ASR M1 Crypto.

La couche Transport

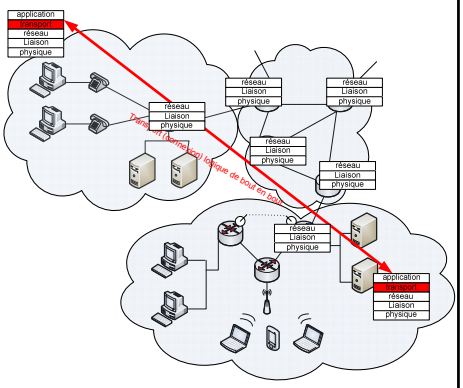
Adlen Ksentini

Introduction

- But :
  - Comprendre les principes de fonctionnement des protocoles de la couche Transport d'Internet
    - Numéro de port/processus
    - Transport fiable des données
  - Comprendre :
    - TCP : mode connecté
    - UDP : mode non-connecté

Les services de la couche Transport

- Assure un transport d'un flux (potentiellement infini) d'octets entre processus fonctionnant sur deux machines distantes
- Les protocoles de transport fonctionnent sur les systèmes terminaux
  - Coté émetteur : découper les messages en segments et les passer à la couche Réseau
  - Coté récepteur : rassembler les segments en messages, et les passer à la couche Application



Couche Transport vs. couche Réseau

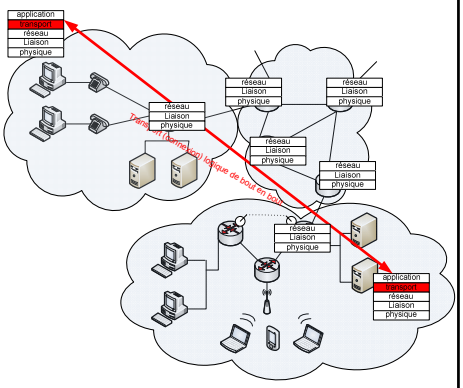
- Couche Réseau : communication entre machines
- Couche Transport : communication entre processus
  - Se base sur (et améliore) les services de la couche Réseau
  - Par ex. contrôle (détection uniquement ou non) d'erreur sur les données

Analogie avec une famille :
12 enfants envoyant des lettres à 12 autres enfants

- Processus = enfant
- Messages couche Appl. = les enveloppes contenant les lettres
- Hôte = maison
- Protocole de transport = Toto et Titi qui échange une lettre
- Couche Réseau = les services de la poste

Les services de transport d'Internet

- TCP : transport fiable, assure que l'ordre de transmission des données est celui d'émission
  - Contrôle de congestion
  - Contrôle de flux
  - Établissement d'une connexion
- UDP : transport non fiable
- Services non offerts
  - Garantie du délai de bout-en-bout
  - Garantie de la bande passante (débit)
  - Différenciation de service

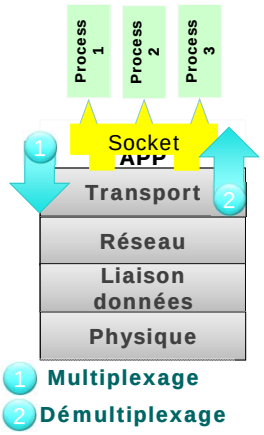


Ports

- **Couche Réseau:** les adresses IP désignent les machines entre lesquelles les communications sont établies.
  - Cependant doit pouvoir identifier les processus s'exécutant sur cette machine.
- **Couche Transport:** l'adressage de processus est effectué selon un concept abstrait: **le numéro de port**
  - Les processus sont créés/détruits dynamiquement sur les machines
  - Il faut pouvoir remplacer un processus par un autre (exemple reboot) sans que l'application distante ne s'en aperçoive
  - Il faut identifier le processus destinataire selon le service offert, sans connaître l'identité locale du processus qui le met en œuvre

Ports (suite)

- La transmission d'un message se fait sur la base d'un port source (celui du processus source) et un port destinataire (celui du processus de destination)
  - L'émission se base sur le port Source (port de la machine locale) … Multiplexage
  - La réception se base sur le port Destination (port de la machine distante) … Démultiplexage
- Interface de programmation offert par le système d'exploitation (socket, ...).
  - Port source choisit automatiquement par le système.
- Les accès aux ports sont généralement bloquants, les transmissions sont asynchrones (files d'attente).

