Chapitre 4

Notions de protocole

4.1. Introduction

Nous avons défini la notion de protocole comme un ensemble de règles établies entre des entités homologues (en général deux) pour améliorer les propriétés du service de communication. Dans ce chapitre, nous allons regarder les principaux mécanismes protocolaires utilisés, et ce quelle que soit la couche considérée. Nous décrirons un protocole et un protocole de communication simple sera réalisé à la fin du chapitre.

4.1.1. *Protocole chef cuisinier - gâte-sauce*

Un protocole est un ensemble de règles. Dans le chapitre 2.1.3.1, nous avions décrit de manière informelle mais somme toute assez précise le protocole chef cuisinier-gâte-sauce. Il faut formaliser les actions et les objets (informations) utilisés pour mettre en œuvre ce protocole. Les deux entités homologues sont le chef cuisinier et le gâte-sauce. Le prestataire de services est la Poste. Les interactions entre prestataire de services et les deux utilisateurs que sont le gâte-sauce et le chef cuisinier sont :

— chef cuisinier lettre.requête poster une lettre
— chef cuisinier lettre.indication réception d'une lettre

La figure 4.1. rappelle l'architecture en couche mise en œuvre dans cet exemple. Cette figure fait apparaître pour chaque entité du service cuisine un contexte. En effet, pour réaliser la recette, une association est établie entre le chef cuisinier et le gâte-sauce. Cette association établit un contexte de communication. Ce contexte de communication contient les structures de données nécessaires à la gestion des échanges protocolaires. Dans cet exemple, le contexte est composé des informations suivantes :

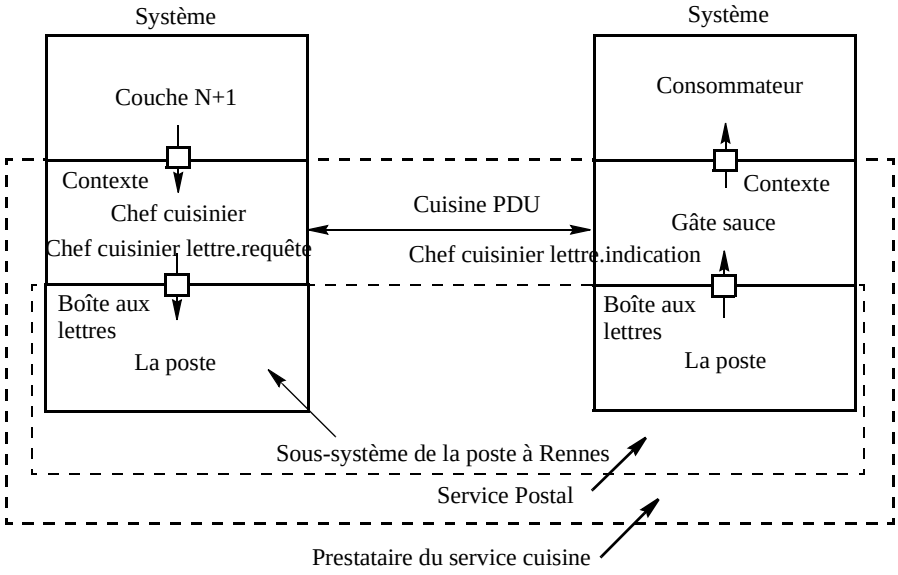

Figure 4.1. *Détail des couches dans le modèle ISO/OSI*

— chef cuisinier :
  • adresse du gâte-sauce
  • N_s = le numéro de la dernière lettre envoyée
  • N_l le nombre de lettres à envoyer
  • liste des lettres à envoyer
  • un état « Envoi d'ordre » : dans cet exemple, le chef cuisinier reste toujours dans cet état, ce qui n'est pas le cas général.

— gâte-sauce :
  • N_l nombre de lettres attendues
  • N_r numéro de lettre attendue
  • place pour stocker les lettres à recevoir
  • liste des lettres reçues.
  • un état qui est soit « Attente ordre de travail », soit « Au travail » (il réalise une recette). Nous verrons plus loin à quoi servent ces états.

Les actions du chef cuisinier sont les suivantes quand il décide d'envoyer un ordre d'exécution de recette.

a / fragmenter la recette en autant de lettres que nécessaire et construire une liste ordonnée des lettres à envoyer,

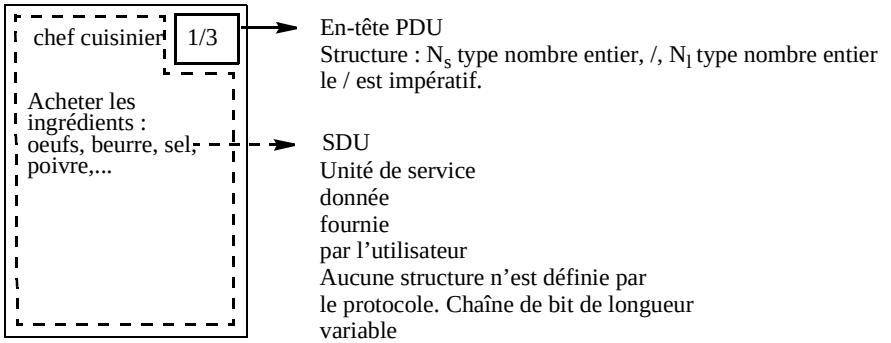

Figure 4.2. *Format de la Cuisine PDU*

b / compter ces lettres et ranger la valeur dans N_l,

c / faire pour N_s =1 à N_l

retirer la première lettre de la liste ;

fabriquer la Cuisine PDU : ajouter le texte N_s/N_l en haut à droite sur la lettre ;

chef cuisinier lettre.requête (adresse du gâte-sauce, lettre) /* poster la lettre */

fin pour

On notera, par la suite, Lettre.N_s et Lettre.N_l, les deux valeurs du champs N_s/N_l de la lettre (la Cuisine PDU). L'adresse ne fait pas partie des informations de service du protocole cuisinier, mais des informations de service du protocole de la Poste. L'adresse est donc passée en paramètre de la primitive et non ajoutée par le protocole cuisinier sur la lettre. On notera que dans ce protocole le chef cuisinier peut faire toutes ses actions en séquence sans avoir à attendre de réaction du cuisinier. On rappelle que l'exemple 2.1.3.1 ne traite que les déséquencements.

4.1.2. *Format de la Cuisine PDU*

L'algorithme nécessite l'insertion dans le message transmis, le PDU, du texte N_s/N_l. On appelle format d'un message, d'une trame ou d'un PDU la structure de donnée décrivant les informations (enveloppes) ajoutées par le protocole (cf figure 4.2.). Le protocole décrit :

— la place de ses enveloppes : en début, en fin, au milieu, en plusieurs morceaux,

— la structure de ses enveloppes,

— le type et la taille de chaque information, souvent même l'ordre dans lequel les bits doivent être transmis,

— la sémantique des informations qu'elle contient.

