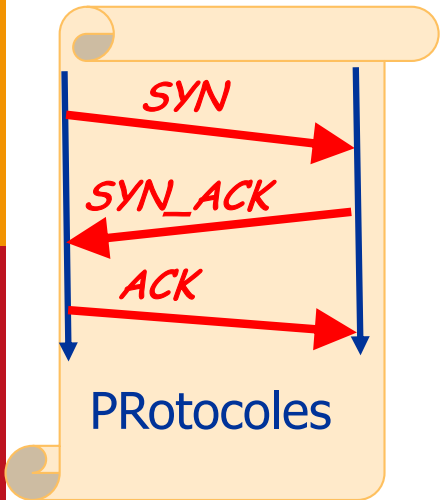
GRENOBLE INP UGA Ensimag

Chapitre PR2
Initiation aux problèmes d'algorithmes répartis
Parallélisme asynchrone

Transfert fiable malgré pertes & erreurs

Problème du consensus réparti

Preuve de protocoles



Parallélisme et répartition

- Parallélisme: **Architectures et algorithmes** qui **coopèrent** pour traiter des informations de manière plus ou moins **simultanée**
- Différentes formes de parallélisme
  - Inhérent dans architecture matérielle (accès concurrents au(x) bus, multi-coeurs...)
  - Calculs parallèles (ex: make –j), calcul Haute Performance (HPC), clusters, grilles de calcul
  - Applications et systèmes répartis
  - Protocoles de réseaux

Problèmes fondamentaux du //isme

$e^{i\pi}$

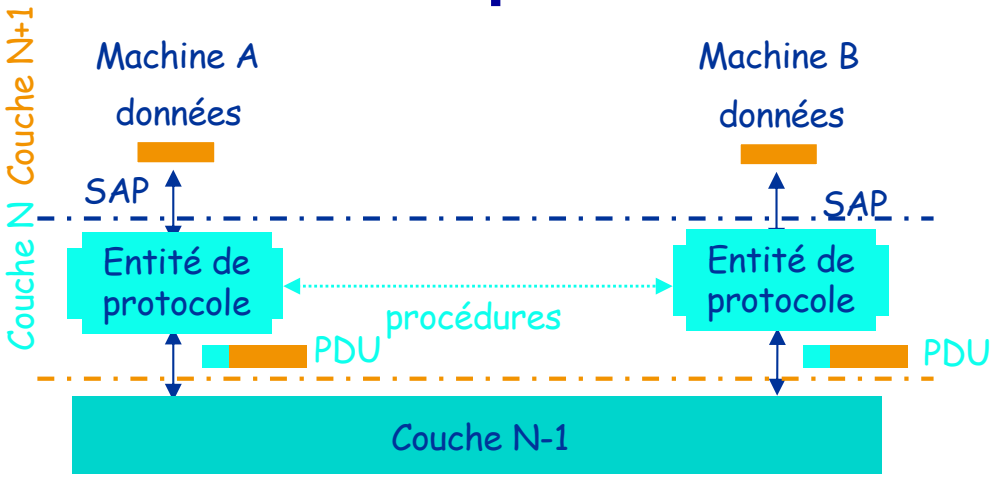
- Information locale, partielle vs calcul global
  - Difficultés: cf pb des 2 armées (plus loin)
  - Optimisations: répartition de l'information et du calcul
- Synchronisation entre les parties
- Non-déterminisme (délais variables)
  - programmation adaptée: commandes gardées, programmation par événement, appels non bloquants...
- Causalité et ordres partiels
  - Sémantiques d'ordres partiels

Contenu du chapitre PR2

NB: PR2 ne présente pas des protocoles, mais des éléments de solutions intervenant dans les protocoles réels sur quelques problèmes fondamentaux

- Acheminement fiable
  - Pb1: En présence d'erreurs de transmission (bits corrompus)
    - Codes détecteurs/correcteurs
  - Pb2: En présence de pertes de messages
- Consensus réparti fiable
  - Pb3: En présence de pertes de messages
    - Le problème des deux armées
  - Pb4: En présence de processeurs défectueux
    - Le problème des généraux byzantins
- Formalisation et preuve de protocoles
  - Automates, logiques temporelles, model-checking, simulation…
  - Liens avec cours TL1

Pb1: Fiabiliser le transfert en présence d'erreurs de transmission



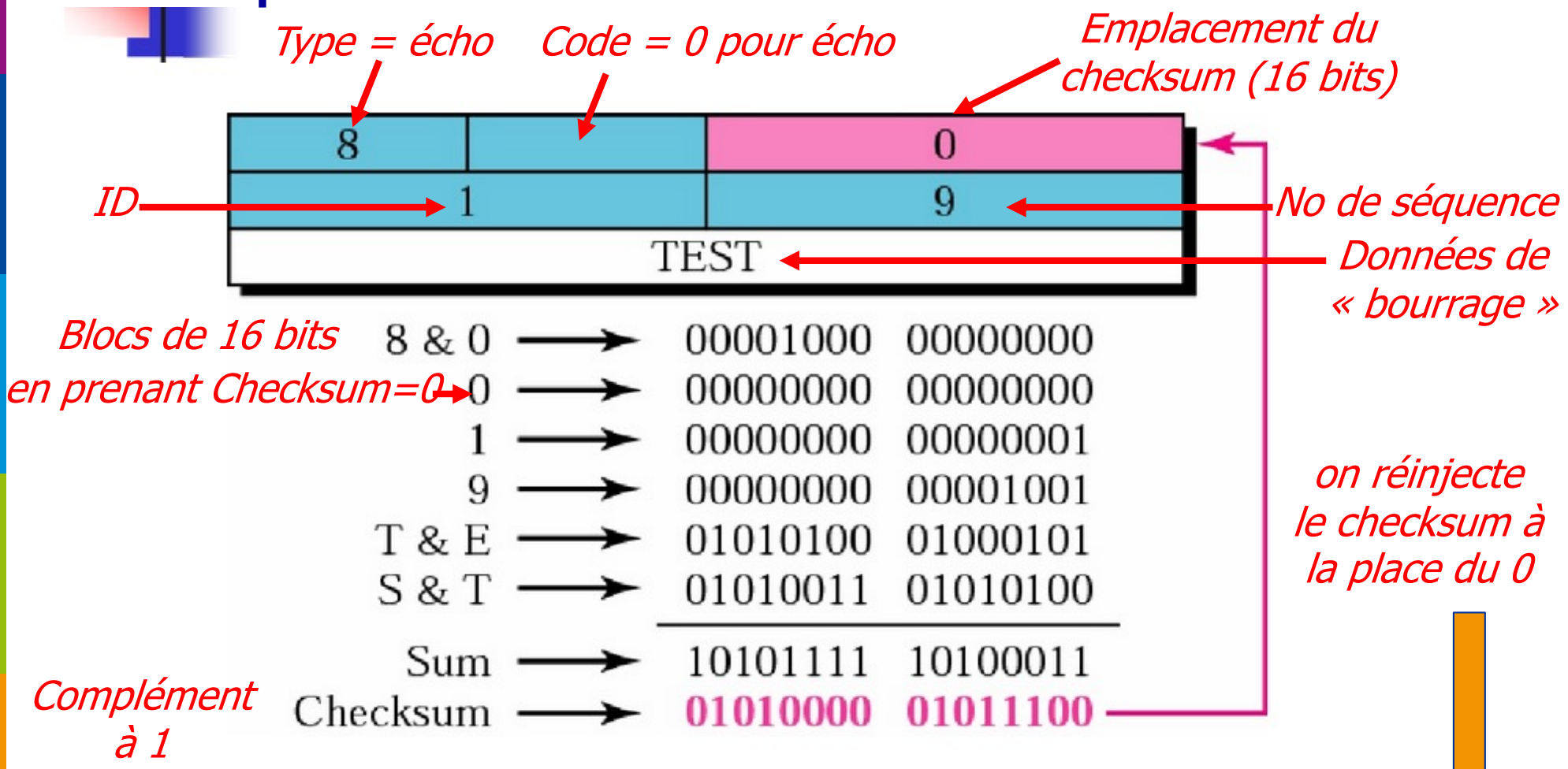
- *Problème: concevoir un protocole de couche N qui détecte et récupère les erreurs de transmission laissées par le service N-1*

  Bits corrompus entre l'envoi et la réception (avec probabilité faible)
- Détection
  - addition d'un champ de contrôle (checksum: code détecteur d'erreur) au contenu du message
  - détection d'incohérence par le récepteur, et signalisation vers l'émetteur (accusé de réception négatif: NACK)
- Récupération
  - Retransmission en cas d'erreur détectée (réception NACK)

