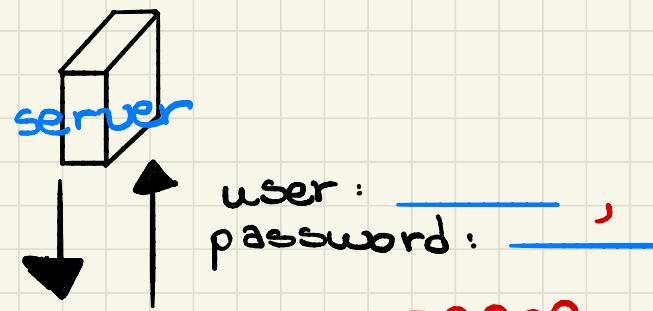


utente
che fa
log in:



login -
result:

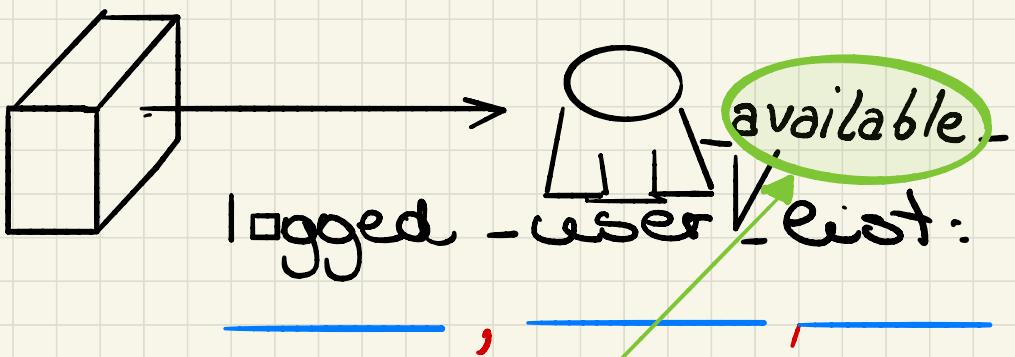
Possibilità:

- "OK" (0)
- "invalid user" (1)
- "wrong password" (2)

ERROR - MSG:
CODE: _____
DESCRIPTION: _____

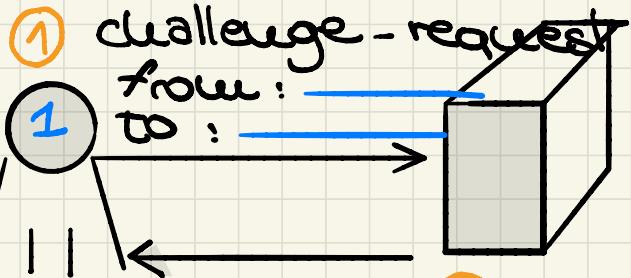
Dopo il login il server invia
a chi' utente che esiste degli
utenti online:

Server: If (login - result == "OK") {
 1- Sends login - result -
 to - user; available -
 in questo modo
 you
 invia
 all'utente
 se stesso
 2- Sends logged - user
 3- add - user - to -
 logged - user (user)



Assumendo che un utente non possa giocare con un utente che già sta giocando.

Inviando una sfida e risposte delle utenze:



