Réseaux

&

Protocoles

Protocole TCP

TCP

- **TCP : Transmission Control Protocol**
  Défini par la RFC 793[(1)]

  IP : Protocole non fiable ⇒ "Best effort protocol"

  TCP est indépendant de la couche IP
  TCP gère la fiabilité des échanges

[(1)] RFC 793 (1981) → Nombreuses adaptations
RFC 7474 → Synthèse des modifications apportées à la RFC 793

Objectifs initiaux[1]

- Définir un protocole de communication caractérisé par
  - Fonctionnement au dessus de la couche IP
  - Gestion d'un circuit logique fiable et sécurisé entre processus
  - Transmission d'un flux de données bidirectionnelles
  - Gestion de la fiabilité des données
  - Contrôle de flux
  - Accepte les connexions multiples

[1] Cerf & Kahn "A protocol for packet network intercommunication" Mai 1974

Propriétés de TCP

- Protocole orienté connexion
  Établissement d'un chemin virtuel entre source et destination
  - → Chemin virtuel indépendant de la couche IP
  - → Mécanisme d'identification du chemin virtuel
  - → Acheminement sans erreur des données
  - → Données délivrées dans l'ordre d'émission
  - → Communication full-duplex

  Gestion de congestion
  Optimisation de l'efficacité réseau

Interface TCP

- Paquet TCP = (En-tête + données) transmises par la couche TCP

TCP propose une interface permettant :

Ouvrir / Fermer une connexion
Émettre / Recevoir des données
Obtenir le statut d'une connexion

TCP utilise des buffers pour échanger les données avec les couches adjacentes

Interface TCP

- Interface utilisateur fournie par TCP
  - `open ( port, foreign socket, mode [, timeout] [, précédence] [, security] [, options] ) → con_name`
  - `send ( con_name, buffer, byte_count, push, urgent [, timeout] )`
  - `receive ( con_name, buffer, byte_count ) → byte_count, urgent, push`
  - `close ( con_name )`
  - `status ( con_name )`
  - `abort ( con_name )`

En-tête TCP

<table>
<tr><td colspan="4">0 4 8 16 31</td></tr>
<tr><td colspan="2">Source port</td><td colspan="2">Destination port</td></tr>
<tr><td colspan="4">Sequence Number</td></tr>
<tr><td colspan="4">Acknowledgement Number</td></tr>
<tr><td>Data Offset</td><td colspan="2">Flags</td><td>Window</td></tr>
<tr><td colspan="2">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options</td></tr>
<tr><td colspan="4">Data
...</td></tr>
</table>

En-tête TCP

- Port source — Port hôte émetteur
- Port destination — Port récepteur

Gestion des ports idem UDP
Port associé à un service

TCP retransmets les paquets dans la pile FIFO du service destination

- Intégrité ⇒ Tous les paquets transmis sont valides
- Respect des flux ⇒ Les paquets sont transmis dans l'ordre

En-tête TCP

- Sequence Number
- Acknowledgement Number

Identifiants permettant de gérer la connexion

- Identifie la connexion en cours
- Indication de flux
- Validation des paquets reçus

En-tête TCP

- Data offset Dimension du champ option
  Indique la position des données

  Valeur = Position du 1er octet de données / 1er octet en-tête
  1 ligne = 4 octets
  Bourrage si taille ≠ options Modulo 4

- Pas d'option ⇒ Data Offset = 5

  Valeur maxi codée sur 4 bits ⇒ Data offset maxi = 15
  ⇒ 40 octets d'options au maximum

En-tête TCP

- Drapeaux[(1)]
  - CWR Congestion Window Reduced
  - ECE Explicit Congestion Notification Echo
  - URG Signale un traitement urgent
  - ACK Message d'acquittement
  - PSH Activation de la fonction push
  - RST Demande de Reset
  - SYN Message de synchronisation de connexion
  - FIN Fin de la connexion

(1) Champ drapeaux codé sur 12 bits. Les bits de poids fort sont positionnés à 0
La signification de chacun sera présentée ultérieurement.

En-tête TCP

- Window Fenêtre de réception

  ⇒ Indique à l'émetteur la quantité de données qui peuvent être transmises ⇔ Espace disponible dans le buffer de réception.

- La valeur est significative si le drapeau ACK est positionné

Remarque : Les mécanismes lié à la charge du réseau seront présentés ultérieurement

En-tête TCP

- Checksum Validation du paquet[(1)]
  → en-tête TCP + Données + pseudo-en-tête IP

- Urgent Pointer Position des données dont le traitement est prioritaire
  ⇒ Si drapeau URG est positionné

[(1)] Contrairement à UDP, checksum est obligatoire à l'émission, et doit être validé à la réception

Pseudo en-tête IPv4

0	8	16	31
Adresse source			
Adresse destination			
Zero	Protocole	TCP Length	

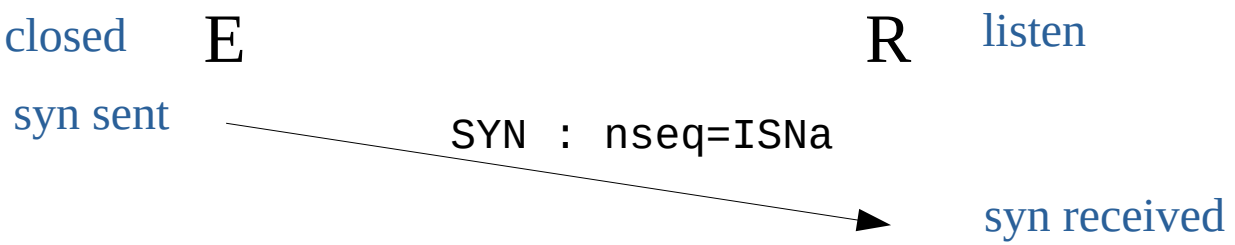
- Protocole TCP ⇒ 6

Pseudo en-tête IPv4 identique pour UDP et TCP

Principes de base

Établissement de la connexion

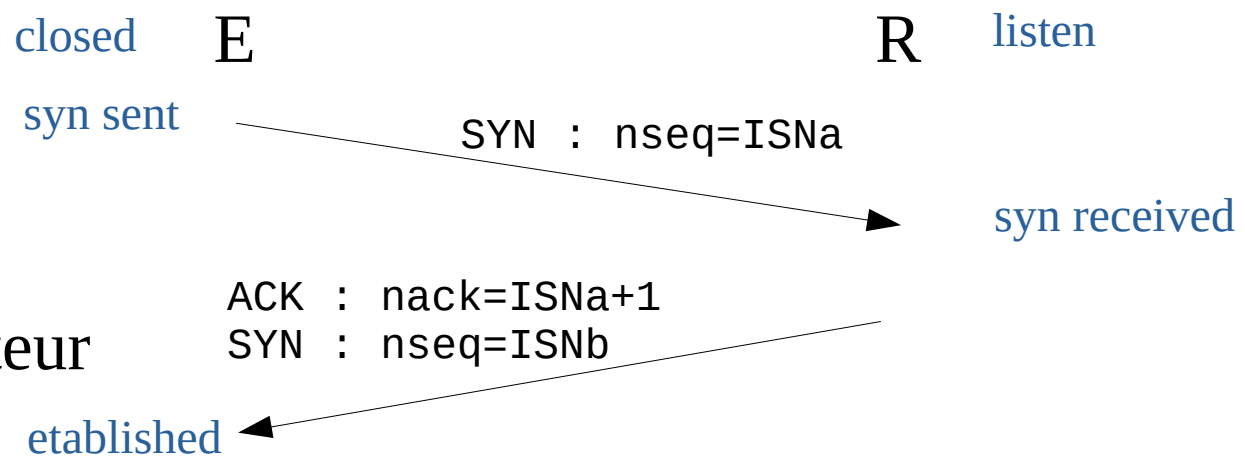
- Poignée de mains à 3 voies[(1)]



[(1)] Three-way handshaking

Établissement de la connexion

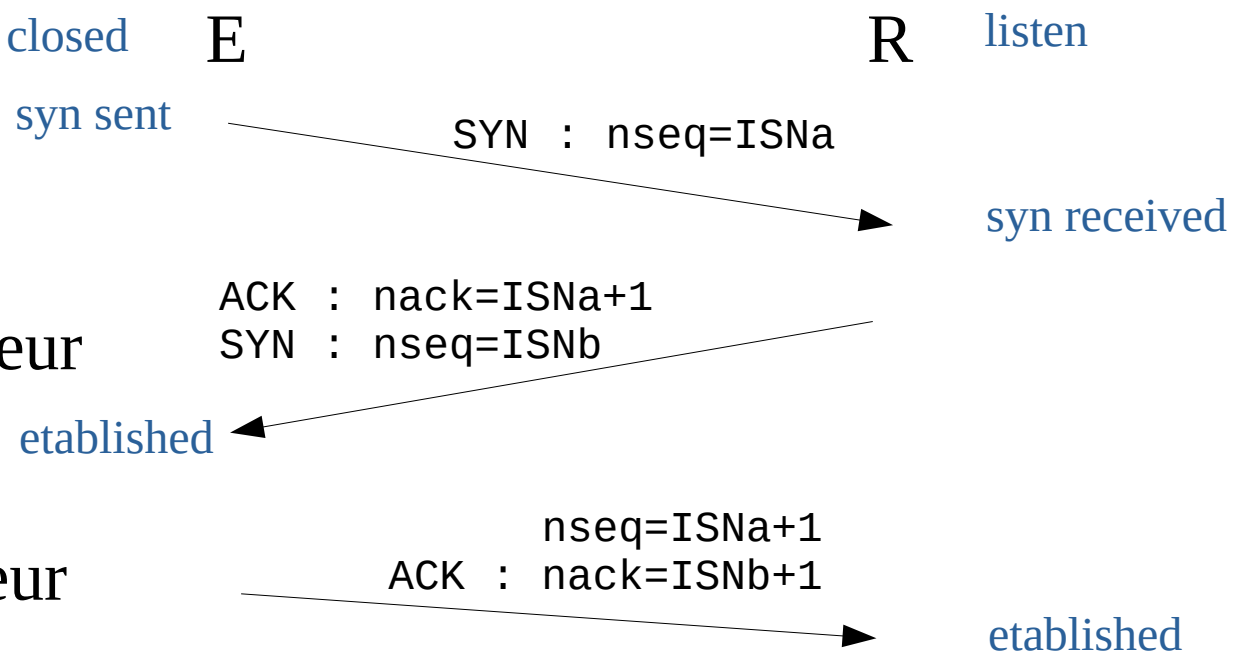
- Poignée de mains à 3 voies[1]



[1] Three-way handshaking

Établissement de la connexion

- Poignée de mains à 3 voies[1]



[1] Three-way handshaking

Gestion des connexions

- TCP gère les connexions en full-duplex
- Chaque sens de communication est indépendant
  - `NSEQ` = N° séquence ⇒ Donnée émises

    `NSEQ` = position du 1er octet de données dans le flux
  - `NACK` = N° acquittement ⇒ Données reçues

    `NACK` = Position du prochain octet attendu dans le flux

Gestion des connexions

- TCP peut gérer des flux séparés

- Un flux est identifié par le couple (@IP , port) (1)

Remarque :

Processus répertoriés par IANA

⇒ Établissement de la connexion sur le port dédié

Service reçoit requête sur port dédié

En retour, TCP spécifie le n° de port pour les échanges suivants

(1) Enregistrement des flux dans la structure TCB

TCB : Transmission Control Block

- Structure de gestion des connexions établies contenant :
  - Identification du processus utilisant la connexion
  - Pointeurs sur les buffers / fenêtre
  - Timers de connexion
  - Indicateurs de statut

- TCP utilise une table de structures TCB

! A chaque requête SYN, une entrée TCB est définie
⇒ Waiting TCB Connexion en cours d'établissement

Remarque : La structure des TCB est décrite dans RFC 675

SYN Flood Attack

- **Attaque de type DOS par saturation des TCB**

- Rappel : Une connexion TCP est établie par un protocole de Handshake en 3 temps

1) Demande d'ouverture de connexion par le client → SYN
2) Réponse du serveur → SYN / ACK
   Une entrée TCB est réservée dans le backlog SYN
3) Validation par le client → SYN / ACK
   L'entrée TCB du backlog est supprimée → Transfert vers gestion des connexion établie

SYN Flood Attack

- Attaque de type DOS par saturation des TCB

1) Demande d'ouverture de connexion par le client → SYN
2) Réponse du serveur → SYN / ACK
   Réservation entrée TCB
3) !! Pas de validation par le client
   L'entrée TCB du backlog reste en attente jusqu'au timeout
4) Le client boucle en 1) pour multiplier les entrées
5) Le serveur est saturée lorsque l'espace mémoire alloué au TCB backlog est plein

SYN Flood Attack

- Prévention des attaques de type SYN Flooding
- Augmenter la taille mémoire allouée au backlog
  ⇒ Nb requêtes maxi dim. mémoire / timeout [1]
- Si requêtes émise par le même client (@IP, …)
  ⇒ Blocage de l'@IP [2]
- Recycler les connexion TCP semi-ouvertes les plus anciennes
- Lorsque le backlog est plein, les entrées TCB ne sont pas conservées mais les ports restent ouverts. Si une réponse valide est reçue (validation par pare-feu), alors reconstruction (partielle?) de l'entrée TCB

[1] Le trafic attaque SYN Flood < trafic attaque DDoS usuelles
[2] Protection inefficace contre les botnet

