celle envoyée par i – et ce même si le général i est loyal. Il ne faut pas permettre cela si nous souhaitons vérifier la condition B. Nous avons alors de plus la nécessité suivante :

2) Si le $i^{ème}$ général est loyal, alors la valeur qu' il a envoyé doit être utilisée par tous les généraux loyaux comme étant v_i.

Nous pouvons alors réécrire la condition 1 comme ceci (que le $i^{ème}$ général soit loyal ou pas) :

1') Deux généraux loyaux quelconques utilisent la même valeur pour v_i.

E. Simplification du problème

Soit :

> 1') Deux généraux loyaux quelconques utilisent la même valeur pour v_i
> 2) Si le $i^{ème}$ général est loyal, alors la valeur qu' il a envoyé doit être utilisée par tous les généraux loyaux comme étant v_i.

Les conditions 1' et 2 portent toutes deux sur une même valeur envoyée par le $i^{ème}$ général. On peut alors restreindre notre étude du problème à : **comment un seul des généraux envoie-t-il sa valeur aux autres** ?

On formulera cela en terme de commandant qui envoie un ordre à ces lieutenants.

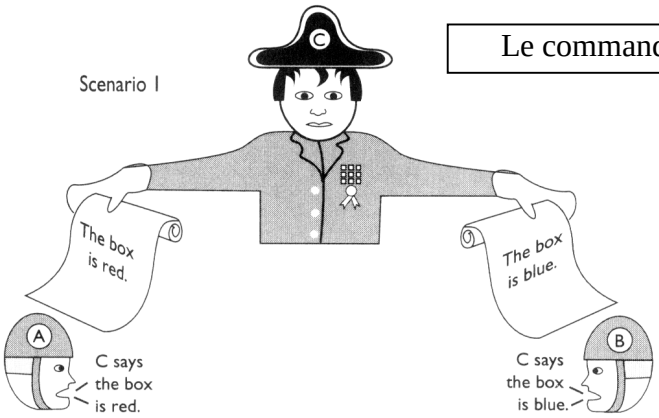
Selon la simplification, décrite ci-dessus, un commandant doit envoyer un ordre à ses n-1 lieutenants, de manière à respecter les conditions 1' et 2. Ces conditions sont connues sous le nom de consistance interactive (**Interactive Consistency** conditions).

Ces conditions sont définies comme suit :

> **IC1** : Tous les lieutenants loyaux obéissent au même ordre.
> **IC2** : Si le commandant est loyal, alors chaque lieutenant doit obéir à l'ordre qu'il a envoyé.

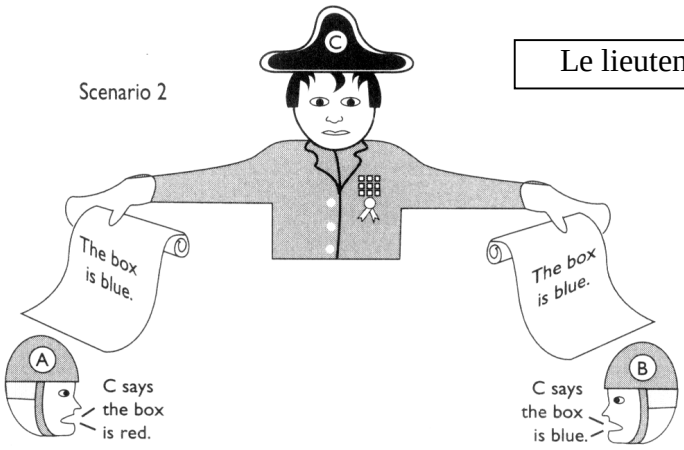
Remarque : si le commandant est loyal, alors IC2 implique IC1. Mais, comme cela a été abordé précédemment, le commandant peut être un traître.

F. Solutions impossibles



Le commandant est un traître

Dans ce scénario, le lieutenant A, s'il respecte IC1, doit dire « red ». Cependant, il reçoit « blue » d'un autre commandant. Alors que doit-il choisir ?



Le lieutenant A est un traître

Dans ce scénario, le lieutenant B, s'il respecte IC1, doit dire « blue ». Cependant, il reçoit « red » d'un autre commandant. Alors que doit-il choisir ?

Il n'existe pas de solutions pour 3 généraux en la présence d'un traître.

