

Module RSA

Partie Réseaux

Isabelle Chrisment



2013-2014



Isabelle Chrisment (TELECOM Nancy) 2013-2014

1

Plan

① Contrôle de congestion et TCP

② Programmation réseaux

• socket, select(), options de socket, socket raw

③ Communication de groupe (multicast)

④ Introduction IPv6

2

Contrôle de congestion et TCP 



① Contrôle de congestion et TCP

• TCP : quelques rappels

• Congestion Internet

• TCP Tahoe

• TCP Reno

• RTT et TCP

• TCP Vegas

• Equité TCP

3

Contrôle de congestion et TCP TCP : quelques rappels

TCP : quelques rappels RFCs: 793, 1122, 1323, 2018, 2581

• Point-à-point

• Orienté connexion

• Fiable et ordonné

• Pas de frontière de message

• Full duplex

• Flux contrôlé

4

Contrôle de congestion et TCP TCP : quelques rappels

Structure d’un segment 32 bits

URG: urgent data (generally not used)

ACK: ACK #

valid

source port # dest port # sequence number

acknowledgement number

counting

by bytes

of data

(not segments!)

PSH: push data now

head len

not

used

UAP R S F

rcvr window size

# bytes

(generally not used)

RST, SYN, FIN: connection estab (setup, terminaison commands)

Internet

checksum

(as in UDP)

checksum ptr urgent data Options (variable length)

application

data

(variable length)

rcvr willing

to accept

Kurose & Ross

5

Contrôle de congestion et TCP TCP : quelques rappels

TCP seq. #’s - ACKs

Seq. #’s:

– Flux d’octets

“numéro” du premier

octet dans les données du segment

ACKs:

– seq # du prochain

octet attendu de

l’autre côté

– ACK cumulatif

Host A Host B

User

types

‘C’

host ACKs

receipt

of echoed

‘C’

host ACKs receipt of ‘C’, echoes back ‘C’

simple telnet scenario Kurose & Ross

time

6

Contrôle de congestion et TCP TCP : quelques rappels

TCP Etablissement d’une connexion

Client Réseau Serveur

**CLOSED CLOSED**

Demande d’Ouverture Active

Emet SYN seq = x

**SYN\_SENT**

Reçoit le segment SYN + ACK

**ESTABLISHED** Emet ACK = y+1

Accepte toute les connexion Ouverture Passive

**LISTEN**

Reçoit le segment SYN **SYN\_RCVD**

Emet SYN seq=y, ACK= x+1

Reçoit le segment ACK **ESTABLISHED**

****7

Contrôle de congestion et TCP TCP : quelques rappels

TCP syn cookies

• Choix particulier du numéro de séquence côté serveur :

Client

Réseau

Serveur

( cf http://cr.yp.to/syncookies.html )

**CLOSED CLOSED**

• 5 bits = t mod32 (t compteur de temps sur 32 bits)

Demande d’Ouverture Active

Accepte toute les connexion Ouverture Passive

• 3 bits : encodage de la valeur du MSS du serveur • 24 bits F(adr. src, port src,

Emet SYN seq = x **SYN\_SENT**

**LISTEN**

Ne pas stocker d’état

adr. dest, port dest, t, secret)

avec F = fonction crypto

Reçoit le segment SYN

**SYN\_RCVD**

Emet SYN seq=cookie,

Reçoit le segment ACK= x+1 SYN + ACK

**ESTABLISHED**

Emet ACK = cookie+1

Vérification cookie

Reçoit le segment ACK **ESTABLISHED**

****8

Contrôle de congestion et TCP TCP : quelques rappels

TCP fermeture connexion

Client Réseau Serveur

Demande de

déconnexion

close

**FIN\_WAIT\_1**

**FIN\_WAIT\_2**

Fermeture connexion Client

**TIME\_WAIT**

Timeout

**CLOSED**

FIN SEQ 5150 (reste 20 octets)

