

## Protocoles de groupes et diffusion

---

**Sacha Krakowiak**  
Université Joseph Fourier  
Projet Sardes (INRIA et IMAG-LSR)  
<http://sardes.inrialpes.fr/~krakowia>

## Plan

- Protocoles de groupes**
  - motivations, définitions
  - spécifications
- Diffusion fiable**
  - diffusion asynchrone
  - diffusion temporisée
- Diffusion totalement ordonnée (atomique)**
  - diffusion totalement ordonnée et consensus
- Algorithmes d'appartenance**
  - vues synchrones
- Diffusion à grande échelle**
  - algorithmes épidémiques
- Applications de la diffusion**
  - cohérence, tolérance aux fautes

## Protocoles de groupes

---

Un **groupe** de processus est un ensemble spécifié de processus, pour lequel on définit des fonctions liées à

- l'**appartenance** (*group membership*) : changement de composition du groupe, connaissance à tout instant de la composition courante du groupe
- la **diffusion** (*broadcast* ou *multicast*) : communication d'information à un ensemble de processus (avec des propriétés spécifiées)

La composition d'un groupe peut changer soit par entrée ou sortie volontaire soit par suite de défaillances ou réinsertions ; certaines spécifications restreignent les changements possibles

### Motivations

- ensemble de processus se comportant comme un processus unique, avec tolérance aux fautes : groupe de serveurs, gestion de données dupliquées, systèmes sur grappes de machines, etc.
- travail coopératif, partage d'information

## Protocoles de groupe : difficultés

---

Les protocoles de groupe posent des problèmes difficiles

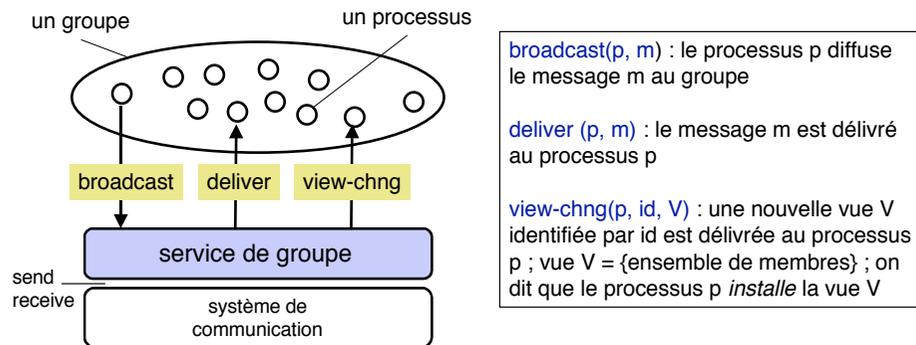
**Difficultés de spécification** : la spécification **rigoureuse** d'un protocole de groupe est une tâche délicate. La plupart des spécifications publiées sont incomplètes, incorrectes ou ambiguës. Voir Chockler et al, 2001

**Difficultés algorithmiques** : la réalisation des protocoles de groupe est difficile en présence de pannes. Certains problèmes sont insolubles en asynchrone (appartenance, diffusion atomique, vues synchrones). La réalisation de nombreux protocoles de groupe implique l'application répétée du consensus

En outre, on constate une grande **instabilité** de la difficulté algorithmique par rapport aux spécifications. Une modification apparemment mineure d'une spécification peut entraîner une variation importante de la difficulté. Les protocoles de groupe restent un domaine mal compris

G. V. Chockler, I. Keidar, R. Vitenberg. Group Communication Specifications : A Comprehensive Study, *ACM Computing Surveys*, Vol. 23, no. 4, Dec. 2001, pp. 427-469

## Protocoles de groupe : interfaces



**broadcast**( $p, m$ ) : le processus  $p$  diffuse le message  $m$  au groupe

**deliver**( $p, m$ ) : le message  $m$  est délivré au processus  $p$

**view-chng**( $p, id, V$ ) : une nouvelle vue  $V$  identifiée par  $id$  est délivrée au processus  $p$  ; vue  $V = \{\text{ensemble de membres}\}$  ; on dit que le processus  $p$  *installe* la vue  $V$

Le service de groupe réalise les protocoles de groupe au-dessus d'un système de communication. Les primitives indiquées constituent le minimum. Certains services peuvent fournir des primitives supplémentaires

Les primitives indiquées sont génériques. Des spécifications précises sont données plus loin

## Diffusion : définitions (1)

La **diffusion** est un mode de communication dans lequel un processus émetteur envoie un message à un **ensemble** de processus destinataires.

Dans la **diffusion générale** (*broadcast*), les destinataires sont tous les processus d'un seul ensemble défini implicitement (mais qui peut ou non être rigoureusement identifié, cf plus loin). L'émetteur est également destinataire. Exemples : les membres d'un groupe unique, vus de l'intérieur du groupe ; "tous" les processus du système

Dans la **diffusion de groupe** (*multicast*), les destinataires sont les membres d'un ou plusieurs groupe(s) spécifiés, les groupes pouvant ne pas être disjoints. L'émetteur peut ne pas appartenir au(x) groupe(s) destinataires(s)

Les primitives sont notées **broadcast**( $p, m$ ) et **multicast**( $p, m, \{g\}$ )

$p$  : émetteur  
 $m$  : message  
 $\{g\}$  : ensemble des groupes destinataires

## Diffusion : définitions (2)

La diffusion est réalisée au-dessus d'un système de communication sous-jacent, qui peut être point à point, ou déjà réaliser une forme de diffusion.

