

---

# Cours ASPD

## Protocoles de Communications

### Telecom Nancy 2A IL

---

Dominique Méry  
Telecom Nancy  
Université de Lorraine

---

Année universitaire 2024-2025  
2 janvier 2026(3:16pm)

- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP
- ⑥ Summary

\_\_\_\_\_

- 1

# Problèmes des communications

---

- La communication de données entre deux entités est en général non fiable
- Le support physique de communication peut perdre, dupliquer, réordonnancer ou détériorer les messages
- Un protocole de communication est une méthode permettant de communiquer des données, en veillant à détecter et à corriger les éventuelles erreurs de transmission
- Modèles d'architectures en couches : des couches basses du réseaux physiques aux couches les plus hautes réalisant les services.
- Modèles d'architecture : OSI et TCP/IP
- Modèles d'empilement des couches avec une relation de raffinement ou de simulation entre une couche supérieure et une couche inférieure : un service de niveau  $n$  est simulé par la couche de niveau  $n-1$

# Modèles de référence OSI

---

- OSI signifie *Open Systems Interconnection*
- Modèle de Référence proposé par l'organisation mondiale de normalisation ISO
- Ce modèle concerne la connexion entre systèmes ouverts à la communication avec d'autres systèmes ouverts
- Le modèle OSI a sept couches :
  - ▶ Une couche correspond à un nouveau d'abstraction pour les communications
  - ▶ Chaque couche possède et effectue des fonctions spécifiques
  - ▶ Les fonctions de chaque couche sont choisies en fonction de la définition de protocoles normalisés internationaux
  - ▶ Le choix des frontières entre les couches doit minimiser le flux d'informations aux interfaces
  - ▶ Le nombre de couches doit être rationnel et permettre de maîtriser l'architecture et d'éviter la cohabitation dans une même couche de fonctions très différentes.

- **Couche Physique** : elle gère la transmission des bits qui peuvent être altérés par des problèmes de transmission physique.
- **Couche Liaison de données** : elle gère les communications sous forme de trames de données et assure donc la communication entre deux entités, en visant à corriger les problèmes du niveau inférieur (gestion de trames en séquences et des trames d'acquittements) ; elle assure aussi une régulation des émetteurs.
- **Couche réseau** : elle gère le sous-réseau, en particulier elle gère les routes ; elle gère aussi les congestions de ce sous-réseau.
- **Couche transport** : elle assure le découpage des données de la couche session et ls passe à la couche réseau. Elle s'assure que les morceaux arrivent correctement au sens des couches supérieures.
- **Couche session** : elle permet d'établir des sessions par les utilisateurs et donc d'utiliser la couche transport : transfert de fichiers par exemple.
- **Couche présentation** : elle concerne la syntaxe et la sémantique de l'information transmise
- **Couche application** : elle compose les protocoles développés

- Couche hôte-réseau ou liens de données : connexion de l'hôte à la couche Internet via un protocole permettant d'envoyer des paquets IP ; driver du système d'exploitation et d'une carte d'interface de l'ordinateur aux réseaux.
- Couche Réseau ou Internet : elle permet l'acheminement de paquets dans n'importe quel réseau et dans n'importe quel ordre ; les questions de réacheminement sont réglées par les couches supérieures. Le format est celui du protocole IP.
- Couche transport : elle permet à des paires ou entités connectées deux à deux, de maintenir une conversation ou une communication ; deux protocoles ont été définis :
  - ▶ le protocole TCP qui assure l'acheminement fiable d'un flux d'octets à une autre entité et
  - ▶ le protocole UDP qui est un protocole non-fiable, sans connexion, pour les applications qui ne veulent pas de séquences ou de contrôles de flux.
- Couche Application : elle contient les protocoles de haut niveau comme FTP, TELNET ; SMTP, DNS, SNMP

- 1

- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP
- ⑥ Summary

- Modélisation par échange de messages :
  - ▶ des actions internes ou locales à un processus
  - ▶ des actions de communications : envoi ou réception
- Canaux de communication :
  - ▶ tout message envoyé est reçu fatalement
  - ▶ tout message envoyé est reçu fatalement mais pas dans l'ordre d'envoi
  - ▶ tout message peut être perdu
- Variables partagées
- communication synchrone ou asynchrone
- modélisation en  $TLA^+$  à partir d'une description des actions en Event-B.