ACK 5170 + 1

FIN SEQ 2250, ACK 5171 ACK 2250 +1

Remise des 20 octets + Fin demandée par serveur **CLOSE\_WAIT**close

**LAST\_ACK**

Fermeture Serveur

**CLOSED**

****9

Contrôle de congestion et TCP TCP : quelques rappels

TCP: scénario de retransmission

t

u

o

e

m

i

t

0

0

1

=

q

e

S

time

Host A

t

u

o

e

m

i

t

2

9

=

q

e

S

Host B

10

Contrôle de congestion et TCP TCP : quelques rappels

Silly window 

avoidance

TCP Illustrated

Stevens

11

Contrôle de congestion et TCP Congestion Internet

Congestion Internet - Problématique

• Dans les années 80, avenir de TCP/IP en question

– Trop de congestion

Données + Retransmissions

Congestion!

Paquets jetés!!

Encore plus de pertes !!!

12

Contrôle de congestion et TCP Congestion Internet

Causes/coûts de la congestion : scénario 1



Kurose & Ross

13

Contrôle de congestion et TCP Congestion Internet

Causes/coûts de la congestion : scénario 2

Kurose & Ross

14

Contrôle de congestion et TCP Congestion Internet

Causes/coûts de la congestion : scénario 2



Kurose & Ross

15

Contrôle de congestion et TCP Congestion Internet

Causes/coûts de la congestion:

scenario 3



Kurose & Ross

16

Contrôle de congestion et TCP Congestion Internet



• Comment améliorer TCP afin qu’il envoie moins de trafic quand le réseau est

congestionné ?

– Comment détecter la congestion ?

– Comment rendre le flux de données sensible au niveau de congestion ?

17

Contrôle de congestion et TCP Congestion Internet

Détection de la congestion

• Elimination de paquets = indicateur de

congestion ?

Paquet

Src Dst

Drop

Ack

Timeout! Pas de

Ack = Congestion!

18

Contrôle de congestion et TCP Congestion Internet

Contrôle de Congestion –

l’effet de la taille de la fenêtre

• Noter que la fenêtre de l’émetteur est égale au nombre de paquets en transit dans le réseau

Window Source Destination X packets

Jorg Liebherr

19

Contrôle de congestion et TCP Congestion Internet

Contrôle de congestion TCP

• Contrôle de bout-en-bout (pas d’assistance du réseau)

• Débit de transmission limité par taille de fenêtre



Kurose & Ross

20

Contrôle de congestion et TCP Congestion Internet

Contrôle de congestion TCP

• w segments, chacun avec MSS octets émis en un RTT:

throughput = w \* MSS

RTTBytes/sec

21

Contrôle de congestion et TCP Congestion Internet

Contrôle de congestion TCP

•“sonder” pour la bande passante utilisable:

– idéalement: transmettre aussi vite que possible

(fenêtre la + grande possible) sans perte

– *augmenter* la fenêtre (🡪 plus de paquets dans le

réseau ) jusqu’à l’apparition de pertes (congestion)

– pertes: *diminuer* la fenêtre (🡪 moins de paquets

dans le réseau) et recommencer à sonder

(augmenter) à nouveau

22

Contrôle de congestion et TCP Congestion Internet

Contrôle de Congestion

• Idée: Concept de la *congestion window*

– fenêtre qui est petite quand la congestion est importante – fenêtre qui est grande quand la congestion est petite

• Fenêtre autorisée = minimum (fenêtre récepteur, fenêtre de congestion)

23

Contrôle de congestion et TCP Congestion Internet

Contrôle de congestion de Van Jacobson

• Van Jacobson a introduit le contrôle de

congestion dans TCP en 1988-1989

• La version d’origine de TCP qui implante le contrôle de congestion de Van Jacobson est connue sous le nom de TCP Tahoe

24

Contrôle de congestion et TCP TCP Tahoe

Éléments de contrôle de congestion dans TCP Tahoe

• AIMD

– Additive increase, multiplicative decrease

• Slow start

• Congestion Avoidance

25

Contrôle de congestion et TCP TCP Tahoe

Additive Increase, Multiplicative Decrease

• Chaque fois qu’un paquet est détruit (perdu), réduire la taille de la fenêtre de manière significative

(multiplicative decrease)

– Multiplicative decrease est nécessaire pour éviter la

congestion

• Si aucune perte n’est observée, augmenter de

manière graduelle la taille de la fenêtre (additive increase)

26

Contrôle de congestion et TCP TCP Tahoe

Problèmes

• Quelle doit être la taille de la fenêtre

– Initialement ?