La diffusion est un outil important car elle est liée aux notions de **prise de décision concertée** (donc de consensus) et de **cohérence des données**

Il existe de nombreuses formes de diffusion, selon les propriétés requises. Au minimum, on demande la propriété de **fiabilité** (tout ou rien), déjà vue, qui garantit une forme de connaissance partagée (propriété d'accord)

V. Hadzilacos, S. Toueg, Fault-Tolerant Broadcast and Related Problems, in S. Mullender (ed.), *Distributed Systems* (2nd edition), Addison-Wesley, 1993

## Diffusion : propriétés (1)

Les propriétés sont présentées ici informellement, définition rigoureuse plus loin

**Propriétés indépendantes de l'ordre d'émission** (concernent uniquement les récepteurs).

**Diffusion fiable** : un message est délivré à tous ses destinataires ou à aucun

**Diffusion totalement ordonnée (ou atomique)** : la diffusion est fiable et les messages sont délivrés dans le même ordre à tous leurs destinataires

**Propriétés liées à l'ordre d'émission**

**Diffusion FIFO** : deux messages issus du même émetteur sont délivrés à tout récepteur dans leur ordre d'émission

**Diffusion causale** : pour tout récepteur, l'ordre de réception de deux messages respecte leur ordre causal d'émission (implique FIFO)

Les deux classes de propriétés sont indépendantes : 6 combinaisons possibles : fiable sans ordre, fiable FIFO, ... , atomique causale

## Diffusion : propriétés (2)

### Propriétés liées au temps

Certaines spécifications imposent un **délai maximal** au-delà duquel aucun message n'est plus délivré. Ce type de contraintes ne peut être satisfait que si le système de communication sous-jacent est **synchrone**

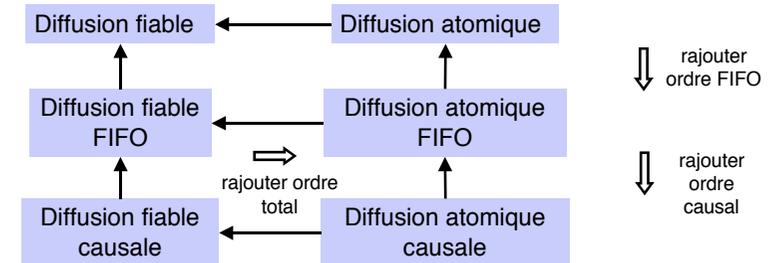
### Propriétés d'uniformité

Une propriété (accord, intégrité, ordre) est dite **uniforme** si elle s'applique à **tous** les processus (**corrects ou fautifs**) et pas seulement aux processus corrects.

L'uniformité est nécessaire lorsqu'un processus fautif peut (avant sa défaillance) exécuter des actions irréversibles comme conséquence de la délivrance d'un message

Ces deux classes de propriétés sont indépendantes des précédentes, et indépendantes entre elles

## Diffusion : propriétés (3)



La flèche simple indique une implication (atomique implique fiable, etc)  
On peut en outre ajouter les propriétés de temporisation et/ou d'uniformité

Nous avons vu la réalisation de la diffusion fiable en asynchrone avec pannes franches

Nous avons vu des techniques permettant de rendre causale une diffusion non causale (horloges vectorielles). Pour FIFO, il suffit de compteurs scalaires

Reste à voir : uniformité, temporisation, diffusion atomique

## Diffusion : spécifications (1)

N.B. Dans les définitions, le terme "processus" signifie "processus membre de l'ensemble de destinataires considéré"

### Diffusion fiable (rappel) [entre crochets, version uniforme]

- **Accord** : si un processus correct [ou fautif] délivre un message  $m$ , tous les processus corrects délivrent  $m$  (au bout d'un temps fini)
- **Validité** : si un processus correct diffuse un message  $m$ , tous les processus corrects délivrent le message  $m$  (au bout d'un temps fini)
- **Intégrité** : Quel que soit le message  $m$ , il est délivré au plus une fois à tout processus correct [ou fautif], et seulement s'il a été diffusé par un processus

**Ordre FIFO** : si un processus diffuse un message  $m$  avant un message  $m'$ , aucun processus correct ne délivre  $m'$  s'il n'a pas au préalable délivré  $m$

**Ordre causal** : si la diffusion d'un message  $m$  précède causalement la diffusion d'un message  $m'$ , aucun processus correct ne délivre  $m'$  s'il n'a pas au préalable délivré  $m$

## Diffusion : spécifications (2)

### Diffusion temporisée

Deux formes de temporisation [entre crochets, version uniforme]

**Temps réel** : il existe une constante  $\Delta$  telle que si la diffusion de  $m$  a été lancée au temps réel  $t$  (vu par un observateur extérieur), aucun processus correct [ou fautif] ne délivre  $m$  après le temps réel  $t + \Delta$

**Temps local** : chaque message est estampillé par l'heure locale d'émission  $te(m)$  (lue sur une horloge physique locale); il existe une constante  $\Delta$  telle qu'aucun processus  $p$  correct [ou fautif] ne délivre  $m$  après le temps réel  $te(m) + \Delta$  (lu sur l'horloge locale de  $p$ )

Avec des horloges synchronisées sur le temps universel (UTC), avec dérive limitée et correction, la version temps local est une approximation de la version temps réel