ACK
id-sfida

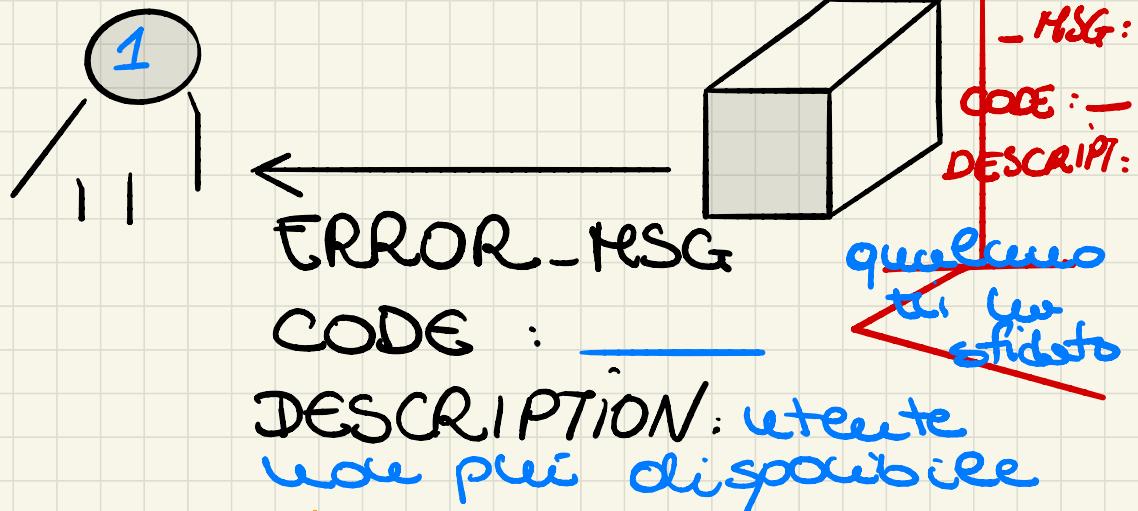
Appena il server riceve una request user 1 e 2 viene in stato pending

ma anche il mio stato

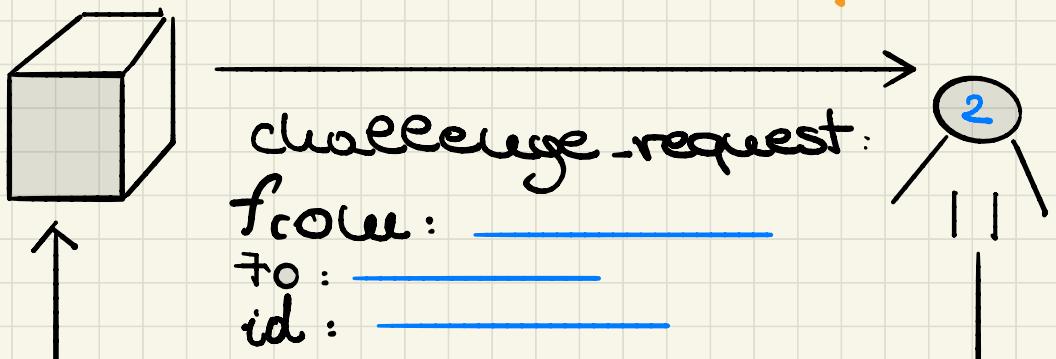
② controllo che l'avversario non ha già iniziato una nuova partita o ha fatto logout

3a (l'avversario ha iniziato una nuova partita / ha fatto Logout)

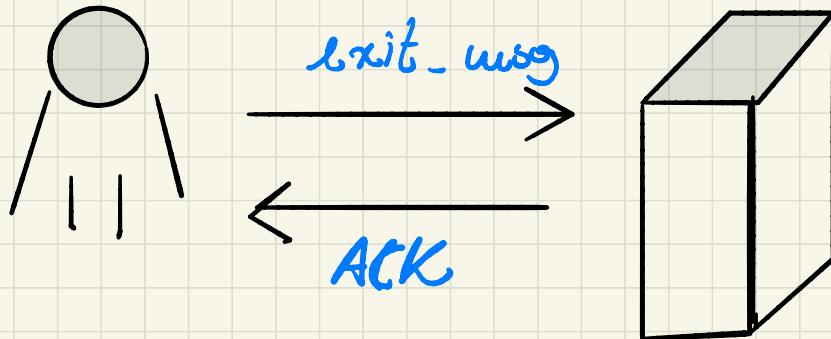
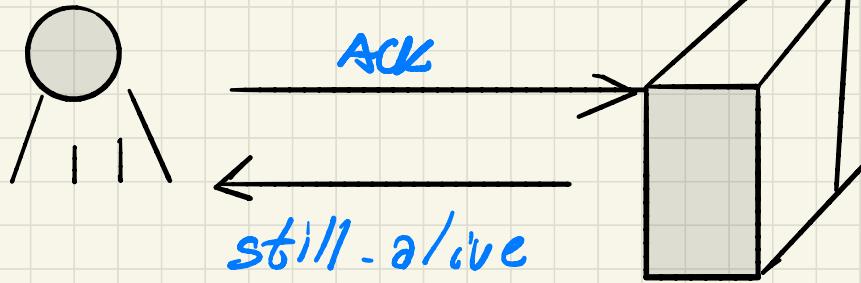
In questo caso secondo me
che oggi avete le possibilità
per un utente di richiedere
che liste aggiornate