– Quand il y perte de paquet ?

• Taille de fenêtre pessimiste? (e.g., 1)

– Additive increase est trop lent

– Les connexions courtes n’utiliseront jamais la bande

passante disponible

• Taille de fenêtre optimiste?

– Un flux trop important de données peut impliquer un

overflow dans les files des routeurs

27

Contrôle de congestion et TCP TCP Tahoe

Slow Start

• Objectif: déterminer la capacité

disponible rapidement

• Idée:

– Utiliser **CongestionThreshold** comme une estimation optimiste de

**CongestionWindow**

– Commencer avec **CongestionWindow** = 1 segment

– doubler **CongestionWindow** à chaque RTT (incrémenter de 1 paquet pour chaque ACK)

– Quand **CongestionThreshold** est atteint utiliser additive increase

Source Destination

28

Contrôle de congestion et TCP TCP Tahoe

Slow-start TCP

Algorithme du slow start Host A

initialise: Congwin = 1

T

Host B

Pour (chq segment ACKed)

T

R

Congwin++

Jusqu’à (evt de perte OR

CongWin >= threshold)

• Croissance exponentielle (par RTT) de la taille de fenêtre (pas si lent!) • Evt de perte: timeout et/ou trois ACKs dupliqués (Tahoe TCP)

time

Kurose & Ross

29

Contrôle de congestion et TCP TCP Tahoe

Illustration du Slow Start

Hypothèse

**CongestionThreshold = 8**

**14**

***n***

***i***

***(***

***w***

***o***

***d***

***n***

***i***

***W***

***n***

***o***

***i***

***t***

***s***

***e***

***g***

***n***

***o***

***C***

**12**

**10**

**8**

**6**

***)***

***s***

***t***

**4**

***n***

***e***

***m***

***g***

**2**

***e***

***s***

**0**

*threshold*

**t= 0t= 2t= 4t= 6**

***Roundtrip times***

******Figures by Jorg Liebeherr Kurose & Ross 30

Contrôle de congestion et TCP TCP Tahoe

AIMD + congestion avoidance • Algorithme :

– Incrémenter **CongestionWindow** d’un paquet par RTT (*linear increase*)

– Réduire **CongestionThreshold** et **CongestionWindow** (*multiplicative decrease*) quand perte

Source

Destination

31

Contrôle de congestion et TCP TCP Tahoe

Evitement de congestion TCP Congestion avoidance

Evitement de congestion

/\* slowstart est terminé \*/ 

/\* Congwin >= threshold \*/

Until (evt de perte) {

chq w segments ACKed:

Congwin++

}

threshold = Congwin/2

Congwin = 1

1

effectuer slowstart

1: TCP Reno évite slowstart (fast

recovery) après 3 ACKs dupliqués

Kurose & Ross

32

Contrôle de congestion et TCP TCP Reno

TCP Reno

• TCP Reno (1990) et ses dérivés sont les plus utilisés aujourd’hui

• TCP Reno implémente :

– Fast Retransmit

– Fast Recovery

• RFC 2581

33

Contrôle de congestion et TCP TCP Reno

Fast Retransmit

Emetteur Récepteur

• Problème : les timers TCP peuvent conduire à des périodes d’inactivité

Packet 1

Packet 2

Packet 3

Packet 4

Packet 5

Packet 6

Retransmit packet 3

ACK 1 ACK 2

ACK 2

ACK 2 ACK 2

ACK 6

34

Contrôle de congestion et TCP TCP Reno

Fast Retransmit

• Envoi d’un acquittement sur réception d’un

paquet

• Envoi d’une duplication du dernier acquittement quand un paquet est reçu en désordre