## Diffusion fiable uniforme

**Rappel du protocole de diffusion fiable** : (peut être broadcast ou multicast)

Tout processus  $p$  exécute le programme suivant :

pour exécuter **broadcast**( $p, m$ ) :

- estampiller  $m$  avec
- sender( $m$ ) (processus émetteur) et seq( $m$ )
- send**( $m$ ) à tous les voisins de  $p$ , et à  $p$

**deliver**( $p, m$ ) se produit comme suit :

- sur exécution de **receive**( $m$ )
- par processus  $p$
- if  $p$  n'a pas précédemment exécuté **deliver**( $m$ ) then
- if sender( $m$ )  $\neq p$  then **send**( $m$ ) à tous les voisins de  $p$
- deliver**( $m$ )

Le protocole ci-dessus réalise aussi la diffusion fiable **uniforme**. Il tolère les pannes franches et les pertes de messages en réception, tant que la communication reste possible entre deux processus corrects

La propriété qui assure l'uniformité est que tout processus qui **délivre** un message l'a **au préalable** envoyé à ses voisins. Si ce n'était pas le cas, un processus pourrait défaillir après avoir délivré un message, sans que celui-ci soit délivré à un autre processus

## Diffusion fiable temporisée

Le protocole précédent réalise aussi la diffusion fiable **temporisée**, avec une borne  $\Delta = (f + d) \delta$  en temps réel, sous les conditions suivantes :

- le système de communication est **synchrone**, et le temps de transmission est borné par  $\delta$
- il y a au plus  $f$  processus défaillants (pannes franches)
- un processus correct **[ou fautif]** quelconque est relié à tout processus correct par un chemin de longueur (nombre de liens) au plus égale à  $d$ , comprenant uniquement des processus et liens corrects
- La durée de l'exécution locale de l'algorithme est négligeable devant  $\delta$

Rappel : nous avons utilisé cette forme de diffusion pour la réalisation du protocole de **validation atomique non bloquant**

Avec la condition **[ou fautif]**, ce protocole peut être rendu **uniforme** (aucun processus correct **ou fautif** ne délivre après  $t+\Delta$  un message émis au temps  $t$ )

## Protocoles de groupe : appartenance

### Le problème de l'appartenance (*membership*)

**Objectif.** Fournir à tout instant à chaque membre du groupe une **vue** de la composition courante du groupe (primitive **view-chng**)

**Spécifications.** On distingue deux classes de spécifications selon que l'on considère ou non la possibilité de partitionnement du groupe

#### Composant (ou partition) primaire (*primary component*)

La suite des vues fournies aux membres du groupe est totalement ordonnée. On ne considère qu'une vue à la fois. Cela correspond à un groupe sans partition, ou un groupe avec partition dans lequel on ne considère qu'un des sous-groupes

#### Partitions multiples (*partitionable*)

La suite des vues fournies aux membres du groupe est partiellement ordonnée. On peut considérer plusieurs vues disjointes à la fois. Cela correspond à un groupe avec partition dans lequel on peut considérer plusieurs sous-groupes

Nous ne considérons que les algorithmes à partition primaire  
L'appartenance est incluse dans les protocoles à vues synchrones (voir plus loin), mais peut aussi être traitée à part

## Protocoles d'appartenance : difficultés

La spécification rigoureuse du problème de l'appartenance (partition primaire) est une tâche délicate.

Un grand nombre des spécifications qui ont été publiées sont imprécises, incomplètes, ou peuvent être satisfaites par des protocoles sans intérêt.

**Résultat** (\*) : Un protocole d'appartenance minimal (spécifications très peu contraignantes) **n'est pas réalisable** dans un système asynchrone avec une seule panne franche (résultat analogue à FLP).

Une des sources du problème est la non-exactitude de la détection de pannes. On pourrait penser qu'un protocole qui tue tout processus soupçonné échappe à l'impossibilité ci-dessus. Contrairement à l'intuition, ce n'est pas le cas.

L'appartenance (asynchrone, pannes franches) peut être résolue en utilisant le consensus

(\*) T. D. Chandra, V. Hadzilacos, S. Toueg, B. Charron-Bost. On the Impossibility of Group Membership, *Proc. ACM Symp. on Principles of Distributed Computing (PODC)*, May 1996

## Protocoles d'appartenance : spécifications

Hypothèse : communication asynchrone fiable, pannes franches, partition primaire (un groupe)

L'ensemble de processus  $\{p_1, p_2, \dots\}$  n'est pas limité. Un processus peut rejoindre (*join*) ou quitter (*leave*) le groupe. Il peut aussi avoir une défaillance (panne franche). Quand un processus quitte le groupe ou défaille, il ne revient pas sous son identité initiale (donc toute sortie est définitive)

Une vue  $v$  est caractérisée par son numéro  $v.id$  et la liste de ses membres  $v.members$  (liste des numéros des processus de la vue).

Le service de groupe délivre les vues successives aux processus. Quand une vue  $v$  est délivrée au processus  $p$ , on dit que  $p$  *installe* la vue

La spécification comprend des propriétés de sûreté (*safety*) et des propriétés de vivacité (*liveness*)

## Protocoles d'appartenance : sûreté (1)

**Validité.** Si un processus  $p_i$  installe une vue  $v$ , alors  $i \in v.members$  ; un processus installe une vue donnée une fois au plus.