Numéro de séquence

- Établissement de la connexion

Valeur pseudo-aléatoire non prédictive codée sur 32 bits

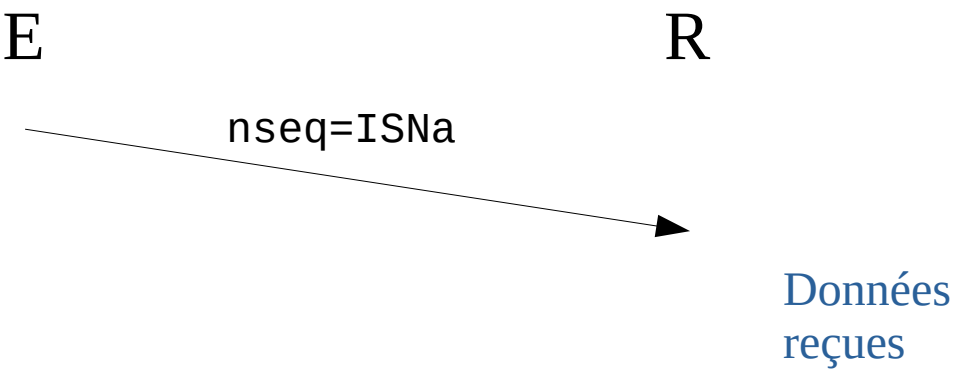
RFC 793 → Le choix de ISN n'est pas arbitraire
→ Générateur 32 bits, incrémenté chaque 4µs
→ Délai de 2 * MSL entre 2 connexions
MSL = Maximum Segment Lifetime

⇒ Un attaquant peut prédire le numéro de séquence

RFC 6528 - Defending against sequence number attacks

Envoi de données

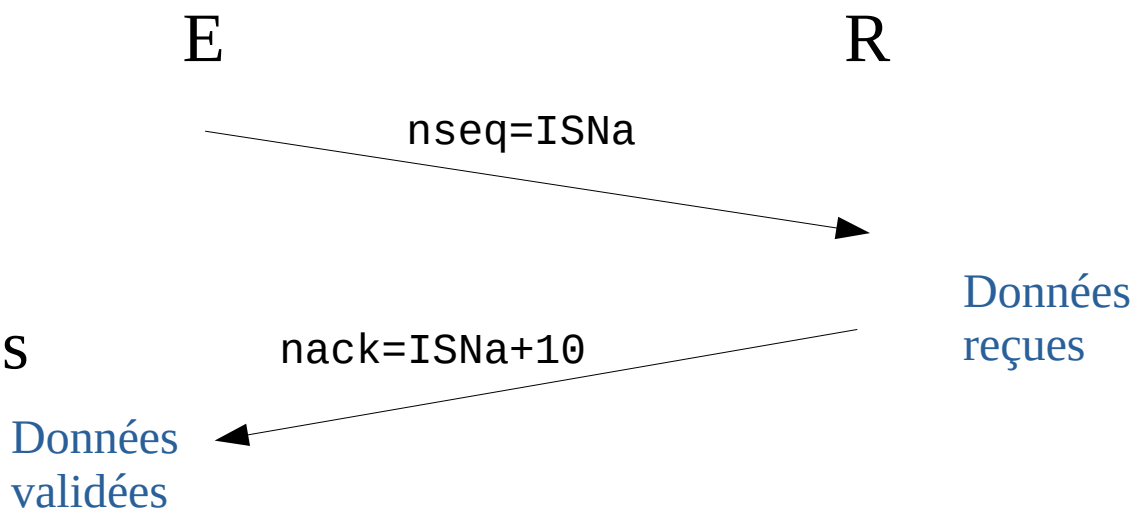
1) Envoi de 10 octets



Envoi de données

1) Envoi de 10 octets

2) Acquittement données



Communication fiable

- TCP assure la validité des données et leur remise dans l'ordre

⇒ Gestion réalisée par les numéro de séquence et d'acquittement

- Émetteur envoie de données → N° séquence identifie 1er octet

  Émetteur enregistre les données émises dans un buffer

  \+ déclenche un timer

- Récepteur envoie un acquittement

  n° acquittement = n° séquence + taille des données

  ⇒ n° acquittement = position du prochain octet attendu

- Après acquittement, émetteur supprime les données de son buffer

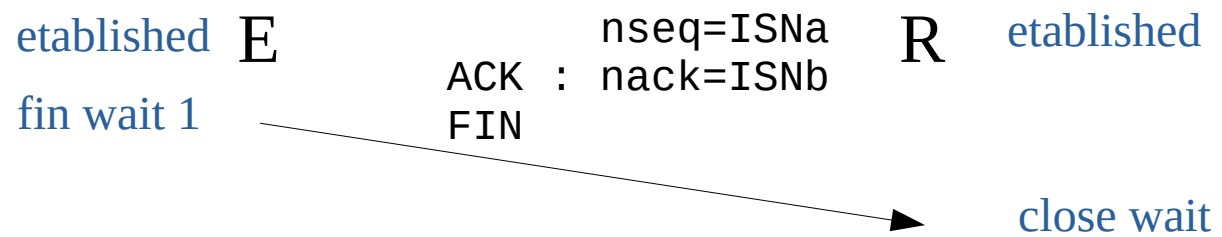
Numéro d'acquittement

- Le numéro de séquence permet d'identifier chaque données émise
  Chaque donnée peut donc être validée par le récepteur
- Mécanisme d'acquittement cumulatif
  ⇒ Acquittement valide toutes les données précédentes
- Numéro de séquence défini sur 32 bits ⇒ Utiliser modulo 2^{32}
- Règle de gestion de nseq et nack
  - nack doit correspondre à une valeur nseq non encore validée
  - Les données à émettre ne sont retirées du buffer qu'après validation
  - nseq du segment reçu doit correspondre aux données attendues