Checksum: code détecteur d'erreurs

- Somme de contrôle:
  - découper le PDU de longueur M=p x n en p blocs de n bits
  - additionner les p nombres (modulo 2^n), total = S
  - en général on complémente à un: C = $(2^n-1) - S$
  - ajouter C à l'en-tête du PDU
  - vérifier à la réception que la somme S' des p blocs de bits reçus est toujours égale à S, i.e. que S' +C = 1111....11 = 2^n-1
- Détection presque parfaite
  - soit r << 1 la probabilité qu'un bit soit mal transmis
  - si r très faible, par exemple r= 10^{-6}, probabilité de PDU incorrect ~ M r + M r^2 +...
  - probabilité de non-détection ~ $Mr^2/ 2^n$ car si un seul bit modifié, la somme ne sera pas la même (il faut que 2 erreurs se compensent)
  - A.N. pour n=16, r= 10^{-6} et p=10, proba ~2,5 x 10^{-15}, soit deux erreurs par petabit de données transmises

Exemple: calcul checksum de ICMP Echo



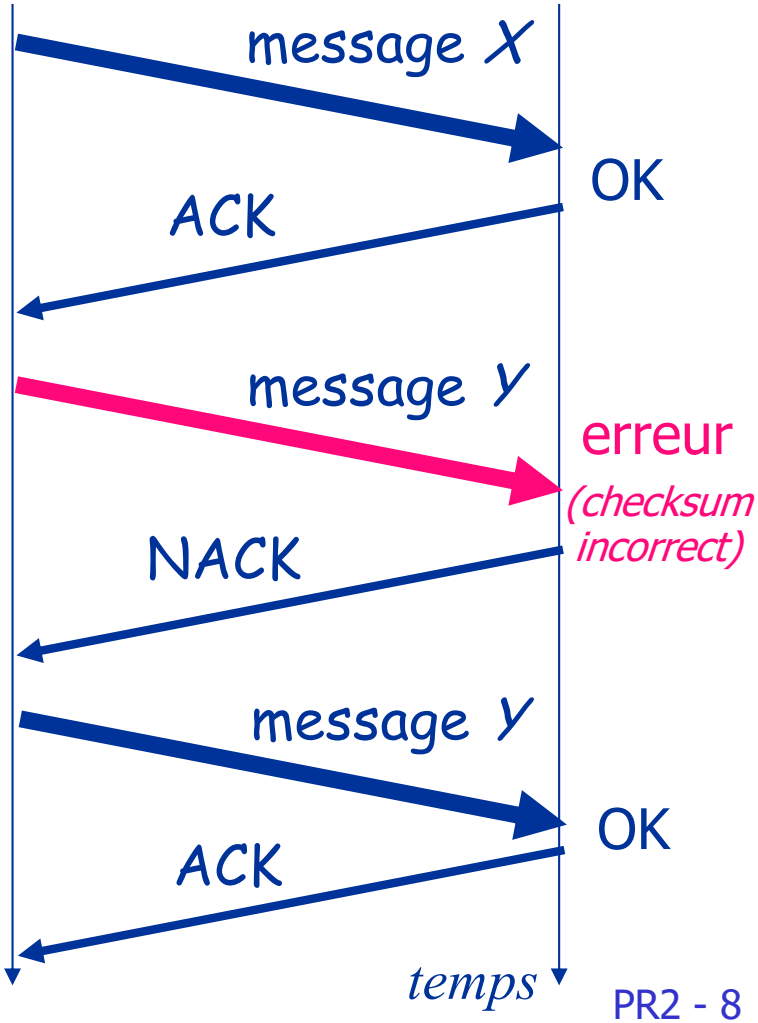
Donc à l'arrivée, le récepteur doit juste vérifier que la somme pour le paquet reçu est une suite de 1

Protocole « arrêt et attente » simple

pour résoudre les erreurs de transmission

- Transmission fiable en présence d'erreurs *(détectées par checksum)*
- Fonctionne bien en l'absence de pertes
- ACK/NACK=1 bit

- + Contrôle de flux
  - l'émetteur peut envoyer le message suivant dès qu'il reçoit ACK



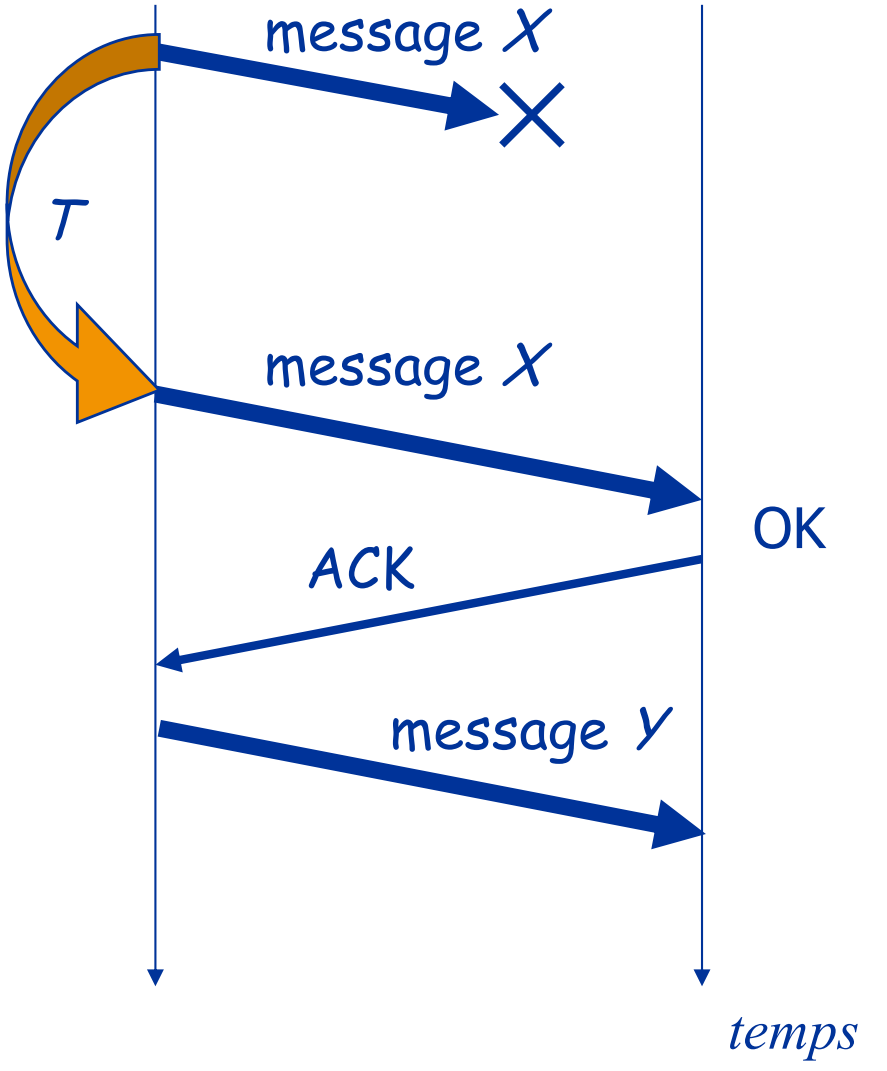
Pb2: Fiabiliser le transfert en présence de pertes

- Transmission fiable malgré des pertes
  - retransmettre en cas de perte - ARQ *(Automatic Repeat reQuest protocol)*
  - protocole : règles de coopération pour organiser la transmission

- Détection des pertes
  - Définition d'un temps d'attente (par ex. d'une réponse): *timer* (temporisation)
  - Le programme « entité de protocole » sera « réveillé » par l'O/S à échéance d'une temporisation