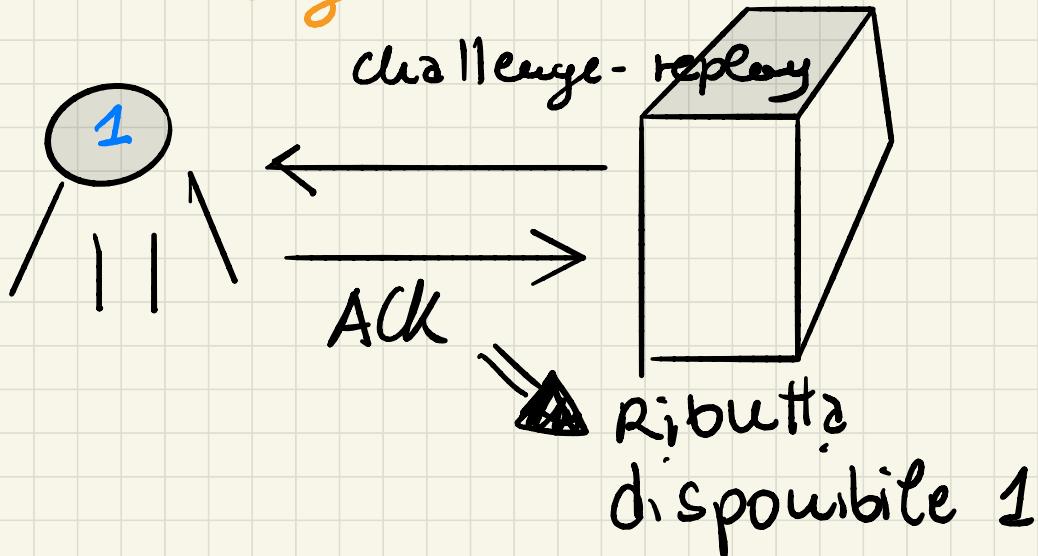
3b - L'utente e' disponibile



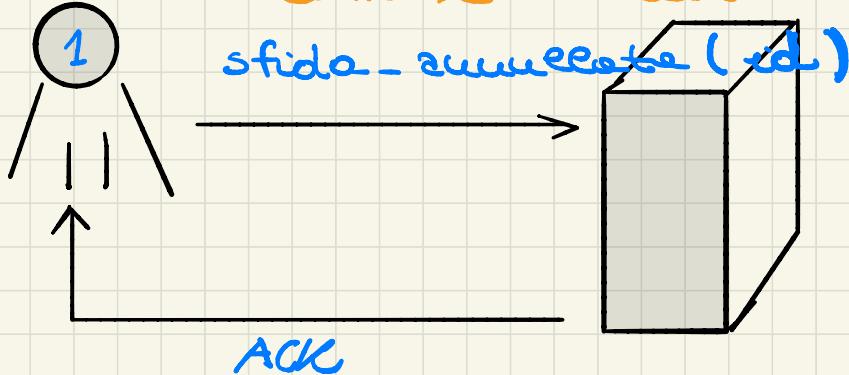
challenge - replay :
from : service
to : user
status : disponibile id : 1000