• Utilisation des acquittements dupliqués pour

déclencher des retransmissions

– Si 3 (ou plus) d’acquittements dupliqués reçus, le

segment est considéré comme perdu

– TCP réalise une retransmission du segment manquant sans attendre l’expiration du timer

35

Contrôle de congestion et TCP TCP Reno

Fast Recovery

• Fast Recovery est implémenté avec Fast Retransmit • Fast Recovery supprime la phase de slow start après retransmission due à un Fast Retransmit

• Fast Recovery gère la transmission des données jusqu’à l’arrivée d’un ACK non dupliqué

36

Contrôle de congestion et TCP TCP Reno

Fast Recovery

• Algorithme :

– Quand le 3ème ACK dupliqué est reçu,

• positionner le seuil (ssthresh) à la moitié de la

congestion window (cwnd) courante

• retransmettre le segment manquant

• Positionner cwnd à la valeur du ssthresh + 3\*MSS

– Chaque fois qu’un autre ACK dupliqué arrive

• Incrémenter cwnd

• Transmettre un paquet si possible

– Quand ACK arrive qui acquitte la nouvelle donnée • Positionner cwnd à la valeur de ssthresh

37

Contrôle de congestion et TCP TCP Reno

Reno

Tahoe



Kurose & Ross

38

Contrôle de congestion et TCP TCP Reno

Résumé Congestion Control

• Essayer d’éviter que la source applique le slow-start – Source utilise slow-start au début et sur timeout uniquement – Source divise la fenêtre de congestion window de la moitié suite à un fast retransmit

• La perte d’un seul paquet peut être détectée par les mécanismes de fast retransmit/fast recovery algorithm • La perte de plusieurs paquets consécutifs forcera la source à appliquer le slow start

• TCP New Reno (RFC3782)

39

Contrôle de congestion et TCP RTT et TCP

Tahoe/Reno

Timers de retransmission

• Les timers de retransmission sont basés sur les calculs du round-trip time (RTT) que TCP effectue

ack5

pkt1ack2pkt2 pkt3 pkt4

pkt5 pkt6 pkt7 ack8

Jorg Liebherr

40

Contrôle de congestion et TCP RTT et TCP

TCP Round Trip Time et Timeout

• Comment positionner le Timeout TCP ?

• Plus long que RTT

– note: RTT va varier

• Trop court: timeout prématuré

– Retransmissions inutiles

• Trop long: réaction lente à la perte de segments

41

Contrôle de congestion et TCP RTT et TCP

TCP Round Trip Time et Timeout

Comment estimer RTT ?

• **SampleRTT**: temps mesuré entre l’envoi d’un segment et la réception d’un ACK

• **SampleRTT** va varier, besoin d’un RTT moyen, plus lissé – Utilisation de plusieurs mesures récentes, pas

uniquement le **SampleRTT courant**

****42

Contrôle de congestion et TCP RTT et TCP

TCP Round Trip Time et Timeout

• Moyenne fluctuante

• L’influence d’un échantillon diminue de

façon exponentielle

• Valeur typique de α: 0.125 (1/8)

• Extrait du rfc 2988

**EstimatedRTT = (1-**α**).EstimatedRTT +** α**.SampleRTT**

****43

Contrôle de congestion et TCP RTT et TCP

TCP Round Trip Time et Timeout

• Valeur du timeout

– **EstimatedRTT** plus “marge de sécurité”

– large variation dans **EstimatedRTT** 🡪

marge de sécurité accrue

**RetransmissionTimeout ou RTO = EstimatedRTT + 4\*Deviation**

**Deviation = (1-**β**).Deviation +**

β**.|SampleRTT-EstimatedRTT|**

**Avec** β **= 0.25**

****44

Contrôle de congestion et TCP RTT et TCP

Problème

• Si un segment est retransmis, TCP ne peut pas calculer le RTT

– Pourquoi ?

ack2

pkt1 pkt1

45

Contrôle de congestion et TCP RTT et TCP

Algorithme de Karn

• Ne pas mettre à jour le RTT sur des segments qui ont été retransmis.