IV. La gestion des pannes

A. *Définitions*

Soit n, le nombre de processus impliqués dans la communication.
Soit m, le nombre de processus défaillants.
Soit v_{def}, une valeur par défaut définie au cas où un message serait absent.
Soit f, une fonction définissant la valeur à appliquer au processus.
Soit P_i, $i \in [1 ; n]$, tous les processus du système d'informations.

B. *Pease & Al (1980) : Unsigned Messages (messages non signés) ou Oral Message*

1. Pré requis

Cet algorithme récursif s'applique si $m < n/3$ (Cf. Solutions impossibles) ou $n > 3m$.

2. Hypothèses

S1 : *Chaque message envoyé est délivré correctement (pas de pertes de messages),*
Dans l'analogie avec les systèmes informatiques, j'ai indiqué que si un canal était défectueux, nous considérions que l'émetteur était un traître.

S2 : *Le destinataire d'un message sait qui lui a envoyé (réseau complètement connecté),*
Pour éviter qu'un composant interfère dans la communication entre deux composants, il est nécessaire que l'information ne passe pas par celui-là.

S3 : *L'absence d'un message peut être détecté (réseau synchrone),*
Ici, il s'agit d'éviter qu'un composant défectueux ne sème la confusion parmi les composants en état de marche en n'envoyant pas de messages. Dans ce cas, nous considérerons une valeur par défaut (comme substitut de réponse).

Soit P_g, le processus émettant l'ordre (Général).
Soit <v>, le message envoyé par un processus (Ordre du général).

3. Algorithme

a) OM(0) : Cas où il n'y a pas de traître

(1) : Le processus P_g envoie <v> à chacun de ses voisins

(2) : Chaque processus P_j utilise la valeur <v> reçue du processus P_g
ou v_{def} si il n' a rien reçu.

b) OM(m) : Cas où il y a m traîtres (m > 0)

(1) : Le processus P_g envoie <v> à chacun de ses voisins.

(2) : Pour chaque processus P_j ($j \neq i$),
Soit v_j = valeur reçue <v> du processus P_g ou v_{def} si aucune valeur n' a été reçue
Envoyer <v_j> à ses voisins (sauf P_g)
en utilisant OM(m-1), P_j se comporte alors comme P_g

(3) : Pour chaque i & chaque $j \neq i$,
Soit v_j = valeur que P_i a reçue de P_j à l' étape (2) ou v_{def} si il n' a rien reçu.
P_i utilise la valeur de $f(v_1,...,v_{n-1})$.

4. Exemple où un lieutenant est un traître

f : fonction majorité
P_1 est le général, P_2 est le traître

Etape 1 : P_1 envoie <v> à P_2, P_3 et P_4

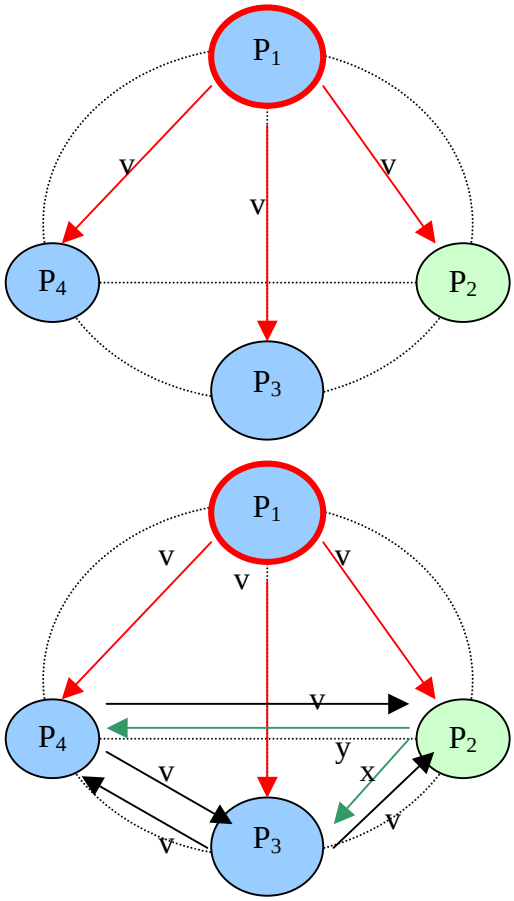
Etape 2 : P_2, P_3 et P_4 reçoivent <v>

Pour P_2, v_2 = v
Pour P_3, v_3 = v
Pour P_4, v_4 = v

P_2 envoie <x> à P_3
P_2 envoie <y> à P_4
P_3 envoie <v> à P_2 et à P_4
P_4 envoie <v> à P_2 et à P_3