Fermeture de la connexion

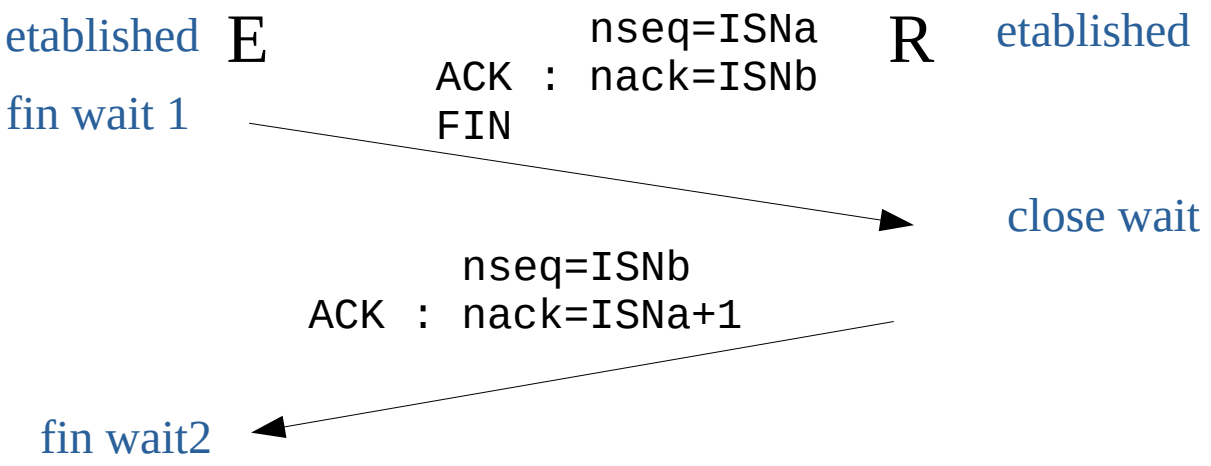
- Fermeture normale de la connexion

1) Demande de fin



Fermeture de la connexion

- Fermeture normale de la connexion

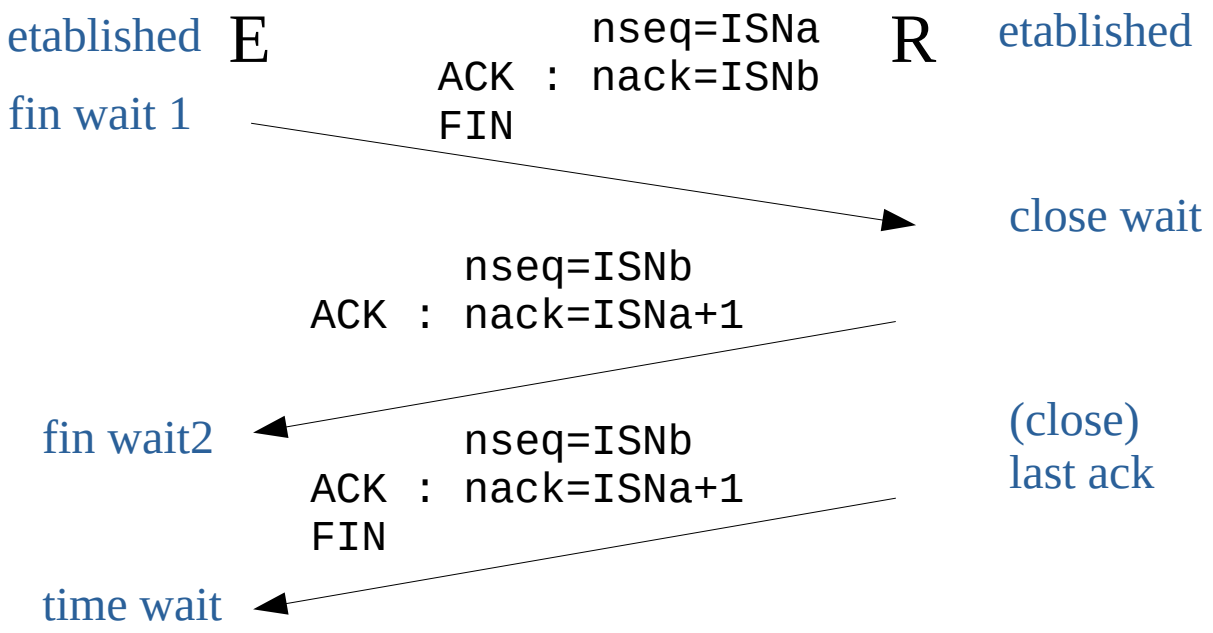


Fermeture de la connexion

- Fermeture normale de la connexion

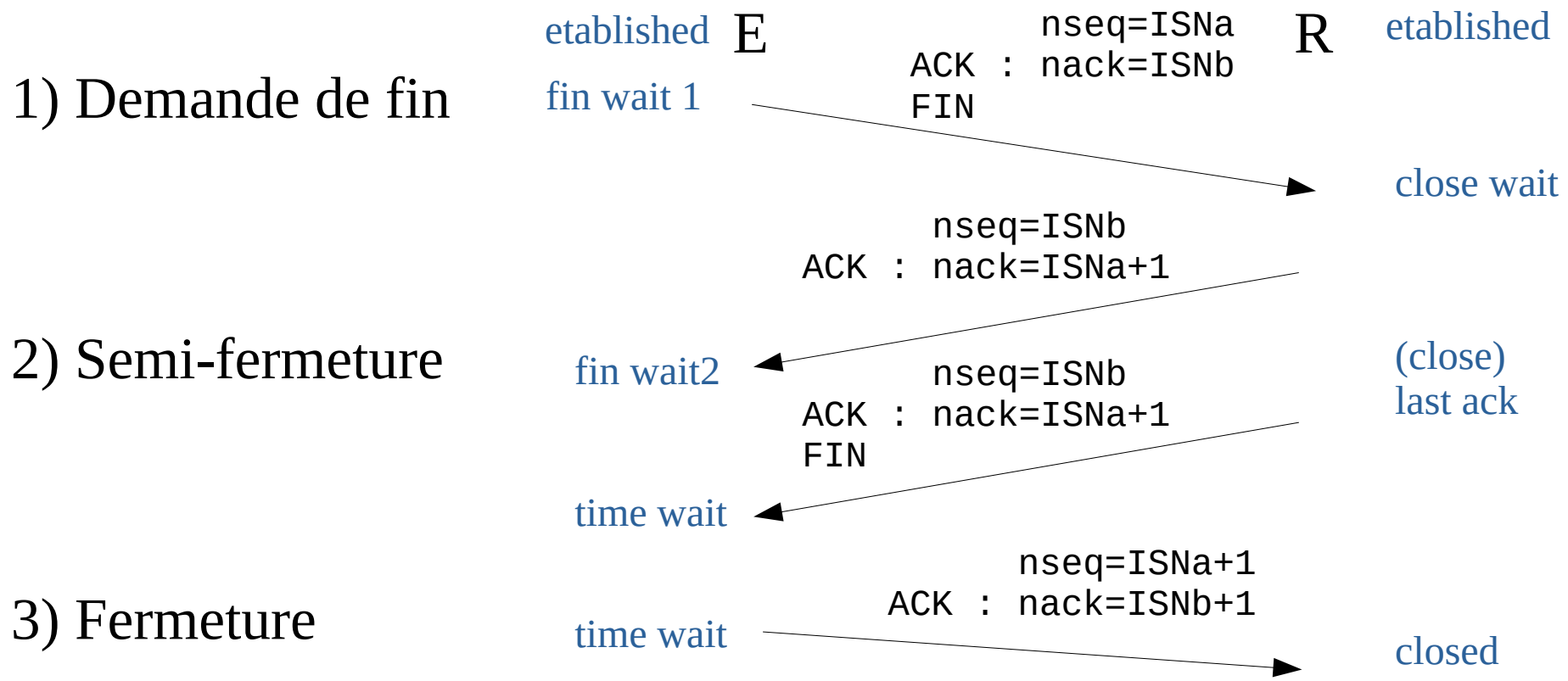
1) Demande de fin

2) Semi-fermeture



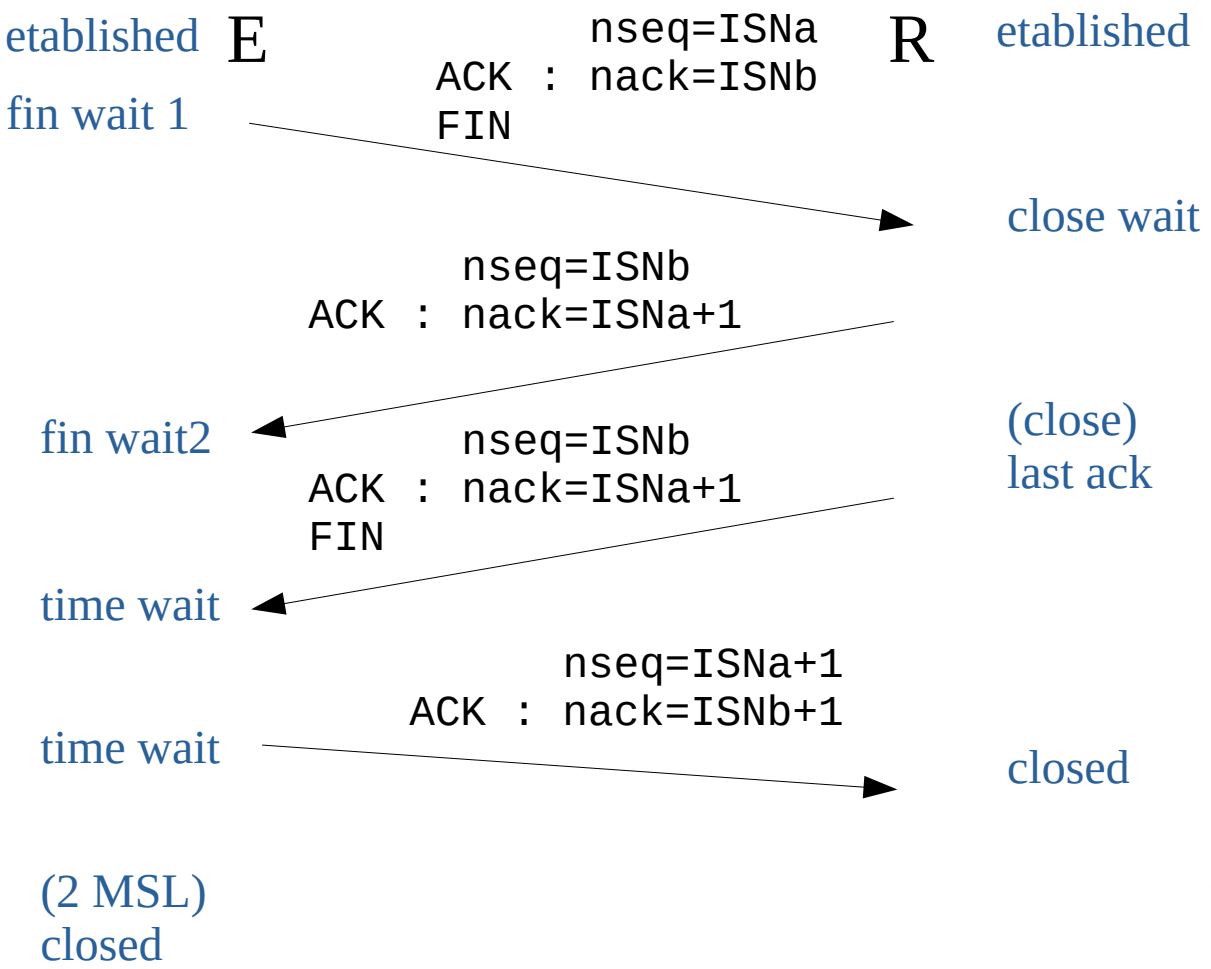
Fermeture de la connexion

- Fermeture normale de la connexion



Fermeture de la connexion

- Fermeture normale de la connexion



Quiet Time concept

- Après fermeture de connexion
  ISN >> dernière valeur nseq Pour éviter confusion
  Recommandation RFC Incrémenter chaque 4µs[(1)]

- MSL = Maximum Segment Lifetime
  ⇒ Durée de vie normale d'un segment < 10s (selon débit)
  Temps attente obligatoire après crash système

- Quiet Time ⇒ Attente minimale pour éviter confusion nseq
  ⇒ MSL = 2mn par défaut

(1) 2^{32} bits ⇒ Durée cycle environ 4,55 heures

Half-duplex

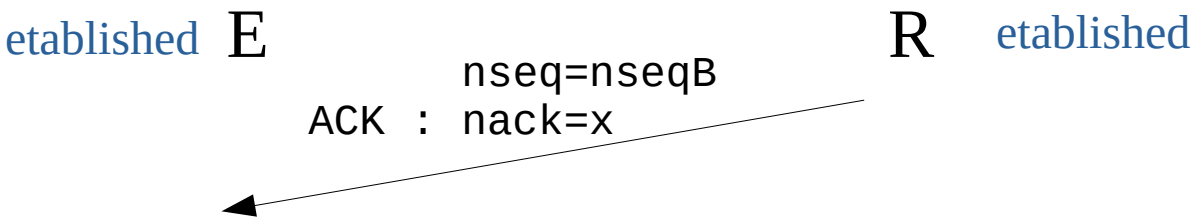
- La fermeture de connexion est uni-directionnelle
  Si fermeture → R ne souhaite plus recevoir de données

- Fermeture Half-Duplex
  - La connexion reste établie
  - La transmission de données est unidirectionnelle
  - Les signaux de validations restent bi-directionnels

- Fermeture Full-Duplex
  - Fin de la connexion

Reset de la connexion

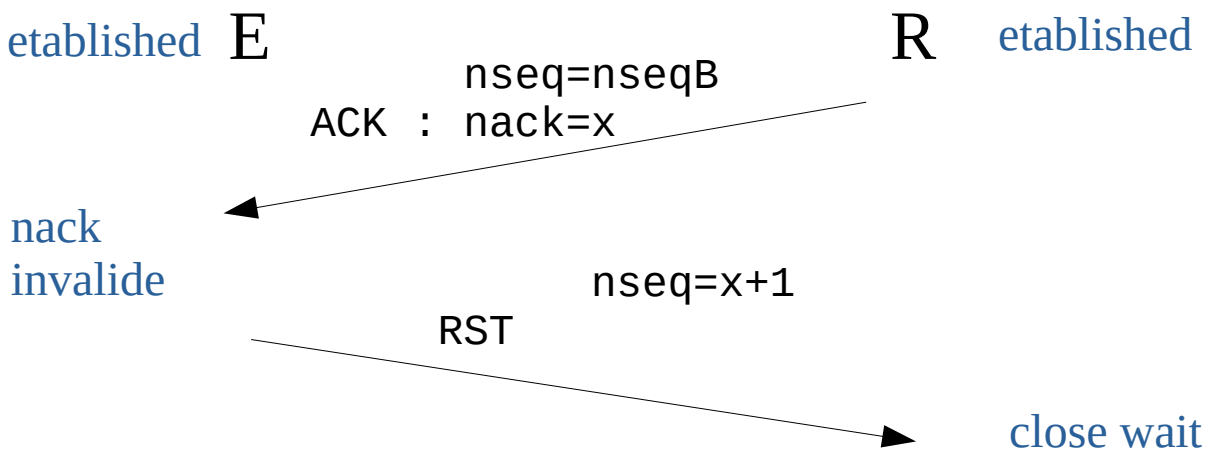
- Reset si détection d'anomalie



Reset de la connexion

- Reset si détection d'anomalie

1) Demande reset



Reset

- Demande de fermeture immédiate de la connexion

- Si la connexion établie (sauf état SYN-SENT)
  → Reset est valide si
  - nseq compris dans la fenêtre normale de connexion

- État SYN-SENT → demande de connexion en cours
  → Reset valide si
  - ACK acquitte message SYN

Blind Attack

- Une connexion TCP est identifiée par 5 informations :
  Protocole, Adresses source et destination, Ports source et destination
- Possibilité de créer une attaque (en aveugle[1]) si l'attaquant est en mesure de générer ces tuples avec une probabilité élevée.
  ⇒ Génération d'un message ICMP comportant les identifiants TCP pour engendrer un reset de la connexion ⇒ Deni de service
- Contre-mesures :
  - Générateur aléatoire pour sélectionner le numéro de port
  - Filtrage des message ICMP : Ne traiter que ceux associés aux messages en cours (non acquitté)

(1) Attaque en aveugle : Les données ne sont pas connues mais un nombre limité de tentatives permet de les créer (Exemple : Séquence aléatoire prévisible)

Références : rfc5927, rfc6056, http://www.gbppr.net/2600/SlippingInTheWindow_v1.0.pdf

Gestion des fenêtres

Fenêtres TCP

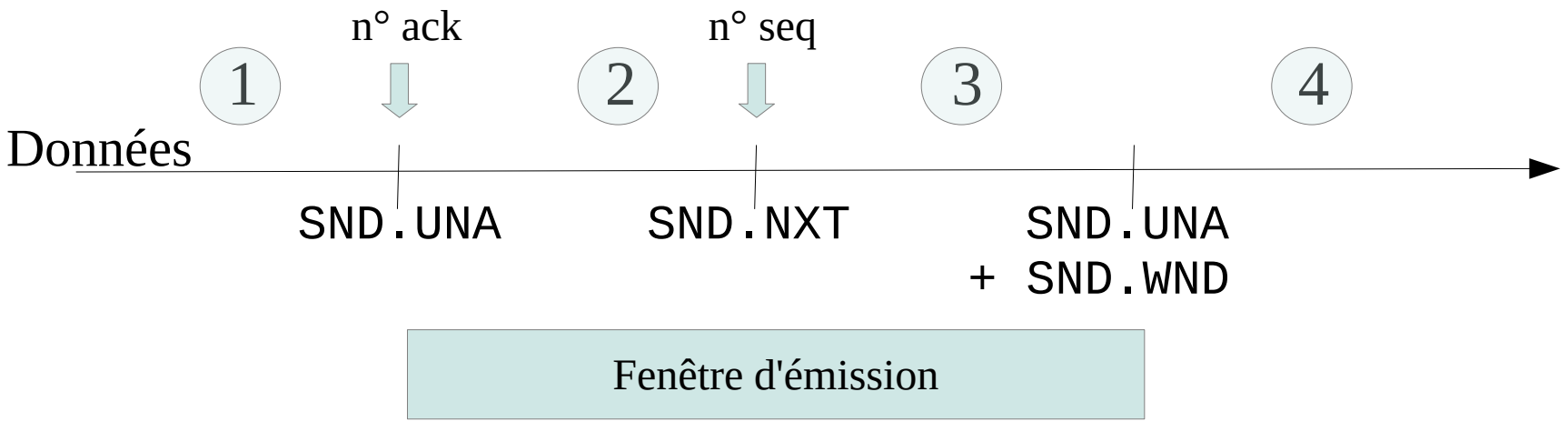
- Synchronisation des échanges gérée par l'utilisation de fenêtres glissantes[1]
- Fenêtre d'émission
  Buffer des données disponibles pour l'émission
- Fenêtre de réception
  Buffer de données disponible pour la réception
- n° seq & n° ack ⇒ Indicateur de position / buffer

[1] *Fenêtre glissante : Sa position de référence se déplace*

TCB : Transmission Control Block

- Send Sequence Variables
  - `SND.UNA` Send unacknowledged
  - `SND.NXT` Send next
  - `SND.WND` Send window
  - `SND.UP` Send urgent pointer
  - `SND.WL1` Segment sequence number use for last window update
  - `SND.WL2` Segment acknowledgement number use for last window update
  - `ISS` Initial send sequence number

Fenêtre d'émission



1 - Données émises et acquittées

2 - Données émises, non acquittées

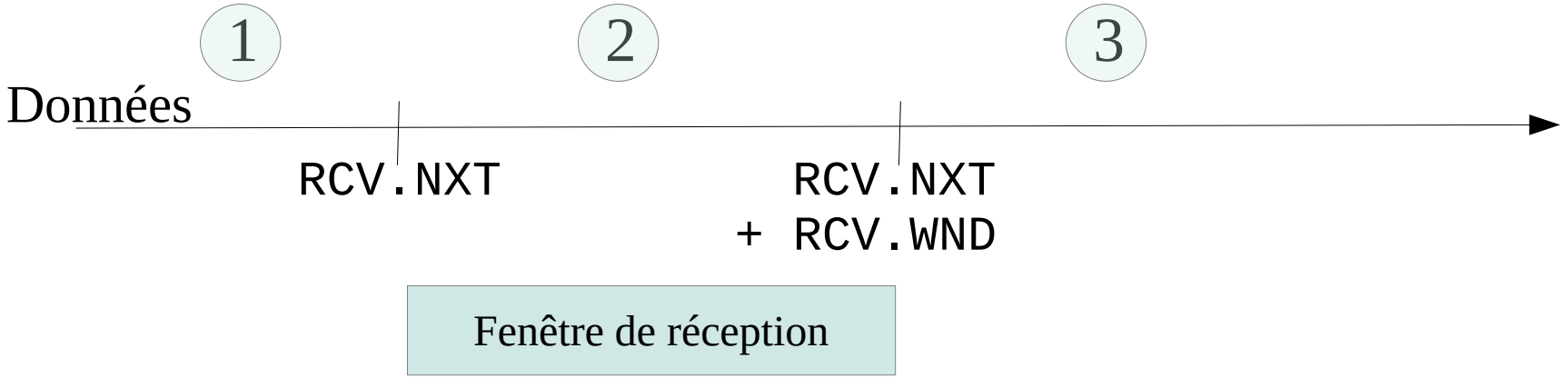
3 - Données prêtes, non encore émises

4 - Données non prêtes ⇐ Fournies par application

TCB : Transmission Control Block

- Receive Sequence Variables
  - `RCV.NXT` Receive next
  - `RCV.WND` Receive window
  - `RCV.UP` Receive urgent pointer
  - `IRS` Initial receive sequence number