## Choix de modélisation

- Donner le « quoi » : spécification de ce que fait le protocole

# Choix de modélisation

---

- Donner le « quoi » : spécification de ce que fait le protocole
  - ▶ envoi d'un message  $m$  par un processus  $P$  à un processus  $Q$

- Donner le « quoi » : spécification de ce que fait le protocole
  - ▶ envoi d'un message  $m$  par un processus  $P$  à un processus  $Q$
  - ▶ décomposition en plusieurs phases

- Donner le « quoi » : spécification de ce que fait le protocole
  - ▶ envoi d'un message  $m$  par un processus  $P$  à un processus  $Q$
  - ▶ décomposition en plusieurs phases
- Donner le « comment » : simulation du protocole par des événements et des phases des couches plus basses
- Modélisation par raffinement à partir du service attendu pour mettre en œuvre dans les couches plus basses ou concrètes

# Observation d'un système réparti

- $u_0 \xrightarrow{e_0} u_1 \xrightarrow{e_1} \dots \xrightarrow{e_{i-1}} u_i \xrightarrow{e_i} u_{i+1} \xrightarrow{e_{i+1}} \dots$
- $e_0$  ou  $e_1$  ou  $\dots$  ou  $e_{i-1}$  ou  $e_i$  ou  $e_{i+1}$  ou  $\dots$
- $e \in \{e_0, e_1, \dots, e_{i-1}, e_i, e_{i+1}, \dots\}$
- $e \in E$  :  $E$  est l'ensemble fini des actions ou des événements observés sur le système modifiant l'état courant.
- $u_0 \xrightarrow{g} \dots \xrightarrow{f} u \xrightarrow{e} u' \xrightarrow{g} \dots$
- Chaque événement modélise la transformation d'une liste de variables d'états appelées *frame* et notée  $u$  :

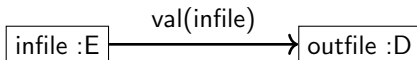
**if**  $cond(u)$  **then**  $u := f(u)$  **fi**

## Non-déterminisme et entrelacement

Les événements de  $E$  sont observés les uns à la suite des autres en veillant à ce qu'un événement est observé quand sa *garde* est vraie. On peut ajouter une hypothèse d'équité sur la trace produite.

# Modélisation du protocole de communication

- La valeur d'un fichier *infile* est transmise d'un émetteur  $E$  à un destinataire  $D$



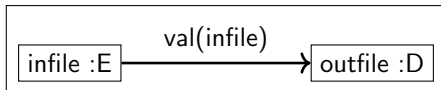
- Un service associé à un protocole de communication revient à effectuer une affectation du type :

`outfile := infile`

- Le nombre d'items ou d'enregistrements du fichier est  $n$  et ces données sont des éléments de DATA
  - ▶ DATA est un ensemble de données
  - ▶ *infile* est un fichier de longueur  $n$  :
    - $n \in \mathbb{N}_1$
    - $infile \in 1 .. n \rightarrow \text{DATA}$

# Modélisation du protocole

- Spécification du service attendu par les deux partenaires  $E$  et  $D$  : transmission du fichier *infile* de  $E$  via un canal de communication et réception dans un fichier *outfile* de  $D$ .



- *outfile* est une variable localisée en  $D$
- $inv1 : outfile \in 1 .. n \mapsto DATA$  : pendant la transmission, *outfile* contient une partie des données et n'a pas tout reçu.



## Service du protocole

- communication  $\stackrel{def}{=} \text{begin } outfile := infile \text{ end}$
- Initialement, le fichier *outfile* contient n'importe quoi.
- L'événement communication réalise le service attendu en un coup !

## Observation des communications

Le protocole de communication est construit selon des hypothèses de l'environnement : hostilité,

# Service du protocole

---

- communication  $\stackrel{def}{=} \text{begin } outfile := infile \text{ end}$
- Initialement, le fichier *outfile* contient n'importe quoi.
- L'événement communication réalise le service attendu en un coup !

## Observation des communications

Le protocole de communication est construit selon des hypothèses de l'environnement : hostilité, fautes,

## Service du protocole

- communication  $\stackrel{def}{=} \text{begin } outfile := infile \text{ end}$
- Initialement, le fichier *outfile* contient n'importe quoi.
- L'événement communication réalise le service attendu en un coup !

## Observation des communications

Le protocole de communication est construit selon des hypothèses de l'environnement : hostilité, fautes, pertes,

# Service du protocole

---

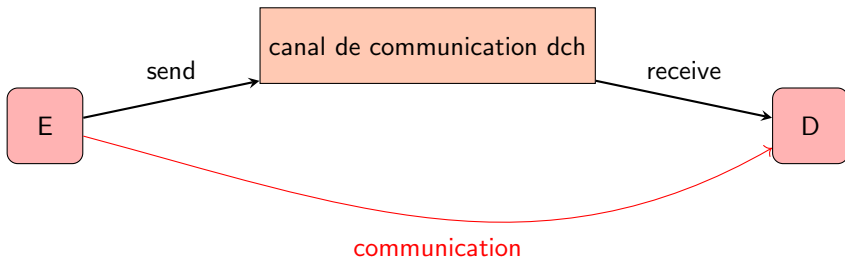
- communication  $\stackrel{def}{=} \text{begin } outfile := infile \text{ end}$
- Initialement, le fichier *outfile* contient n'importe quoi.
- L'événement communication réalise le service attendu en un coup !