• Quand TCP retransmet, positionner :

***RTO = min (2 RTO, 64)***

******46

Contrôle de congestion et TCP TCP Vegas

Stratégies de contrôle de congestion

• Stratégie TCP

– Contrôle de congestion une fois détectée

– De manière répétitive, augmente la charge pour trouver le point de congestion puis diminution de la charge quand

congestion

• Stratégie alternative

– Prédire quand la congestion va se produire

– Réduire le débit avant que le paquet soit détruit

– Deux possibilités :

• Orientée routeur : mécanisme RED

• Orientée hôte : TCP Vegas

47

Contrôle de congestion et TCP TCP Vegas

TCP Vegas

• Vegas: Prédit et évite la congestion avant qu’elle se produise

• Tahoe, Reno: Réagit à la congestion une fois qu’elle s’est produite

• Question: comment prédire la congestion ?

48

Contrôle de congestion et TCP TCP Vegas

Principe

• L. Brakmo, S.O’Malley et L. Peterson

(SIGCOMM’94)

• Idée :

– La source observe si des signes indiquent que les files des routeurs se remplissent et qu’une

congestion risque de se produire

• RTT augmente

• Le débit d’émission fluctue

49

Contrôle de congestion et TCP TCP Vegas

Observation

• L’accumulation de paquets dans le réseau peut se déduire en observant le RTT et débit en émission

*cwnd*

Liaison surchargée Routeur

Congestionné

Taux émission = *cwnd* / RTT

Jorg Liebherr

50

Contrôle de congestion et TCP TCP Vegas

Algorithme

• Soit **BaseRTT** le minimum de tous les RTT mesurés (généralement RTT du premier paquet)

**ExpectRate = CongestionWindow/BaseRTT**

• Source calcule **ActualRate** une fois par RTT

51

Contrôle de congestion et TCP TCP Vegas

Algorithme

• Source compare **ActualRate** avec **ExpectRate**

**Diff = ExpectedRate - ActualRate**

**if Diff < *a***

**augmente CongestionWindow linéairement**

**else if Diff > *b***

**décroître CongestionWindow linéairement**

**else**

**laisser CongestionWindow inchangée**

****52

Contrôle de congestion et TCP Equité TCP

Equité TCP

Si N sessions TCP partagent le même lien goulot d’étranglement, chacun doit avoir 1/N de la

capacité du lien

TCP connexion 1

routeur

capacité R TCP

connexion 2

Kurose & Ross

53

Contrôle de congestion et TCP Equité TCP

Pourquoi TCP est équitable ?

Deux sessions en compétition:

• Augmentation additive donne une pente de 1, lorsque le débit augmente

• Diminution multiplicative diminue le débit

proportionnellement

54

Contrôle de congestion et TCP Equité TCP

Pourquoi TCP est équitable ?

Partage équitable de la bande

R

passante

perte: diminution fenêtre par facteur de 2

congestion avoidance: augmentation linéaire

Connexion 1 débit

R

Kurose & Ross

55

Programmation réseaux

Plan

① Contrôle de congestion et TCP

② Programmation réseaux

• socket, select(), options de socket, socket raw

③ Communication de groupe (multicast)

④ Introduction IPv6

56



Programmation réseaux

Modèle Client Serveur



② Programmation réseaux

• Modèle Client Serveur

• Introduction socket

• Client-Serveur TCP

• Client-Serveur UDP

• Entrée/sortie multiplexée : select

• Entrée/sortie asynchrone

• Options de socket

• Socket raw

57

Programmation réseaux Modèle Client Serveur

Le modèle Client Serveur

Application Client Application Serveur requête

Attente 

(synchrone)

• Modèle asymétrique :

– Le client :

• demande l’exécution d’un service

traitement 

réponse

• est bloqué en attente de la réponse (mode synchrone)

– Le serveur :

• est en attente de requêtes (processus en arrière plan)

• réalise le service demandé d’un client et envoie la réponse (traitement

séquentiel ou concurrent)

• peut gérer plusieurs clients en même temps (serveur multi-clients)