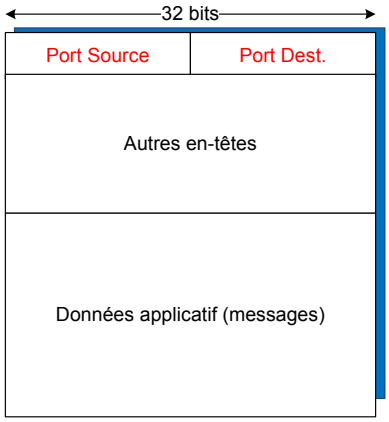


1 Multiplexage
2 Démultiplexage

Fonctionnement du démultiplexage

- Un hôte reçoit un datagramme IP
  - Chaque datagramme contient une adresse IP source et une adresse IP destination
  - Chaque datagramme transporte un seul segment de la couche Transport
  - Chaque segment contient le port source et le port destination (well-known port pour les appli. connues)
- Un hôte, utilise alors les @IP et les numéros de port pour remettre le segment au processus (via la bonne socket)

Format d'un segment TCP/UDP



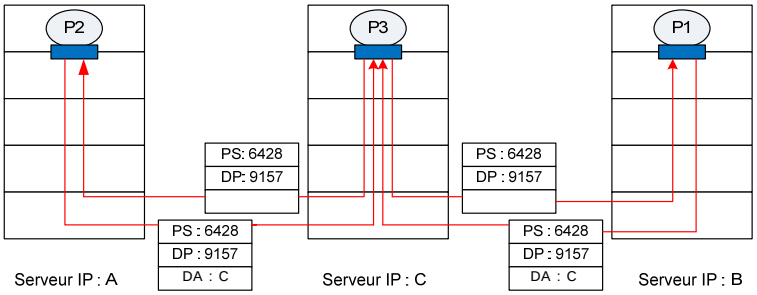
En mode non connecté

- Création d'une socket associée au numéro de port local spécifié :

```
DatagramSocket socket1 = new DatagramSocket(9001);
DatagramSocket socket2 = new DatagramSocket(9002);
```

- La socket UDP est identifiée par : (@IP dest., numéro de port dest.)
- Lorsqu'une machine reçoit un segment UDP
  - Vérifier le port de destination spécifié dans le segment
  - Passer le segment UDP au processus qui est associé à ce port
- Tous les datagrammes IP reçus sur une machine et pour un numéro de port donné peuvent avoir des @IP source ou/et numéro de port source totalement différents

Démultiplexage en mode non connecté

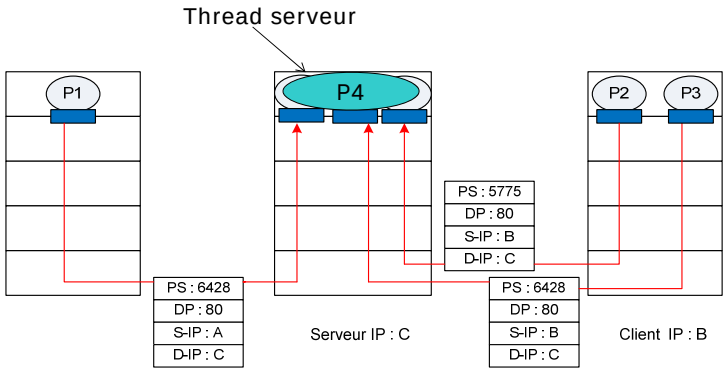


Port source : il sert de port de destination lors de la réponse

Démultiplexage en mode connecté

- Une connexion TCP est identifiée par :
  - L'@IP source
  - Le numéro de port source
  - L'@IP destination
  - Le numéro de port destination
- La machine réceptrice utilise ces 4 composants pour remettre les données au bon processus
- Le serveur peut supporter plusieurs connexions TCP en parallèle
  - Chaque connexion est identifiée par ses 4 composants
  - Par exemple : un serveur web a une (ou plusieurs) socket(s) pour chaque client connecté
    - Dans HTTP non persistant, une connexion TCP pour chaque ressource d'une requête HTTP

Démultiplexage en mode connecté (suite)

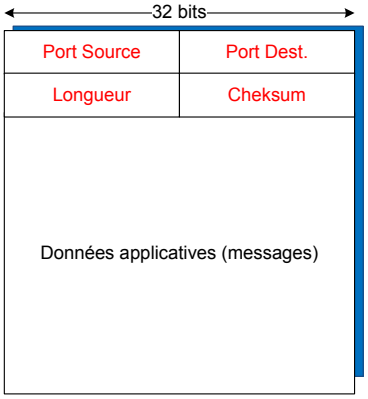


UDP – User Datagram Protocol

- Service Best-effort ("au mieux"), les segments UDP peuvent :
  - Se perdre
  - Arriver dans le désordre
- Chaque segment UDP est traité indépendamment des autres
  - Pas d'établissement de connexion entre l'émetteur d'un segment UDP et le récepteur du segment UDP
- Pourquoi utiliser UDP ?
  - Pas d'établissement de connexion (cela diminue le délai)
  - Segment allégé (faible taille des en-têtes, débit efficace augmenté)
  - Pas de contrôle de congestion, l'émetteur UDP peut utiliser toute la bande passante disponible
  - Peu de traitement