Etape 3 : P_2 reçoit <v> de P_3 et P_4
P_3 reçoit <x> de P_2 et <v> de P_4
P_4 reçoit <y> de P_2 et <v> de P_3

Pour P_2, v_2 = v, v_3 = v, v_4 = v ➔ majorité(v,v,v) : v
Pour P_3, v_3 = v, v_2 = x, v_4 = v ➔ majorité (v,x,v) : v
Pour P_4, v_4 = v, v_2 = y, v_3 = v ➔ majorité (v,y,v) : v



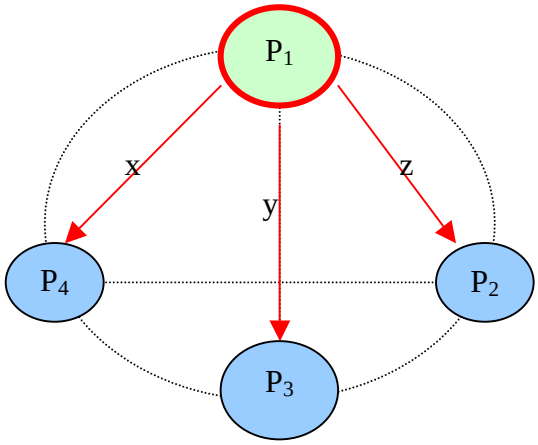
P_2, P_3 et P_4 aboutissent à la même décision (IC1). La valeur <v> choisie est bien la valeur envoyée initialement par P_1 (IC2).

5. Exemple où un général est un traître

f : fonction majorité

P_1 est le général et le traître

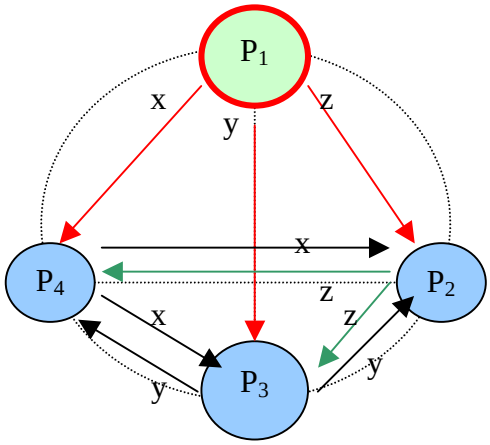
Etape 1 : P_1 envoie <z> à P_2, <y> à P_3 et <x> à P_4



Etape 2 : P_2 reçoit <z>, P_3 reçoit <y> et P_4 reçoit <x>

Pour P_2, $v_2 = z$
Pour P_3, $v_3 = y$
Pour P_4, $v_4 = x$

P_2 envoie <z> à P_3 et à P_4
P_3 envoie <y> à P_2 et à P_4
P_4 envoie <x> à P_2 et à P_3



Etape 3 : P_2 reçoit <x> de P_4 et <y> de P_3
P_3 reçoit <x> de P_4 et <z> de P_2
P_4 reçoit <z> de P_2 et <y> de P_3

Pour P_2, $v_2 = z$, $v_3 = y$, $v_4 = x$ ➔ majorité(x,y,z) : v
Pour P_3, $v_3 = y$, $v_2 = z$, $v_4 = x$ ➔ majorité (x,y,z) : v
Pour P_4, $v_4 = x$, $v_2 = z$, $v_3 = y$ ➔ majorité (x,y,z) : v

P_2, P_3 et P_4 aboutissent à la même décision (IC1). (IC2) est respecté car le général n'est pas loyal.

6. Lemmes et théorèmes

Lemme de Terminaison (Termination) :

Si l'algorithme commence au temps 0, tous les processus décident au temps m+1.

Preuve : L'algorithme est récursif et son argument m est décrémenté à chaque appel. Le temps 0 est l'étape où le processus P_g initialise l'algorithme en envoyant le message <v> à tous ses voisins. Le temps 1 est l'étape de la première réception des processus P_j ($j \neq g$).

Lemme de dépendance (Dependence) :

Pour tout m et k, l' algorithme OM(m) satisfait IC2 (Si P_g est fiable, chaque P_j voisin de P_g doit avoir la même valeur <v> que P_g) si $n > 2k+m$ avec au plus k traîtres.