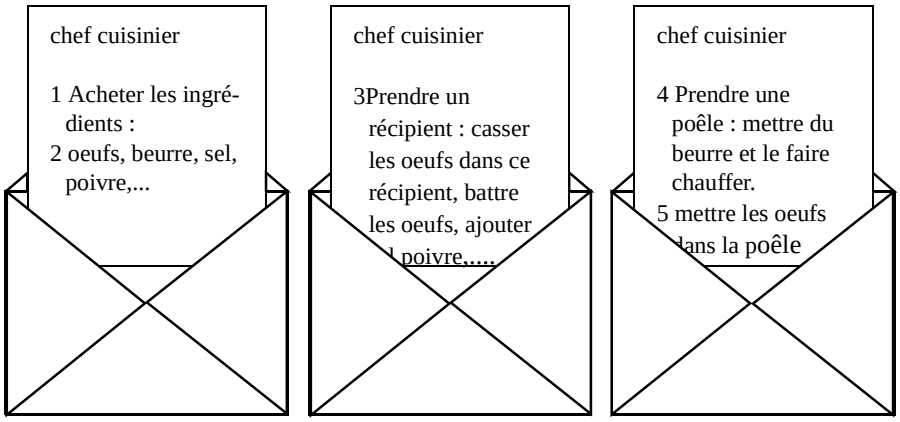

Figure 4.3. *Le protocole numérote les ligne du texte et non les lettres*

La précision de ces descriptions est essentielle. N'oubliez pas que les entités homologues vont être développées par des programmeurs différents. La moindre erreur d'interprétation de ces structures de données fera échouer le protocole. La place du « / », dans cet exemple, est dans un protocole de la plus haute importance et son absence ou sa présence inopinée risque de faire échouer le protocole. Que va interpréter le gâte-sauce s'il trouve 2//3 au lieu de 2/3 ? Un ordinateur, lui, se trompera à coup sûr.

Par contre, le protocole ne définit que très peu de chose quant à la partie données utilisateur. Il s'agit de la copie de la SDU fournie par la couche supérieure. Le protocole définit sa place relativement aux enveloppes dans la suite d'octets qui composent le PDU. La plupart des protocoles imposent une taille maximale aux données utilisateur. Dans notre exemple, cette taille maximale est en fait le poids accepté pour une lettre ordinaire par la Poste. C'est donc une contrainte qui provient de la couche inférieure.

D'autres choix de données de protocole pourraient aboutir à réaliser un service équivalent, mais il s'agira alors d'un autre protocole. La figure 4.3. montre un autre choix : les phrases du texte complet sont numérotées. L'information de service ici est donc le numéro de ligne.

4.1.3. *Description algorithmique du protocole*

Les actions du gâte-sauce sont décrites sur la figure 4.5. Au départ il attend un ordre. Un processus dans un état attend un événement. Un événement dans cet exemple peut uniquement être une indication d'arrivée de message. On notera que les seules informations utiles pour le protocole sont le numéro de la lettre, Lettre.$N_{s,}$ et le nombre total de lettres émises par le chef cuisinier, Lettre.N_l. La seule difficulté de

l'algorithme est dans la reprise des lettres stockées précédemment. Le gâte-sauce n'a pas besoin des informations d'adresse : si la lettre lui est parvenue, il s'agit forcément d'un ordre à exécuter.

Le protocole ci-dessous souffre des défauts évoqués au paragraphe 2.1.4. Prouver les propriétés d'un protocole ne relève pas de sa description. Il faudrait utiliser pour cela un langage formel de description de protocoles tel que Estelle ou Lotos.

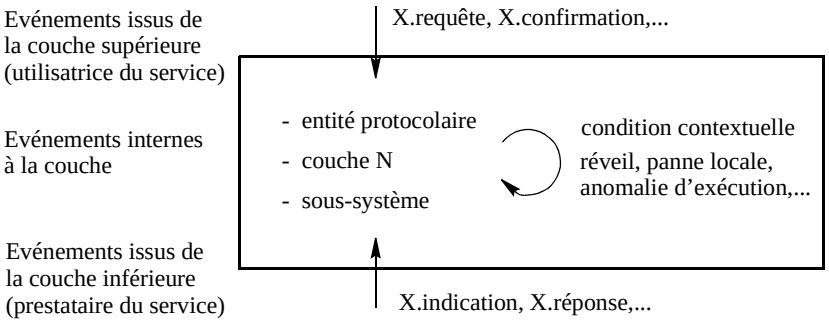

Figure 4.4. *Origine des événements capables de solliciter une entité protocolaire*

Notion d'événement

Une entité protocolaire, autrement dit un sous-système ou encore une couche (figure 4.4.), est sollicitée par des événements qui sont :

— externes : les primitives d'interaction entre couches. Celles-ci sont issues des couches adjacentes, donc soit de la couche immédiatement supérieure, soit de la couche immédiatement inférieure ;

— internes : le résultat de processus internes à l'automate ou au traitement fait par celui-ci. Ce sont essentiellement les sonneries de réveil.

4.1.4. *Description sous forme d'automate d'états finis*

On remarque que la description du protocole fait intervenir des états et des événements. La description algorithmique d'un protocole est souvent fastidieuse et imprécise et on lui préfère une description sous forme d'automate d'états finis. La figure 4.6. décrit cet automate sous forme de tableau. Chaque ligne contient un état, et chaque colonne un événement, généralement l'arrivée d'un message (indication) ou d'une requête. Dans chaque case ainsi définie, on note les actions entreprises et l'état suivant. Cet automate est décrit par un sextuplet <Etat de départ, Evénement, Etat d'arrivée, Actions vers la couche inférieure, Actions vers la couche supérieure, Actions internes>. Seuls les trois premiers éléments sont nécessaires à la description de l'automate. Les trois derniers champs peuvent être vides. Ils servent à décrire le comportement du protocole.

	— Attente ordre de travail :	Etat
	faire	
	attendre : chef cuisinier lettre.indication	Attend une lettre
d /	$N_r = 1$;	Initialisations
e /	N_l= Lettre.N_l	
f /	Réserver la place pour recevoir N_l lettres.	
g /	Cas : Lettre.N_s == N_r	Traitement de la lettre. Lettre en séquence
	exécuter le contenu de cette lettre ;	
	incrémenter N_r ;	
	fin cas	
	Cas : Lettre.N_s # N_r	Lettre hors séquence conservée
	ranger la lettre	
	fin cas	
	Autre cas : ne rien faire ;	
h /	Etat = “Travail”.	
	Autre cas : rester dans l’état “Attente ordre de travail”	Nouvel état
	fin faire	
	— Travail :	Etat
	faire	
	attendre : Chef cuisinier lettre.indication	Attend une lettre
i /	Cas : Lettre.N_s == N_r	Lettre en séquence
	exécuter le contenu de cette lettre ;	
	incrémenter N_r ;	
	tant que Première_Liste_Lettres_recues.N_s == N_r faire	On lit les lettres reçues précédemment hors séquence
	retirer cette lettre de la liste et	
	exécuter son contenu ;	
	incrémenter N_r ;	
	finfaire	
	fin cas	Lettre hors séquence conservée
	Cas : Lettre.N_s # N_r	
	ranger la lettre dans liste lettres reçues	
	dans l’ordre des Lettre.N_s croissant ;	
	fin cas	
	Autre cas : ne rien faire ;	Tout la recette à été reçue
j /	Cas : N_l == N_r	
	Etat “Attente ordre de travail”.	
	fin cas	
	Autre cas: ne rien faire;	
	fin faire	

Figure 4.5. *Algorithme du gâte-sauce*

Evénements / Etats	Lettre.indication
Attente ordre de travail Etat suivant	Action d Figure 4.5. Travail
Travail Etat suivant	Action f Figure 4.5. Si N_l == N_r alors Attente ordre de travail sinon Travail

Figure 4.6. *Description de l'automate sous forme de tableau*

— Etat courant : c'est l'état où est arrivé le protocole ; le protocole attend dans cet état un événement.

— Evénement : il déclenche l'exécution des actions associées.

— Etat suivant : après l'exécution des actions associées, l'automate passe dans cet état.

— Action vers la couche inférieure (envoi d'un PDU).

— Action vers la couche supérieure (envoi d'une primitive d'indication ou de confirmation).

— Actions internes (armer un réveil, modifier une variable du contexte...).

La notation sous forme de tableau est compacte et peut contenir un grand nombre de lignes et de colonnes. Chaque case peut être aisément documentée quant aux actions à entreprendre. Par contre, elle n'est pas vraiment parlante. Aussi utilise-t-on souvent, en complément, une notation sous forme de graphe état transition (cf. figure 4.7.) ou de réseaux de Petri.