UDP (suite)

- Utilisé pour les applications de streaming multimédia
  - Tolérant aux pertes
  - Avide de bande passante
- Exemple d' application :
  - DNS
- Pour assurer la fiabilité des données transportées sur UDP
  - Rajouter le contrôle au niveau de la couche Application



Checksum UDP

Emetteur

- Le segment est vu une séquence de nombres entiers de 16 bits:
  - y compris les champs d'en-tête
  - sauf le champ Checksum qui est supposé nul
- Checksum:
  - l'inverse du résultat de l'addition (en complément à un) du contenu du segment
- Mettre la valeur obtenue dans le champ Checksum

Récepteur

- Calcul du checksum de la totalité du segment reçu (checksum reçu inclus)
- Vérifier si le résultat du checksum calculé est nul
  - Non: détection d'une erreur (c'est sûr)
  - Oui: pas d'erreur détectée (n'élimine pas complètement la possibilité d'existence d'une erreur)

Checksum UDP: exemple

Ex.: addition de deux entiers de 16 bits en complément à un, puis on prend son complément

Somme en complément à 1 :	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Ajout de la retenue	(1)	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Résultat de la somme	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum	1	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

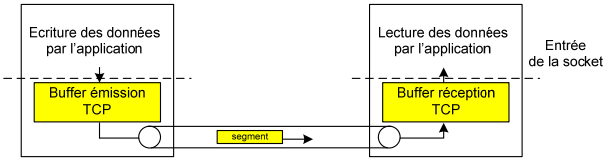
complément à 1 du résultat (c'est l'inverse)

Code d'entiers relatifs en complement à un

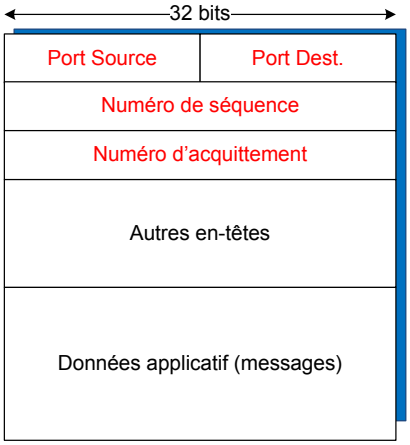
- Exemple du codage sur 3 bits
  - 3 : 011
  - 2 :010
  - 1 : 001
  - 0 : 000 (111)
  - -1 : 110
  - -2 : 101
  - -3 : 100
- Facile à implémenter électroniquement
  - L'addition de deux entiers relatifs est similaire à l' addition de deux entiers
  - Exemple : 3 + (-1) := 011 + 110 = [1] 001 = 010 := 2
- Peu dense : 2 zéros !
- Le codage de l'inverse d'un nombre est l'inverse du codage du nombre. (1 : 001 et -1 : 110)

TCP

- RFC : 793, 1122, 1323, 2018, 2581
- Fiable, livraison des octets dans l'ordre où ils ont été émis
- Etablissement d'une connexion
  - Echange de messages de contrôle pour initialiser l' état du récepteur et celui de l'émetteur
    - MSS (Maximum Segment Size) : taille maximum des segments négociée à l'établissement de la connexion
  - Avant l'envoi des messages de données
- Contrôle de congestion
  - Contrôle la capacité du réseau à accepter les données
  - Fenêtre d'émission
- Contrôle de flux
  - Contrôle la capacité du récepteur à accepter les données
- Utilise des buffers au niveau de l'émetteur et du récepteur

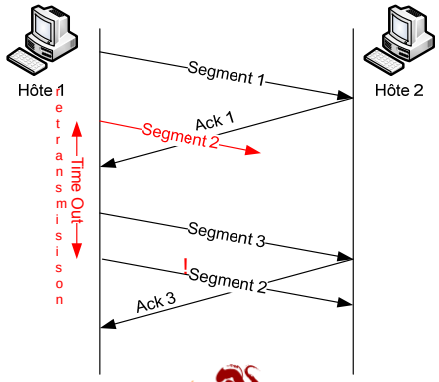


TCP – Format du segment



TCP - Fiabilité

- Segment est-il bien arrivé ?
  - Accusé de réception (Acquittement)
  - Timeout (temporisateur)



Retransmission timeout (RTO)