Fenêtre de réception



1 - Données reçues et acquittées

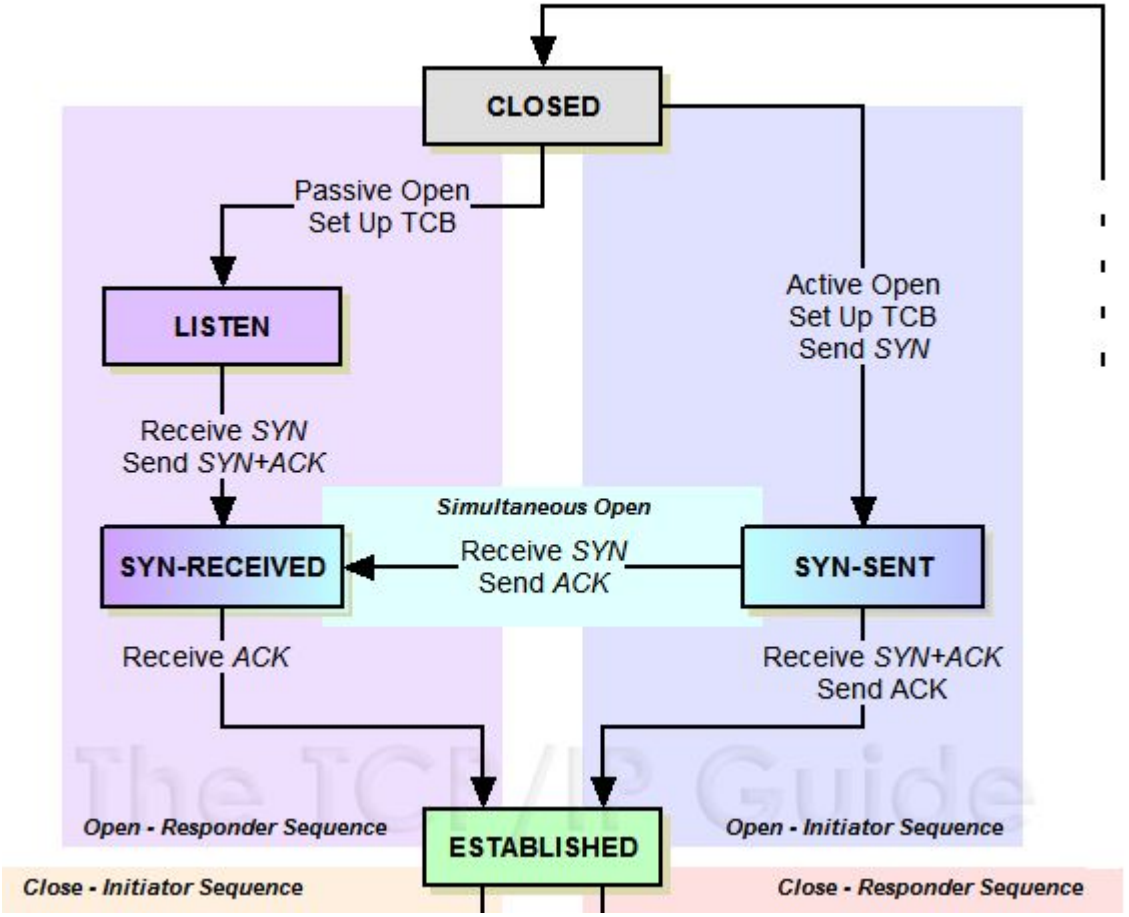
2 - Zone de données prête à la réception

3 - Zone non prête, pour futures données

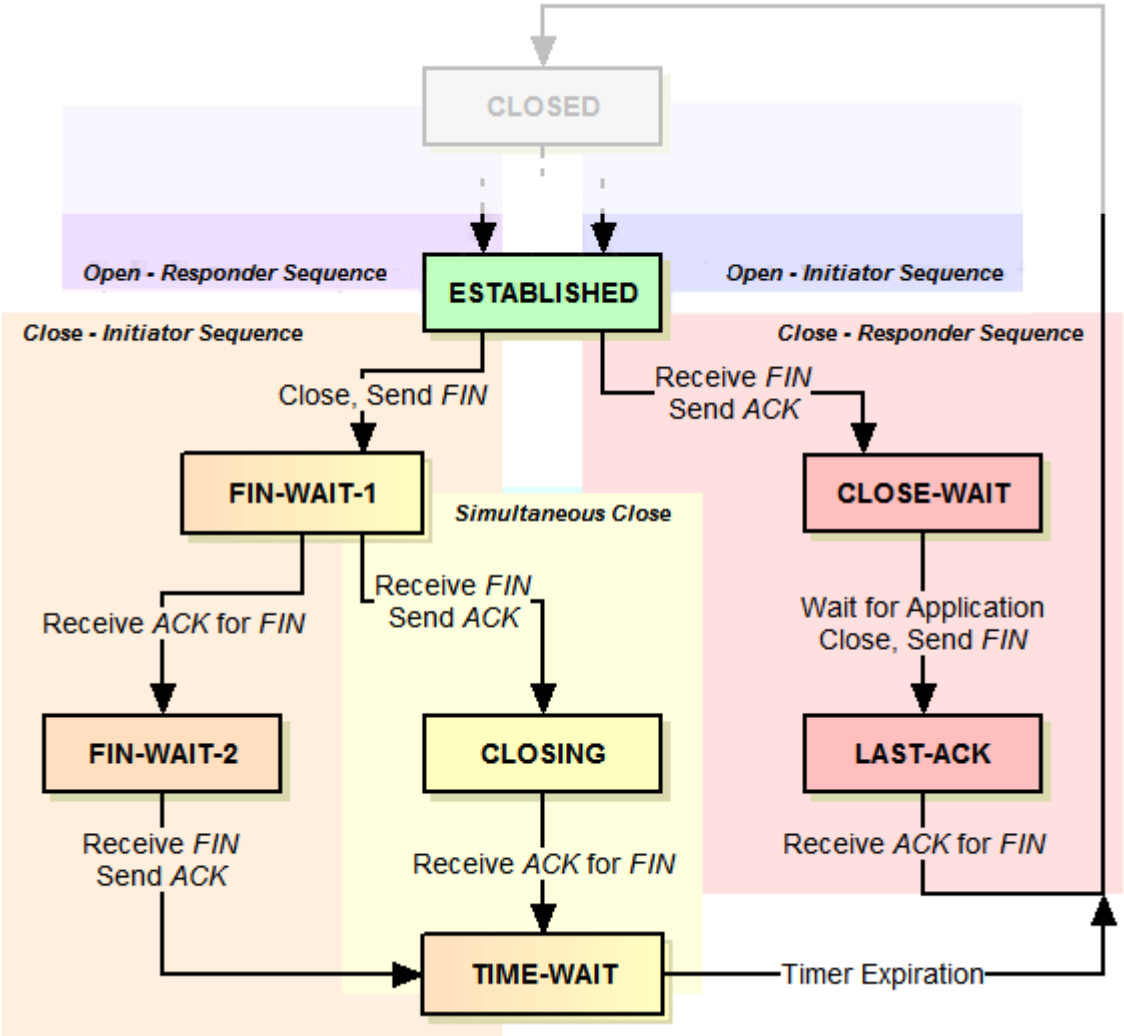
TCB : Transmission Control Block

- Current Segment Variables
  - `SEQ.SEQ` Segment sequence number
  - `SEQ.ACK` Segment acknowledge number
  - `SEQ.LEN` Segment length : Dimension des données dans le segment
  - `SEQ.WND` Segment window
  - `SEQ.UP` Segment urgent pointer
  - `SEQ.PRC` Segment precedence value

Statut de la connexion - Ouverture



Statut de la connexion - Fermeture



Validation des données

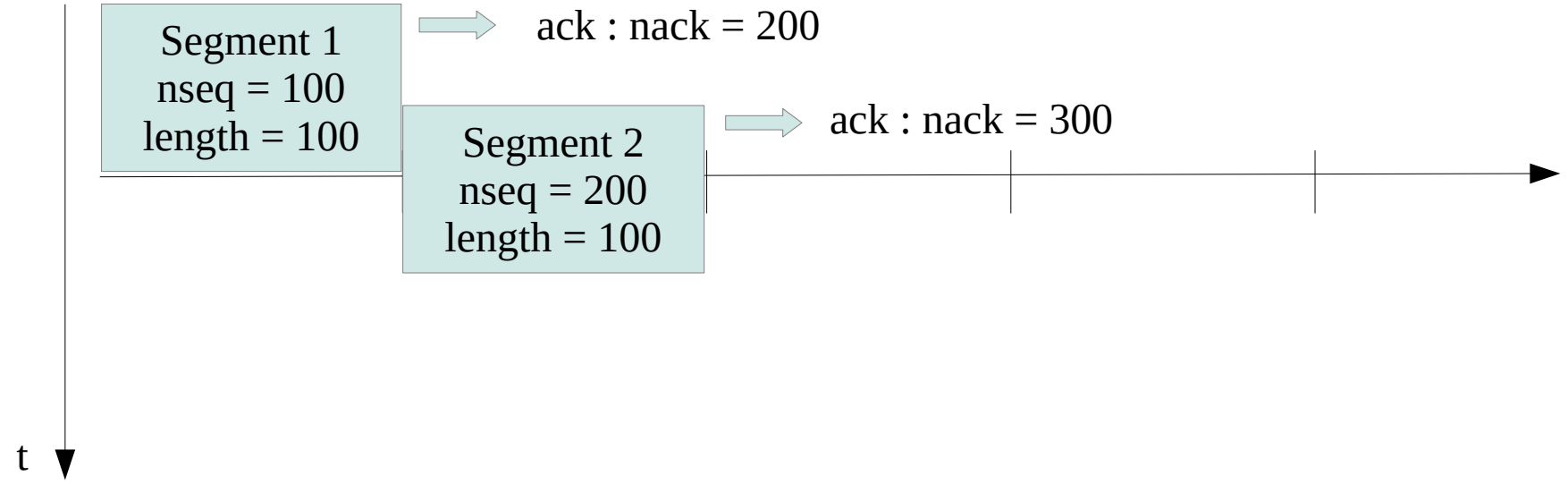
Validation des données

- TCP propose des mécanismes pour valider les données
  nseq / nack ⇒ Acquittement des données reçues & valides