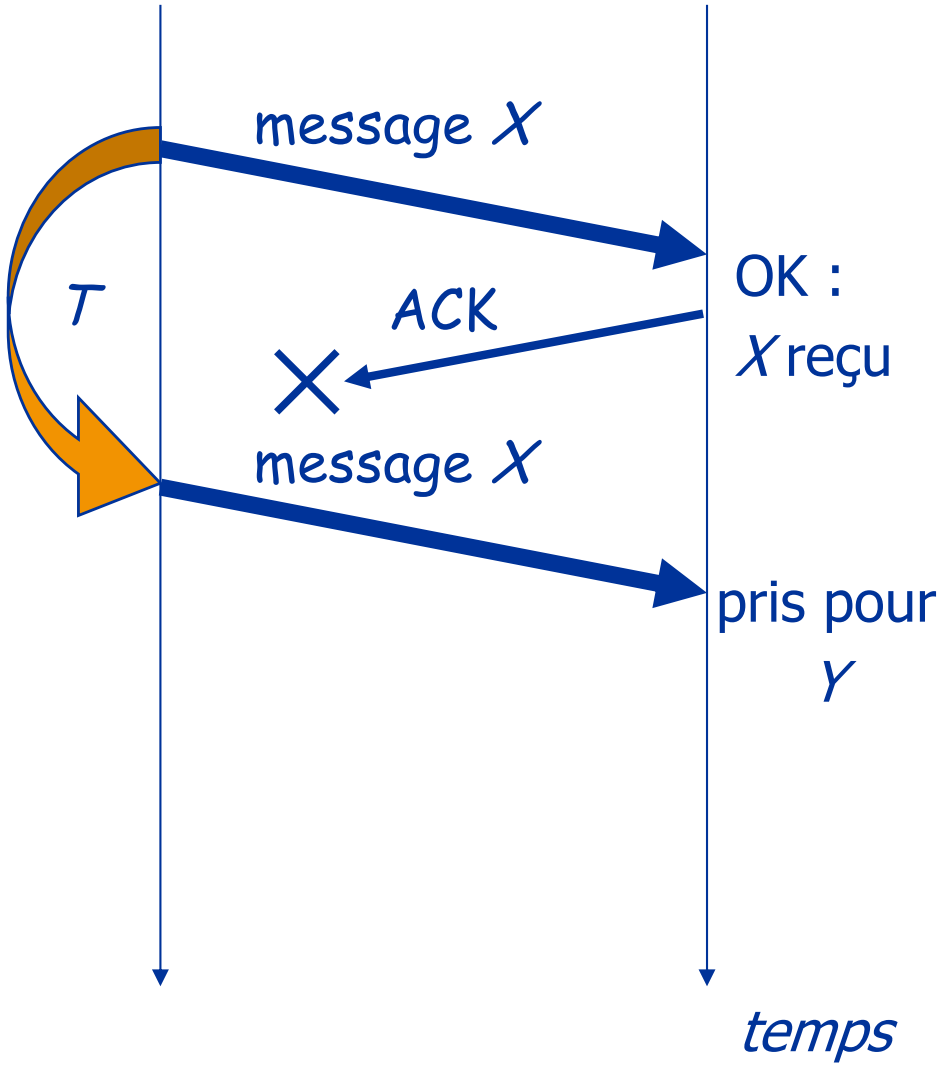
Cas de pertes: récupération avec timer

- Temporisation
  - si pas de réponse, retransmettre
  - l'intervalle T doit être supérieur au temps aller-retour



Problème de duplication
induit par la retransmission

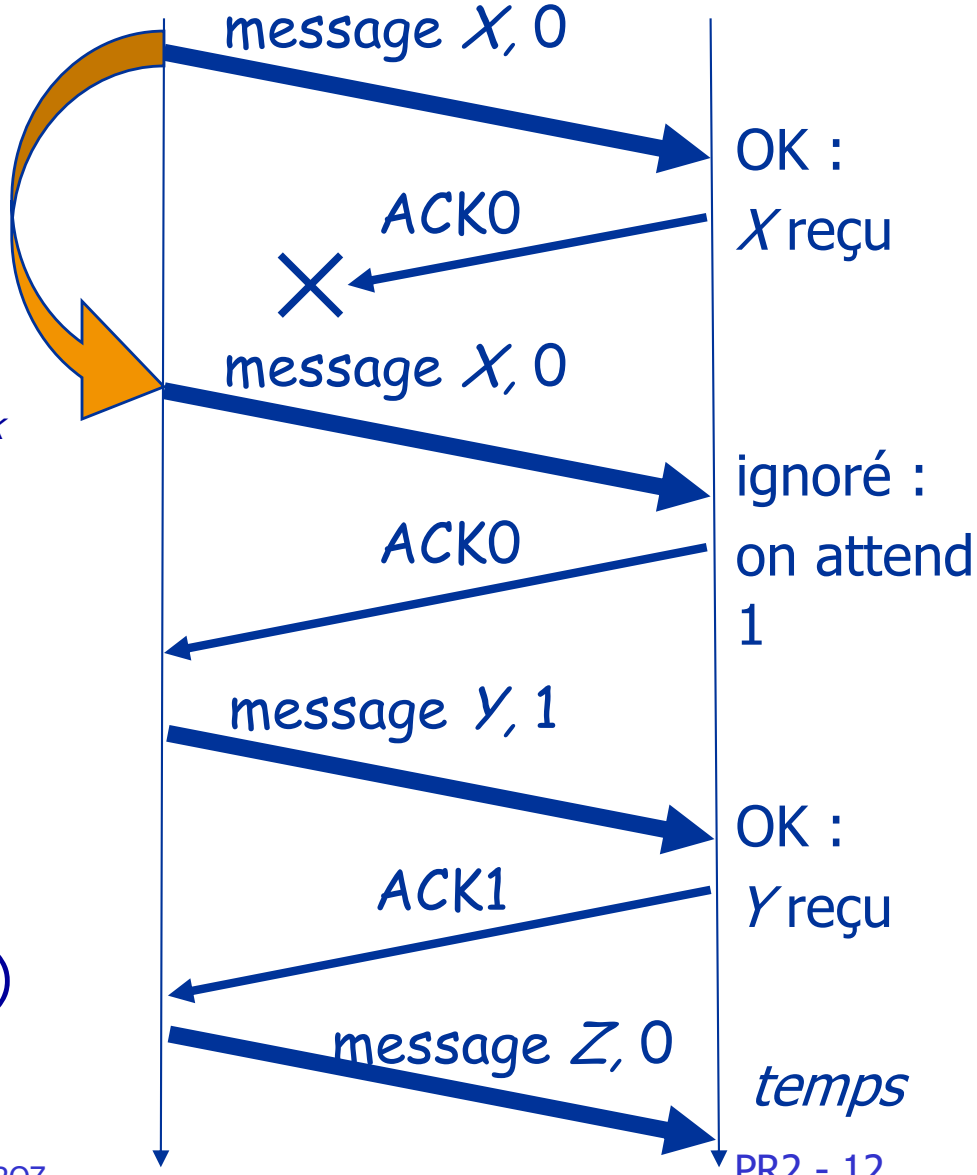
- Retransmission
  - on confond le message suivant avec le message retransmis
- Solution
  - numérotation de messages



Numérotation (résout la duplication)

- Numéros des messages et des ACKs
  - si un message en désordre, on l'ignore
  - compteur sur k bits - mod 2^k
- Simplification en arrêt et attente

- 1 bit suffit (protocole du bit alternant).
- ACK 0 -> 0
  ACK 1 -> 1
  NACK: on renvoie le bit (numéro) associé au dernier message correctement reçu: (NACK 0)=1

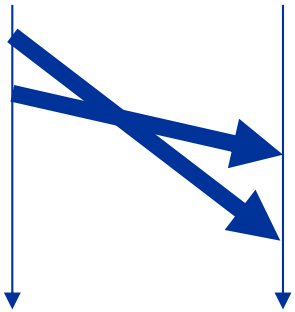


Arrêt et attente avec numéros mod 2
le protocole du bit alternant

- Protocole du Bit Alternant (ou « alterné ») *(Alternating Bit Protocol)*
  - récupère pertes et erreurs: transmission fiable
  - proposé en 1969 pour la couche 2 (liaison): « A note on reliable full-duplex transmission over half-duplex links » CACM May 1969
  - *Idée cruciale*: le bit n'a pas une signification fixe (0/1: ACK/NACK) MAIS il alterne (change de signification à chaque nouveau message)
    - NB: avec signification fixe (1ère idée naturelle), ÇA NE FONCTIONNERAIT PAS
    - Economie: le bit a une double(-triple) fonction, numéroter+accuser réception et il numérote AUSSI les messages en sens retour (piggybacking)
- Tampons ?
  - émetteur : 1 message
- Performances ?
  - on attend pendant tout le temps d'aller-retour