## Observation des communications

Le protocole de communication est construit selon des hypothèses de l'environnement : hostilité, fautes, pertes, mensonges ...

- 1 Communications entre processus
- 2 **Modélisation de protocoles de communication**
  - Service d'un protocole
  - Protocole monodirectionnel fiable**
  - Protocole fiable avec réordonnancement possible
- 3 Protocole de Stenning
- 4 Protocole ABP
- 5 Protocole Sliding Window SWP
- 6 Summary

- Un émetteur envoie des trames à un récepteur
- La communication a lieu dans un seul sens
- Le canal de communication est parfait
- Le récepteur reçoit toutes les données transmises
- L'émetteur envoie les données le plus rapidement possible
- Une trame contient les informations suivantes : type (indicateur de données ou non), séquence (numéro de trame), ack (ack éventuel), info (information transportée ou octet)
- Événements :
  - ▶ `SENDING(trame,dest)` via la couche physique
  - ▶ `RECEIVING(trame)` via la couche physique



- $E$  envoie à  $D$  par le canal de communication modélisé par la variable  $dch$
- `sendingdata` dépose la valeur  $infile(s)$  dans le canal de communication
- `receivingdata` récupère la valeur suivante se trouvant dans le canal  $dch$ .
- On conserve l'ordre d'envoi de type fifo

- *outfile* est localisée sur *D* et reçoit les valeurs de *E* via le canal.
- *r* et *s* sont deux indicis de contrôle pour gérer en type fifo
- *dch* modélise le canal de communication.

$inv1 : outfile \in 1 .. n \rightarrow DATA$

$inv2 : r \in 0 .. n$

$inv3 : s \in 1 .. n+1$

$inv4 : r \leq s$

$inv5 : dch \in 1 .. n \rightarrow DATA$

$inv6 : outfile = 1 .. r \triangleleft infile$

$inv7 : dch \subseteq 1 .. s-1 \triangleleft infile$

$inv8 : outfile \subseteq dch$

- (inv6) : *outfile* contient la copie du fichier *infile* entre 1 et *r*.
- (inv4) : le curseur de réception *r* est plus petit que le curseur d'envoi *s*.
- (inv8) : les données reçues sont des copies de données transmises dans *dch*
- (inv7) : le canal *dch* ne transmet que des valeurs du fichier *infile* dans l'ordre fifo avec comme borne courante *s*

- INITIALISATION  $\stackrel{def}{=}$   $\left( \begin{array}{l} outfile := \emptyset \\ r := 0 \\ s := 1 \\ dch := \emptyset \end{array} \right)$
- transmissionover  $\stackrel{def}{=}$  if  $r = n$  then skip end
- sendingdata  $\stackrel{def}{=}$  if  $s \leq n$  then  $\left[ \begin{array}{l} dch(s) := infile(s) \\ s := s+1 \end{array} \right]$  end
- receivingdata  $\stackrel{def}{=}$   
if  $r+1 \in dom(dch)$  then  $\left[ \begin{array}{l} outfile(r+1) := dch(r+1) \\ r := r+1 \end{array} \right]$

# Protocole FIFO-FIABLE

---

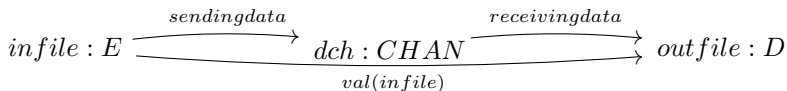
- $E :: \text{if } s \leq n \text{ then } dch(s), s := infile(s), s+1 \text{ fi}$
- $D :: \text{if } r+1 \in dom(dch) \text{ then } outfile(r+1), r := dch(r+1), r+1 \text{ fi}$

k

- $E :: \text{if } s \leq n \text{ then } \left\{ \begin{array}{l} \text{send } infile(s) \text{ to } D \text{ in } dch \\ s := s+1 \end{array} \right. \text{ fi}$
- $D :: \text{if } r+1 \in dom(dch) \text{ then } \left\{ \begin{array}{l} \text{receive } dch(r+1) \text{ in } ioutfile(r+1) \\ r := r+1 \end{array} \right. \text{ fi p}$

# Sommaire du protocole

- $E :: \text{if } s \leq n \text{ then } \left\{ \begin{array}{l} \text{send } infile(s) \text{ to } D \text{ in } dch \\ s := s+1 \end{array} \right. \text{ fi}$
- $D :: \text{if } r+1 \in dom(dch) \text{ then } \left\{ \begin{array}{l} \text{receive } dch(r+1) \text{ in } outfile(r+1) \\ r := r+1 \end{array} \right. \text{ fi}$



- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP
- ⑥ Summary

## fiable avec réordonnancement possible

- variables *outfile*, *s*, *dch*
- invariants

$inv1 : s \in 1 .. n+1$

$inv2 : dch \in 1 .. n \rightarrow DATA$

$inv4 : dom(dch) \cap dom(outfile) = \emptyset$

$inv3 : dch \cup outfile = 1 .. (s-1) \triangleleft infile$

- INITIALISATION  $\stackrel{def}{=} \left[ \begin{array}{l} outfile := \emptyset \\ s := 1 \\ dch := \emptyset \end{array} \right]$
- communication2  $\stackrel{def}{=} \text{if } \left[ \begin{array}{l} s = n+1 \\ dch = \emptyset \end{array} \right] \text{ then skip end}$
- sendingdata  $\stackrel{def}{=} \text{if } [s \leq n] \text{ then } \left[ \begin{array}{l} dch(s) := infile(s) \\ s := s+1 \end{array} \right] \text{ end}$
- receivingdata  $\stackrel{def}{=} \text{if } \left[ \begin{array}{l} r \in dom(dch) \\ m \in DATA \\ dch(r) = m \\ r \notin dom(outfile) \end{array} \right] \text{ then } \left[ \begin{array}{l} outfile := outfile \cup \{r \mapsto m\} \\ dch := dch \setminus \{r \mapsto m\} \end{array} \right] \text{ end}$

- Hypothèse de fiabilité excessive
- Le canal de communication peut ne pas protéger les données transmises



Prise en compte des pertes de messages.

# Section Courante

---

- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP

- ⑥ Summary

## Protocole fiable avec attente

- Le protocole envoie des trames mais attend que le récepteur lui signale la réception
- Le canal de communication est toujours supposé fiable mais le flux est contrôlé
- Événements :
  - ▶ `SENDING(trame,dest)` via la couche physique
  - ▶ `WAITING(trameack)` via la couche physique
  - ▶ `RECEIVING(trame)` via la couche physique
  - ▶ `SENDING(trameack,eme)` via la couche physique

# Protocole fiable avec attente

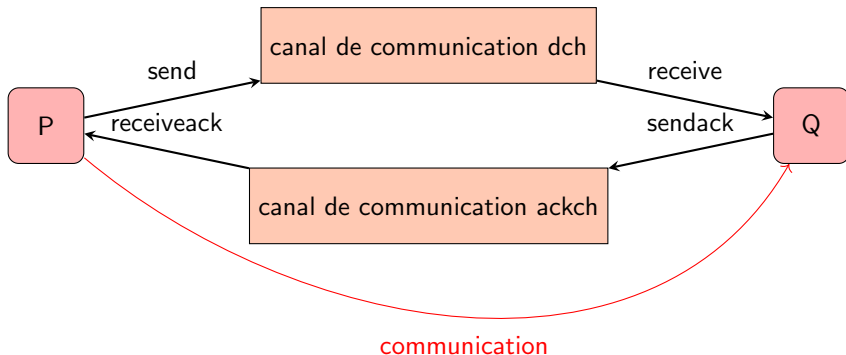
---

- Le protocole envoie des trames mais attend que le récepteur lui signale la réception
- Le canal de communication est toujours supposé fiable mais le flux est contrôlé
- Événements :
  - ▶ `SENDING(trame,dest)` via la couche physique
  - ▶ `WAITING(trameack)` via la couche physique
  - ▶ `RECEIVING(trame)` via la couche physique
  - ▶ `SENDING(trameack,eme)` via la couche physique
- Hypothèse de fiabilité irréaliste

# Protocole avec acquittement et retransmission

- Le protocole précédent est non fiable dans la mesure où la couche physique est peut-être non fiable ou bruitée
- L'idée est de contrôler si la trame envoyée est bien reçue sinon on renvoie mais le récepteur doit être capable de savoir qu'il s'agit d'une trame dupliquée.
- Événements :
  - ▶ `SENDING(trame,dest)` via la couche physique avec un numéro de trame
  - ▶ `WAITING(trameack)` via la couche physique du numéro de trame et ré-émission de la trame si nécessaire
  - ▶ `RECEIVING_ACK` via le canal physique de la trame ack avec le numéro
  - ▶ `RECEIVING(trame)` via la couche physique
  - ▶ `SENDING(trameack,eme)` via la couche physique avec le numéro de trame reçue

# Protocole de Stenning



- Soient deux entités P et Q : P veut envoyer des messages à Q
- P envoie une suite de données  $d_1, \dots, d_n$  à Q
- P répète les deux opérations :
  - ▶ phase d'envoi : P envoie la donnée  $(d_i, i)$  à Q
  - ▶ phase d'attente : P attend de recevoir la valeur  $i$
  - ▶ phase de confirmation : si  $i$  est reçue, alors le protocole reprend à la phase d'envoi