Preuve : La preuve se fait par récurrence sur m.
Posons, la fonction majorité comme fonction d'évaluation (c'est plus facile à comprendre).

* Pour m = 0, OM(0) fonctionne uniquement si P_g est fiable et si le message est délivré correctement (S1). Dans ce cas, l'algorithme respecte IC2.

* Pour m > 0, supposons que OM(m-1) satisfait IC1 (Tous les P_j fiables ont la même valeur).
Prouvons que cela est aussi vrai pour m.

A l'étape (1), le processus P_i envoie une valeur <v> à ses voisins P_j.
A l'étape (2) chaque P_j applique OM(m-1).

Par hypothèse, nous avons : n>2k+1 ou n-1>2k+(m-1).

Par récurrence, chaque P_l obtient v_j = v de chaque P_j (P_l et P_j, processus fiables)

Comme il y a au plus k processus défectueux et que *n-1 > 2k+(m-1) >= 2k* c'est à dire que k<(n-1)/2, la majorité des n-1 processus sont fiables. Ainsi chaque processus fiable a v_g = v comme majorité des n-1 valeurs pour g, il obtient majorité(v_1,...,v_{n-1}) = v à l'étape (3), ce qui vérifie IC2.

Lemme d'arrangement (Agreement) :

Tous les processus fiables ont de la même valeur (IC1).

Preuve : Ceci a été démontré dans le lemme précédent.

Théorème :

Pour tout m, l'algorithme OM(m) satisfait les conditions IC1 et IC2 si nous avons n > 3m.

Preuve : La preuve se fait par récurrence sur m.
Posons, la fonction majorité comme fonction d'évaluation (c'est plus facile à comprendre).

* Pour m = 0, OM(0) satisfait IC1 et IC2. Tous les processus sont fiables et ont tous la même valeur.

* Pour m > 0, nous supposons alors que le théorème est vrai pour OM(m-1)

Considérons tout d'abord le cas où le P_g est fiable.

En prenant k = m dans le Lemme de dépendance, nous pouvons voir que OM(m) satisfait IC2.
IC1 est impliqué par IC2 puisque P_g est fiable (Cf. Les conditions nécessaires et suffisantes).
Donc nous n'avons en fait qu'à considérer le cas où P_g est défaillant.

Maintenant, considérons le cas où il y a au plus m processus défaillants et P_g est l'un d'entre eux.

Donc, il y au plus m-1 processus défaillants (hormis P_g).
Puisqu'il y a plus de 3m processus, il y a plus de 3m-1 processus (hormis P_g), et 3m-1>3(m-1).
Nous pouvons alors appliquer l'hypothèse de récurrence pour déduire que OM(m-1) satisfait IC1 et IC2. Ainsi pour chaque j, chaque groupe de deux processus obtient la même valeur pour v_j à l'étape (3).
Cela se déduit par IC2 si l'un des deux processus est le processus P_j et par IC1 dans les autres cas.

Ainsi, chaque groupe de deux processus obtient le même vecteur de valeurs v_1,..., v_{n-1} et obtient donc la même valeur majorité(v_1,..., v_{n-1}) à l'étape (3).
Ce qui prouve IC1.

7. Complexités

En temps, la complexité est optimale : Θ(m+1)

m	Nb messages pour OM(m)
0	(n-1)
1	(n-1)(n-2)
…	…
m	$\prod_{i=1}^{m+1}(n-i)$

En messages, la complexité est exponentielle, elle est de l'ordre de $O(n^{m+1})$.
En taille de message, la complexité est la même que la complexité en message.

C. *Lamport, Shostak & Pease (1980) : Signed Messages (messages signés)*

Dans l'algorithme précédent, le problème des généraux Byzantins était résolu si nous avions moins d'un tiers de processus défaillants dans le système. Ici, nous allons le résoudre pour un nombre quelconque. Nous allons faire disparaître cette contrainte en intégrant des messages signés au protocole.

1. Hypothéses

Pour cela, nous allons ajouter 2 hypothèses supplémentaires à celles décrites ci-dessus.