Donc une vue ne peut être installée que par ses membres (propriété d'auto-inclusion)

**Ordre total.** L'ensemble des vues installées par les processus est totalement ordonné

On note  $V$  la suite ordonnée des vues, et  $succ(v)$  la vue qui suit  $v$ . Si  $v$  est la  $k$ -ième vue,  $v.id = k$ .

**Vue initiale.** Il existe une vue initiale, dont les membres sont prédéfinis.

Ces participants initiaux n'ont pas appelé *join*. L'état initial du groupe est l'état de l'ensemble des membres initiaux

## Protocoles d'appartenance : sûreté (2)

**Accord.**  $\forall p_i$ , soit  $V_i$  la séquence des vues installées par  $p_i$ . Alors  $V_i$  est une sous-séquence d'éléments contigus de  $V$ .

Cette propriété définit la cohérence globale mais autorise les défaillances

**Justification.** Soit  $v$  une vue telle que  $succ(v)$  existe. Alors

$v.members \neq succ(v).members$

$\forall j \in (succ(v).members \setminus v.members) : p_j$  a appelé *join*

$\forall j \in (v.members \setminus succ(v).members) : p_j$  a appelé *leave* ou est défaillant

Un changement de vue doit être justifié par une entrée ou une sortie. Une entrée ou une sortie ne peut résulter que des événements qui la justifient.

**Transfert d'état.**  $\forall v, v'$  tels que  $v' = succ(v)$ ,  $\exists i \in v.members \cap v'.members$ , tel que  $p_i$  installe  $v'$

L'état du système (matérialisé par la vue) peut être transféré d'une vue à la suivante, sauf si tous les processus sont en panne

## Protocoles d'appartenance : vivacité

**Terminaison.**

Si  $p_i$  appelle *join*, alors ou bien  $\exists v$  tels que  $p_i$  installe  $v$ , ou bien  $p_i$  défaille.

Si  $p_i$  appelle *leave* ou défaille dans la vue  $v$ , alors :  
 $\exists v'$  telle que  $v' \in succ^+(v)$  et  $i \notin v'.members$

[on note  $succ^+(v)$  la séquence des successeurs de  $v$ ]

Cette propriété indique que tout événement concernant l'appartenance d'un processus au groupe (départ, arrivée ou panne) finit par être répercuté dans une vue ultérieure (sans spécifier dans laquelle)

## Diffusion à vues synchrones : spécifications (1)

La diffusion à vues synchrones a été citée à propos de la disponibilité des serveurs. Les spécifications de la diffusion à vues synchrones sont une extension de celles de l'appartenance

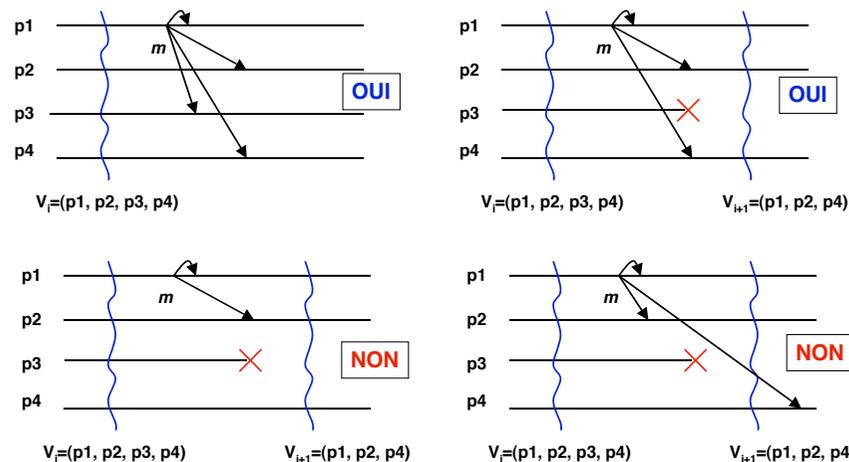
Nous considérons un groupe unique  $g$ , sans partition. L'extension principale concerne l'**accord** :

- Les processus de  $g$  (tant qu'ils ne sont pas en panne) voient **la même** séquence totalement ordonnée de vues  $V_0(g), V_1(g), \dots$
- Les processus de  $g$  (tant qu'ils ne sont pas en panne) voient **le même** ensemble de messages délivrés entre la délivrance d'une vue  $V_i(g)$  et la délivrance d'une vue  $V_{i+1}(g)$

La condition de **justification** est étendue pour spécifier que tout message délivré a été envoyé.

La condition de **vivacité** concerne d'une part le changement de vue (comme plus haut) et d'autre part la délivrance des messages

## Diffusion à vues synchrones : spécifications (2)



## Diffusion à vues synchrones : principe de réalisation

**Hypothèse** : système asynchrone avec pannes franches et réinsertion. Néanmoins une hypothèse de synchronisme est nécessaire pour les réalisations pratiques (détecteur de pannes utilisant un délai de garde)

**Résultat** : la diffusion à vues synchrones est réalisable avec le consensus

On procède en deux phases lors du changement de vues :

- définir une valeur initiale (estimation) pour l'ensemble de messages délivrés lors de la vue courante et pour la prochaine vue ;
- lancer le consensus sur ces estimations.