# Protocole de Stenning

- VARIABLES  $dch, s, r, ackch, outfile$

$inv1 : r \in 0 .. n$

$inv2 : s \in 1 .. n+1$

$inv3 : dch \in 1 .. n \rightarrow DATA$

$inv4 : ackch \subseteq 1 .. n$

$inv5 : r \leq s$

$inv6 : s \leq r+1$

$inv7 : outfile = 1 .. r \triangleleft infile$

$inv8 : dch \subseteq 1 .. s \triangleleft infile$

$inv9 : s \leq n+1$

$inv10 : r \leq n$

$inv11 : outfile \in 1 .. n \rightarrow DATA$

$inv12 : s \notin dom(outfile) \Rightarrow s = r+1$

$inv13 : s \in dom(outfile) \Rightarrow s = r$

$inv14 : s \in ackch \Rightarrow s \in dom(outfile)$

$inv15 : ackch \subseteq 1 .. r$

## THEOREMS

$th : r = n \Rightarrow outfile = infile \text{ safety property}$

# Protocole de Stenning

- INITIALISATION  $\stackrel{def}{=}$  
$$\left[ \begin{array}{l} dch := \emptyset \\ ackch := \emptyset \\ s := 1 \\ r := 0 \\ outfile := \emptyset \end{array} \right]$$
- servicedone  $\stackrel{def}{=}$  if  $[ r = n ]$  then  $[ \text{skip} ]$  end
- sendingadata  $\stackrel{def}{=}$  if  $[ s \leq n ]$  then  $[ dch(s) := infile(s) ]$  end
- receivingadata  $\stackrel{def}{=}$   
if  $[ r+1 \in dom(dch) ]$  then 
$$\left[ \begin{array}{l} outfile(r+1) := dch(r+1) \\ r := r+1 \end{array} \right]$$
 end
- sendingack  $\stackrel{def}{=}$  if  $[ r \neq 0 ]$  then  $[ ackch := ackch \cup \{r\} ]$  end
- receivingack  $\stackrel{def}{=}$  if  $[ s \in ackch ]$  then  $[ s := s+1 ]$  end

# Protocole de Stenning

---

- $\text{daemondch} \stackrel{def}{=} \text{if } \left[ \begin{array}{l} i \in 1 .. n \\ m \in DATA \\ i \mapsto m \in dch \end{array} \right] \text{ then } [ dch := dch \setminus \{i \mapsto m\} ] \text{ end}$
- $\text{daemonack} \stackrel{def}{=} \text{if } [ i \in \text{ackchackch} := \text{ackch} \setminus \{i\} ] \text{ then } [] \text{ end}$
- Les deux événements introduisent des erreurs du type perte de messages
- Le modèle permet donc de décrire à la fois le protocole et l'environnement.

# Section Courante

---

- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP

- ⑥ Summary  
]

# Protocole ABP (Alternating Bit Protocol)

---

- Le protocole ABP (Alternating Bit Protocol) est un protocole de communication qui vise à fournir un canal de communication fiable sur un canal peu fiable.
- ABP est une version simplifiée du protocole TCP (Transmission Control Protocol) utilisé sur Internet.
- ABP est une évolution du protocole de Stenning.

# Protocole bidirectionnel à fenêtre de taille 1

---

- Ce protocole est aussi appelé le protocole du *bit alterné*
- Pour régler les problèmes de codage des numéros de trame, on observe que l'on a besoin uniquement d'un bit pour contrôler si la trame est effectivement répétée et reçue in fine.

# Section Courante

---

- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP

- ⑥ Summary  
]

# Protocole bidirectionnel à fenêtre de taille $n$

---

- Ce protocole est aussi appelé le protocole du *sliding window protocol*
- Le protocole précédent peut être amélioré par une fenêtre de longueur  $n \geq 1$  et cette fenêtre glisse en suivant les trames reçues.