Sol. à pbs 1&2: Protocole du bit alterné

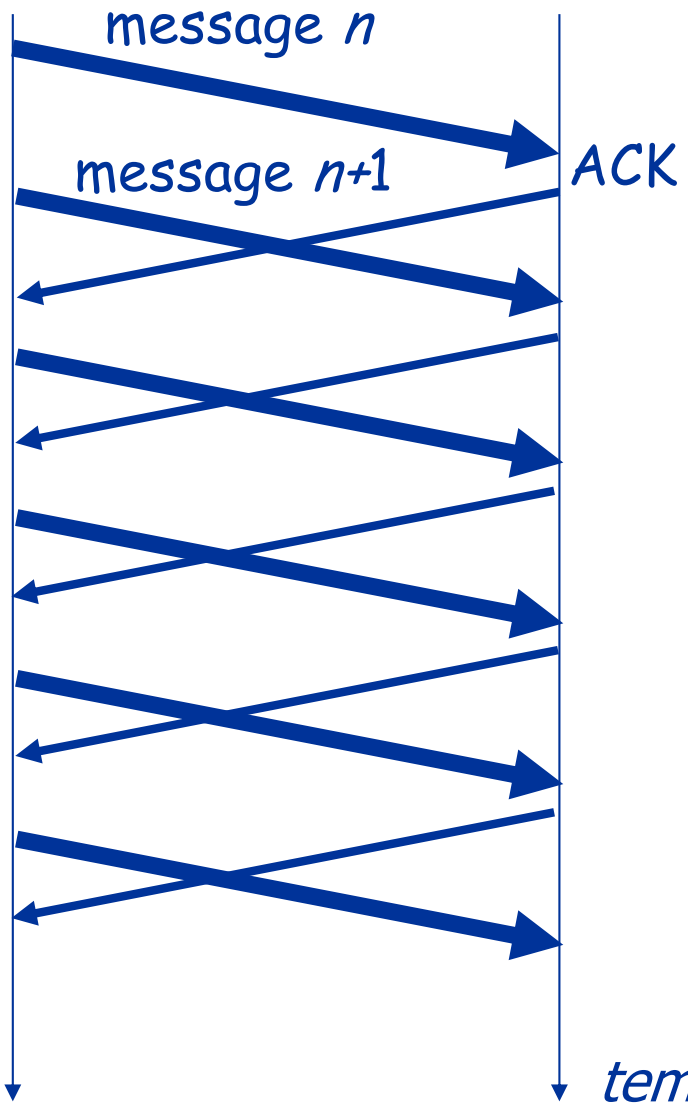
- Emetteur, var bit_courant init à 0
  - `Transmet (b,m) avec ajout 1 bit b=bit_courant`
  - `Si reçoit message (B,M)`
    - `Si B=bit_courant alors bit_courant:=1-bit_courant, peut envoyer mess suivant m'`
    - `Sinon renvoie (bit_courant,m)`
  - `Si ne reçoit rien, renvoie (bit_courant,m) au bout de T`
- Récepteur: idem
- Protocole prouvé correct (transmission fiable) si
  - Message peuvent être perdus par couche N-1
  - Bit d'en-tête non corrompu par N-1
  - Quel que soit T
- Protocole incorrect si ordre des messages non préservé par couche N-1 *(cf TD)*

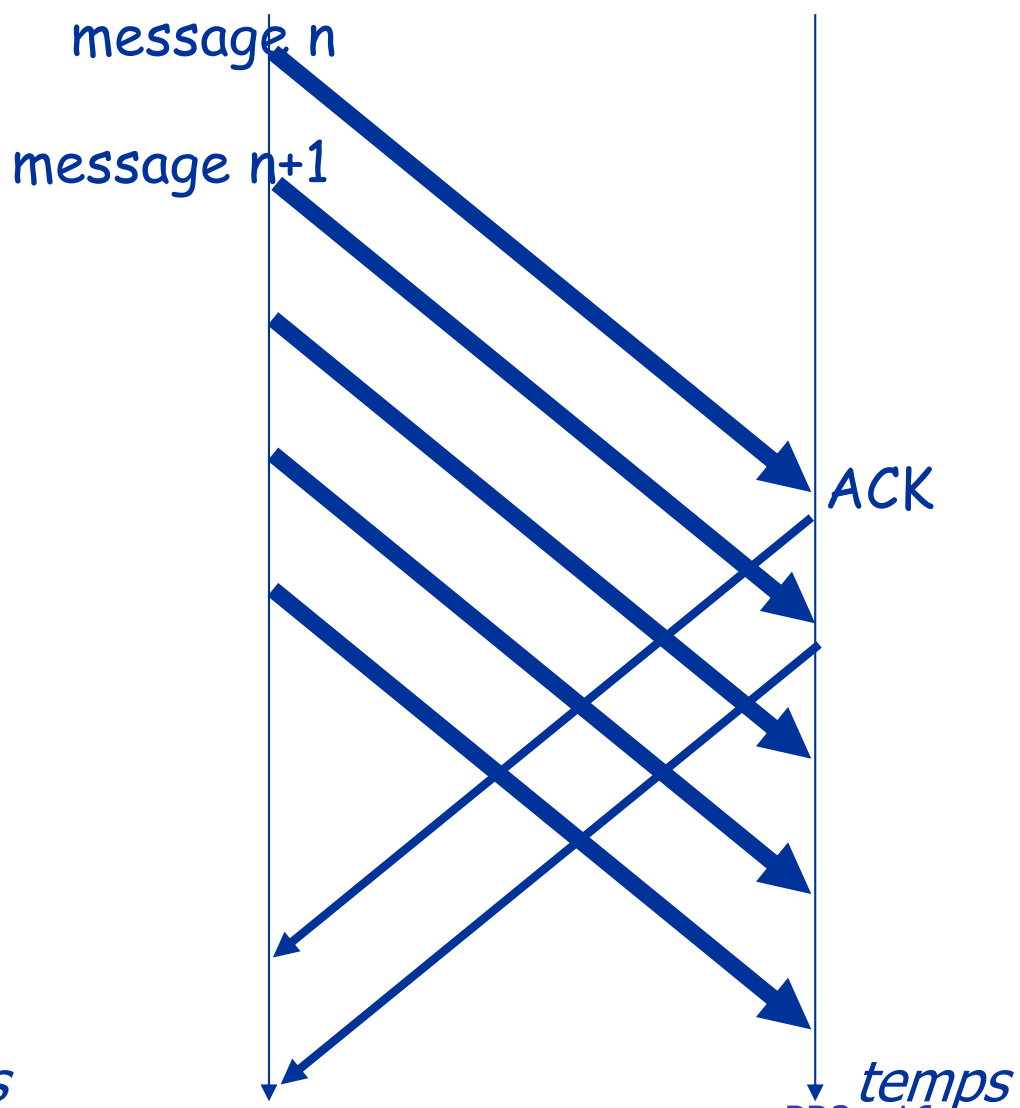


Solution généralisée: Protocoles à fenêtre

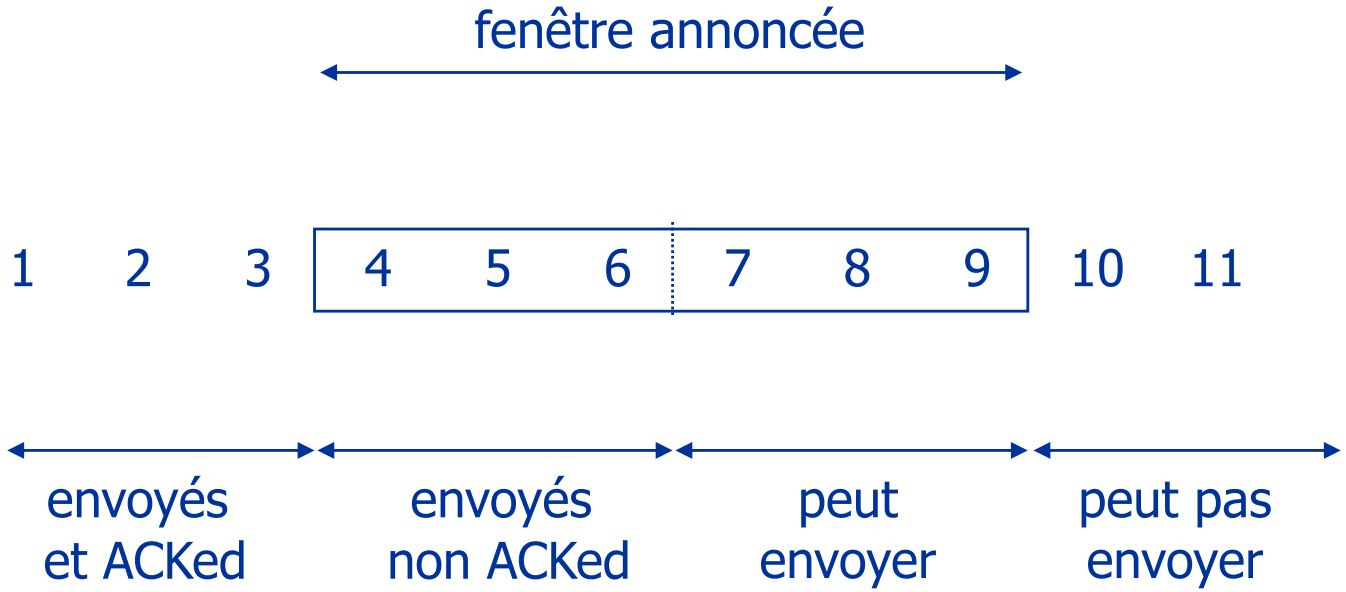
- Améliorer les performances
  - l'émetteur peut envoyer plusieurs messages sans attendre l'ACK
  - fenêtre d'émission : les messages à envoyer
  - taille de fenêtre ?
    - dépend du temps aller-retour
    - idéal : pouvoir envoyer (temps de transmission) le nombre de messages correspondant à un RTT
      - Mais le RTT pas forcément connu à l'avance
      - En TCP: négociation de « Window Size » et ajustement en fonction de la variation des temps (RTT, congestion)