S4 : *La signature d'un processus fiable ne peut pas être imitée et toute modification du contenu d'un message peut être détectée (un message peut être supprimer, mais pas modifié).*

S5 : *N'importe quel processus peut vérifier l'authenticité d'un message (personne ne peut tromper le donneur d'ordre).*

Ces hypothèses peuvent s'appliquer facilement à des systèmes informatiques utilisant la signature numérique. Elles permettent, aussi, de faire disparaître la contrainte sur le nombre de processus défaillants. Cependant, il est évident que ce problème n'a pas d'intérêts si il y a au plus un seul processus fiable (Il faut au moins un général et un lieutenant fiable).

Ces deux hypothèses impliquent, contrairement aux messages oraux, que le seul processus donneur d'ordre, P_g, peut être un traître.
En effet, si P_g est fiable, il a envoyé le même message signé à tous ses voisins. Si un voisin de P_g modifie le message (ce voisin est un traître) et l'envoie aux autres processus, il viole l'hypothèse S4.
Si P_g envoie des messages contradictoires à ses voisins (P_g est un traître), il est le seul à avoir pu les signer et si deux processus reçoivent des ordres contradictoires, ils sont sûrs que P_g est le traître. Ils peuvent donc décider soit d'exécuter une valeur par défaut, soit de ne rien faire.

2. Algorithme

a) Principe

Cet algorithme nécessite qu'un processus qui envoie un message y appose sa signature. En d'autres termes, cela signifie qu'un processus qui reçoit un message doit le copier, le signer et puis l'envoyer à ses voisins.

Ici, nous avons, également, besoin d'une fonction d'évaluation. Mais, le domaine change. Ce n'est pas une fonction qui calcule une valeur sur un ensemble. Mais plutôt, une fonction qui permet de faire un choix sur toutes les valeurs reçues.

Soit P_g, le processus donneur d'ordre
Soit f, une fonction de choix telle que : $f(\varnothing) = v_{def.}$, $f(\{v\}) = v$, $f(\{v_1,\ldots,v_n\}) = v_i$ ($i \in [1 ;n]$).
Soit <v :j :i>, un message contenant la valeur signée par P_j, reçue de P_i.
Soit V_i, l'ensemble des ordres signés corrects reçus du processus P_i ($|V| \leq 1$, si P_i est fiable)

b) SM(m)

Initialement $V_i = \{\}$

(1) Le commandant envoie sa valeur signée <v :i> à chacun de ses voisins.

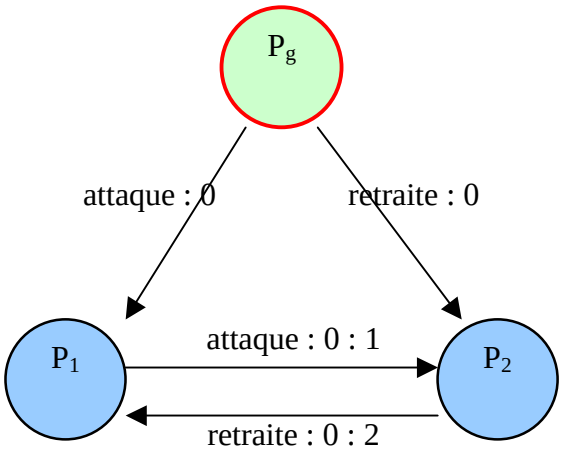
(2) Pour chaque processus P_i,

P_i reçoit un message <v :0> (c'est le premier message qu'il reçoit).
$V_i = \{v\}$
Pour chaque processus P_j, voisins de P_i
Envoyer <v :0 :i> à P_j
Fin pour

P_i reçoit un message <v :0 :j_1 :…j $_k$> et $v \notin V_i$
$V_i = V_i \cup \{ v \}$
Si k < m
Pour chaque processus P_j, voisins de P_i, tel que $j \notin j_1,\ldots j_k$
Envoyer <v :0 :j_1 :…. :j_k :i> à P_j
Fin pour
Fin si

(3) Pour chaque processus P_i
Quand il n'y a plus de messages, le processus a comme valeur Choix(V_i)

3. Exemple



Etape 1 : P_g envoie <attaque : 0> à P_1 et <retraite : 0> à P_2

Etape 2 :

P_1 reçoit <attaque : 0>, V_1 = {attaque}
et envoie <attaque : 0 : 1> à P_2
P_2 reçoit <retraite : 0>, V_2 = {retraite}
et envoie <retraite : 0 : 2> à P_1

Etape 3 :

P_1 reçoit <retraite : 0 : 2>, V_1 = {attaque, retraite }
P_2 reçoit <attaque : 0 :1>,V_2 = {retraite, attaque }

P_1 et P_2 savent que P_g est un traître (S4 : La signature n'est pas falsifiable). Ils peuvent alors décider soit d'exécuter une valeur par défaut, soit de ne rien faire.