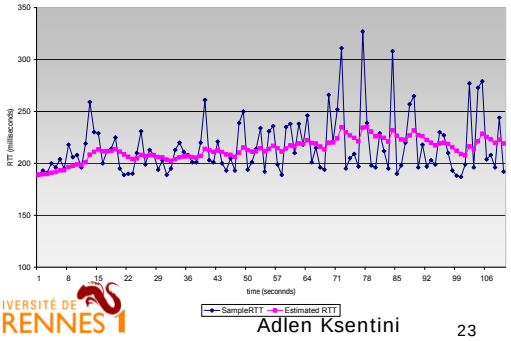
- Comment le RTO est-il défini ?
  - Supérieur à un RTT, mais les RTTs varient
    - Trop petit => des retransmissions inutiles
    - Trop grand => réaction tardive aux pertes
- Comment estimer le RTT ?
  - Mesurer ce RTT en temps réel
  - Ces mesures peuvent varier
    - Utiliser une moyenne

Estimation du RTT

Moyenne pondérée exponentiellement :

`RTT_estimé(t) = (1-α)*RTT_estimé(t-1) + α*RTT_mesuré(t)`

- Influence des mesures antérieures du RTT
- Valeur typique de α = 0.125



RTO (suite)

- Définition du RTO
  - RTT_estimé plus une valeur de garde
    - La variation du RTT estimée
  - Calculer la variance du RTT_estimée par rapport au RTT_mesuré

`DevRTT(t) = (1-β)*DevRTT (t-1) + β (RTT_estimé(t) - RTT_mesuré(t))`

- Alors le timeout est défini par :

`RTO(t) = RTT_estimé(t) + 4*DevRTT(t)`

Livraison dans l'ordre – N° de séquence

- La conservation de l'ordre des octets est nécessaire pour recomposer le message

L'acheminement dans Internet

=> commutation par paquets

  - Ne garantit pas l'ordre
    - Chaque segment TCP est acheminé séparément
  - Des chemins différents pour des segments TCP successifs

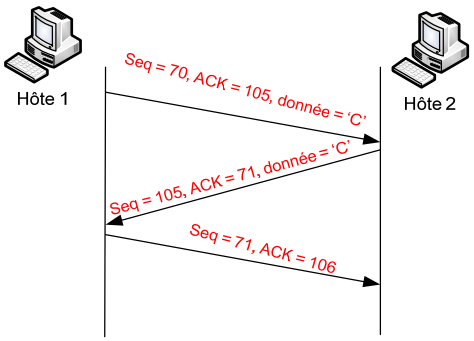
- La retransmission des segments TCP
  - Perturbe l'ordre des octets contenus dans ces segments

Numéro de séquence

- On numérote les octets
  - Le numéro du segment est celui de son premier octet
  - Introduit un ordre sur les segments
  - Permet d'accuser réception d'un segment (de tous les octets contenus par ce segment)
    - L'acquittement indique le prochain octet attendu
  - Permet de détecter les segments perdus, désordonnés ou dupliqués
- Un même numéro de séquence ne doit pas être réutilisé (modulo 2^{32})

Numéro de séquence - exemple

- N° de séquence
  - Le numéro d'emplacement du premier octet du segment dans le flux (byte stream)
- ACK
  - Le numéro de séquence du prochain octet attendu



TCP établissement de la connexion

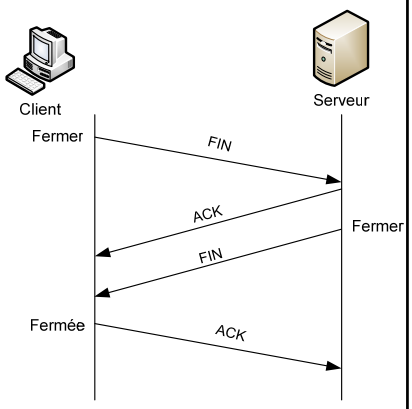
- Etape 1 : le client envoie un segment TCP le bit SYN au serveur
  - Spécifie le numéro de séquence initial du flux de données allant du client au serveur
  - Pas de données
- Etape 2 : le serveur répond avec un segment TCP avec les bits SYN + ACK
  - S'il accepte la connexion
  - Spécifie le numéro de séquence initial du flux inverse
  - Pas de données
- Etape 3 : le client répond par un segment ACK, qui peut contenir des données

TCP fermeture de la connexion

- Le client ferme la connexion

```
ClientSocket.close();
```

- Etape 1 : Le système du premier Pus envoie un segment TCP avec le bit FIN au serveur
- Etape 2 : Le second Pus reçoit le segment FIN, ferme la demi-connexion et répond par un segment avec le bit ACK.
- Il faut répéter les deux étapes pour l'autre demi-connexion.



TCP : Contrôle de flux

- Le coté récepteur d'une connexion TCP gère un buffer de réception
  - Le processus (coté application) peut être lent à lire dans le buffer
- Contrôle de flux
  - => l'émetteur ne doit pas faire déborder le buffer de réception
- Le récepteur envoie la valeur de l'espace disponible du buffer de réception dans chaque segment ayant un bit ACK (c-à-d. servant d'acquittement)