Taille de fenêtre





Transfert



- Fenêtre glissante
  - avance à la réception de ACK
  - bloquée quand limite haute de la fenêtre est atteinte

Stratégies de retransmission

- Stratégies
  - Arrêt et attente *(Send and Wait)*
  - Retransmission contiguë *(Go-back-N)*
  - Retransmission sélective *(Selective Retransmit)*
- TCP
  - utilise une combinaison de Go-back-N et de retransmission sélective
  - numérote les octets transmis, et non pas les PDU (appelés segments)

Contenu du chapitre PR2

- Acheminement fiable
  - Pb1: En présence d'erreurs de transmission (bits corrompus)
    - Codes détecteurs/correcteurs
  - Pb2: En présence de pertes de messages
- Consensus réparti fiable
  - Pb3: En présence de pertes de messages
    - Le problème des deux armées
  - Pb4: En présence de processeurs défectueux
    - Le problème des généraux byzantins
- Formalisation et preuve de protocoles
  - Automates, logiques temporelles, model-checking, simulation…

Pb3: Le problème des deux armées

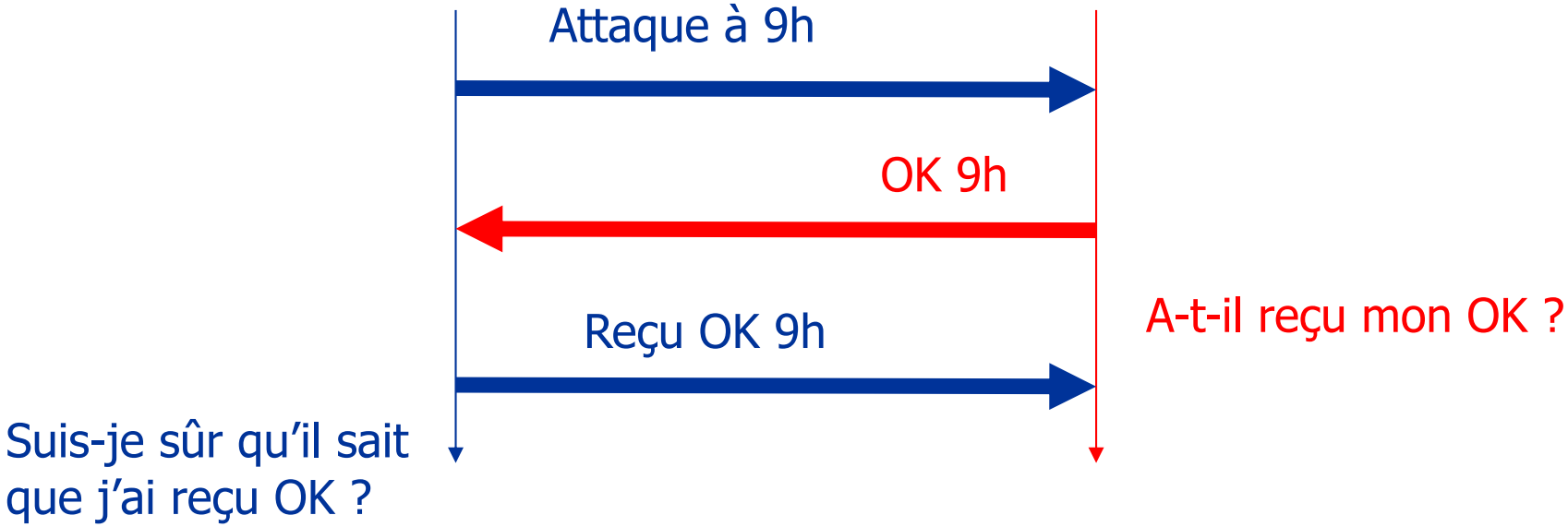
Armée bleue

Armée blanche

Armée rouge

- L'armée bleue et l'armée rouge (alliées) ne peuvent gagner contre l'armée blanche que si elles attaquent simultanément.
- Si une des deux armées attaque seule, elle est écrasée.
- Leurs messagers peuvent être interceptés par l'armée blanche.
- Pb: concevoir un protocole de synchronisation entre l'armée bleue et l'armée rouge pour attaquer ensemble

Tentatives de synchronisation



- Il ne peut exister de protocole garantissant que les deux parties attaqueront en même temps.
- Solutions approchées: algorithme probabiliste, limiter le risque d' échec (sans pouvoir le réduire à 0)
- *Application: TCP (fermeture de connexion)*

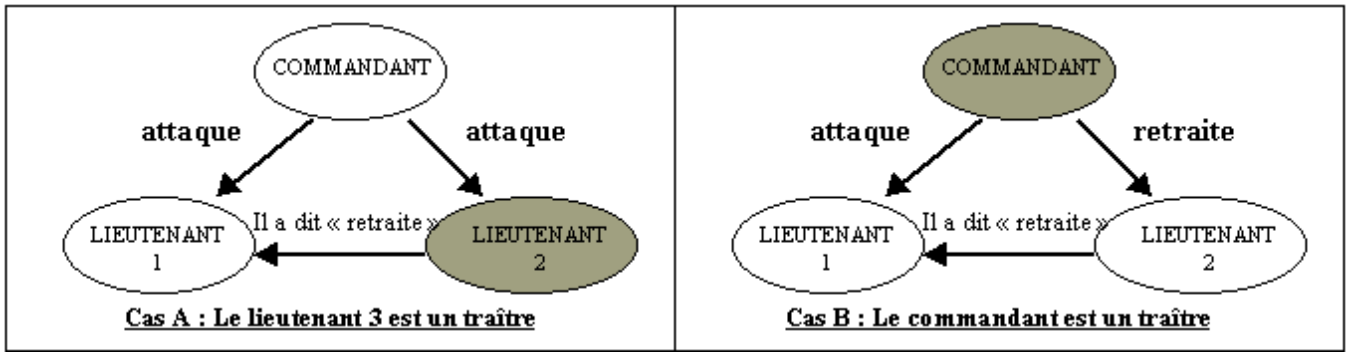
Pb 4: Le problème des généraux byzantins

Lamport, Pease, Shostak ACM TOPLAS 1982

- N généraux doivent attaquer la ville
- m généraux félons, N-m loyaux
- Les généraux félons(traîtres) peuvent mentir, et donner des ordres incohérents
- Objectif:
  - Tous les généraux loyaux doivent exécuter la même action
  - Un petit nombre de traîtres ne doit pas conduire à l'exécution d'un mauvais plan

Le problème des généraux byzantins

- Ex: 1 commandant coordonne attaque, les autres généraux (lieutenants) doivent décider d'une action commune



- Théorème (cas général): pas de solution si m >= N/3; il faut au moins N= 3m+1 généraux (donc au moins 2m+1 loyaux pour m félons)
- *Application: systèmes tolérants aux fautes, calcul du nombre de processeurs répliqués nécessaire*

Contenu du chapitre PR2

- Acheminement fiable
  - Pb1: En présence d'erreurs de transmission (bits corrompus)
    - Codes détecteurs/correcteurs
  - Pb2: En présence de pertes de messages
- Consensus réparti fiable
  - Pb3: En présence de pertes de messages
    - Le problème des deux armées
  - Pb4: En présence de processeurs défectueux
    - Le problème des généraux byzantins
- Formalisation et preuve de protocoles
  - Automates, logiques temporelles, model-checking, simulation…

Validation des protocoles

- Difficulté à concevoir des protocoles:
  - algorithmes répartis, l'esprit a de la peine à suivre plusieurs fils indépendants
  - asynchronisme, non-déterminisme (délais variables etc)
  - Possibilité de:
    - *réceptions non spécifiées* (machine dans un état où on ne lui a pas dit comment traiter tel message qu'elle reçoit)
    - interblocages « *deadlock* » (chaque entité attend une réponse d'une autre)
    - cycles non productifs « *livelock* » (chacun redemande à l'autre)
    - etc...
- Besoin de méthodes formalisant les comportements
  - pour raisonner et prouver
  - pour traitement automatique (preuve, génération de code...)