Cette représentation en tableau garantit que l'on n'oublie aucun cas puisque tous les états et tous les événements sont énumérés. Il est fréquent que la plupart des cases correspondent à des situations état-transition impossibles. Aussi la case correspondante est-elle vide, pour indiquer qu'il n'y a ni action ni changement d'état. Il reste à démontrer que la situation est réellement impossible.

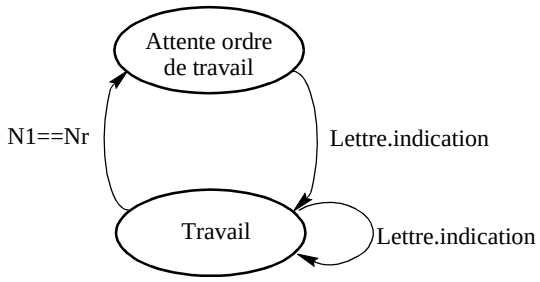

Figure 4.7. *Automate sous la forme états-transitions*

Cet exemple ne fait pas apparaître d'émission de messages, mais, dans la plupart des protocoles réseau, c'est le cas. Comme l'action d'émettre un message vers l'entité homologue est particulièrement importante, on la met en exergue dans la liste des actions.

4.1.5. *Description sous forme état-transition*

Cette description graphique permet de visualiser plus aisément qu'avec un tableau la position relative d'un automate. Par contre, il est moins aisé de décrire les actions entreprises sur un événement. La figure 4.7. décrit notre automate. Les cercles dénotent des états, avec leur nom au milieu. Les conditions de transition sont écrites à côté. On note par une flèche partant de l'état d'origine vers l'état terminal les transitions d'état. Une flèche partant d'un état et revenant au même état dénote que l'événement associé fait l'objet d'un traitement particulier, mais ne provoque pas de changement d'état. Un événement qui n'apparaît sur aucune flèche en sortie d'un état ne provoque aucun traitement ni changement d'état.

Pour chaque flèche (arc orienté dans le vocabulaire de la théorie des automates), il y a un texte ou algorithme qui décrit les actions à faire. En général, on associe la représentation graphique à la représentation sous forme de tableau.

4.2. Protocole de détection des erreurs

Nous avons vu que toute voie de transmission est sujette à erreur. Il n'est pas acceptable que la modification d'un ou de plusieurs bits pendant le transfert d'un message ne soit pas détectée. Imaginez les conséquences d'un message de transfert de fonds contenant un ordre de virement de 1 000,00 F, dont le contenu est modifié par transformation de la virgule en un chiffre quelconque, 9 par exemple. Cela conduirait à un transfert de 1 000 900 F, ce qui est tout à fait différent. S'il est impossible de supprimer totalement l'occurrence d'une erreur sur la voie de transmission, par contre il est possible de réduire presque totalement le risque de délivrer un message erroné.

Les taux d'erreurs, P_v, sur des lignes physiques, sont typiquement de l'ordre de : 10^{-5} pour une liaison téléphonique, 10^{-7} à 10^{-8} pour un câble coaxial, 10^{-10} à 10^{-12} pour une fibre optique. L'objet du protocole de détection d'erreurs est de fournir une probabilité encore plus faible de délivrer un message erroné (modifié sans que l'on s'en aperçoive). Il ne s'agit pas de corriger les erreurs. Nous verrons ces protocoles aux paragraphes suivants.

Les protocoles de détection sont tous basés sur l'ajout d'une information de contrôle aux données transmises. La donnée de contrôle est calculée par l'entité émettrice qui l'ajoute dans son PDU. Cette donnée de contrôle est vérifiée par l'entité homologue réceptrice, par exemple en faisant le même calcul que l'entité émettrice, et en comparant les résultats. L'entité réceptrice accepte le PDU si le résultat de la comparaison est correct. Sinon, le PDU est mis à la poubelle. Aucune information contenue dans ce PDU n'est crédible. Il n'y a en général aucun moyen de localiser l'erreur dans le PDU[1]. L'algorithme de l'entité réceptrice est donc très simple : acceptation de tout le PDU ou abandon (destruction) de tout le PDU.

Le protocole de détection remplace donc la modification de quelques bits par une perte pure et simple du PDU. Le PDU ne sera ni interprété plus avant ni a fortiori délivré à une autre couche. Il est radicalement perdu. Malgré tout, ces protocoles ne sont pas parfaits et l'on appelle taux d'erreur non détecté, P_{nd}, la probabilité pour qu'une information modifiée soit remise malgré le protocole. En général, $P_{nd} << P_v$.

Voyons quelques exemples de protocoles de détection d'erreur couramment rencontrés dans les réseaux.

4.2.1. *Parité*

Les messages sont découpés en blocs de i bits (i = 7 ou 8). Pour chaque bloc, on calcule le nombre de bit à un (ou à zéro) et l'on ajoute un $i + 1^{\text{-ième}}$ bit de telle sorte que parmi les i + 1 bits il y ait toujours un nombre pair de bits à 1 (ou impair). Pour une parité paire, s'il y avait 5 bits à 1 parmi les i bit, le $i + 1^{\text{-ième}}$ bit vaudrait 1. Par contre, s'il y avait 4 bits à 1, le $i + 1^{\text{-ième}}$ bit vaudrait 0.

P_{nd} pour les bits de parité est assez médiocre. En effet, si un nombre impair de bits est modifié, le protocole détecte l'erreur. Par contre, si un nombre pair est modifié, le protocole ne verra rien.

Or, malheureusement, la probabilité d'erreur n'est pas uniformément répartie. Sur une voie physique, un parasite a de grandes chances de modifier un groupe de bits consécutifs et non un seul bit. La définition d'un modèle d'erreur est complexe et nécessite une analyse détaillée de la structure de la voie. Dans une mémoire d'ordinateur au contraire, les bits peuvent être abîmés individuellement de manière apparemment aléatoire pour les messages.

---

1. On verra toutefois dans les exercices proposés des protocoles de détection localisation et correction éventuelle des erreurs.

Le rendement du protocole d'insertion de parité est i/ i+1 ; si i est petit (7 ou 8), le rendement est médiocre.

4.2.2. *Codes à redondance cyclique*

Les codes à redondance cyclique, CRC, sont aussi appelés Frame Check Sequence, FCS. Ils possèdent trois propriétés essentielles :

— Simplicité du codage et décodage,

— Capacité à détecter efficacement aussi bien des erreurs indépendantes ou se présentant par trains (suites de bits),

— Faible sensibilité à la taille du bloc de données pour offrir un rendement raisonnable.

L'information est prise comme un entier (polynôme). Appelons I(x) la suite de bits qui constitue le message à fiabiliser. Comme pour le bit de parité, le PDU produit sera l'ajout à I(x) d'une information appelée CRC, code à redondance cyclique, de taille fixe. Cette information est souvent concaténée à la fin : PDU = I(x) - CRC.

Le CRC est produit en divisant I(x) par une constante, généralement un nombre premier, appelée polynôme générateur, de taille déterminée (4, 8, 16 ou 32 bits), appelée « ordre du polynôme ». Le reste de cette division constitue le CRC. L'intérêt de cette solution est que la division d'un entier de taille variable par une constante, en conservant uniquement le reste, se réalise de manière très simple et efficace.

Intuitivement, si vous divisez un entier, I(x), par une constante, C, et que vous en retirez le reste $(I(x) - [I(x)/C]_{reste})$, alors le résultat est un multiple de C. En arithmétique binaire sans report, l'addition et la soustraction donnent le même résultat.