58

Programmation réseaux Modèle Client Serveur 

Le modèle Client Serveur

Contrôlé par programmeur

Application Client

socket

socket

Protocole applicatif

Application Serveur

socket

socket

Contrôlé par programmeur

Contrôlé par le système d’exploitation

TCP

UDP

IP

TCP

UDP

IP

Contrôlé par le système d’exploitation

Interface réseau

Interface réseau

59

Programmation réseaux 



② Programmation réseaux

• Modèle client serveur

• Introduction socket

• Client-Serveur TCP

• Client-Serveur UDP

• Entrée/sortie multiplexée : select

• Entrée/sortie asynchrone

• Options de socket

• Socket raw

60

Programmation réseaux Introduction socket 

Définition d’un(e) socket

• Interface de programmation (API) permettant d’envoyer et recevoir des données

• Point de communication bidirectionnel

– à l’intérieur d’une même machine 🡺 domaine Unix

– à travers les réseaux 🡺 domaine Internet

• Initialement liée à la version BSD (Berkeley) de Unix • Descripteur de socket 🡺 similaire descripteur fichier (ressources système)

• Accessible par des appels systèmes

61

Programmation réseaux Introduction socket 



Table des

fichiers

ouverts

sd

Table des

inodes

socket

File des

messages

en réception

File des

messages

en émission

Table des Boîte aux lettres descripteurs

Utilisateur

Système

62

Programmation réseaux Introduction socket 

Les types de sockets • Précise le type de

communication utilisé

– SOCK\_STREAM

• transmission fiable, ordonnée, sans

SOCK\_STREAM socket

socket

duplication

• en mode connecté

• autorise messages urgents

• pas de préservation des limites de messages

• TCP

– SOCK\_DGRAM

– SOCK\_RAW

TCP

UDP

IP

63

Programmation réseaux Introduction socket 

Les types de sockets • Précise le type de

communication utilisé – SOCK\_STREAM – SOCK\_DGRAM

SOCK\_DGRAM

socket

socket

• transmission non fiable, non ordonnée, avec possibilité de duplication

• en mode non connecté

• préservation des limites de messages

• UDP

– SOCK\_RAW

TCP

UDP

IP

64

Programmation réseaux Introduction socket 

Les types de sockets

• Précise le type de

communication utilisé

– SOCK\_STREAM

– SOCK\_DGRAM

SOCK\_RAW

socket socket

– SOCK\_RAW

• protocoles de bas niveau (super user)

• permet de ne pas passer par la couche transport et d’utiliser IP directement

• ICMP

TCP

UDP

IP

65

Programmation réseaux Introduction socket 

Les familles de sockets

• Une famille ou un domaine définit :

– La convention d’adressage ou format des adresses à utiliser – Le ou les protocoles réseaux supportés

• AF\_UNIX (AF\_LOCAL)

– Local à la machine : domaine UNIX

• AF\_INET

– Domaine Internet et adresses IPv4

• AF\_INET6

– Domaine Internet et adresses IPv6

66



Programmation réseaux

Introduction socket

Structure générique d’une socket

Format général décrit dans

**<sys/socket.h>**

**struct sockaddr**

**struct sockaddr {**

**uint8\_t sa\_len;**

**sa\_family\_t sa\_family;**

**sa\_len**

16 bits

**sa\_family**

**char s\_data[14]; }**

**s\_data**

**sa\_len** 🡺longueur de la structure **sa\_family** 🡺 AF\_UNIX ou AF\_INET,….

**s\_data** 🡺 jusqu’à 14 octets pour décrire l’adresse spécifique au protocole

**(14 octets)**

****67

Programmation réseaux Introduction socket 

Structure spécifique d’une socket

• Format spécifique au protocole de communication

**struct sockaddr\_xx {**

**}**

• xx est un suffixe unique fonction du protocole xx = **in** pour Internet (IPv4)