**CONTEXT** *DATA*  
**SETS**

*D*

**CONSTANTS**

*n, IN, l*

**AXIOMS**

*axm1 :  $n \in \mathbb{N}_1$*

*axm2 :  $IN \in \mathbb{N} \rightarrow D$*

*axm3 :  $\text{dom}(IN) = 0 \dots n$*

*axm4 :  $l \in \mathbb{N}_1$*

*axm5 :  $l \leq n$*

**END**

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \rightarrow D$

$inv2 : i \in 0 .. n+1$

$inv3 : 0 .. i-1 \subseteq dom(OUT)$

$inv3bis : dom(OUT) \subseteq 0 .. n$

$inv7 : chan \in \mathbb{N} \rightarrow D$

$inv8 : ack \subseteq \mathbb{N}$

$inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$

$inv10 : got \subseteq \mathbb{N}$

$inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$

$inv13 : got \subseteq dom(OUT)$

$inv14 : ack \subseteq dom(OUT)$

$inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$

$inv17 : chan \subseteq IN$

$inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \leftrightarrow D$

$inv2 : i \in 0 .. n+1$

$inv3 : 0 .. i-1 \subseteq dom(OUT)$

$inv3bis : dom(OUT) \subseteq 0 .. n$

$inv7 : chan \in \mathbb{N} \leftrightarrow D$

$inv8 : ack \subseteq \mathbb{N}$

$inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$

$inv10 : got \subseteq \mathbb{N}$

$inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$

$inv13 : got \subseteq dom(OUT)$

$inv14 : ack \subseteq dom(OUT)$

$inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$

$inv17 : chan \subseteq IN$

$inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \rightarrow D$

$inv2 : i \in 0 .. n+1$

$inv3 : 0 .. i-1 \subseteq dom(OUT)$

$inv3bis : dom(OUT) \subseteq 0 .. n$

$inv7 : chan \in \mathbb{N} \rightarrow D$

$inv8 : ack \subseteq \mathbb{N}$

$inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$

$inv10 : got \subseteq \mathbb{N}$

$inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$

$inv13 : got \subseteq dom(OUT)$

$inv14 : ack \subseteq dom(OUT)$

$inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$

$inv17 : chan \subseteq IN$

$inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre
- Réception progressive des données au cours du glissement

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \leftrightarrow D$

$inv2 : i \in 0 .. n+1$

$inv3 : 0 .. i-1 \subseteq dom(OUT)$

$inv3bis : dom(OUT) \subseteq 0 .. n$

$inv7 : chan \in \mathbb{N} \leftrightarrow D$

$inv8 : ack \subseteq \mathbb{N}$

$inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$

$inv10 : got \subseteq \mathbb{N}$

$inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$

$inv13 : got \subseteq dom(OUT)$

$inv14 : ack \subseteq dom(OUT)$

$inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$

$inv17 : chan \subseteq IN$

$inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre
- Réception progressive des données au cours du glissement
- Deux canaux d'échanges :

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \rightarrow D$   
 $inv2 : i \in 0 .. n+1$   
 $inv3 : 0 .. i-1 \subseteq dom(OUT)$   
 $inv3bis : dom(OUT) \subseteq 0 .. n$   
 $inv7 : chan \in \mathbb{N} \rightarrow D$   
 $inv8 : ack \subseteq \mathbb{N}$   
 $inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$   
 $inv10 : got \subseteq \mathbb{N}$   
 $inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$   
 $inv13 : got \subseteq dom(OUT)$   
 $inv14 : ack \subseteq dom(OUT)$   
 $inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$   
 $inv17 : chan \subseteq IN$   
 $inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre
- Réception progressive des données au cours du glissement
- Deux canaux d'échanges :
  - ▶  $chan$  est le canal de communication des données à partir de  $IN$  vers  $OUT$

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \leftrightarrow D$   
 $inv2 : i \in 0 .. n+1$   
 $inv3 : 0 .. i-1 \subseteq dom(OUT)$   
 $inv3bis : dom(OUT) \subseteq 0 .. n$   
 $inv7 : chan \in \mathbb{N} \leftrightarrow D$   
 $inv8 : ack \subseteq \mathbb{N}$   
 $inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$   
 $inv10 : got \subseteq \mathbb{N}$   
 $inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$   
 $inv13 : got \subseteq dom(OUT)$   
 $inv14 : ack \subseteq dom(OUT)$   
 $inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$   
 $inv17 : chan \subseteq IN$   
 $inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre
- Réception progressive des données au cours du glissement
- Deux canaux d'échanges :
  - ▶  $chan$  est le canal de communication des données à partir de  $IN$  vers  $OUT$
  - ▶  $ack$  est le canal de retour.

# PROCESSUS Sliding-Window Protocol

**VARIABLES** :  $OUT$ ,  $i$ ,  $chan$ ,  $ack$ ,  $got$

$inv1 : OUT \in \mathbb{N} \leftrightarrow D$   
 $inv2 : i \in 0 .. n+1$   
 $inv3 : 0 .. i-1 \subseteq dom(OUT)$   
 $inv3bis : dom(OUT) \subseteq 0 .. n$   
 $inv7 : chan \in \mathbb{N} \leftrightarrow D$   
 $inv8 : ack \subseteq \mathbb{N}$   
 $inv9 : ack \cup got \subseteq i .. i+l \cap 0 .. n$   
 $inv10 : got \subseteq \mathbb{N}$   
 $inv12 : dom(chan) \subseteq 0 .. i+l \cap 0 .. n$   
 $inv13 : got \subseteq dom(OUT)$   
 $inv14 : ack \subseteq dom(OUT)$   
 $inv16 : 0 .. i-1 \triangleleft OUT = 0 .. i-1 \triangleleft IN$   
 $inv17 : chan \subseteq IN$   
 $inv18 : OUT \subseteq IN \wedge got \subseteq dom(chan)$

- Idée du protocole : faire glisser une fenêtre
- Réception progressive des données au cours du glissement
- Deux canaux d'échanges :
  - ▶  $chan$  est le canal de communication des données à partir de  $IN$  vers  $OUT$
  - ▶  $ack$  est le canal de retour.