Soit le polynôme générateur x^3+x+1, donc polynôme d'ordre 3, correspondant à l'entier ou séquence binaire 1011 ($1^3 0^2 1^1 1^0$). Le CRC est calculé en effectuant la division modulo 2 du message multiplié par x^3 (décalage de 3 bits à gauche et remplacement par des 0 - ou ajout de 3 bits 0 à droite du message). Le reste obtenu (sur 3 bits) est retiré (en pratique ajouté à la fin du message). La figure 4.8. montre divers exemples de calcul du CRC.

La probabilité que la modification de plusieurs bits donne le même reste est très faible. Pour de plus amples détails, nous renvoyons le lecteur à des ouvrages de mathématiques qui décrivent les codes linéaires et cycliques.

La taille de I(x) n'a pas d'influence sur le reste de la division ni sur la probabilité que des erreurs multiples donnent le même reste. Néanmoins, cette probabilité n'est pas nulle. Un message peut donc arriver alors que des erreurs s'y sont introduites et n'ont pas été détectées. La probabilité pour que des erreurs multiples donnent le même reste est d'autant plus faible que l'ordre du polynôme est grand.

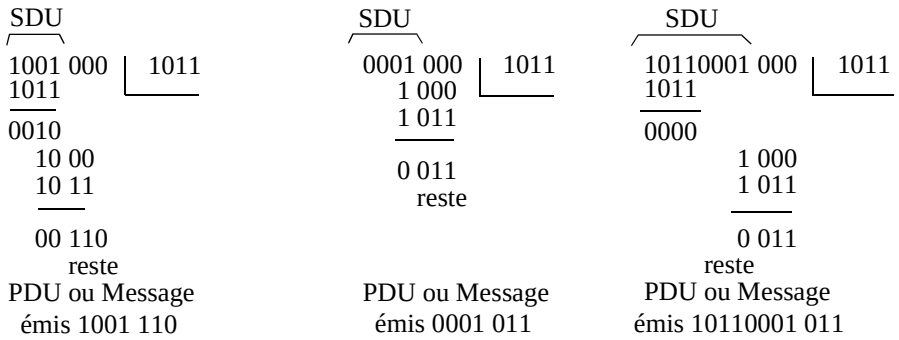

Figure 4.8. *Exemple de calcul du CRC avec le polynôme générateur x^3+x+1 pour deux messages de 4 bits et un message de 8 bits*

4.3. Principe des protocoles de correction des erreurs en réseau

Les PDU (trames, messages, paquets) peuvent se perdre ou arriver erronés. Dans le cas d'un PDU détecté « erroné », aucun champ de ce PDU ne peut être utilisé puisque l'on ne sait pas où est l'erreur. Si le PDU provient d'une voie raccordée à plusieurs sources (réseau à diffusion - radio ou réseau local), on ne peut même pas être sûr de la provenance (source-origine) du PDU. Il doit être jeté. Si le PDU provient d'une voie point à point (reliant directement la source au récepteur), alors le fait de recevoir un PDU erroné informe sur une tentative d'émission du partenaire situé au bout de la voie. Il est alors possible de lui signaler par un PDU particulier, que nous appellerons accusé de réception négatif NACK, la mauvaise réception de son PDU. Mais le cas d'une voie point à point est un cas particulier, et nous en ferons abstraction dans la suite de cet exposé. D'autre part, le PDU-NACK doit être transmis. Il est donc lui aussi sujet au risque de perte ou d'altération. Il n'est pas possible de garantir que ce PDU-NACK arrivera à coup sûr.

Nous allons d'abord prendre un protocole très simple appelé « Send and Wait », envoi et attente, pour illustrer les problèmes rencontrés dans la conception et la mise au point des protocoles. Nous proposons d'étudier en exercice un autre protocole simple, le protocole du bit alterné, qui est une variante de ce protocole.

4.3.1. *Protocole « Send and Wait »*

Soit un réseau général capable d'émettre des messages à un destinataire parmi un ensemble d'abonnés. Imaginons un réseau dont les propriétés du service sont similaires à celui de la Poste, décrites au paragraphe 2.1.1. Les propriétés du service de communication offert à une entité sont importantes pour définir et choisir le protocole. La figure 4.9. montre deux abonnés A et B dont les entités (processus) Send and Wait (S & W) vont mettre en œuvre ce protocole. Chaque entité peut émettre et

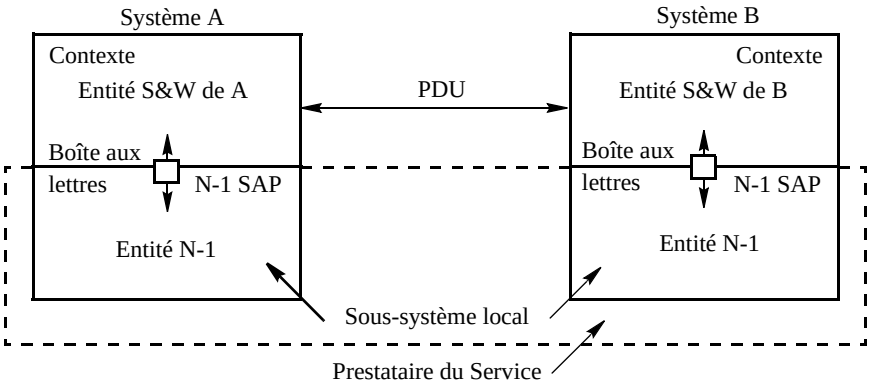

Figure 4.9. *Architecture de mise en œuvre du protocole « Send and Wait »*

recevoir des PDU par l'intermédiaire de la boîte aux lettres, locale au sous-système, notée N-1 SAP (cf. figure 4.9.).

Pour illustrer le protocole, nous allons utiliser un diagramme d'échanges temporel (cf. figure 4.10.). Les entités, S & W de A et S & W de B, sont placées respectivement à gauche et à droite du prestataire de services dont les entités ne sont pas représentées. Les flèches obliques indiquent les PDU échangées avec leur nom. Dans notre schéma, l'épaisseur du trait de la flèche simule la durée d'émission, et la pente de celle-ci le délai d'acheminement (éventuellement réduit au délai de propagation plus la durée d'émission si le prestataire de services se résume à une voie physique). Les diagrammes temporels suivants ne représenteront plus ces délais que l'on doit toutefois conserver à l'esprit.

4.3.1.1. *Protocoles sans fautes*

L'entité S & W attend une donnée à transmettre. Quand elle en a une, elle prépare son émission :

— construction du S & W PDU de données, noté PDU-DATA. L'enveloppe contient au moins le type DATA du PDU ;

— mise à jour des variables de son contexte, en particulier l'état du protocole.

Puis elle émet ce PDU-DATA à l'aide de la primitive de service envoi-request (PDU-DATA). L'adresse de B est ici omise car le protocole fonctionne entre A et B exclusivement. L'entité S & W de A se met en attente d'un acquittement provenant de B. L'entité S & W de A ne prendra aucun nouveau bloc de données à transmettre tant qu'elle n'aura pas reçu cet acquittement. L'automate de la figure 4.11. décrit le comportement de cette entité.