Automate (Mealy) décrivant un protocole

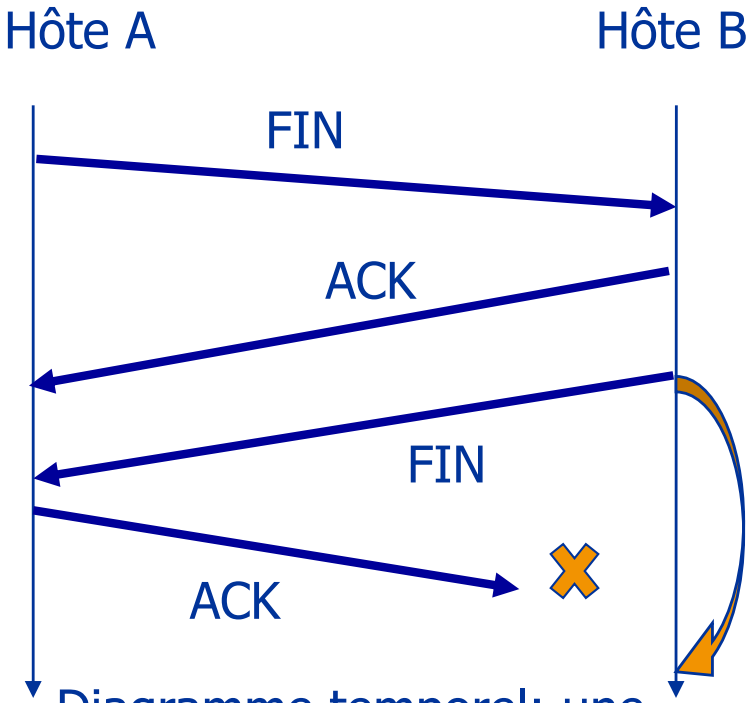
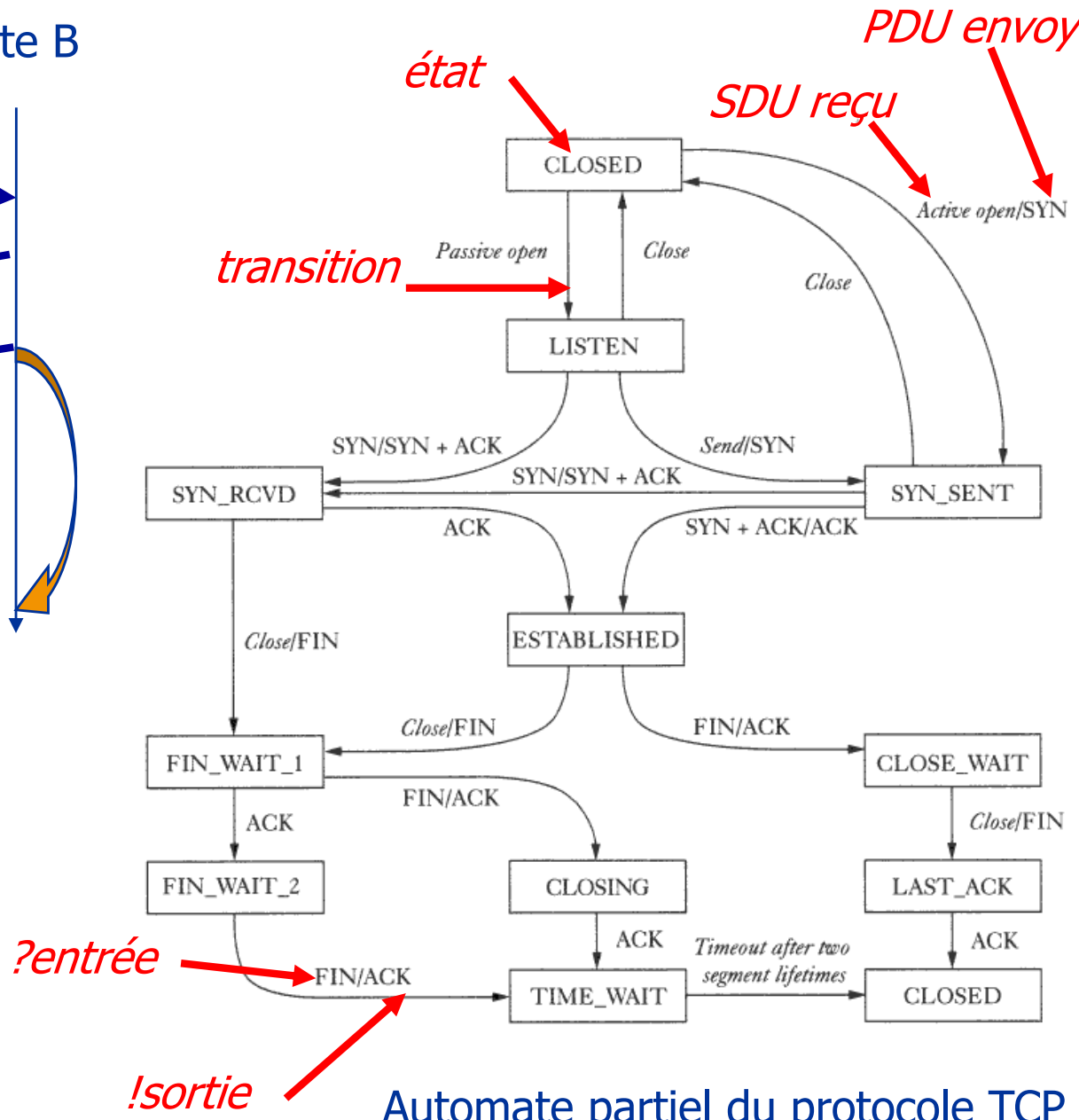


Diagramme temporel: une instance possible (vision globale)

Automate: comportement d'une entité de protocole (local)
enchaînements possibles d'états et de transitions étiquetées: ?entrée / !sortie



Automate partiel du protocole TCP

Définition d'un calcul par un langage

Correspondance TL

- Alphabet: ensemble des types de PDU
- Mot: séquence de PDU
  (un diagramme temporel
  = 1 exemple)
- Langage: enchaînements permis par le protocole
  (cf procédures pair à pair)
  = Tous les exemples possibles
- Calcul séquentiel: mot=ordre total

Difficultés => complexité

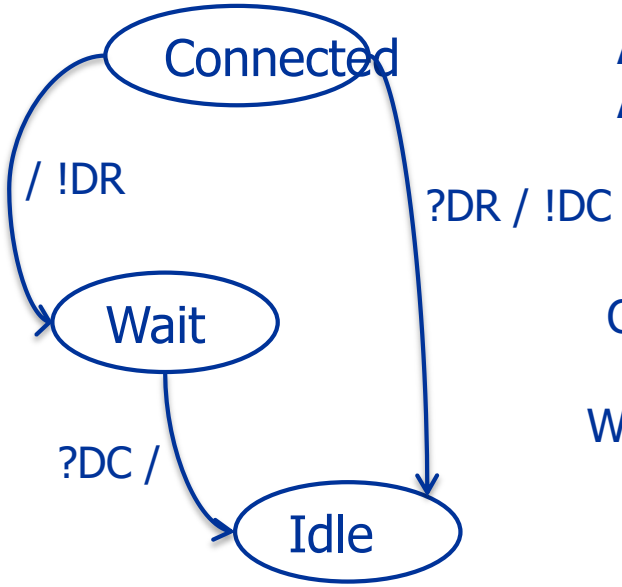
- Alphabet structuré (types complexes)
- séquence + temps réel
- Langage défini par une grammaire:
  - automate: langage régulier
  - automates+variables: Turing complete
- //isme asynchrone:
  - ordre partiel
  - mélange de langages