**in6** pour Internet (IPv6)

**un** pour Unix

68



Programmation réseaux

Introduction socket

Structure dans le domaine UNIX

Format décrit dans **<sys/un.h>**

**struct sockaddr\_un { uint8\_t sun\_len;**

**struct sockaddr\_un** 16 bits

**sa\_family\_t sun\_family; char sun\_path[104];**

**sun\_len**

**AF\_UNIX**

**}**

**sun\_family** 🡺 AF\_UNIX

**sun\_path** 🡺 /\* chemin = nom de fichier \*/

**sun\_path**

**(jusqu’à 104 ou 108 octets)**

****69

Programmation réseaux Introduction socket 

Structure dans le domaine INET

Format décrit dans **<netinet/in.h> struct sockaddr\_in { struct in\_addr {**

**struct sockaddr\_in** 16 bits

**uint8\_t sin\_len; in\_addr\_t s\_addr;**

**sin\_len**

**AF\_INET**

**sa\_family\_t sin\_family; }**

**in\_port\_t sin\_port; typedef uint32\_t in\_addr\_t; struct in\_addr sin\_addr;**

**char sin\_zero[8];**

**}**

**sin\_family** 🡺 AF\_INET

**sin\_port** 🡺 numéro sur 16 bits dans **l’ordre du réseau (network byte ordered)**

**sin\_addr** 🡺 adresse Internet sur 32 bits dans **l’ordre du réseau sin\_zero** 🡺 inutilisé (doit être mis à zéro)

**#port**

**@IPv4 inutilisé**

****70

Programmation réseaux Intoduction socket 

Structure dans le domaine INET Numéro de port

• Identifie de manière unique, sur une machine, un processus exécutant un service donné

– Il existe des ports prédéfinis (cf **/etc/services**)

• serveur web : port 80

• serveur mail : port 25

• appel de la fonction **getservbyname()**

– Le port peut être choisi par le programmeur d ’application >= IPPORT\_RESERVED (1024)

– Le port peut être attribué par le système **(sin\_port** 🡸**0)**

****71

Programmation réseaux 

Structure dans le domaine INET Numéro de port – fonction getservbyname()

#include <netdb.h>

struct servent **\* getservbyname** (const char \*name, const char \*proto)

**name** 🡺 pointeur sur la chaîne décrivant le nom du service

**proto** 🡺 pointeur sur la chaîne où se trouve le nom du protocole (tcp) **valeur de retour** -1 si erreur

**struct servent{**

**char \*s\_name; /\* nom du service\*/**

**char \*\*s\_aliases; /\* liste alias \*/**

**int s\_port; /\* numéro port (ordre réseau)\*/**

**int s\_proto ; /\* protocole utilisé \*/**

**}**

****72

Programmation réseaux Introduction socket 

Structure dans le domaine INET

Adresse IP

• Identifie de manière unique une machine sur un réseau

– soit obtenue par les fonctions :

• **gethostbyname(), gethostbyname2() getipnodebyname(),**

**getaddrinfo()**

– dans le cas d’un programme serveur la valeur

INADDR\_ANY est choisie : les messages peuvent être

reçus sur n’importe quelle interface réseau

• Exprimée sur la forme ddd.ddd.ddd.ddd

– exemple : 193.54.8.4

– Dans la structure 🡺entier format réseau

73



Programmation réseaux

Introduction socket

Structure dans le domaine INET

Adresse IP – fonction gethostbyname()

#include <netdb.h>

struct hostent \***gethostbyname** (const char \*name)