Cet automate a deux états, « Attente nouvelle donnée » et « Attend ACK ». L'événement correspondant à une requête de transmission de la couche supérieure, événement généré par la primitive « envoi.request », fait passer l'automate de

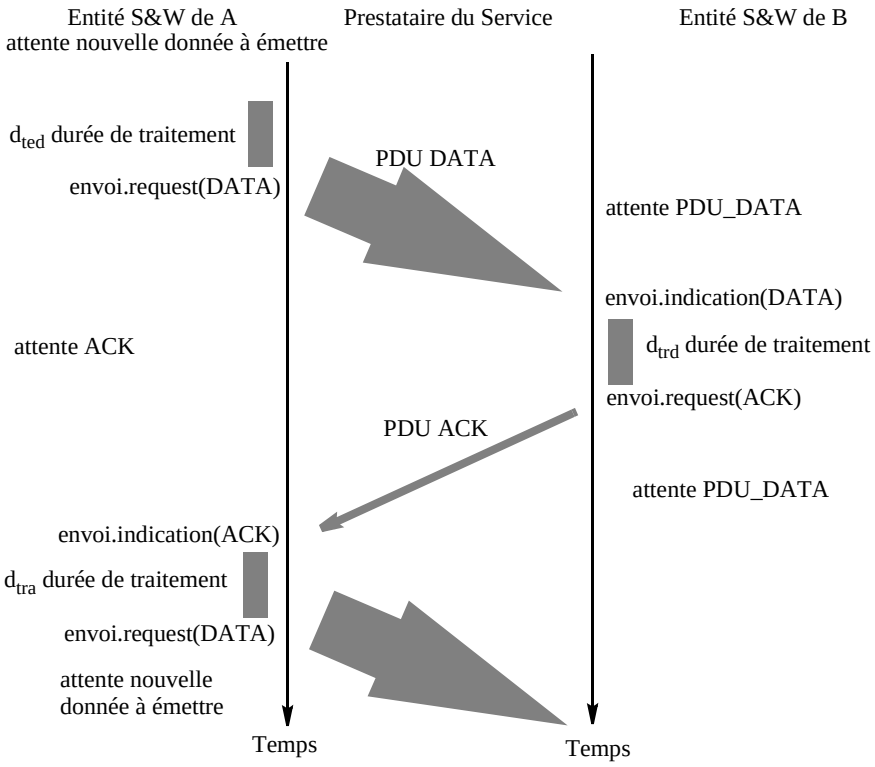

Figure 4.10. *Diagramme d'échange du protocole « Send and Wait »*

l'état « Attente nouvelle donnée » à l'état « Attend ACK ». L'événement N-1 envoi.indication, issu de la couche inférieure (N-1) porteur d'un PDU-ACK, provoque la transition inverse. Quand l'automate est dans l'état « Attente nouvelle donnée », seul l'événement envoi.indication fait changer d'état l'automate. Quand l'automate est dans l'état « Attend ACK », seul l'événement PDU-ACK fait changer d'état l'automate.

On notera que le temps pris par le processeur qui supporte l'entité S & W pour exécuter ces travaux n'est pas nul. Nous avons noté d_{ted} la durée de traitement de ce PDU-DATA par l'entité S & W de A. Cette durée dépend fortement de la puissance de traitement du processeur sur lequel l'entité est mise en œuvre et de l'efficacité des mécanismes de gestion des interruptions du système d'exploitation utilisé. Il y a lieu de prendre en considération d_{ted} lors de l'évaluation du délai de transit.

L'entité S & W de B est, elle, toujours en attente d'un PDU-DATA. Lorsqu'elle reçoit un PDU-DATA (sans erreur) :

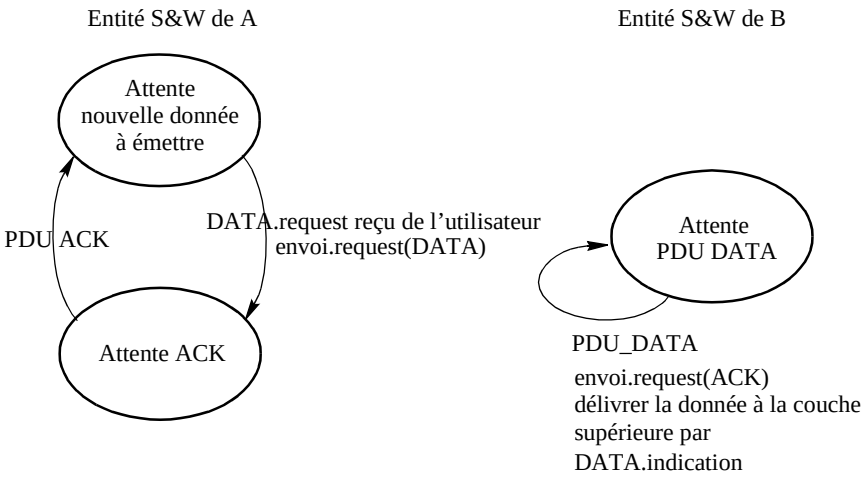

Figure 4.11. *Automate sous la forme états-transition du protocole Send and Wait*

— elle délivre la donnée à l'utilisateur destinataire dans la boîte à lettres qui sert d'interface par une primitive d'indication. Ces éléments ne sont pas notés sur les figures 4.10. et 4.11.

— elle prépare et envoie un PDU-ACK. L'envoi est fait par la primitive envoi-request (PDU-ACK). En effet, pour le prestataire de service, le PDU-ACK est une donnée ordinaire. Il n'y a donc pas lieu de prévoir une primitive spécifique pour envoyer des acquittements. Par contre, l'enveloppe du PDU contient le type ACK.

— elle met à jour son contexte.

La figure 4.11. montre l'automate de l'entité S & W de B réceptrice. Cet automate est très simple, il ne comporte qu'un état et traite un seul événement. Notre protocole construit une voie unidirectionnelle de A vers B. L'utilisateur de l'entité S & W de B n'a pas la possibilité de transmettre des données.

On notera que le prestataire de service doit offrir une voie de communication bidirectionnelle pour que les entités S & W en A et B puissent émettre et recevoir. Réaliser un service bidirectionnel dans notre entité implique, comme indiqué dans le paragraphe 2.3.1.6, de mettre en œuvre deux voies unidirectionnelles. C'est-à-dire que chaque entité mettra en œuvre les deux automates décrits sur la figure 4.11. Ces deux automates fonctionnent de manière indépendante l'un de l'autre, éventuellement en parallèle, ce qui réalise le schéma du paragraphe 2.3.1.6.

Le temps pris par le processus qui supporte l'entité S & W de B pour exécuter ses travaux est d_{trd}, la durée du traitement de réception du PDU-DATA. Ce temps n'est pas nul et dépend fortement de la réalisation (processeur, qualité du programme, délai de prise en compte et de gestion des interruptions par le système d'exploitation…). Nous avons aussi noté d_{tra}, durée du traitement de réception de l'acquittement pour l'entité S & W de A. Ces différents temps interviennent dans le calcul des délais

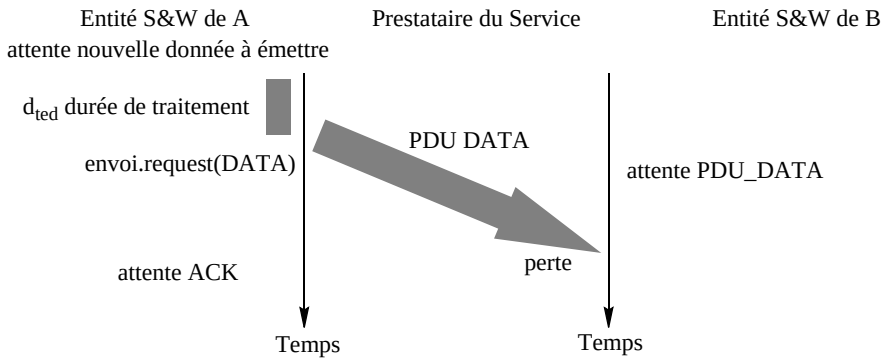

Figure 4.12. *Diagramme d'échange du protocole « Send and Wait » avec perte du PDU-DATA*

d'acheminement (transmission unidirectionnelle) et d'aller-retour du service fourni par des entités S & W.