Les propriétés du consensus (accord, validité, terminaison) sont alors applicables au changement de vue

Ch. Malloth, A. Schiper, View Synchronous Communication in Large Scale Networks. Proc. 2nd Open Workshop, ESPRIT project Broadcast

## Diffusion à vues synchrones : exemple de réalisation (1)

### Etape 1 : choix des valeurs initiales

Soit  $c$  la coupure (état du système) au lancement du protocole. Soit  $\text{MsgSet}_{k,i}(c)$  l'ensemble de messages délivrés par  $p_i$  dans la vue  $k$  et la coupure  $c$ . Chaque processus  $p_i \in V_k$  envoie  $\text{MsgSet}_{k,i}(c)$  à tous les processus de  $V_k$ . Alors  $p_i$  attend qu'il existe une coupure  $c'$  telle que

- sur  $c'$ ,  $\text{CommSet}_i(c')$  (l'ensemble de processus non soupçonnés par  $p_i$ ) est une majorité dans  $V_k$  :  $|\text{CommSet}_i(c')| > |V_k|/2$
- si sur  $c'$ ,  $p_i$  n'a pas reçu  $\text{MsgSet}_{k,j}$  de  $p_j \in V_k$ , alors  $p_j \in \text{Susp}_i(c')$  (la liste des processus soupçonnés par  $p_i$  dans  $c'$ )

Si  $p_i$  est correct, si moins de la moitié des processus de  $V_k$  sont fautifs, et si le détecteur possède la complétude forte, alors  $c'$  existe. Alors  $p_i$  propose les estimations suivantes

- messages :  $m_i =$  l'union des  $\text{MsgSet}_{k,j}(c)$  reçus par  $p_i$  dans  $c'$
- vue suivante :  $\text{nextv}_i = \text{CommSet}_i(c')$

Notons que si  $p_j \in \text{nextv}_i$ , alors  $\text{MsgSet}_{k,j}(c) \subseteq m_i$

Le cas d'une sortie volontaire est simple (équivalent à une panne certaine)

## Diffusion à vues synchrones : exemple de réalisation (2)

### Etape 2 : consensus

Le consensus est lancé à partir des propositions  $m_i$  et  $nextv_i$  des processus de  $V_k$ . Le résultat du consensus est le couple  $(MsgSet_k, V_{k+1})$ . Notons que ce résultat vérifie aussi la propriété d'inclusion :

$$\text{si } p_i \in V_{k+1}, \text{ alors } MsgSet_{k,i} \subseteq MsgSet_k$$

Quand un processus reçoit ce résultat, il délivre les messages de  $MsgSet_k$  qu'il n'a pas déjà délivrés et délivre la vue suivante  $V_{k+1}$ .

### (Ré)intégration de processus

Soit  $p_i \notin V_k$ . Pour rejoindre  $g$ ,  $p_i$  envoie une requête  $join\text{-}req(p_i)$  à un membre au moins de  $V_k$ . De même qu'une nouvelle suspicion, la réception de cette requête déclenche le protocole de changement de vue. Le message  $join$  est inclus dans l'estimation des messages.

Soit  $JoinSet_k =$  ensemble des  $p_j$  tels que  $join\text{-}req(p_j) \in MsgSet_k$ . Alors  $V_{k+1} = V_{k+1}^{temp} \cup JoinSet_k$ , où  $V_{k+1}^{temp}$  est le résultat du consensus, noté  $V_{k+1}$  plus haut

## Diffusion totalement ordonnée : spécifications

### Diffusion totalement ordonnée (ou atomique)

- fiable
- **Accord** : si un processus correct délivre un message  $m$ , tous les processus corrects délivrent  $m$  (au bout d'un temps fini)
  - **Validité** : si un processus correct diffuse un message  $m$ , tous les processus corrects délivrent le message  $m$  (au bout d'un temps fini)
  - **Intégrité** : Quel que soit le message  $m$ , il est délivré au plus une fois à tout processus correct, et seulement s'il a été diffusé par un processus
- **Ordre total** : si les processus corrects  $p$  et  $q$  délivrent tous les deux les messages  $m$  et  $m'$ , alors  $p$  délivre  $m$  avant  $m'$  seulement si  $q$  délivre  $m$  avant  $m'$

Donc tous les processus corrects délivrent **le même ensemble** de messages **dans le même ordre**. Cette propriété d'ordre total garantit une propriété forte : les processus ont la même vue de l'état du système  
On peut rajouter l'**uniformité** de l'accord, de l'intégrité et de l'ordre

## Réalisation de la diffusion totalement ordonnée (1)

**Résultat fondamental** : dans un système asynchrone avec pannes franches, la diffusion totalement ordonnée est équivalente au consensus

- a) Si on dispose d'un algorithme de diffusion totalement ordonnée, on sait réaliser le consensus.

Pour proposer une valeur, chaque processus la diffuse atomiquement à tous les membres du groupe. Tous les processus corrects reçoivent le **même ensemble** de valeurs dans le **même ordre**. Ils décident la première valeur, la même pour tous.

- b) Si on dispose d'un algorithme de consensus, on sait réaliser la diffusion totalement ordonnée.