# Phases du protocole

- Initialisation du protocole

INITIALISATION  $\stackrel{def}{=}$   $\left[ \begin{array}{l} OUT := \emptyset \\ i := 0 \\ chan := \emptyset \\ ack := \emptyset \\ got := \emptyset \end{array} \right]$

- Phase d'envoi et de réception

send  $\stackrel{def}{=}$  if  $\left[ \begin{array}{l} j \in i .. i+l \\ j \leq n \\ j \notin got \end{array} \right]$  then  $[ chan(j) := IN(j) ]$  end

receive  $\stackrel{def}{=}$   
if  $\left[ \begin{array}{l} j \in dom(chan) \\ j \in i .. i+l \end{array} \right]$  then  $\left[ \begin{array}{l} OUT(j) := chan(j) \\ ack := ack \cup \{j\} \end{array} \right]$  end

# Phases du protocole

- Phase d'accusé de réception et de complétion

receiveack  $\stackrel{def}{=}$  if  $[k \in ack]$  then  $\left[ \begin{array}{l} got := got \cup \{k\} \\ ack := ack \setminus \{k\} \end{array} \right]$  end

completion  $\stackrel{def}{=}$  if  $[i = n+1 \wedge got = \emptyset]$  then  $[skip]$  end

- Gestion de la fenêtre

sliding  $\stackrel{def}{=}$   
if  $\left[ \begin{array}{l} grd1 : got \neq \emptyset \\ grd3 : i \in got \\ grd4 : i+l < n \end{array} \right]$  then  $\left[ \begin{array}{l} act1 : i := i+1 \\ act2 : got := got \setminus \{i\} \\ act3 : ack := ack \setminus \{i\} \end{array} \right]$  end

La fenêtre ne glisse plus quand elle ne peut plus mais elle se vide et fond en quelque sorte ( $i+l \geq n$ ).

emptywindow  $\stackrel{def}{=}$   
if  $\left[ \begin{array}{l} grd1 : got \neq \emptyset \\ grd2 : i \in got \\ grd3 : i+l \geq n \\ grd4 : i \leq n \end{array} \right]$  then  $\left[ \begin{array}{l} act1 : i := i+1 \\ act2 : got := got \setminus \{i\} \\ act3 : ack := ack \setminus \{i\} \end{array} \right]$  end

# Pertes de messages sur les deux canaux

$\text{loosingchan} \stackrel{\text{def}}{=}$   
if  $\left[ \begin{array}{l} \text{grd1} : j \in i .. i+l \\ \text{grd2} : j \in \text{dom}(\text{chan}) \\ \text{grd3} : j \notin \text{got} \end{array} \right]$  then  $\left[ \text{act1} : \text{chan} := \{j\} \triangleleft \text{chan} \right]$  end  
 $\text{loosingack} \stackrel{\text{def}}{=}$   
if  $\left[ \text{grd1} : k \in \text{ack} \right]$  then  $\left[ \text{act1} : \text{ack} := \text{ack} \setminus \{k\} \right]$  end

# Le protocole du bit alterné en TLA<sup>+</sup>

- Le protocole du bit alterné est une instance de ce protocole pour  $l = 0$ .
- Le choix de  $l$  est important puisqu'il intervient dans le codage du témoin de transmission.
- On peut imaginer que les choix suivants sont pertinents :
  - ▶  $l = 0$  : codage sur 0 bits
  - ▶  $l = 2$  : codage sur 1 bits
  - ▶  $l = 4$  : codage sur 2 bits
  - ▶  $l = 2^k$  : codage sur  $k$  bits
- Dans le cas d'un codage sur  $k$  bits, on choisit ce qui reste sur la trame d'envoi comme place.