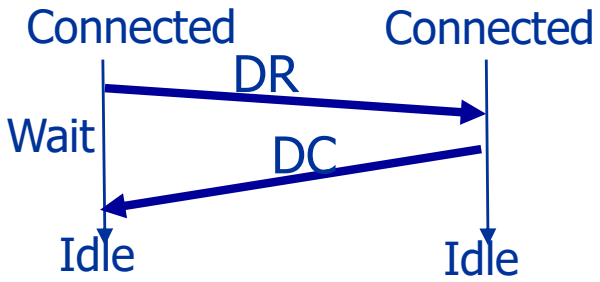
Exemple: petit protocole de déconnexion

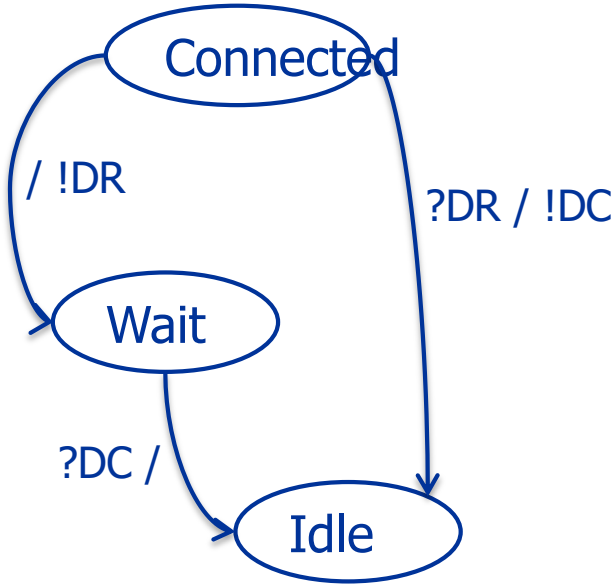
PDU

DR: Disconnect Request

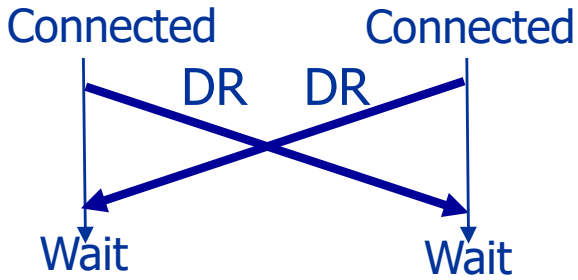
DC: Disconnect Confirm





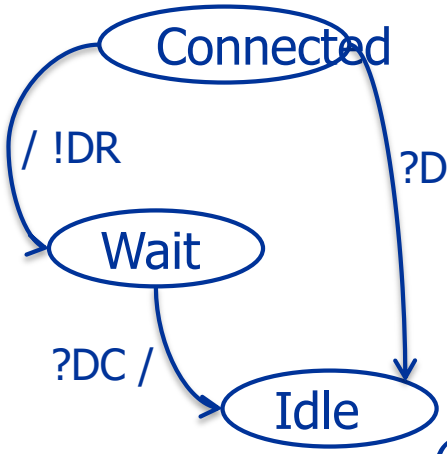


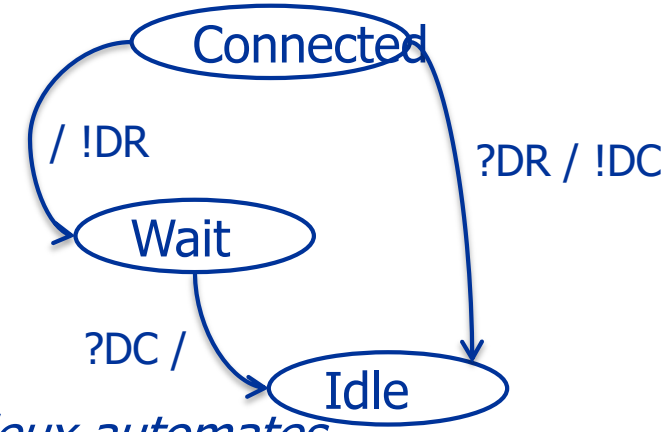
Comportement attendu



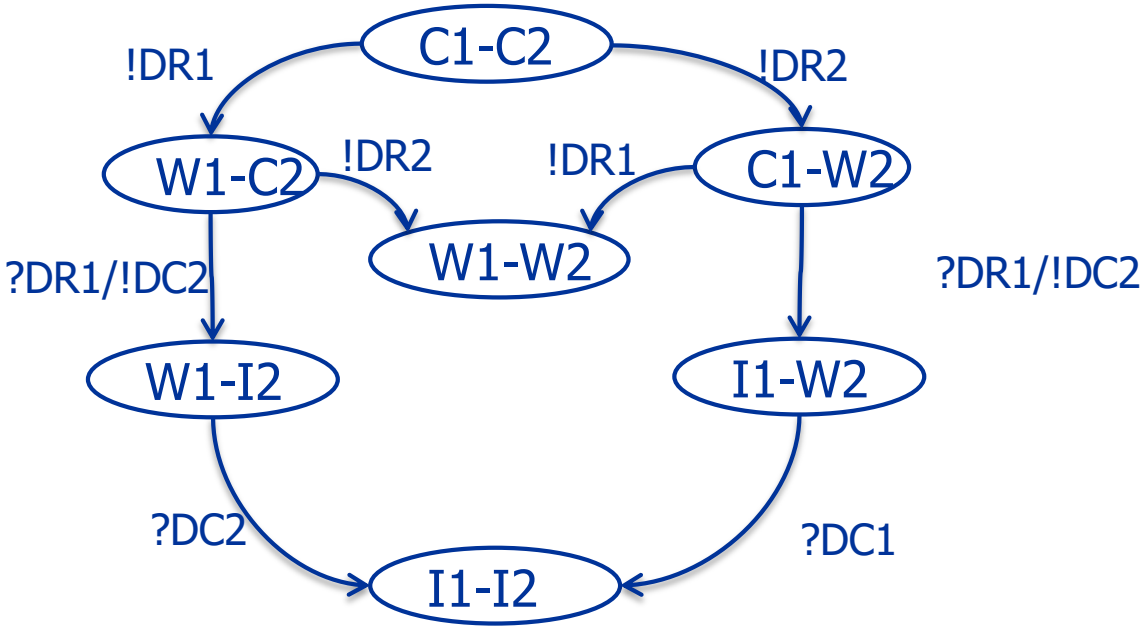
Interblocage

Algorithme pour vérifier le fonctionnement



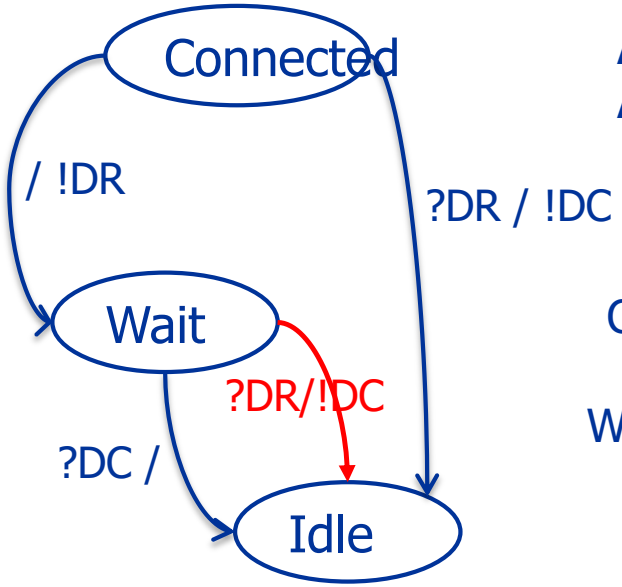


Calcul du produit asynchrone des deux automates

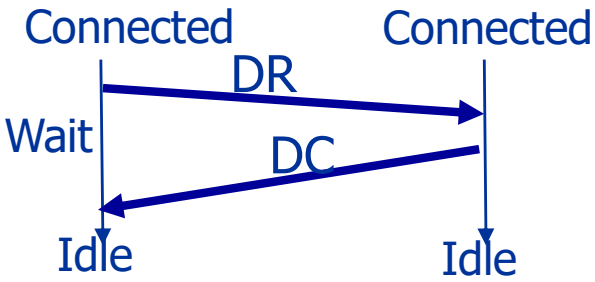


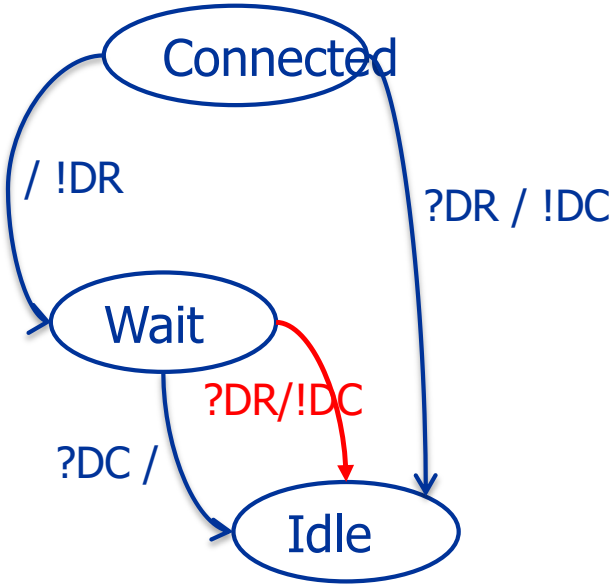
On peut identifier état puits anormal W1-W2