La preuve est plus complexe, cf Chandra, Toueg 1996. L'algorithme de réduction est donné ci-après

## Réalisation de la diffusion totalement ordonnée (2)

L'algorithme ci-dessous est exécuté par chaque processus  $p$  du groupe

Initialisation :

```
R_delivered := ∅
A_delivered := ∅
k := 0
```

Exécution de A-broadcast(m) :

```
R-broadcast(p, m) } (1)
```

R-broadcast est la diffusion fiable  
A-broadcast est la diffusion atomique

A-deliver(...) se produit comme suit :

```
when R-deliver(p, m)
  R_delivered := R_delivered ∪ {m}

when R_delivered - A_delivered ≠ ∅
  // test périodique : ∃ messages non délivrés ?
  k := k + 1
  A_undelivered := R_delivered - A_delivered
  propose(k, A_undelivered)
  wait until decide(k, msgSetk)
  A_deliverk := msgSetk - A_delivered
  A-deliver (tous les messages de A_deliverk
             dans un ordre déterministe)
  A_delivered := A_delivered ∪ A_deliverk
```

} (2)  
} (3)

L'algorithme comporte 3 tâches indépendantes (asynchrones). La délivrance des messages intervient dans la phase 3, qui comprend un nombre indéterminé de tours. Le **consensus** est utilisé pour déterminer quel est l'ensemble de messages (**le même pour tous**) qui sera délivré à chaque tour.

## Réalisation de la diffusion totalement ordonnée (3)

Conséquence de l'équivalence : tous les résultats du consensus s'appliquent à la diffusion totalement ordonnée

- Il n'existe pas d'algorithme déterministe réalisant la diffusion totalement ordonnée en asynchrone avec pannes franches (d'après FLP)
- La diffusion totalement ordonnée est réalisable avec un détecteur de pannes de classe P ou S et tolère  $n - 1$  pannes pour  $n$  processus
- La diffusion totalement ordonnée est réalisable avec un détecteur de pannes de classe  $\diamond$  S et tolère  $\lceil n/2 \rceil - 1$  pannes pour  $n$  processus

Dans la pratique, les algorithmes de diffusion totalement ordonnée utilisent une hypothèse de synchronisme (délai de garde) pour détecter les pannes

## La diffusion totalement ordonnée, en pratique

Un protocole de diffusion totalement ordonnée doit garantir l'ordre et tolérer les défaillances.

L'ordre peut être assuré de plusieurs manières

- **Par un ou plusieurs séquenceur(s) (sites spécialisés) fixe ou tournant**
- **Par les émetteurs des messages**
- **Par les récepteurs des messages**

Diverses méthodes de tolérance aux défaillances peuvent être utilisées (en particulier celles vues précédemment) : stabilité, communications redondantes, détecteurs de pannes, consensus

Source : synthèse sur la diffusion totalement ordonnée : X. Defago, A. Schiper, P. Urbán. Total Order Broadcast and Multicast Algorithms, *Technical Report*, École Polytechnique Fédérale de Lausanne, 2004

## Diffusion totalement ordonnée : ordre par séquenceur

C'est l'analogue des méthodes à base de coordinateur vues précédemment. Pour diffuser un message, on l'envoie à un processus particulier, le **séquenceur**. Celui-ci attribue au message un numéro de séquence (entiers consécutifs) et envoie le message à tous. Les messages sont délivrés dans l'ordre des numéros de séquence.

Schéma de base : **séquenceur fixe prédéfini**. Variantes selon le protocole d'envoi des messages. Ex : Kaashoek et al.

Autre schéma : **séquenceur circulant**. Plusieurs processus (voire tous) peuvent jouer le rôle de séquenceur, et ce rôle (matérialisé par un jeton) tourne entre ces processus. Ex : Chang - Maxemchuk

Comme dans tous les schémas privilégiant un processus particulier, le problème est de traiter la défaillance du séquenceur. Détails plus loin

## Séquenceur simple (sans traitement des pannes)

### émetteur

```
broadcast(p, m)
send(m) to sequencer
```

### séquenceur

```
Initialisation:
seqnum := 1;
when receive(m)
sn(m) = seqnum;
send(m, sn(m)) to all
seqnum := seqnum+1
```

### destinataire $p_i$

```
Initialisation:
nextdeliver_  $p_i$  := 1;
pending_  $p_i$  :=  $\emptyset$ ;
when receive(m, seqnum)
pending_  $p_i$  := pending_  $p_i$   $\cup$  {m, seqnum};
while  $\exists$  (m, seqnum')  $\in$  pending_  $p_i$  : seqnum' = nextdeliv_  $p_i$  do
deliver(m')
nextdeliv_  $p_i$  := nextdeliv_  $p_i$  +1
```

## Diffusion totalement ordonnée : ordre fixé par émetteur

Deux méthodes

**Circulation d'un privilège (jeton).** Un émetteur doit attendre d'avoir le jeton pour diffuser un message (analogie avec l'anneau à jeton - *token ring* - vu pour l'exclusion mutuelle).

Chaque émetteur numérote séquentiellement ses messages. Donc chaque message est identifié par un couple (émetteur, numéro de séquence). Les destinataires délivrent les messages dans l'ordre (émetteur, numéro)

**Ordre utilisant l'histoire des émissions (estampilles d'émission).**

Exemple 1 : ordre causal. Les messages sont estampillés par l'heure logique d'émission et délivrés par chaque récepteur dans l'ordre des estampilles (le numéro de l'émetteur discrimine en cas de conflit).

Exemple 2 : ordre arbitraire uniforme défini par convention. Par exemple : message (premier non délivré) de  $p_1$ , message de  $p_2$ , ... message de  $p_n$ , message suivant de  $p_1$ , etc.