4.3.1.2. *Protocole « Send and Wait » avec les fautes*

Les erreurs possibles sont :

— perte d'un message par le prestataire de service. On peut souligner que l'altération du message est équivalent à une perte. Les deux messages utilisés ici sont le PDU-DATA et le PDU-ACK ;

— panne de l'entité A ou B. Par panne, nous admettrons la cessation de fonctionnement définitive (stable) de l'entité (coupure de courant, plantage système…).

On notera que la panne, au sens ci-dessus défini, du prestataire de services n'est pas prise en compte ici car elle est équivalente à la perte de tous les messages émis et donc se ramène au cas précédent. La panne, transitoire ou permanente, d'une entité peut être aussi considérée comme la perte d'un ou de plusieurs messages.

Le diagramme temporel de la figure 4.12. montre l'état de blocage atteint par notre protocole en cas de perte du PDU-DATA. Les deux automates attendent un événement qui n'arrivera jamais.

Il n'y a aucune raison pour que seules les PDU-DATA se perdent (ou soient victimes d'altérations). La figure 4.13. montre l'état de blocage atteint par notre protocole en cas de perte du PDU-ACK. L'automate de S & W de B a délivré le bloc de donnée et attend le suivant. L'automate S & W de A attend l'ACK émis par B qui s'est perdu.

Afin de nous protéger contre les pertes, nous allons introduire un mécanisme de surveillance. Si aucune perte ne se produit, le délai d'aller et retour, noté d_{ar}, ou délai au bout duquel le PDU-ACK doit arriver, est composé de (cf. figure 4.10.) :

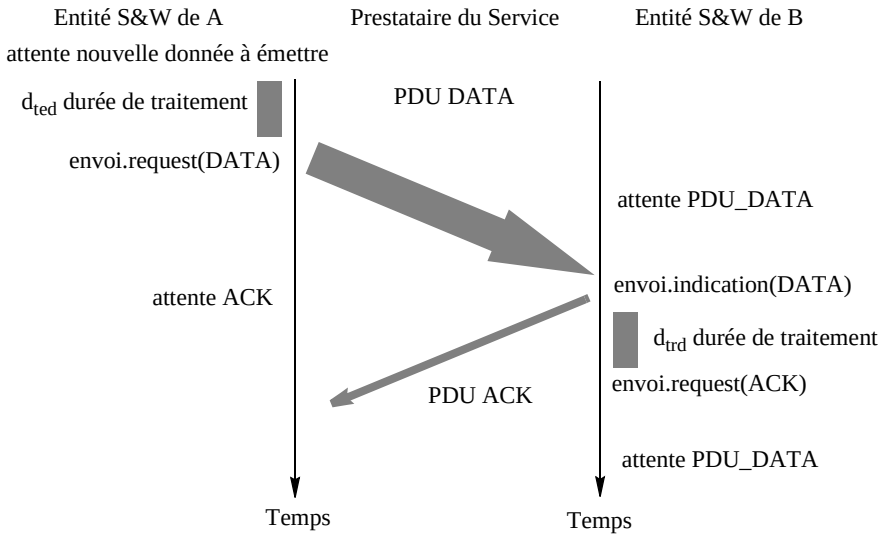

Figure 4.13. *Diagramme d'échange du protocole « Send and Wait » en cas de perte du PDU-ACK*

— durée d'émission PDU-DATA,
— délai de transit du PDU-DATA,
— d_{trd}, délai de traitement dans l'automate S & W de B,
— délai de transit du PDU-ACK,

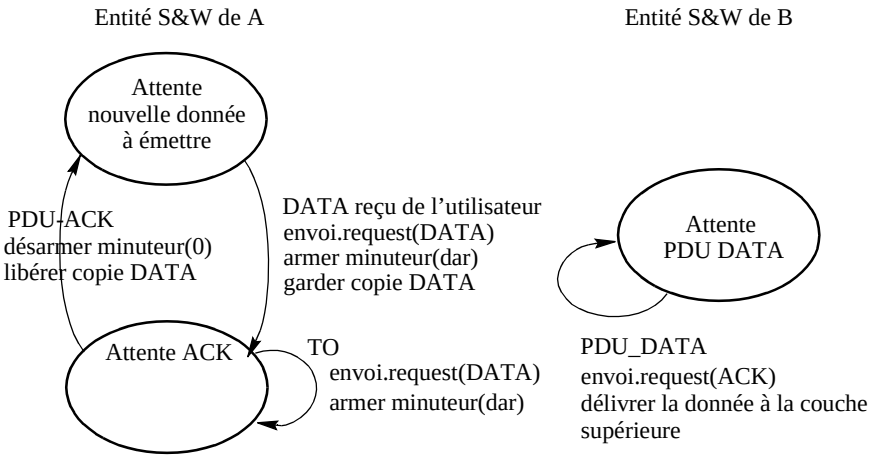

Figure 4.14. *Automate sous la forme états-transitions du protocole « Send and Wait » avec pertes*

Si aucun message n'est reçu au bout du délai d'aller et retour d_{ar}, l'un des deux PDU, DATA ou ACK, a été perdu. Il n'est pas possible pour l'entité S & W de A d'en savoir plus. Elle ne peut en aucun cas savoir ce qu'a fait l'entité homologue S & W en B sans échange de message. Ce qu'on voit sur les diagrammes temporels est une vue abstraite inaccessible aux entités réelles.

Nous allons surveiller les pertes à l'aide d'un minuteur (réveil ou time out). Lorsque l'automate S & W émet un PDU, il arme un minuteur avec la durée d_{ar}. Si le minuteur sonne, le délai d_{ar} est écoulé. L'automate S & W en déduit que soit la donnée, soit l'ACK s'est perdu. Il réémet la dernière donnée émise. Pour cela, il faut qu'il en ait conservé une copie. Le contexte de ce protocole contient pour l'émetteur une zone mémoire pour garder une copie de la dernière PDU émise.

Lorsque le PDU-ACK arrive, l'automate doit désarmer le minuteur afin qu'il ne sonne pas hors de propos. La figure 4.14. décrit l'automate modifié. La figure 4.15. décrit l'automate émetteur complet avec les actions.

Evénements Etats	Donnée	PDU-ACK	TO
Attente nouvelle Donnée Etat suivant	Emettre PDU-DATA Armer minuteur(d_{ar}) Garder copie DATA Attente ACK	Non pris en compte Attente nouvelle Donnée	Non pris en compte Attente nouvelle Donnée
Attente ACK Etat suivant	Non pris en compte Attente ACK	Désarmer minuteur Libérer copie DATA Attente nouvelle Donnée	Emettre PDU-DATA Armer minuteur(d_{ar}) Attente ACK

Figure 4.15. *Description de l'automate émetteur (entité S&W de A) sous forme de tableau*

4.3.1.3. *Les doubles*

Ce protocole utilise le principe des répétitions pour corriger les pertes de messages. Il nous crée un nouveau type d'erreur. En effet, la figure 4.16. et l'étude des automates décrits sur les figures 4.14.et 4.15. montrent clairement que, en cas de perte du PDU-ACK, deux données identiques seront délivrées à l'utilisateur final. La remise d'un double est aussi une erreur. Imaginez le transfert d'un programme pour lequel certaines instructions auraient été dupliquées par le protocole de transfert : votre programme ne fonctionnera plus. Pour se protéger contre les doubles, il faut introduire un mécanisme supplémentaire. Un mécanisme simple, dit du « bit alterné », est proposé en exercice 4.3..