- R → Détection de données non reçue
- E → Données non acquittées

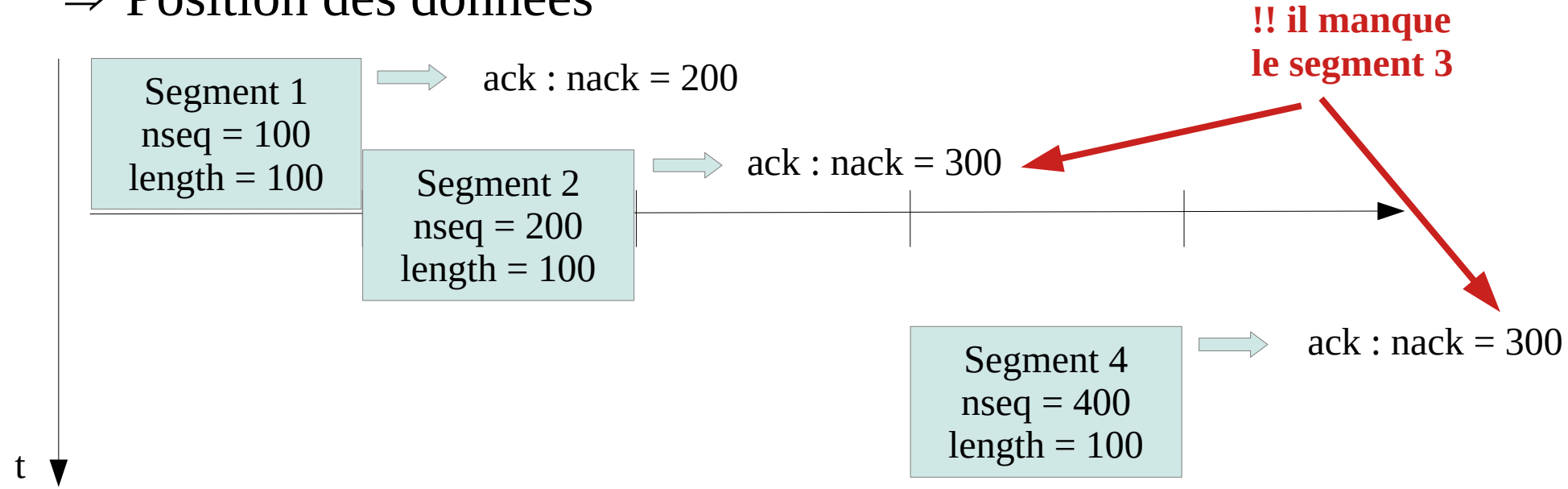
R → Données non reçues

- Chaque segment possède un numéro de séquence
  ⇒ Position des données

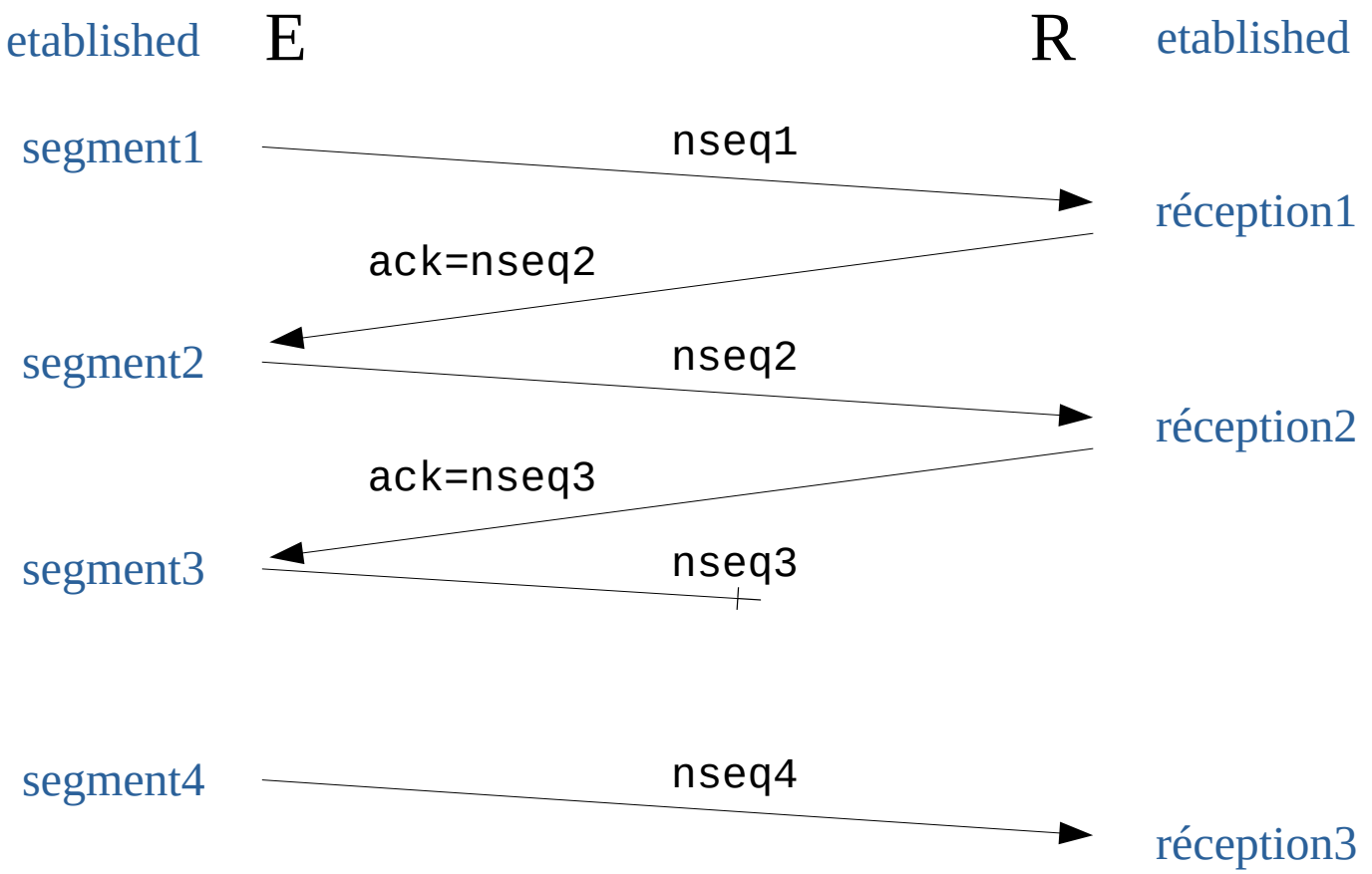


R → Données non reçues

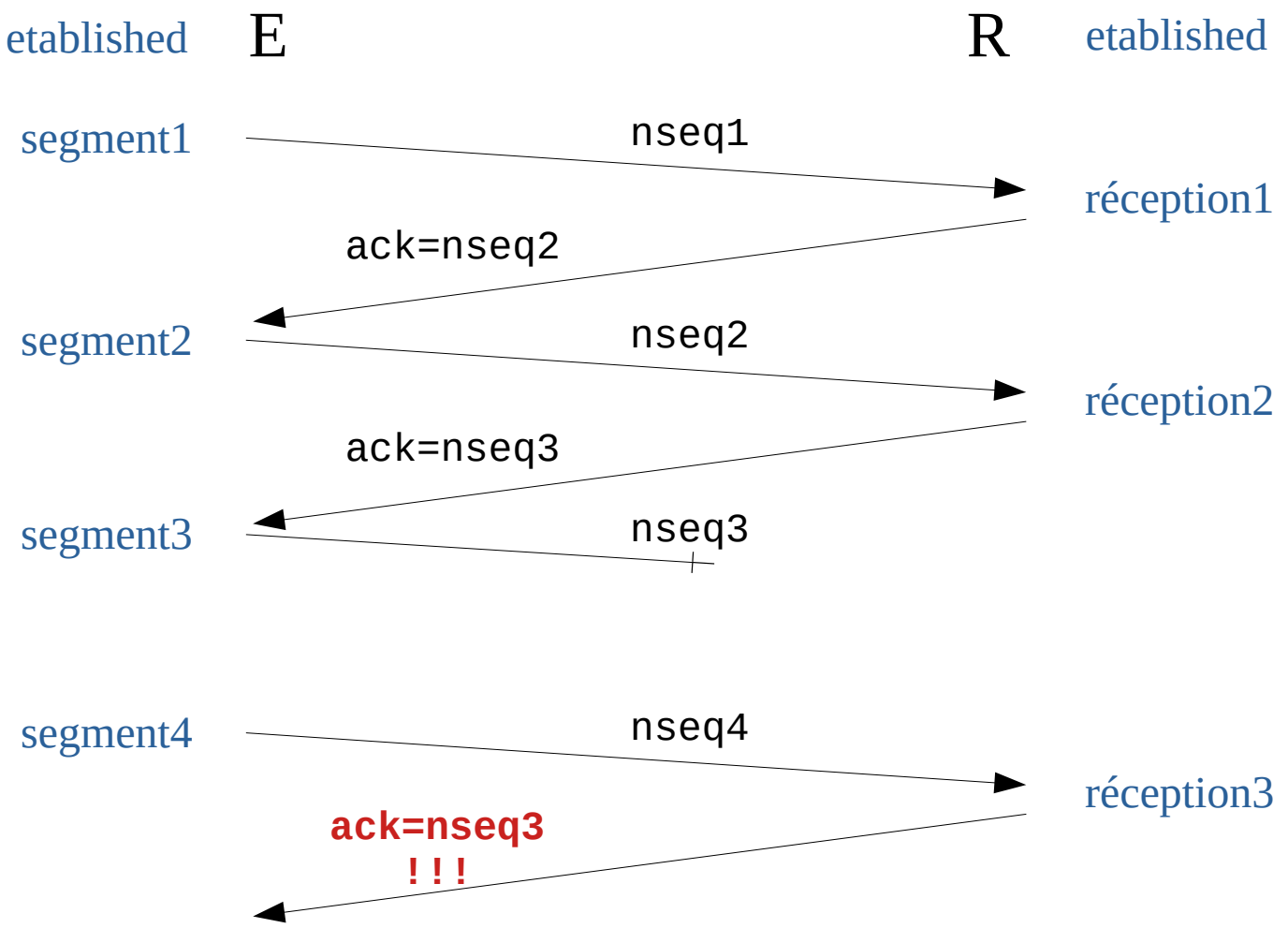
- Chaque segment possède un numéro de séquence
  ⇒ Position des données



R → Données non reçues



R → Données non reçues



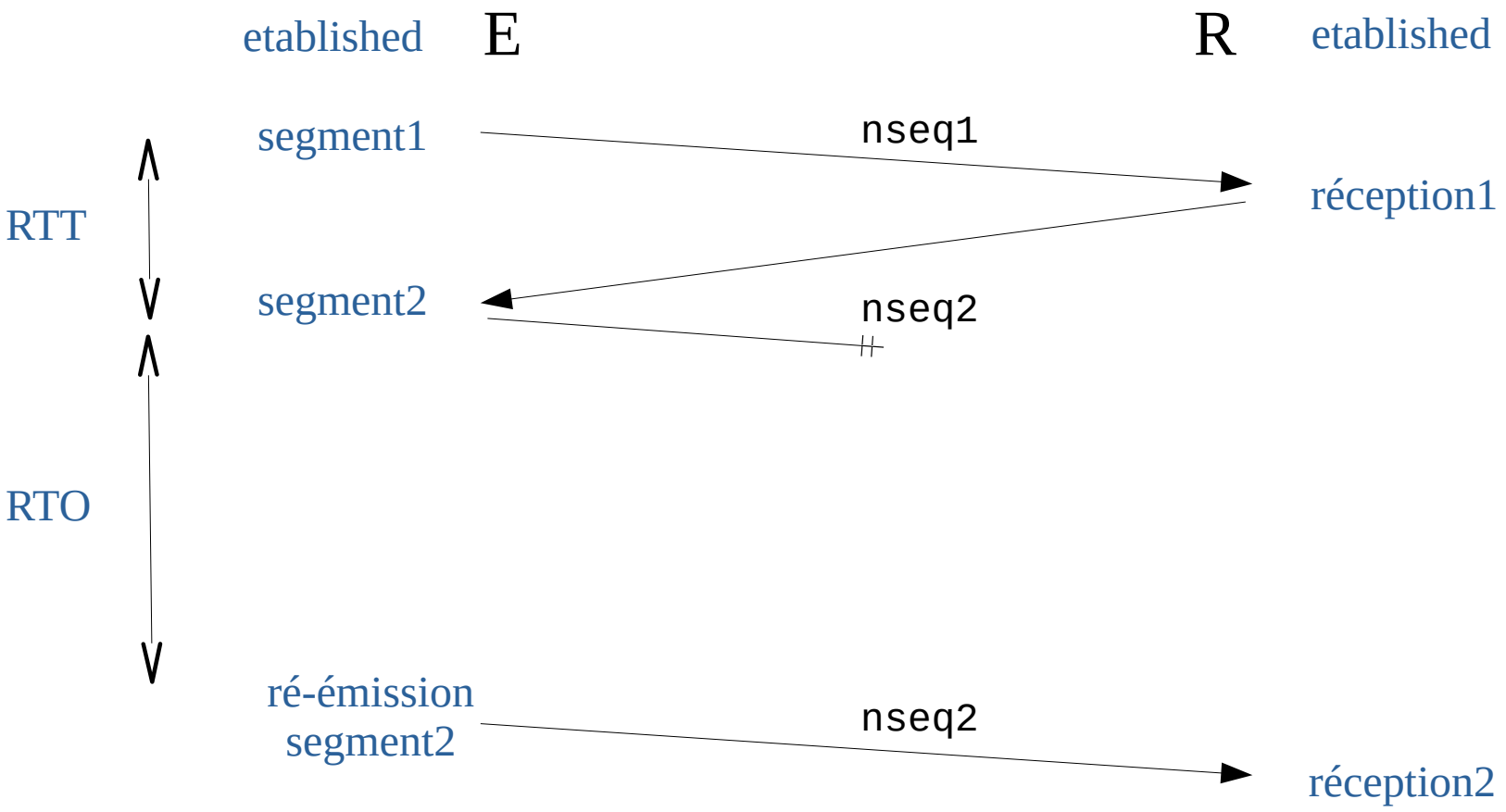
E → Time out

- Détection d'un segment non arrivé à destination
  E ne reçoit pas de message d'acquittement

- Causes possibles de la non réception d'un message :
  - Délai acheminement trop long (routage, congestion ….)
  - Message perdu (congestion, défaillance, ...)
  - Message incorrect (CRC non valide,…)

- Règles TCP : Un segment est considéré perdu si non acquitté :
  a) Après durée au moins égale à Time Out
  b) Après 3 nack identiques

E → Time out (a)


etablished
E
R
etablished
segment1
nseq1
réception1
RTT
segment2
nseq2
RTO
ré-émission
segment2
nseq2
réception2

E → Time out (a)

- ACK non reçu après Time Out
  ⇒ Le segment est re-transmis automatiquement

- RTT Round Trip Time (1)
- RTO Retransmission Time Out (2)

(1) Ré-évaluation régulière (voir plus loin)
(2) Calcul RTO présenté plus loin

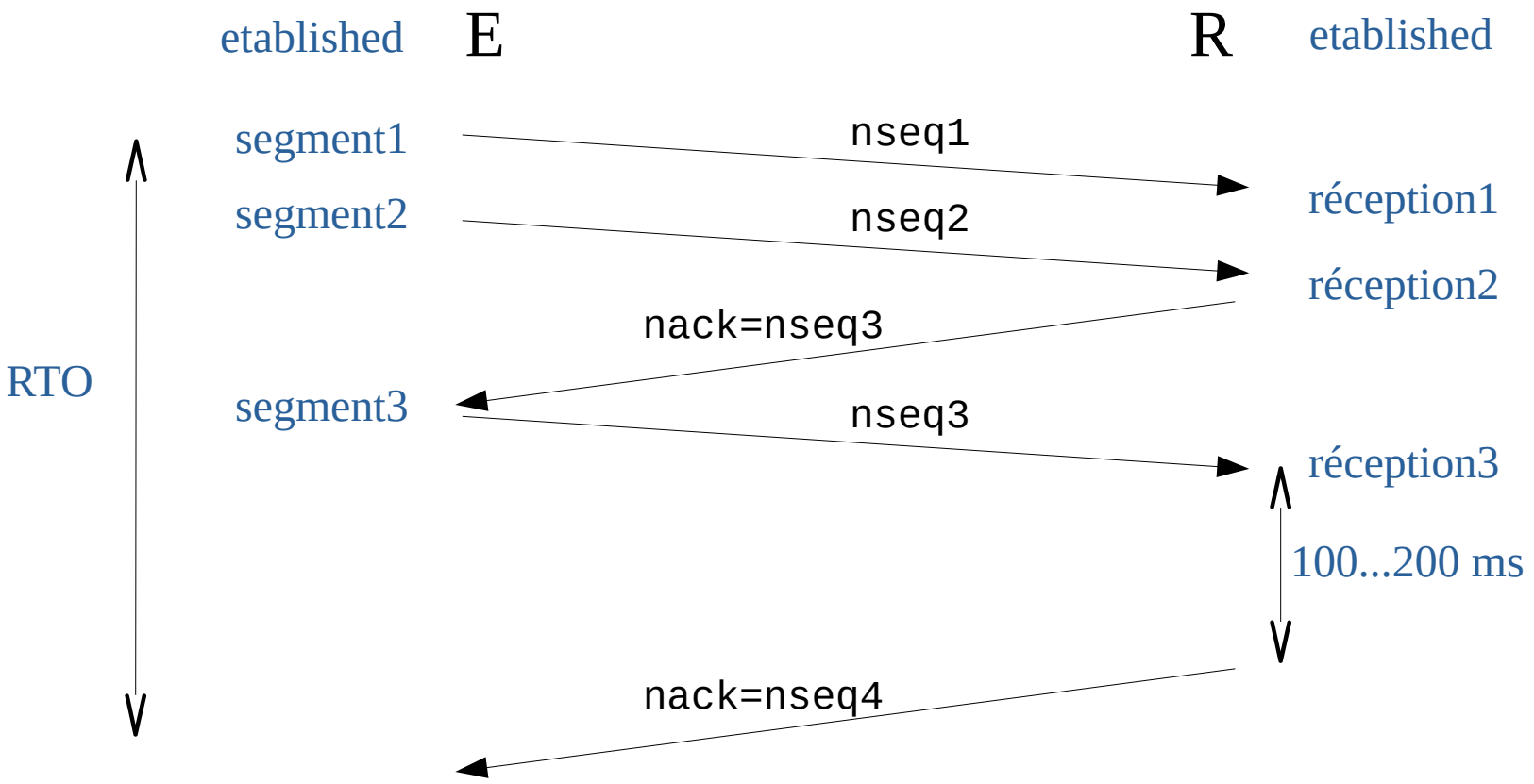
E / R → ACK regroupé

- R ne doit pas émettre un ACK pour chaque segment
- E peut envoyer plusieurs segments