Problème : **vivacité** (si un processus n'émet rien) - on demande à tous d'émettre des messages vides

## Ordre utilisant des estampilles en émission (sans traitement des pannes, canaux FIFO)

### processus p (émetteur et destinataire)

```

Initialisation:
  received_p := ∅ ; delivered_p := ∅ ;
  LC_p[p1, ..., pn] := {0, ..., 0} ; // LC_p[q] : horloge logique de q, vue par p
broadcast (m)
  LC_p[p] := LC_p[p] + 1 ;
  ts(m) := LC_p[p] ; // rôle émetteur
  send FIFO (m, ts(m)) to all
when receive (m, ts(m))
  LC_p[p] := max(ts(m), LC_p[p] + 1) ; // rôle destinataire
  LC_p[sender(m)] := ts(m) ;
  received_p := received_p ∪ {m} ;
  deliverable := ∅ ;
  for each message m' in (received_p \ delivered_p) do // stabilité assurée
    if ts(m') ≤ min LC_p[q] then
      deliverable := deliverable ∪ {m'} ;
  délivrer tous messages dans deliverable, ordre croissant de (ts(m), sender(m)) ;
  delivered_p := delivered_p ∪ deliverable ;
    
```

## Diffusion totalement ordonnée : accord des récepteurs

On peut utiliser un protocole d'accord entre les destinataires pour définir l'ordre de délivrance des messages. Exemples :

**Ordre utilisant des estampilles en réception.** Chaque message  $m$  est estampillé provisoirement à l'arrivée par l'heure logique de réception.

Les différents récepteurs se communiquent leurs estampilles, et quand toutes sont connues, on attribue définitivement à  $m$  la plus grande.

Donc tout message a la même estampille (définitive) sur tous les sites récepteurs. Les messages sont délivrés dans l'ordre de ces estampilles.

N.B. La **stabilité** est garantie par la règle du max : quand un message estampillé par  $e$  est délivré, un autre message non encore délivré ne peut recevoir une estampille inférieure à  $e$ .

Principe utilisé dans la version initiale d'ABCAST (Isis). Problème : le traitement des défaillances est complexe

**Consensus entre les récepteurs.** Algorithme présenté plus haut.

Avantage : modularité. Le traitement des défaillances est inclus dans le service générique de consensus

## Accord des récepteurs (exemple)

$p_1$	$p_2$	$p_3$
estampilles provisoires (après réception)		
m2(2, 3)	m1(1,2)	m3(3, 3)
m3(3, 4)	m2(2, 4)	m2(2, 5)
m1(1, 5)	m3(3, 5)	m1(1, 7)

estampilles définitives (après échange mutuel (chaque message reçu envoyé à tous avec estampille provisoire). Une estampille est définitive quand on a une copie du message en provenance de tous les sites

m2(2, 5)	m1(1, 7)	m3(3, 5)
m3(3, 5)	m2(2, 5)	m2(2, 5)
m1(1, 7)	m3(3, 5)	m1(1, 7)

m2 devient délivrable avant m3 même si m3 a son estampille définitive avant

un message devient **délivrable** s'il a son estampille **définitive** et que celle-ci est inférieure à celle de tous les messages arrivés (avec estampille provisoire)

les messages délivrables sont délivrés dans l'ordre des estampilles définitives

**m2, m3, m1 sur chaque site**

Le conflit entre m2 et m3 (même estampille) est résolu par le numéro de l'émetteur ( $2 < 3$ )

## Diffusion totalement ordonnée : synthèse sur l'ordre

### Ordre déterminé par séquenceur

fixe

circulant ←

### Ordre déterminé par émetteur

privilège ←

historique ←

### Ordre défini par accord entre les récepteurs

numéro en réception ←

consensus

utilise un jeton

utilise des estampilles

## Diffusion totalement ordonnée : défaillances (1)

Plusieurs voies d'approche pour la tolérance aux fautes dans la diffusion totalement ordonnée

Utiliser un mécanisme sous-jacent qui repose sur un détecteur de pannes

Consensus (algorithme vu ci-dessus)

Protocole d'appartenance (installation de vues), et vues synchrones

Intérêt : séparation des problèmes grâce à la modularité

Utiliser directement des techniques de redondance

Dans le cas du séquenceur : détection de la panne du séquenceur et élection d'un nouveau séquenceur. Problème pour le nouveau séquenceur : reconstituer l'histoire, c'est-à-dire déterminer à partir de quel numéro il doit reprendre. Intérêt ici de la diffusion redondante (par l'émetteur **et** par le séquenceur)

## Diffusion totalement ordonnée : défaillances (2)

Plusieurs voies d'approche pour la tolérance aux fautes dans la diffusion totalement ordonnée (suite)

Utiliser la propriété de **stabilité** des messages.

Un message est  $k$ -stable s'il a été reçu par  $k$  processus. Si on tolère  $f$  défaillances au plus, un message  $(f+1)$ -stable a été reçu par **au moins un** processus correct, et donc sa délivrance pourra être garantie. La  $k$ -stabilité, en pratique, est vérifiée par acquittement et délai de garde.

Les **pertes de messages** peuvent aussi être traitées. Par exemple la numérotation du séquenceur permet aux destinataires de détecter les "trous" et de redemander au séquenceur les messages manquants (il doit donc conserver les messages tant qu'ils risquent de lui être demandés).