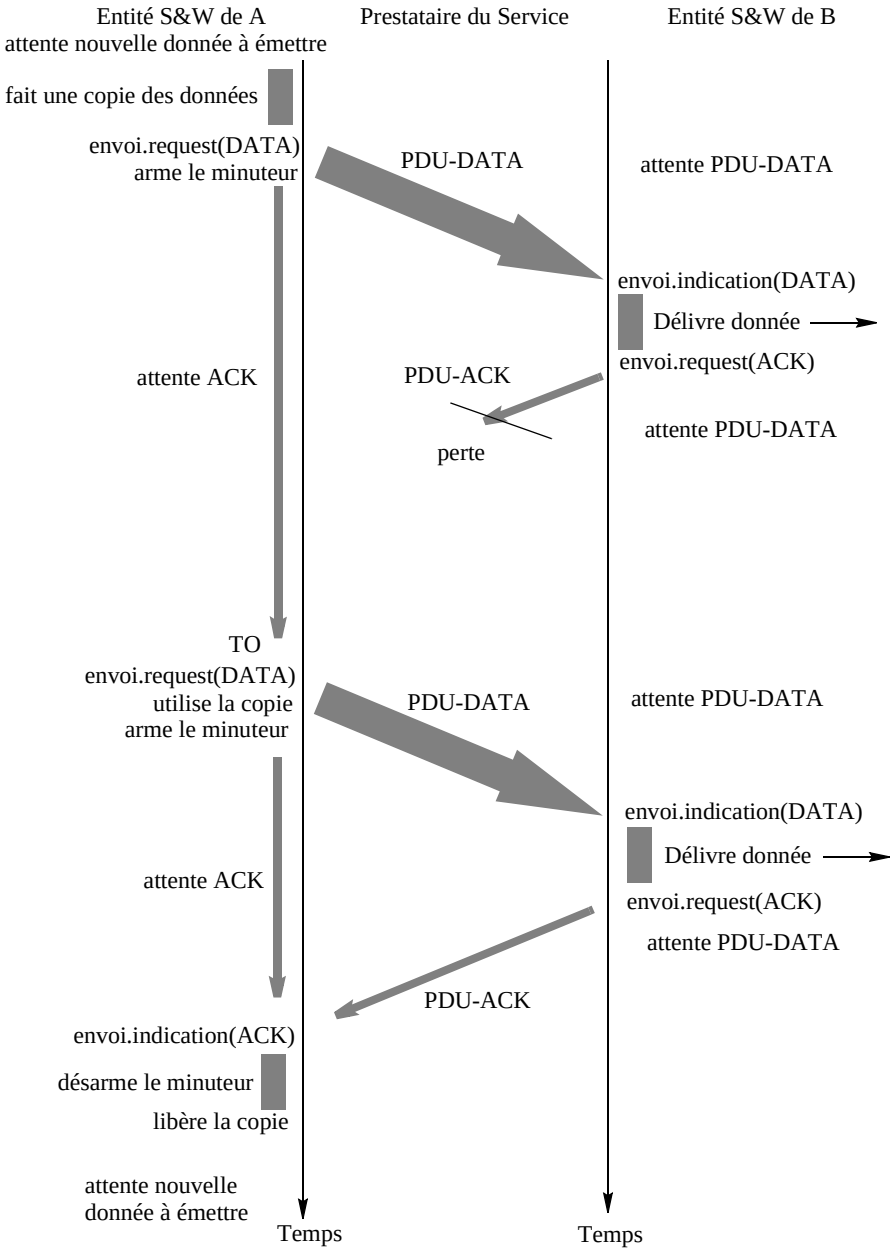

Figure 4.16. *Diagramme d'échange du protocole « Send and Wait » en cas de perte du PDU-ACK. Deux copies de la même donnée sont délivrées à l'utilisateur destinataire*

4.3.1.4. *Performances du protocole « Send and Wait »*

L'utilisation de la voie de communication par ce protocole est fonction des délais. La voie est occupée uniquement pendant la durée d'émission, d_e^{DATA}, du message. Par contre, pour émettre un nouveau message, il faut attendre au minimum, en l'absence de perte : $d_{ar} = d_e^{DATA} + d_e^{ACK} + 2*d_t + [d_{ted} + d_{trd} + d_{tra}]$

où :

— d_e^{DATA} est la durée d'émission du PDU-DATA ;

— d_e^{ACK} est la durée d'émission du PDU-ACK ;

— d_t le délai d'acheminement (si la voie est construite sur un réseau complexe maillé, il se résume au délai de propagation plus la durée d'émission si la voie se résume à un seul support physique), une fois par message. On rappelle que les facteurs qui entrent en compte pour le calcul du délai d'acheminement sont, le temps de transit de chaque entité traversée, et pour chaque voie entre deux entités (voie simple), le délai de propagation, d_p (fonction des distances à parcourir), et le débit sur chaque voie simple. Pour une voie simple (pas d'entité intermédiaire entre l'émetteur et le destinataire), si le débit est élevé, d_p devient prépondérant. A l'inverse, si le débit est faible, d_p devient négligeable ;

— $[d_{ted} + d_{trd} + d_{tra}]$ la somme des temps de traitement. Ceux-ci ont été décrits sur la figure 4.10. Ces temps sont fonction de la puissance des processeurs utilisés, de la mise en œuvre plus ou moins efficace des algorithmes, de l'efficacité des mécanismes de gestion des interruptions de l'OS... Pour une machine donnée, on considérera le temps de traitement comme une constante.

Si deux demandes de transmission se succèdent, l'efficacité d'utilisation de la voie est mesurée par : $E = d_e^{DATA}/d_{ar}$ (toujours <1).

De ce fait, la bande passante que l'on peut offrir à l'utilisateur est, au mieux, le produit ExD, où D est le débit de la voie utilisée.

Ce protocole n'utilise pas du tout le parallélisme potentiel des trois entités (S & W de A, S & W de B, prestataire du service de communication). Il est intéressant de pouvoir émettre les données sans attendre l'acquittement pour tirer profit de ce parallélisme. On appelle anticipation le principe d'émettre la donnée suivante, alors que l'on n'a pas encore reçu l'acquittement de la ou des précédentes.

4.3.2. *Protocole « Go Back N »*

Ce protocole tire profit du parallélisme possible en réseau. Il autorise la source à émettre un nombre F de PDU sans avoir reçu d'acquittement. F est le facteur d'anticipation, ou fenêtre. Ce mécanisme est mis en œuvre sous diverses formes dans la plupart des protocoles.

Dans ce protocole, l'émetteur peut envoyer continuellement ses trames tant qu'il en a à transmettre, sans attendre l'ACK. A l'expiration du timer de la première trame envoyée et non acquittée, cette trame et toutes celles qui suivent sont réémises. On

associe à la première trame non acquittée émise un réveil armé. Lorsqu'un ACK arrive, un réveil est armé relativement à la trame émise après celle qui vient d'être acquittée. La figure 4.17. montre un exemple de retransmissions. Sur cette figure, le PDU-DATA2 se perd. L'entité émettrice A a réussi à transmettre DATA3 et DATA4 lorsque le réveil sonne pour signaler que DATA2 n'est pas arrivé. L'entité retransmet alors toutes les unités de données qu'elle a mémorisées. Sur la figure 4.17., les PDU DATA2, DATA3 et DATA4 sont retransmis en séquence sans autre délai. On remarquera que les unités de données sont numérotées. Il faut donc que le protocole gère ces numéros. Nous verrons dans le chapitre suivant comme exemple d'un tel protocole LAPB utilisé dans le réseau Transpac, entre autres, dans la couche liaison de données. Comme il y a beaucoup de variantes possibles dans la gestion des retransmissions, nous n'irons pas plus loin dans la description d'un protocole Go Back N.

Par un PDU-ACKi, le récepteur indique qu'il attend un PDU-DATAi et que tous les PDU-DATA de numéro inférieur à i ont été reçus.