**name** 🡺 pointeur sur la chaîne où se trouve le nom du système

**valeur de retour** -1 si erreur

**struct hostent{**

**char \*h\_name; /\* nom machine\*/**

**char \*\*h\_aliases; /\* liste alias \*/**

**int h\_addrtype; /\* type famille \*/**

**int h\_length ; /\* longueur adresse \*/**

**char \*\*h\_addr\_list; /\* liste adresses \*/**

**#define h\_addr h\_addr\_list[0]**

**}**

****74



Programmation réseaux

Introduction socket



**peut ne pas être renseigné**

**sizeof (struct**

**sockaddr\_in)**

**gethostbyname() ->h\_addr**

**sin\_addr(4 octets)**

**struct sockaddr\_in**

**sin\_len (1 octet)**

**AF\_INET**

**#port**

**@IPv4**

**gethostbyname() ->h\_addrtype**

**sin\_family (1 octet)**

**getservbyname() ->s\_port**

**sin\_port (2 octets)**

**adresse Internet (IP)**

**inutilisé**

**bzero() ou memset() sin\_zero (8 octets)**

****75

Programmation réseaux Introduction socket 

Structure dans le domaine INET Format de représentation des données

• Les machines hôtes stockent les données (entiers) dans un format

– Little endian

– Big endian

• Pour communiquer il faut un format de transfert des données

– *network byte order :* **big endian**

****76

Programmation réseaux Introduction socket 

Structure dans le domaine INET Fonctions d’ordonnancement des octets

(Byte Ordering Functions)

#include <arpa/inet.h>

uint16\_t **htons** (uint16\_t host16bitvalue);

uint32\_t **htonl** (uint32\_t host32bitvalue);

retournent une valeur dans l’ordre du réseau

uint16\_t **ntohs** (uint16\_t net16bitvalue);

uint32\_t **ntohl** (uint32\_t net32bitvalue);

retournent une valeur dans l’ordre de la machine hôte

77

Programmation réseaux Introduction socket 

Structure dans le domaine INET Fonctions de manipulation d’octets

#include <strings.h>

void **bzero (**void \*dest, size\_t nbytes) ; 

void **bcopy** (const void \*src, void \* dest, size\_t nbytes) ;

int **bcmp** (const void \*ptr1, const void \*ptr2, size\_t nbytes );

retourne 0 si égal

#include <string.h>

void **\*memset (**void \*dest, int c, size\_t len) ;

void **\*memcpy** (void \*dest, const void \*src, size\_t nbytes) ;

int **memcmp** (const void \*ptr1, const void \*ptr2, size\_t nbytes ); retourne 0 si égal

78

Fonctions obsolètes

Programmation réseaux Introduction socket 

Structure dans le domaine INET Fonctions de manipulation d’adresses

#include <arpa/inet.h>

int **inet\_aton** (const char \*strptr, struct in\_addr \*addrptr) ;

*retourne 1 si string valide 0 sinon*

in\_addr\_t **inet\_addr** (const char \*srcptr) ;

retourne une valeur de 32 bits ordonnée réseau sinon INADDR\_NONE

char **\*inet\_ntoa** (struct in\_addr inaddr) ;

retourne un pointeur vers une chaîne contenant une adresse IP « aaa.bbb.ccc.ddd » int **inet\_pton**(af, src, dst)

int af; /\* AF\_INET ou AF\_INET6 \*/

const char \*src; /\* l'adresse (chaine de caract.) à traiter \*/

void \*dst; /\* le tampon où est rangé le résultat \*/

const char \* **inet\_ntop**(af, src, dst, size)

int af; /\* AF\_INET ou AF\_INET6 \*/

const void \*src; /\* l'adresse binaire à traiter \*/

char \*dst; /\* le tampon où est rangé le résultat \*/

socklen\_t size; /\* la taille de ce tampon \*/

79

Programmation réseaux Introduction socket 



**struct** sockaddr\_in serv\_addr;

Struct hostent \*hp;

**bzero**( (&serv\_addr, sizeof(serv\_addr) );

**memset** (&serv\_addr, 0, sizeof(serv\_addr) );

serv\_addr.sin\_port = **htons** ((ushort) atoi(argv[2]));

hp = (struct hostent \*)**gethostbyname** (argv[1]);

if (hp == NULL) {

fprintf(stderr, "%s: %s non trouve dans in /etc/hosts ou dans le DNS\n",

argv[0], argv[1]);

exit(1);

}

serv\_addr.sin\_family = hp->h\_addrtype ; /\* AF\_INET \*/

serv\_addr.sin\_addr = \* ((struct in\_addr \*)(hp->h\_addr));

printf ("IP address: %s\n", **inet\_ntoa** (serv\_addr.sin\_addr));

80