## Diffusion totalement ordonnée : quelques problèmes (1)

Bien que les spécifications de la diffusion atomique soient contraignantes, en particulier dans le cas uniforme, on n'est pas à l'abri d'incohérences dès que le modèle de défaillances s'écarte du modèle (communication fiable, pannes franches)

Exemples :

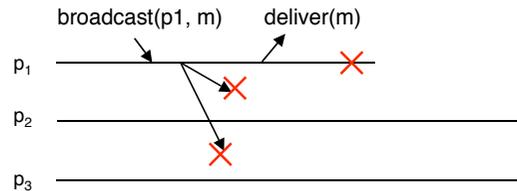
communication avec perte de messages (en nombre fini non borné)  
panne par omission : le processus perd un message lors de l'émission ou de la réception

## Diffusion totalement ordonnée : quelques problèmes (2)

Intérêt de la définition d'un protocole **uniforme**

Si le système de communication peut perdre des messages, un protocole non uniforme peut conduire à une incohérence

### Exemple



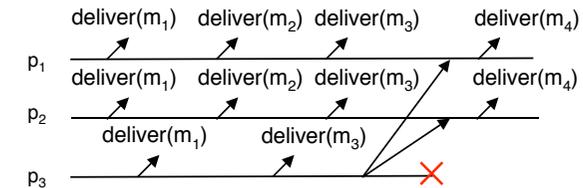
Ce cas respecte la spécification de la diffusion atomique, non celles de la diffusion atomique **uniforme**

## Diffusion totalement ordonnée : quelques problèmes (3)

### La contamination

Même si la diffusion est **uniforme**, des incohérences peuvent se produire pour des pannes de type **omission**. Le problème est que la spécification de l'accord uniforme est nécessairement asymétrique :

Si un processus **[correct ou fautif]**, délivre un message  $m$ , tous les processus **corrects** délivrent le message  $m$



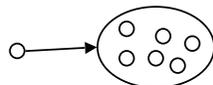
Ce cas respecte la spécification de la diffusion atomique **uniforme**. Néanmoins le message  $m_4$  diffusé par un processus dans un état incohérent peut "contaminer" les processus corrects

## Diffusion totalement ordonnée : groupes multiples (1)

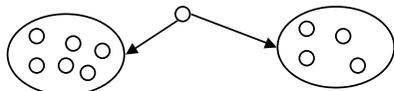
Jusqu'à présent, on a vu la diffusion dans un ensemble unique de processus (*broadcast*). On s'intéresse maintenant aux groupes multiples (*multicast*).

On suppose que l'émetteur peut ne pas appartenir au(x) groupe(s) des destinataires. Diverses formes de spécification sont possibles.

**Groupe unique** : pas de différence avec *broadcast*, sauf que l'émetteur peut être extérieur au groupe



**Groupes disjoints** : l'ordre total est respecté au sein de chaque groupe. Les algorithmes utilisés pour *broadcast* se transposent directement



## Diffusion totalement ordonnée : groupes multiples (2)

**Groupes multiples avec recouvrement** : la difficulté réside dans la spécification de l'ordre pour les processus appartenant à l'intersection. Plusieurs spécifications sont possibles.

**a) Ordre total local**. L'ordre total est respecté au sein de chaque groupe. Les processus de l'intersection, en tant que destinataires, ont une "personnalité" différente dans chaque groupe. Donc peu de différence avec groupes disjoints

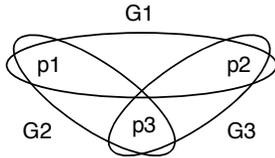
**b) Ordre total par paires**. Si deux processus corrects  $p$  et  $q$  délivrent les messages  $m$  et  $m'$ , alors  $p$  délivre  $m$  avant  $m'$  seulement si  $q$  délivre  $m$  avant  $m'$ .

Cette propriété est vraie en particulier si  $p$  et  $q$  appartiennent à une intersection de groupes. L'ordre (unique) dans l'intersection doit être compatible avec l'ordre au sein de chaque groupe

La réalisation de cette propriété est non-triviale (ne se déduit pas directement des algorithmes pour groupe unique)

## Diffusion totalement ordonnée : groupes multiples (3)

L'ordre total par paires pose un problème de cohérence lorsqu'il y a plus de 2 groupes.



Soit 3 messages  $m_1$ ,  $m_2$ ,  $m_3$

$p_1$  délivre  $m_3$  puis  $m_1$   
 $p_2$  délivre  $m_1$  puis  $m_2$   
 $p_3$  délivre  $m_2$  puis  $m_3$

Compatible avec ordre total par paires, mais incompatible avec ordre total dans chaque groupe.

Pour éviter cette situation, on spécifie :

**c) Ordre total global.** On définit la relation  $m < m'$  par la propriété "tout processus correct délivre  $m$  et  $m'$ , dans cet ordre". Dans l'ordre total global, la relation " $<$ " doit être acyclique.

## Conclusion sur la diffusion et les groupes

La diffusion et les protocoles de groupes constituent un domaine important, tant pour la théorie que pour les applications. C'est un domaine de recherche actif, car il reste des questions ouvertes.

Les algorithmes vus jusqu'ici s'appliquent à un ensemble de processus bien identifié. Ils imposent des propriétés strictes. Mais ils ont des limitations :

- ils s'appliquent mal aux situations où l'ensemble des processus est inconnu à l'avance et change dynamiquement
- ils passent très mal à l'échelle

Pour dépasser ces limitations, des algorithmes fondés sur un principe différent ont été introduits (algorithmes épidémiques). Ils reposent sur une approche statistique. Nous les examinons dans la suite.