⇒ Optimisation du trafic réseau
Évite surcharge réseau dûe aux messages d'acquittement
Optimise le débit

- Règles TCP ⇒ ACK doit être envoyé :
  - Délai < Time Out ⇒ 100 … 200 ms
  - ACK immédiat si détection de segment manquant
  - ACK immédiat pour chaque second segment reçu

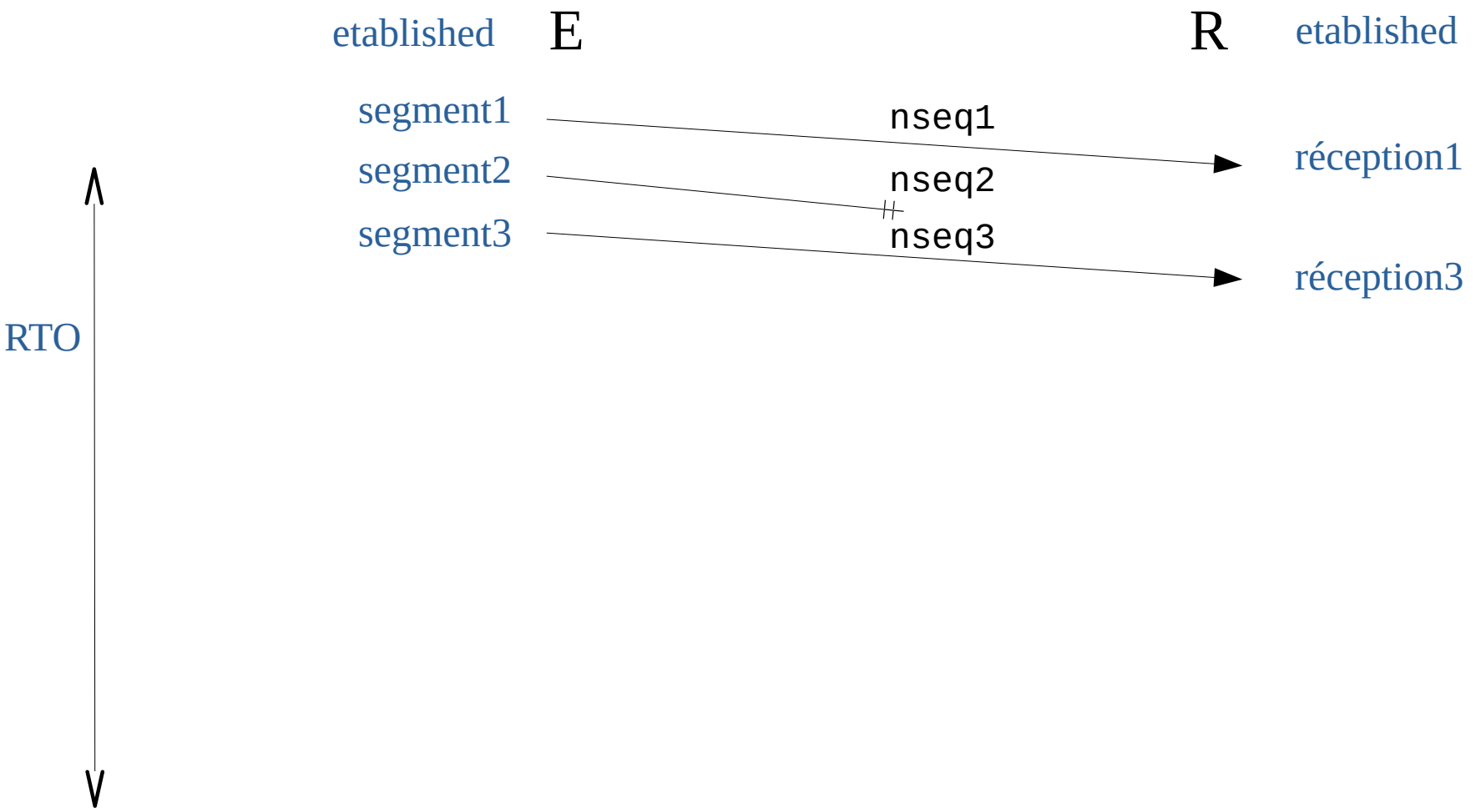
ACK regroupés (1)



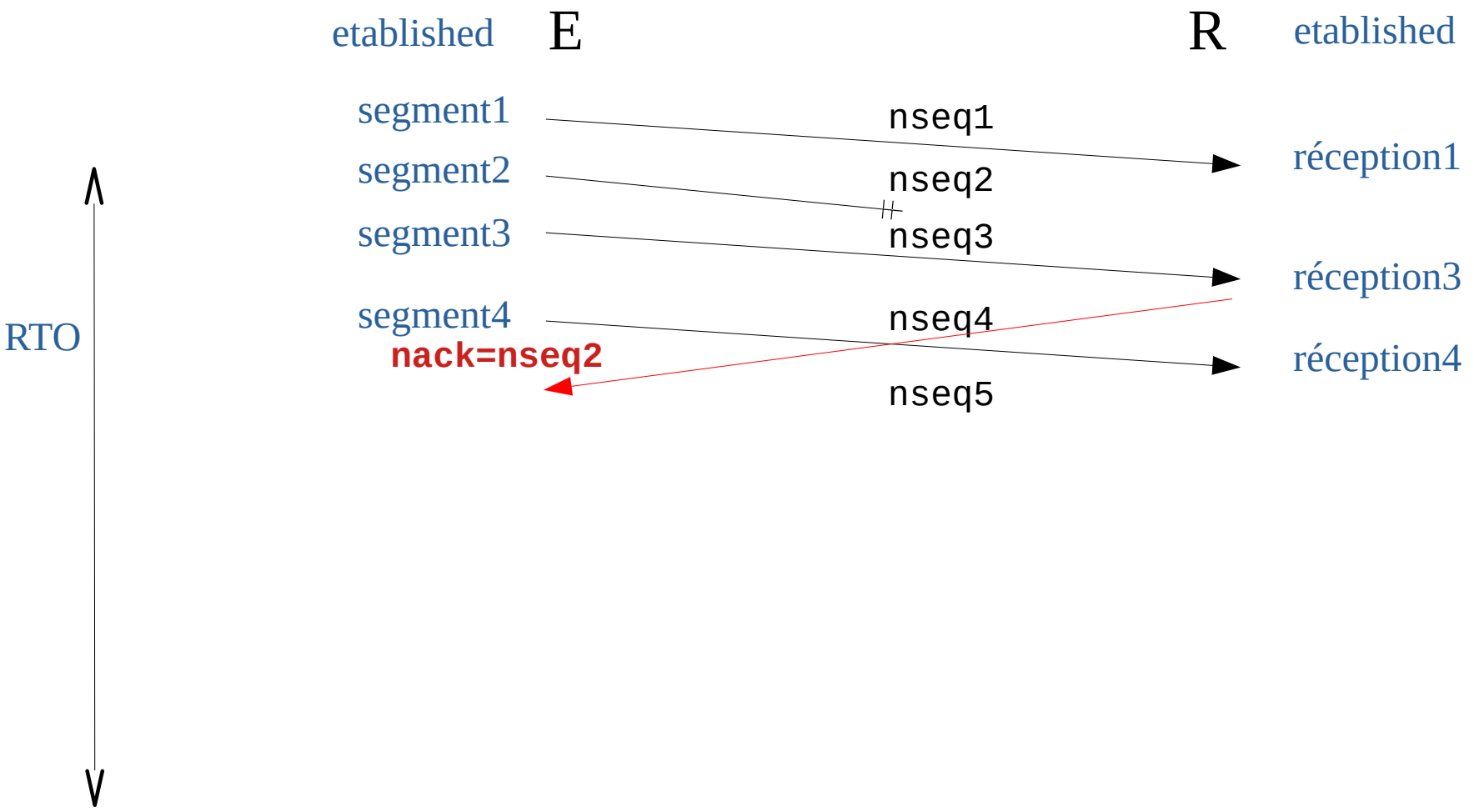
⇒ Fonctionnement normal sans anomalie

- Permet d'augmenter le débit
- Limite la charge réseau / message acquittement

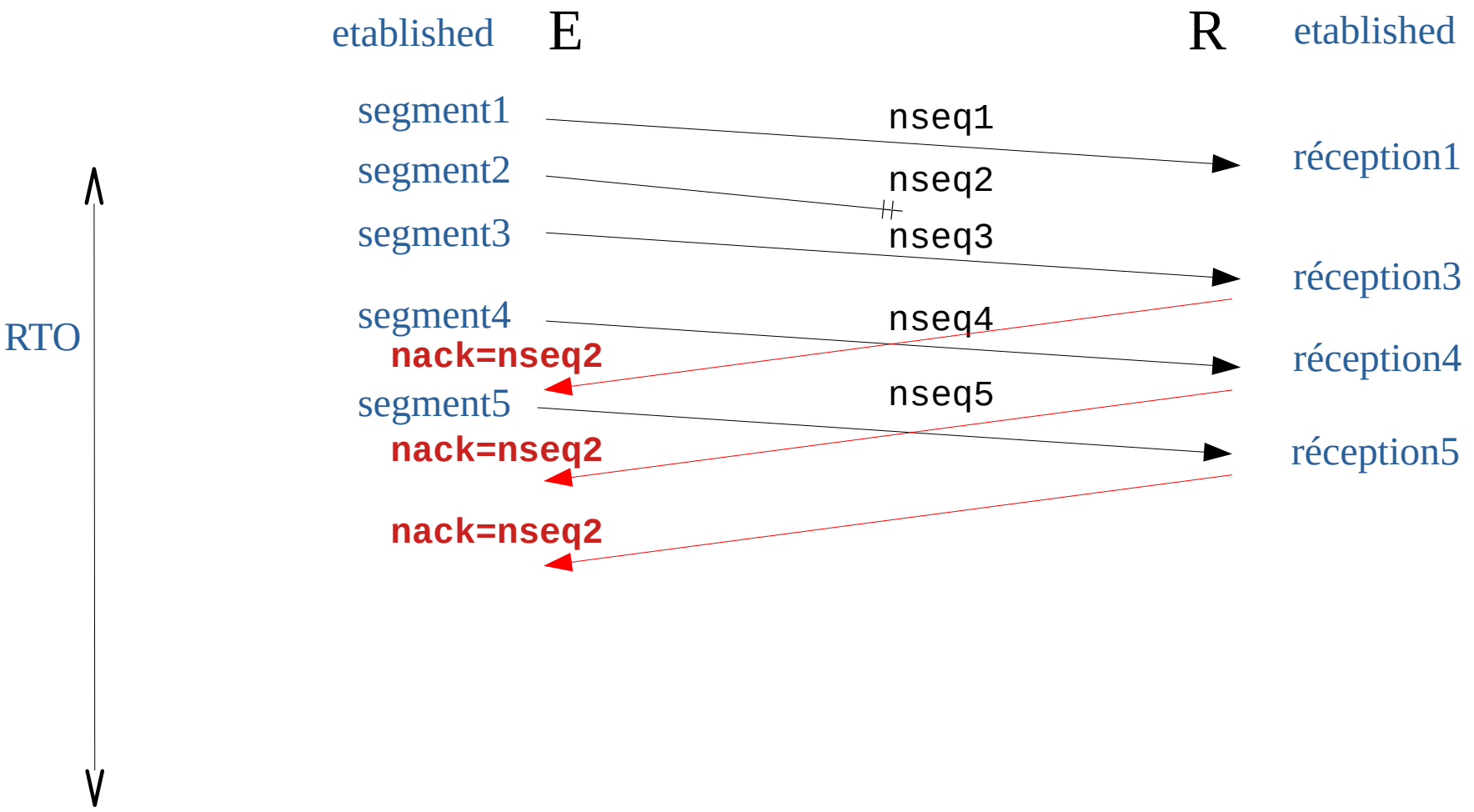
ACK regroupés (2)



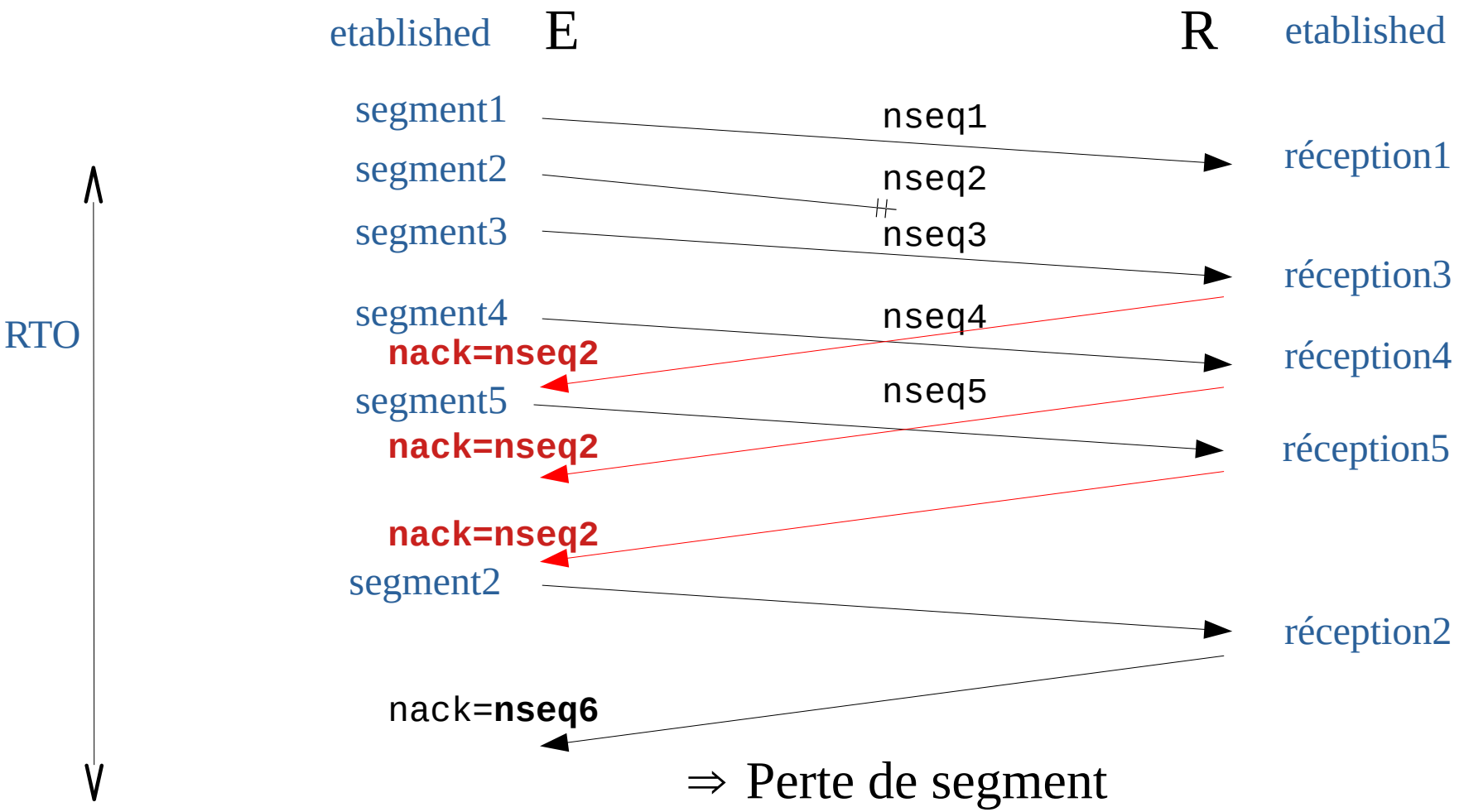
ACK regroupés (2)