Exemple: 1ère tentative de solution

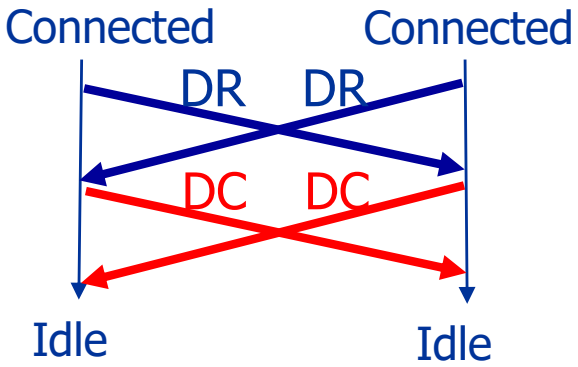


PDU
DR: Disconnect Request
DC: Disconnect Confirm



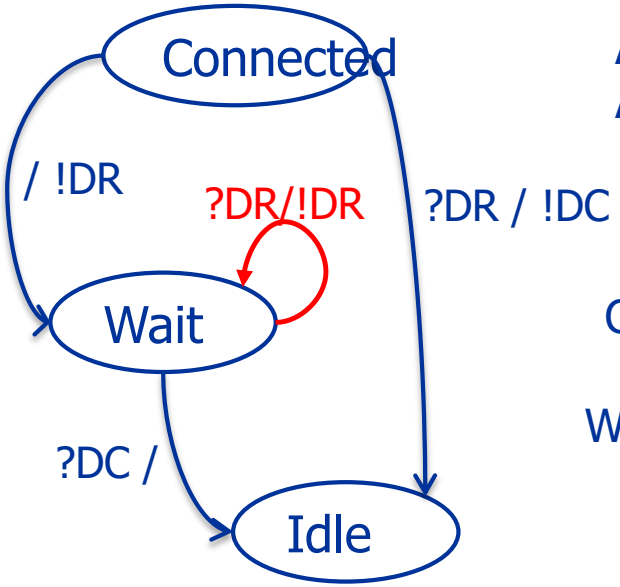


Comportement attendu



Réception non spécifiée

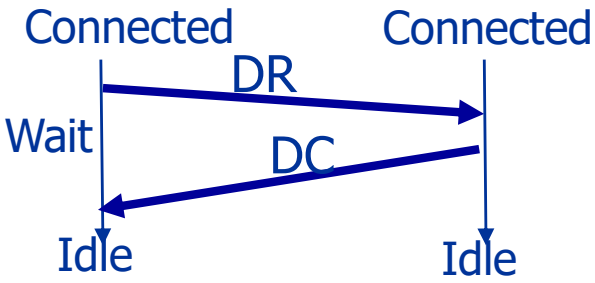
Exemple: 2ème tentative de solution

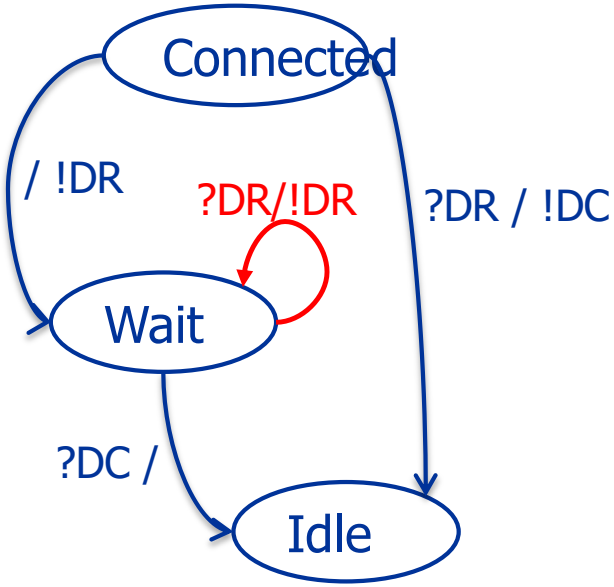


PDU

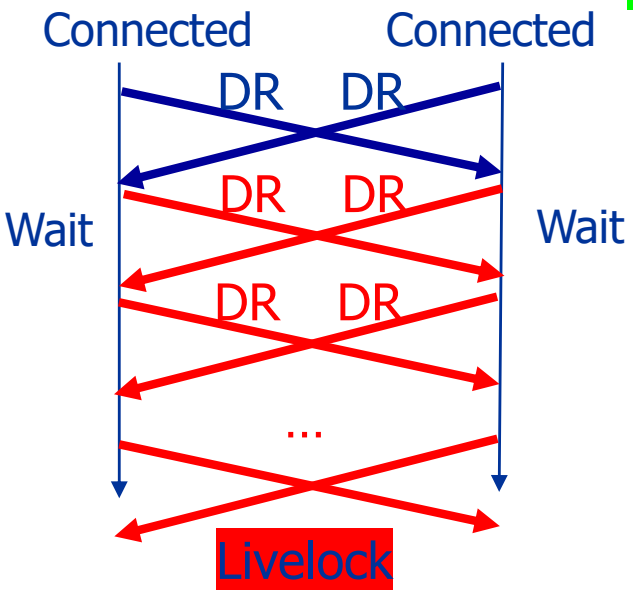
DR: Disconnect Request

DC: Disconnect Confirm





Comportement attendu



Livelock

Model-checking

- Construire le graphe complet des états possibles du système
  - Produits asynchrones ou synchrones d'automates (ou d'états de calcul d'un programme) *-> cf cours TL1*
- Vérifier des propriétés du système:
  - Atteignabilité d'états désirables / indésirables
  - Occurrence d'événements attendus/non-attendus
  - Respect de règles d'enchaînement (ex: un message émis sera toujours reçu)
  - *Cas particulier: vérification de propriétés de sécurité (confidentialité, légitimité…) pour des protocoles cryptographiques (cf TP avec outil spécialisé AVISPA)*

Preuve de protocoles

- Protocoles = algorithmes répartis
  - Difficiles à concevoir corrects
  - Si incorrect, difficile à détecter et à récupérer (N machines)
- Méthodes de vérification
  - Automates (étiquetés, temporisés etc)
  - Utilisation de logiques temporelles pour exprimer les propriétés: $\Box$(Send $\Rightarrow \Diamond$ Receive)
  - Algèbre linéaire pour calculer des invariants
  - Algèbres de processus (e.g. π-calcul) pour calculer des équivalences par réécriture de termes
  - Model-checking, simulation…

Anecdotes sur la vérification

- Joseph Sifakis, prix Turing français & grenoblois
  - 1987 - preuve de protocoles d'exclusion mutuelle répartie, de protocoles de consensus (Delta4) par model-checking, protocoles écrits en Estelle (automates étendus).
- R. Groz, Claude Jard, Claire Lassudrie, France Télécom 1983
  - Preuve manuelle en logique de Hoare d'un protocole d'exclusion mutuelle par vote majoritaire
  - Découverte d'une erreur vicieuse en cas de pannes multiples si N>=5
- 1985: RNIS, protocole LAPB monotrame
  - Quand un comité d'experts mondiaux réinvente le bit alternant (1969) en oubliant d'alterner le bit !
    - Le protocole proposé pour la norme était FAUX !

Bilan chapitre PR2: notions essentielles

- Protocoles = algorithmes parallèles/répartis
  - Assurer la cohérence globale à partir de calculs sur une information locale et partielle
  - Difficultés de conception
  - Importance de méthodes de vérification
- Mécanismes pour fiabiliser
  - Codes détecteurs (& correcteurs) d'erreurs
  - Retransmission, notion de fenêtre d'émission
  - Temporisation
  - Numérotation (de messages, de sites)
  - Redondance, Vote
- Formalisation et vérification de protocoles
  - *Modélisation graphique: automates*
  - Formalisation (TD IRC) et outils de vérification (TP sécurité pour protocoles cryptographiques)