4. Théorème

Pour tout m, l'algorithme SM(m) résout le problème des généraux Byzantins si il y a au plus m traîtres ($m \leq n-1$).

Preuve :

Prouvons tout d' abord IC2.

Si P_g est fiable, alors il envoie sa valeur signée <v:0> à chacun de ses voisins P_i à l' étape (1).
Chaque P_i va alors recevoir l' ordre v à l' étape (2).
De plus, puisqu' aucun processus défaillant ne peut imiter un message de la forme <v' :0> (S4), un processus fiable ne peut pas recevoir d' autre valeur à l' étape (2).
Ainsi, pour chaque P_i fiable, l' ensemble V_i obtenu à l' étape (2) consiste en un ensemble contenant une unique valeur v. A l' étape (3), le processus P_i choisira l'unique valeur de V.
Cela prouve IC2 et cela implique IC1 (de part la propriété)

Considérons le cas où P_g est défaillant.

Deux processus fiables P_i et P_j ont la même valeur à l'étape (3) si les ensembles V_i et V_j obtenus à l'étape (2) sont les mêmes.

Donc, pour prouver IC1, il suffit de prouver que, si P_i place une valeur v dans V_i à l'étape (2), alors P_j doit placer la même valeur v dans V_j à l'étape (2).

Pour cela, il nous faut montrer que P_j reçoit un message correctement signé contenant cette valeur. Si P_i reçoit la valeur v à l'étape (2), alors il l'envoie à P_j à l'étape (2).

Donc P_j la reçoit (du fait de S1).

Si P_i ajoute la valeur v à V_i dans l'étape (2), alors il doit recevoir un message $<v:0:j_1:...:j_k>$. Si j est l'un des j_r, alors, du fait de S4, il a déjà dû recevoir la valeur v. Si ce n'est pas le cas, il faut considérer 2 cas :

1. k<m
   Dans ce cas, i envoie le message $<v:0:j_1:...:j_k:i>$ à P_j ; donc P_j doit recevoir la valeur v.
2. k=m
   Puisque P_g est défaillant, au plus m-1 des voisins de P_g sont des traîtres. Ainsi, au moins l'un des processus $j_1,...,j_m$ est fiable. Ce processus fiable doit avoir envoyer à P_j la valeur v quand il l'a reçue pour la première fois, donc P_j doit avoir reçu cette valeur.

Cela complète la preuve.

5. Complexités

Les complexités en temps et en message restent les mêmes que l'algorithme des messages oraux.

La complexité en taille de messages peut être différente car cela dépend de l'algorithme de chiffrement utilisé.

V. La cryptographie

Cette partie complète la partie précédente. La cryptographie garantit dans un système informatique, la confidentialité des informations. Dans le cas de l'algorithme des messages signés, pour que les hypothèses S4 et S5 soient valides, il faut s'assurer qu'un tiers ne puisse pas modifier l'information.

Les algorithmes qui ont été abordés dans ce document ont été élaborés pour des réseaux complets (tous les composants sont reliés à tous les composants), or c'est rarement le cas. Il existe donc des intermédiaires entre deux composants. La cryptographie doit permettre de s'assurer que l'information émise et bien l'information reçue.

Le chiffrement d'Elgamal (Cf. [PKC]) est un de ces algorithmes.

A. Le chiffrement d'Elgamal (1985) : Algorithme à clés publiques

Il s'agit d'un système à clé publique dont la sécurité repose, comme le protocole de Diffie et Hellman, sur la difficulté de calculer le logarithme discret. Le destinataire, Bob, possède deux clés :

* une clé secrète : un entier ***s***.
* une clé publique, qui consiste en un entier ***p***, un entier ***a*** premier avec ***p***, et l'entier $P=a^s \bmod p$.

Si Alice veut envoyer le message M à Bob, M étant un entier compris entre 0 et p-1, elle procède de la façon suivante : elle tire au hasard un nombre k, et calcule :

$C_1=a^k \bmod p$, et $C_2=MP^k \bmod p$.

Le message chiffré est le couple (C1,C2), qu'elle transmet à Bob. A la réception, celui-ci calcule :

$R_1=C_1^s \bmod p=a^{sk} \bmod p=P^k \bmod p$.