ACK regroupés (2)



ACK regroupés (2)



⇒ Perte de segment

Retransmission après 3 ack dupliqués

Calcul du Time Out

- RTT Mesuré à intervalle régulier
  ⇒ Estimation du réseau → Adaptation TCP

➔ Mesure de RTT par Estampille (1)

- RTO Défini par RFC 6298

(1) Voir options TCP

Calcul du Time Out

- Valeurs initiales
  - RTT = 0
  - RTO = 1s
- Tentative de connexion

  1) Envoyer demande de connexion

  2) Si ACK non reçu ⇒ RTO = RTO x 2 ⇒ reprendre en 1)
- RTO max = 64s

  Si connexion non établie avec RTO = RTO max [1]

  Fin de connexion ⇒ Émission RST

[1] Délai maxi < 9mn si aucune réponse

Calcul du Time Out

- **Valeurs initiales** `RTT = 0` `RTO = 1s`

- **Après 1ère mesure** `RTT → mRTT`
  - `SRTT = mRTT`
  - `RTTVAR = mRTT / 2`
  - `RTO = SRTT + max (G, K * RTTVAR)`

- `mRTT` Dernière valeur mesurée pour RTT
- `SRTT` Valeur de RTT moyennée
- `RTTVAR` Variations entre les mesures de RTT
- `G` Granularité de l'horloge `G optimum ≤ 100ms`
- `K` Constante `K = 4`

Calcul du Time Out

- **Mesures suivantes**
  - `RTTVAR = (1-β) * RTTVAR + β * |SRTT - mRTT|`
  - `SRTT = (1-α) * SRTT + α * mRTT`
  - `RTO = SRTT + max (G, K * RTTVAR)`

    - α Constante `α = 1/8`
    - β Constante `β = 1/4`

- **Limites `RTO`**

  `1s < RTO < 60s`

  Si expiration `RTO` ⇒ `RTO min = 3s`

  Réinitialiser le calcul de `RTO`

Calcul du Time Out

- Algorithme de Karn [(1)]
  Analyse pour optimiser a valeur de RTO

- Mesure de RTT
  → Délai entre émission segment et réception ACK

1) Un segment retransmis ne peut servir pour la mesure de RTT
2) Ne pas effectuer de mesures de RTT en parallèle
3) Si mesure de RTT incorrecte, conserver les valeurs courantes
4) Maintenir RTO >> RTT

[(1)] Improving RTT Estimates in Reliable Transport Protocols - 1987

Delayed ACKs

- TCP permet de regrouper les messages ACK

- Pour optimisation, R peut regrouper les messages ACK
  ! ACK doit être transmis avant RTO
  ! R ne sait par avance quand un segment va arriver

  ⇒ R envoi ACK avec un délai maximal RTO min / 2 = 500ms

  ACK valide le dernier segment reçu correctement

Algorithme de Naggle [1]

- Optimisation de l'efficience réseau
- Remarque : Émission TCP d'un segment avec 1 octet de données
  En-tête TCP = 20 octets minimum
  En-tête IP = 20 octets minimum
  Efficacité réseau = 1 / 41 = 2 ,4 %

Exemple : Telnet ⇒ Envoi immédiat de chaque caractères

[1] RFC 895 (1984) updated by RFC 7805

Algorithme de Naggle

- Émission immédiate du premier octet
- Accumulation des octets suivants dans buffer jusqu'à :
  a) Buffer plein
  b) Acquittement du message précédent

  ⇒ Le message suivant comprendra l'intégralité du buffer

- Option TCP_NODELAY ⇒ Permet de désactiver l'algo de Naggle

Algorithme de Naggle

- Problème : Delayed ACKs
- R envoi ACK après délai variable (100...200 ms)

  ⇒ Delayed ACK génère une latence au niveau de l'application

- TCP ne peut optimiser cette latence

  ⇒ Désactiver algorithme de Naggle ou Interdire delayed ACKs ?

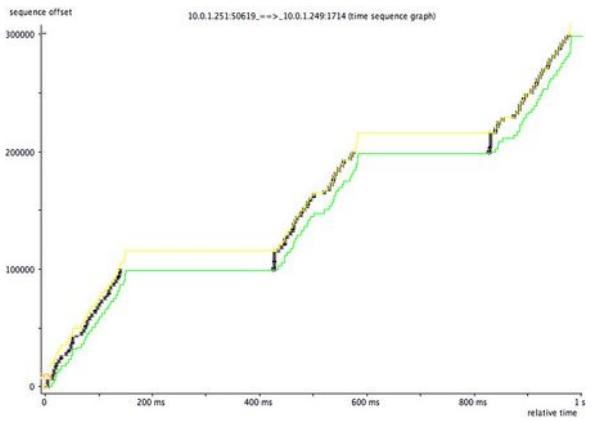
Algorithme de Naggle

- Incidence des ACK regroupés sur l'algorithme de Naggle
  R émet un ACK si :
  a) Second segment reçu
  b) Expiration du délai depuis dernier segment non acquitté

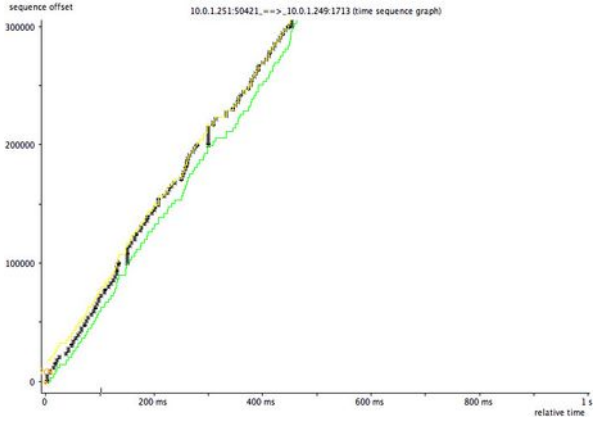
- Proposition de modification
  E peut émettre un segment si
  - Aucune données fournie par l'application n'est en attente
  - 1 seul segment n'a pas été acquitté

Algorithme de Naggle

- Illustration de l'incidence de l'algorithme de Naggle



Transmission / 100 000 octets



Transmission / 99 913 octets

Explication :

100 000 octets Le découpage engendre un nombre de segments impair
⇒ ACK immédiat pour les segments pairs (cas de 99 913 octets!)

Rque : Windows → TCP segment size = 1460 o / Mac OS X → TCP segment size = 1448 o

Flag PUSH

- Demande la transmission "immédiate" des données
  - E → Transmet le contenu du buffer
  - R → Fourni le contenu du buffer de réception à l'application

- Objectif : Éviter une latence au niveau applicatif [1]

Remarque : L'application est censée positionner le flag PUSH lors de l'émission de son dernier segment.

[1] *Force l'envoi immédiat de données de petites dimensions*

Flag URGENT

- Indication de données urgentes
  ⇒ incite R à commuter en mode urgent

- Flag URG ⇒ Spécifie que URG. PTR est significatif
- URGENT POINTER ⇒ Position des données urgentes [1]

  Position des données urgentes indiquées par nseq + urgent pointer

[1] *L'action associée aux données urgentes est gérée par l'application.*

Options

Options

- Champ option complété par du bourrage si nécessaire
- Format d'une option `type [taille] [contenu]`
  - `Type` 1 octet Identifiant de l'option
  - `Taille` 1 octet Nombre d'octets utilisés par l'option
    (Si l'option utilise un contenu)
- TCP doit ignorer les options non implémentées,
  sans générer d'erreur

Options limitées à 1 octet

- Type = 0 Indicateur de fin de la liste des options
  Utilisé si le dernier octet d'option n'est pas aligné
  Ne doit pas être utilisé entre 2 options

- Type = 1 No-operation
  Peut être utilisé entre 2 options (pour alignement)
  ! Alignement des options non imposé par la norme

Maximum segment size [1]

- Type = 2
  Taille = 4
  Data = Max Segment Size

Si cette option est présente

Indique la dimension maximale des données pouvant être traitées
Valeur par défaut = 536 → 576 - 40

MSS doit être transmis à la création de la connexion [2]

[1] RFC 6691

[2] MSS ne peut être supérieur à la valeur de MTU du réseau de l'hôte

Maximum segment size

- `Eff.snd.MSS = min( SendMSS+20, MMS_S )`
  `- TCPhdrsize - IPoptionsize`
- `SendMSS` Valeur transmise par hôte distant (536 par défaut)
- `MMS_S` Valeur max. acceptée par la couche transport(1)
- `TCPhdrsize` Dimension en-tête TCP (>20 si options)
- `IPoptionsize` Dimension de toute option IP qui peut transiter par la couche TCP

Remarque : La valeur MSS émise utilise des en-têtes sans option
⇒ L'émetteur doit ôter les options utilisées

(1) TCP peut définir la dimension MMS_S ou utiliser (par défaut) MTU

RTTM : Round-Trip Measurement

- Type = 8
  Taille = 10
  Data = Timestamp Value (4 octets)
  Timestamp Echo Reply (4 octets)

- Timestamp Echo Reply valide uniquement si ACK

- Delayed ACK ne doit pas être activé si RTTM inclus dans le segment

Remarque : RTTM non valide si segment retransmis

TCP Window Scale Option (1)

- Le champ Window défini dans l'en-tête est limité à 32 bits
- Window Scale Option permet de définir un multiplicateur pour la valeur du champ Window

  ⇒ Option valide uniquement si 2 hôtes ont transmis l'option à la connexion (SYN)

- Type = 3
  Taille = 3
  Data = shift count

Remarque : shift count = 0 ⇒ Autorise émetteur à utiliser option
⇒ Valeur transmise = 1

RFC 5681

Congestion avoidance

Taille des fenêtres

- Champ "Window" Taille de la fenêtre de réception
  - Valeur numérique non signée codée sur 16 bits
  - Dimension de fenêtre de réception définie par l'émetteur

- Permet de limiter la charge réseau aux données exploitables

- Fenêtre de réception = Fenêtre "glissante"
  ⇒ Indique à l'émetteur la quantité maximale de données qui peuvent être transmises avant validation
  ⇒ Valeur spécifiée / ACK transmis dans le même segment

Remarque : voir option MSS

Taille des fenêtres

- Restrictions
- La taille de fenêtre ne peut être diminuée sans validation
  → Acceptation de la nouvelle valeur par émetteur pour les données suivantes (restrict window)

- Tout poste TCP doit pouvoir accepter au moins 1 octet de données

- E doit transmettre un segment à intervalle régulier, même si la taille de la fenêtre de réception est égale à 0 → Période 2 mn
- R doit valider tout segment de données reçu avec `nseq` correspondant aux données attendues

TCP congestion control (1)

- TCP défini 4 algorithmes afin d'optimiser l'efficacité du réseau
  - Slow-start
  - Congestion avoidance
  - Fast retransmit
  - Fast recovery

- RFC 5681 ⇒ Définition des algorithmes

(1) RFC 5681

Définitions

- **SMSS** Sender Maximum Size Segment
- **RMSS** Receiver Maximum Segment Size
- **Full-Sized Segment** Segment avec SMSS octets de données
- **rwnd** Receiver Windows
- **cwnd** Congestion Window
- **IW** Initial Window. Valeur initiale de cwnd
- **LW** Loss Window. cwnd après détection de perte de données
- **RW** Restart Window. cwnd après inactivité
- **FLIGHT SIZE** Taille des données émises sans ACK
- **DUPLICATE ACKNOWLEDGEMENT**
  Valeur de ACK dupliquée

Algorithme Slow Start

- Objectif : Ne pas surcharger le réseau au démarrage

L'algorithme Slow-start s'applique :

a) Au démarrage d'une connexion

b) Après une période d'inactivité (Idle period [1])

c) Après Time Out

⇒ S'applique si cwnd < sstresh

[1] Idle period = Pas de segment reçu pendant durée > RTO

Algorithme Slow Start

- cwnd E → Fenêtre de congestion
  Volume des données pouvant être transmises avant ACK
- rwnd R → Fenêtre de réception
  Volume des données pouvant être reçues
- sstresh Slow Start Treshold
  Valeur initiale choisie arbitrairement
  sstresh mini = segment size maxi (with options) ⇒ MSS
  sstresh doit être modifiable pour s'adapter au réseau

Initial Window (1)

- Taille de la fenêtre de congestion au démarrage

  `IW ≤  min ( 4 * MSS,  max ( 2 * MSS, 4380 ) )` (2)

  Remarque : 4380 octets = 3 paquets IP avec en-têtes minimales

```
MSS ≤ 1095           ⇒ win ≤ 4 * MSS
1095 ≤ MSS < 2190    ⇒ win ≤ 4380
2190 ≤ MSS           ⇒ win ≤ 2 * MSS
```