Sur la figure 4.17, on remarquera que l'entité réceptrice acquitte uniquement le dernier PDU reçu correctement et en séquence. On dit que des PDU sont en séquence lorsque le récepteur a reçu sans interruption tous les PDU numérotés 1 à n. Un PDU est dit hors séquence si son numéro est séparé du PDU précédent de plus de un numéro. Ainsi, sur la figure 4.17, le PDU-DATA3 est hors séquence car l'entité réceptrice a reçu le PDU-DATA1 en dernier. Il lui manque donc le PDU-DATA2. Il répète à l'infini ce PDU-ACK2 sur toute réception de données hors séquence pour indiquer qu'il attend toujours le PDU-DATA2. Celui-ci sera renvoyé lorsque le réveil qui lui est associé sonne dans l'entité émettrice.

Par contre, la perte d'un PDU-ACK est tolérée par ce protocole. En effet, tout PDU-ACKi indique que les données de numéro inférieur à i ont été reçues en séquence par le récepteur. Sur la figure 4.17. la perte du PDU-ACK4 est sans importance du fait qu'un PDU-ACK5 arrive peu après et indique à l'automate émetteur la bonne réception par B des PDU DATA3 et DATA4.

Les performances évaluées en utilisant le même critère que précédemment doivent prendre en compte le nombre de blocs que l'on peut transmettre consécutivement. Soit F ce nombre. Il faut attendre pour avoir l'ACK du $F^{\text{-ième}}$ bloc.

— $D = F[d_e^{DATA} + d_{ted}] + d_e^{ACK} + d_{trd} + 2d_t + F*d_{tra}$.

Cette formule montre que le parallélisme cache le coût dû aux délais d'acheminement de F-1 messages et à la durée d'émission du PDU-ACK.

— $F [d_e^{DATA} + d_{trd} + d_{ted}]$ est le coût pour l'émission des F messages par l'émetteur et le coût de traitement des F acquittements.

— $F*d_{tra}$ est le coût de prise en compte par l'émetteur des F PDU-ACK.

— Le coût d'émission des F PDU-ACK n'apparaît que pour une seule fois, la dernière. Les F-1 émissions précédentes sont faites en parallèle avec l'émission des PDU-DATA, ou leur traitement, ou le traitement du PDU-ACK précédent.

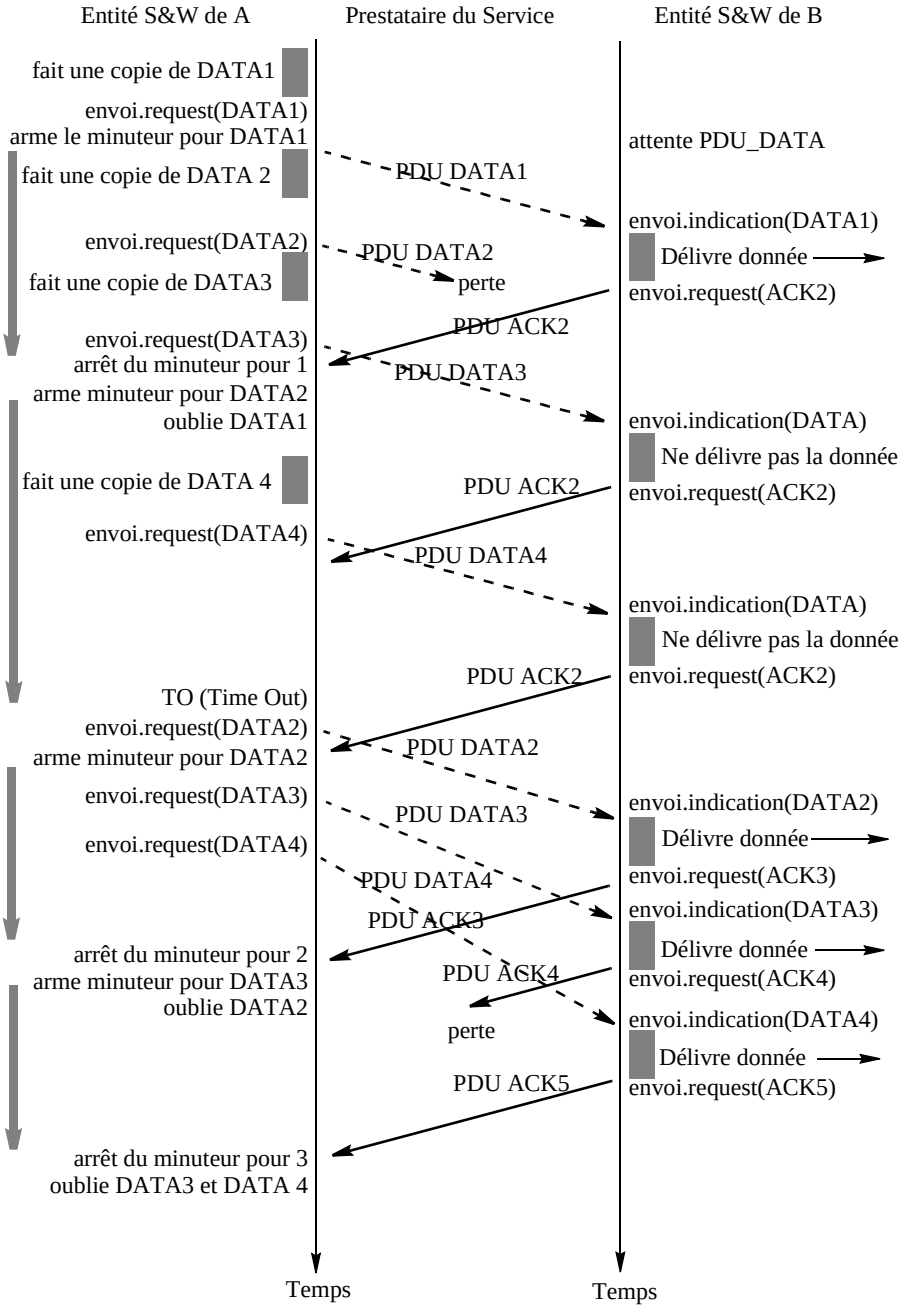

Figure 4.17. *Diagramme d'échange du protocole Go back N simplifié*

— De même, le délai de propagation n'apparaît que pour le premier PDU-ACK et pour le dernier PDU-ACK.

Sur un réseau satellitaire, d_t est principalement dû au délai de propagation d_p, qui est grand. Le gain obtenu par le protocole Go Back N sur le protocole S & W est considérable. On pourrait encore améliorer le protocole en ne retransmettant que les PDU réellement perdus.

Parmi les avantages de ce protocole, on notera que le récepteur peut jeter tout PDU hors séquence. Il n'a pas besoin de les stocker. Cela simplifie sa tâche et économise des ressources mémoire.

4.3.3. *Protocole à répétition sélective*

On utilise ici des NAK pour signaler les PDU erronés en conjonction avec le mécanisme de numérotation. Seul le PDU acquitté négativement, ou dont le réveil a sonné sans recevoir pour lui d'acquittement positif, est renvoyé. A l'évidence, cette technique améliore les performances de débit par rapport au protocole Go Back N. Le récepteur doit faire un réordonnancement de ses tampons (zone où sont stockés les PDU reçus hors séquence), car les SDU doivent être délivrés au récepteur dans le bon ordre. Cette technique est avantageuse dans les réseaux où le délai de propagation est particulièrement long par rapport à la durée d'émission (satellites, réseaux à haut débit...). Le nombre de tampons nécessaires devient important. Ce protocole utilise une PDU appelée SREJ « selective reject ».

4.4. Conclusion

La description d'un protocole sous la forme d'un automate d'état fini doit impérativement être maîtrisée car cette forme de spécification est la plus fréquemment rencontrée [2]. L'automate d'état fini permet de comprendre le comportement dynamique du protocole représenté sur les diagrammes temporels. Une séquence de trames observées sur un support physique est un diagramme temporel.