Il a retrouvé P^k, et il divise C_2 par cette quantité pour retrouver M. Un attaquant éventuel, pour retrouver M, doit pouvoir calculer P^k, connaissant P, a et a^k. Il doit donc découvrir k, et est confronté au problème du logarithme discret.

Le défaut du système d'ElGamal est que le message chiffré est deux fois plus long que le message original. En revanche, le fait d'utiliser un paramètre aléatoire k est un plus en termes de sécurité : le même message M chiffré à 2 moments différents donnera deux messages codés distincts!

B. Exemple

Supposons p = 2579, a = 2, s = 765,

$P = 2^{765} \bmod 2579 = 949$

Supposons que l'expéditeur souhaite transmettre le message M = 1299.

Pour commencer, il choisit au hasard k, disons k = 853 et calcule

$C_1 = 2^{853} \bmod 2579 = 435$

$C_2 = 1299*949^{853} \bmod 2579 = 2396$

Lorsque le destinataire reçoit le texte chiffre C = (435,2396), il calcule :

$M = 2396*(435^{765})^{-1} \bmod 2579 = 1299$

VI. Conclusion

Ce document a constitué une première approche du problème des généraux Byzantins. Les algorithmes vus, ici, sont applicables mais sur des systèmes informatiques adaptés, répondant aux critères définis (réseau complet et synchrone).

Seulement, il faut bien noter que la résolution de ces problèmes n'est pas simple. Pour pouvoir fiabiliser des systèmes, potentiellement, défaillants, il est nécessaire de provoquer des informations redondantes et cela implique un très grand nombre de messages à échanger. Il s'avère que si les systèmes nécessitent une confidentialité, les tailles des messages en sont augmentées.

Aujourd'hui, il existe des algorithmes beaucoup plus efficaces que ceux abordés ici car c'est un domaine qui est au centre d'un enjeu économico-militaro-industriel. Les grandes entreprises possèdent des ordinateurs reliés en réseau, l'objectif est de mieux utilisé cette puissance de calcul potentielle (tous les ordinateurs ne sont pas utilisés à 100%). Sur Internet, la cryptographie est la clé de l'avenir afin de sécuriser au mieux les transactions financières et les échanges de documents confidentiels.

VII. Annexe A : Les personnalités marquantes

Leslie Lamport : http://research.microsoft.com/users/lamport/pubs/pubs.html

Robert Shostak : http://www.vocera.com/people/people.shtm

Marshall Pease : Pas de sites trouvés

Danny Dolev : http://www.cse.huji.ac.il/~dolev/

Raymond Strong : http://theory.lcs.mit.edu/~dmjones/STOC/Authors/stronghraymond.html
http://theory.lcs.mit.edu/~iandc/Authors/stronghraymond.html

Al : Pas de sites trouvés

Tahar Elgamal : http://www.ciao.gov/industry/08-22-00/Elgamal.htm

Dr. Whitfield Diffie : http://research.sun.com/people/diffie/

Martin Hellmann : Pas de sites trouvés

VIII. Annexe B : Références

[HTI] Histoire des technologies informatiques
http://www.scedu.umontreal.ca/sites/histoiredestec/histoire/tdmhiste.htm

[IDA] "**Introduction to distributed algorithms**"
Gérard Tel (2001), 2° édition de Cambridge (Chap. 15 ,p 469-504)

[GDP] Gestion de Pannes : l' algorithme des Généraux Byzantins
(Yann CEZARD, DESS TNI - Université de Montpellier II, décembre 2001)
http://www.lirmm.fr/~ajm/Cours/01-02/DESS_TNI/TER21/

[PBF] Practical Byzantine Fault Tolerance
Miguel Castro and Barbara Liskov
[MIT] MIT Laboratory for Computer Science
http://www.pmg.lcs.mit.edu/~castro/osdi99_html/osdi99.html

[PKC] "**A public key cryptosystem and a signature scheme based on discrete logarithms**"
in IEEE, Transaction on Information Theory, Taher Elgamal (1985)

[IAC] Introduction à la cryptographie
http://www.labouret.net/crypto/

[CRY] La cryptographie
http://www.uqtr.ca/~bellefeu/cours/sif1035/notes/telematique/securite/crypto/index.htm

[UTI] Exemple d'utilisation de ces algorithmes
http://www.cs.huji.ac.il/~dolev/pubs/ftcs-24.pdf