- Discussion dans RFC 3390 sur valeur optimale
  - ➔ Si bande passante élevée Augmenter IW
  - ➔ Si charge réseau élevée Diminuer IW

(1) RFC3390 (2) Valeur recommandée

Algorithme Slow Start

- Tant que `cwnd < sstresh`

  Pour chaque ACK reçu

  `cwnd = cwnd + min ( N, SMSS )`

  avec N = nb octets validés par le dernier ACK

- Fin algorithme slow-start si
  - `cwnd > sstresh`
  - Congestion réseau détectée

- Rappel :

  A tout instant → Données émises `maxi = min ( cwnd, rwnd)`

Algorithme Congestion Avoidance

- Objectif : Ne pas surcharger le réseau
  S'applique lorsque la communication est établie

- si `cwnd ≥ sstresh`
  Pour chaque RTT
  `cwnd = cwnd + SMSS` (1)

Si perte de segment

- Si détection de paquet perdus via RTO
  et aucune retransmission n'a été effectuée

  `sstresh = max ( FlightSize / 2, 2 * SMSS )`
  `FlightSize` = Taille des données émises et non encore validées par ACK

  cwnd = LW
  LW = Loss Window = 1 full-sized segment(Taille maxi)

! Si une retransmission a déjà été effectuée
  ⇒ `sstresh` reste inchangé

Fast Retransmit Algorithme → R

- Si R reçoit un segment dans le désordre (détection de segment manquant)
  ⇒ Envoi immédiat ACK pour indiquer le segment manquant
- A chaque réception de segment dans le désordre
  ⇒ Duplicate ACK
- Si Segment manquant reçu
  => ACK / dernier élément correctement reçu

Fast Retransmit Algorithme → E

- Si E reçoit des duplicate ACK

➔ Si duplicate `ACK ≤ 2` ⇒ E continue les émissions normalement

➔ Si duplicate `ACK = 3`
  - `sstresh = max ( FlightSize / 2, 2 * SMSS )`
  - E réémet le segment manquant
  - `cwnd = sstresh + 3 * SMSS`

➔ Pour chaque duplicate `ACK` reçu (> 3)
  - `cwnd = cwnd + SMSS`

Fast Retransmit Algorithme → E
Fast Recovery

- Si E reçoit ACK > duplicate ACK ⇒ Reprise normale

➔ E envoi 1 segment avec des données non déjà émises

  - A la réception de ACK du dernier segment émis

  ➔ `cwnd = sstressh` ⇒ Fast recovery

Fast Recovery

- Après rétablissement "normal" de la connexion
  - `sstresh = max ( FlightSize / 2, 2 * SMSS )`

  ⇒ Évolution "douce" des valeurs

- Reprise de congestion avoidance
  - `cwnd = cwnd + SMSS`
  - `cwnd = min (cwnd , rwnd)`

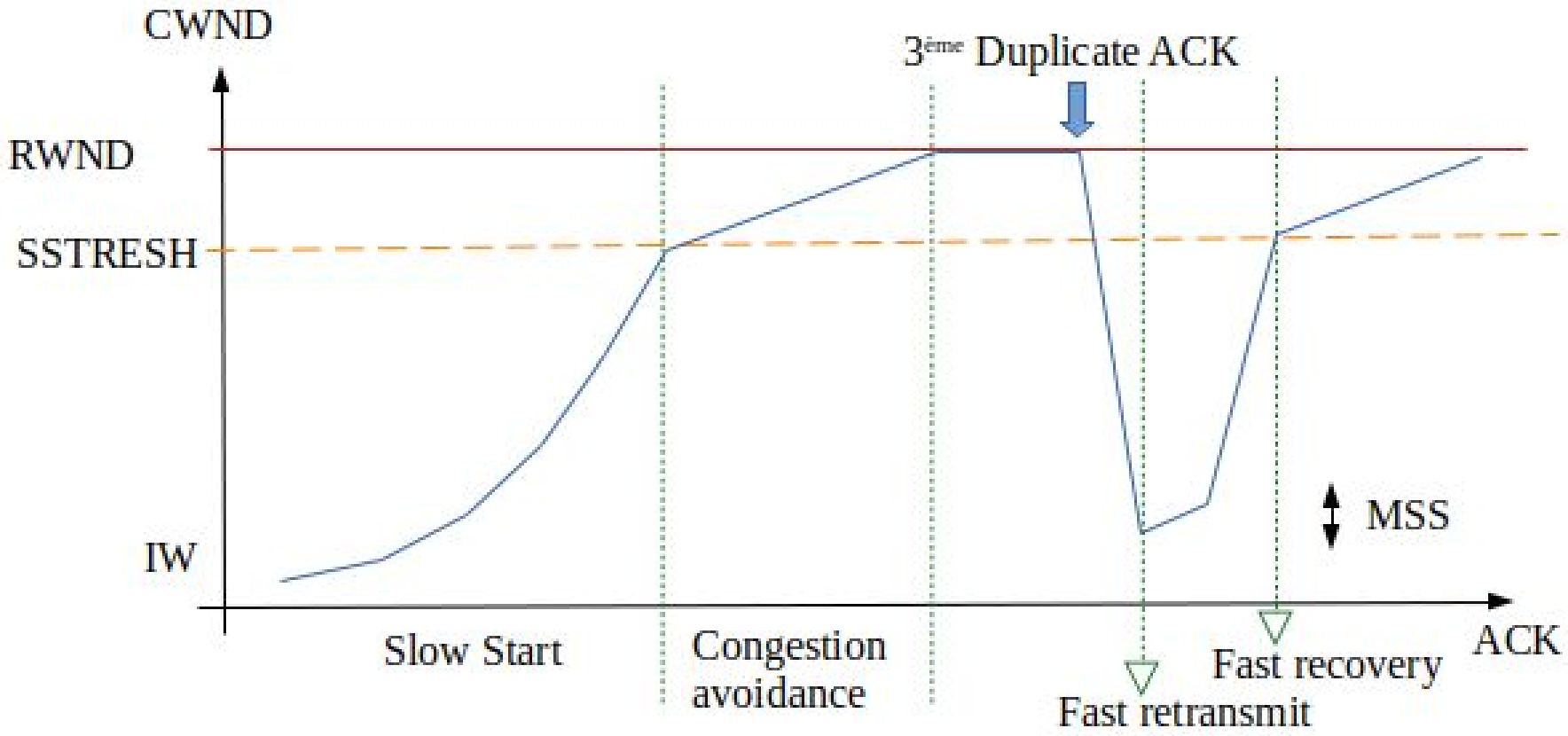
Slow Start / Idle Period

- Après Idle Period, E doit utiliser l'algorithme de Slow Start

  `cwnd = RW`

  avec `RW = Restart window = min ( IW, cwnd)`

Exemple


CWND
3ème Duplicate ACK
RWND
SSTRESH
IW
MSS
Slow Start
Congestion avoidance
Fast retransmit
Fast recovery
ACK

Option SACK (1)

- Selective Acknowledgement Options
  ⇒ Duplicate ACK spécifie le plus ancien segment non reçu
  Pas d'indication sur les segments reçus ultérieurement
- SACK-permitted Transmis lors de la création de la connexion
- SACK Utilisée lors de la perte de segment

! SACK autorisé uniquement si SACK-permitted reçu lors de l'établissement de la connexion

(1) RFC 2018

Option SACK-permitted

- Type = 4 ⇒ Indique que l'hôte peut utiliser l'option SACK
  Taille = 2

Option SACK

- Type = 5
  Taille = variables → Selon données transmises
  Data = Left edge of 1st block
  Right edge of 1st block
  …
  Left edge of nth block
  Right edge of nth block

Option SACK

- ACK Valeur du plus ancien segment manquant
  ⇒ Duplicate ACK

- Block Segments reçus
  - Left edge of block = nseq du bloc
  - Right edge of block = nseq du segment suivant non reçu

  Remarque : SACK est souvent utilisé conjointement avec l'option Timestamp

Option SACK

- Ordre des blocs transmis

➔ Le premier bloc spécifié correspond à celui ayant déclenché le message ACK
⇒ `ACK = nseq` du plus ancien segment non reçu
Left edge of 1er block = nseq du dernier segment reçu

➔ Si réception d'un segment manquant
⇒ Il est inséré dans le bloc adjacent

RFC 3168

Explicit Congestion Notification

Détection de la congestion

- Congestion détectée si suppression de paquets IP
  - ➔ RTO Émetteur
  - ➔ ACKRécepteur

- Ajouts de bits supplémentaires pour anticiper la congestion
  - TCP CWR : Congestion Window Reduced (1)
    ECE : ECN Echo
  - IP ECT : ECN Capable Transport
    CE : Congestion Experienced

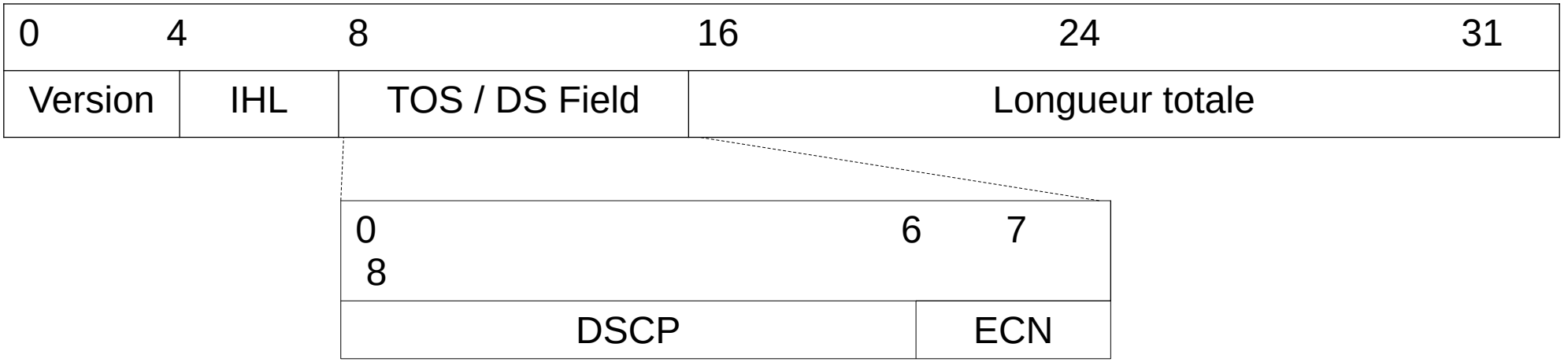
(1) Les drapeaux CWR & ECE doivent être positionné lors de la création de connexion

Indication de congestion dans IP (1)

- Ajout des bits ECN pour indiquer la congestion
  - ECN doit être compatible avec mécanismes existants
  - Gestion de congestion exploitable uniquement si les échanges se poursuivent pendant la phase de congestion
  - En général, communication asynchrone (retour = ACK)
    ⇒ Les chemins aller et retours peuvent être distincts
  - Les routeur sont plus efficients pour traiter les informations comprises dans l'en-tête IP, plutôt que les options

(1) RFC 3168

Indication de congestion dans IP



- DSCP : Differentiated Services Codepoint
- ECT CE ⇒ noms RFC 2481 obsolètes → ECN bits

0	0	ECN non géré
0	1	ECT(1) } Indication → ECN capable
1	0	ECT(0) }
1	1	Indication de congestion (Modifié par routeur)

Active Queue Management

- Active Queue Management [1] permet d'anticiper la détection de congestion avant la suppression des paquets
  Seuil de déclenchement [2] avant saturation de la queue de messages
  - ➔ Si le mécanisme CE est supporté
    ⇒ Marquer les paquets avec CE au lieu de les supprimer
    Notification de réduction du trafic demandé
  - ➔ Si le mécanisme CE n'est pas disponible
    ⇒ Suppression de paquets pour signifier a congestion
  - ➔ Lorsque le routeur est saturé
    ⇒ Suppression de tous les paquets

[1] RFC 7567

[2] Utiliser une valeur moyenne pour limiter les fluctuations

CE et fragmentation

- Si le paquet est marqué CE, alors le flag DF doit être positionné
- Si le paquet a été fragmenté, et que l'un des fragments est marqué CE, alors le marquage doit être reporté sur le paquet après réassemblage
- Le mécanisme CE ne doit par être appliqué si un paquet est supprimé pour une autre raison que la congestion (erreur checksum, …)

Indications TCP

- Bits ECE & CWR sont ajoutés au protocoles TCP
- Si marquage CE du paquet par la couche IP
  ⇒ Le bit ECE du segment est activé pour le prochain message ACK
- Si segment reçu avec flag ECE, alors gestion de congestion activée
  ⇒ Retour avec flag CWR → déclenchement slow-start
  Informe R que la fenêtre de congestion a été réduite

Remarque : Pour activer le mécanisme dans TCP, indication lors de la phase de connexion :

E → `SYN / ECE = 1 / CWR = 1`

R ← `ACK / ECE = 1 / CWR = 0`

Compléments Linux

Gestion réseau

- Pseudo répertoire `/proc/net`

⇒ Pseudo fichiers réseau

- `arp` ⇒ `Table arp`
- `dev` ⇒ `Statut des interfaces`
- `route` ⇒ `Table de routage`
- `sockstat` ⇒ `statistiques sur les sockets`
- `...`

Gestion IP

- Pseudo répertoire `/proc/sys/net/ipv4`

  ⇒ Fichiers de configuration de la communication ipv4
  - `ip_default_ttl`
  - `ip_forward`
  - `ip_unprivileged_port_start`
  - `...`

Gestion TCP

- Pseudo répertoire `/proc/sys/net/ipv4`

⇒ Fichiers de configuration des connexions TCP

- `tcp_base_mss`
- `tcp_frto`
- `tcp_keepalive_time`
- `tcp_sack`
- …

Informations ⇒ `man tcp`

Outil analyse réseau

- Fichiers de configuration & pages `man`
- Outils linux de base
  - `ip ...`
  - `tcpdump`
- Packages complémentaires
  - `netstat` ⇒ `net-tools`
  - `nmap`
  - `netperf`
  - `bmon`
  - `darkstat`
  - `wireshark` ⇒ Analyse réseau détaillée