# Section Courante

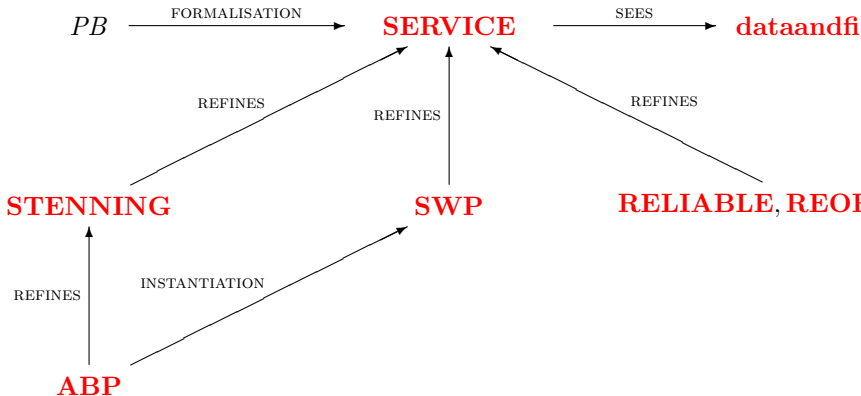
---

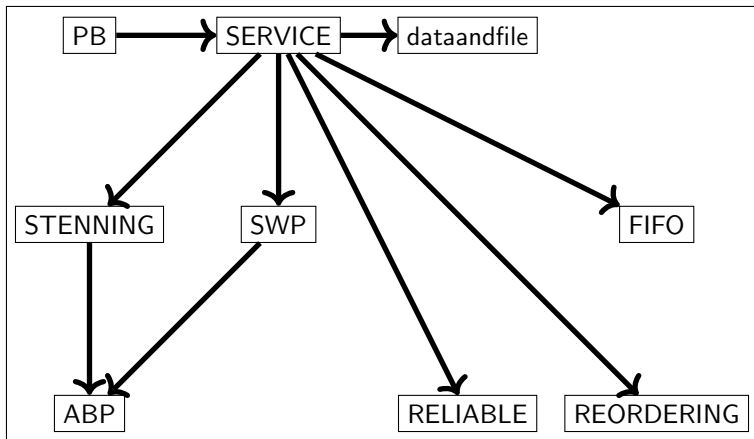
- ① Communications entre processus
- ② Modélisation de protocoles de communication
  - Service d'un protocole
  - Protocole monodirectionnel fiable
  - Protocole fiable avec réordonnancement possible
- ③ Protocole de Stenning
- ④ Protocole ABP
- ⑤ Protocole Sliding Window SWP

## ⑥ Summary

]

# Status of development





# Conclusion

---

- Modèles en couches
- Relation d'abstraction
- Mécanismes de répétition et de contrôle du
- Exemple de TCP/IP

# Transactions

---

- Un protocole de communication est une suite de transactions modifiant des variables d'états.
-

# Transactions

---

- Un protocole de communication est une suite de transactions modifiant des variables d'états.
- 
- Initialisation du protocole

$$\text{INITIALISATION} \stackrel{def}{=} \left[ \begin{array}{l} OUT := \emptyset \\ i := 0 \\ chan := \emptyset \\ ack := \emptyset \\ got := \emptyset \end{array} \right]$$

# Transactions

- Un protocole de communication est une suite de transactions modifiant des variables d'états.

- 

- Initialisation du protocole

$$\text{INITIALISATION} \stackrel{def}{=} \left[ \begin{array}{l} OUT := \emptyset \\ i := 0 \\ chan := \emptyset \\ ack := \emptyset \\ got := \emptyset \end{array} \right]$$

- Phase d'envoi et de réception

$$\text{send} \stackrel{def}{=} \text{if } \left[ \begin{array}{l} j \in i .. i+l \\ j \leq n \\ j \notin got \end{array} \right] \text{ then } [ chan(j) := IN(j) ] \text{ end}$$

# Transactions

- Un protocole de communication est une suite de transactions modifiant des variables d'états.

- 

- Initialisation du protocole

$$\text{INITIALISATION} \stackrel{\text{def}}{=} \left[ \begin{array}{l} OUT := \emptyset \\ i := 0 \\ chan := \emptyset \\ ack := \emptyset \\ got := \emptyset \end{array} \right]$$

- Phase d'envoi et de réception

$$\text{send} \stackrel{\text{def}}{=} \text{if} \left[ \begin{array}{l} j \in i .. i+l \\ j \leq n \\ j \notin got \end{array} \right] \text{ then } [ chan(j) := IN(j) ] \text{ end}$$

$$\text{receive} \stackrel{\text{def}}{=} \text{if} \left[ \begin{array}{l} j \in dom(chan) \\ j \in i .. i+l \end{array} \right] \text{ then } \left[ \begin{array}{l} OUT(j) := chan(j) \\ ack := ack \cup \{j\} \end{array} \right] \text{ end}$$

- Un protocole de communication est une suite de transactions qui peuvent être définies comme des actions ou événements.
- Le service d'un protocole de communication est de communiquer des données dans de bonnes conditions (pas de pertes, pas de disparition de messages, pas d'altération des messages, ...)
- Exemple du formalisme Alice & Bob qui couvre les protocoles cryptographiques.