4a - L'utente sfidato rifiuta la challenge



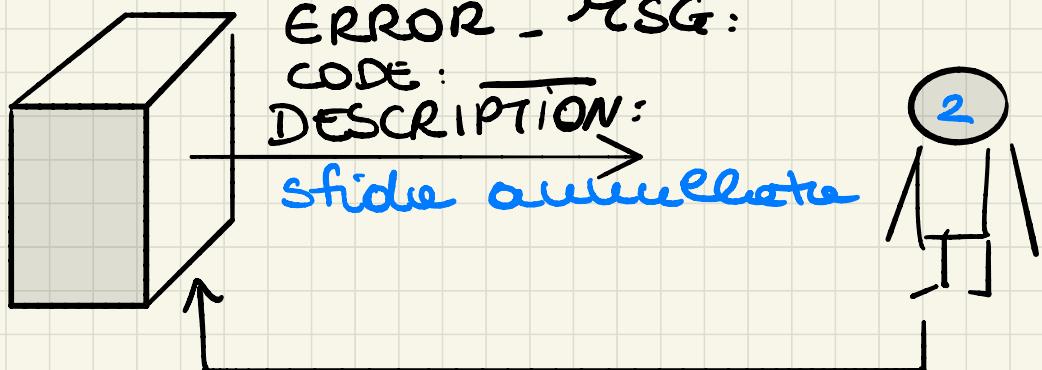
- Caso in cui utente 2 non risponde a 1 entro x secondi



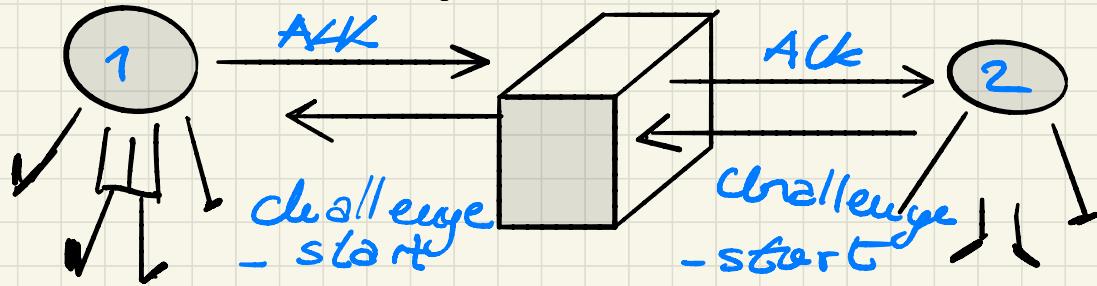
utente 1 di
nuovo online

Mise in rete e ogni challenge

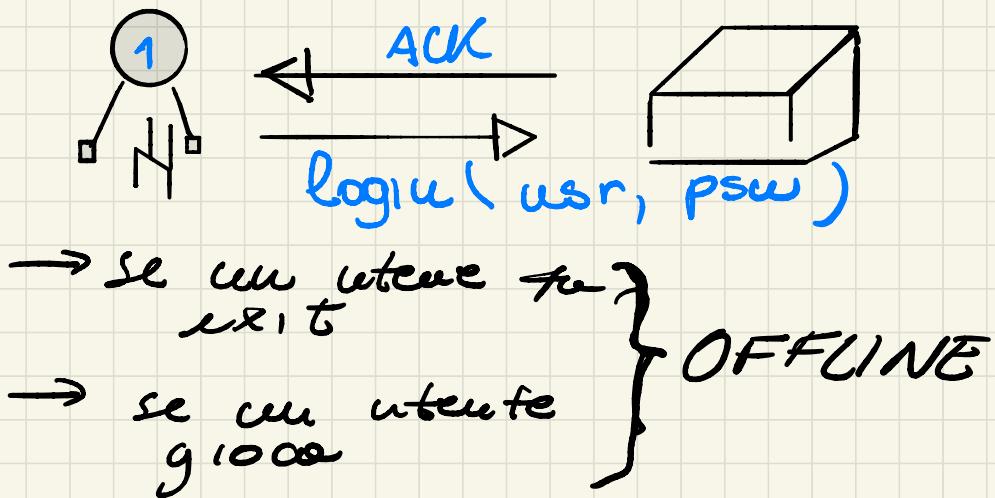
FROM	TO	STATUS
USR 1	USR 2	ANNULLATA



Il server sorteggia e' attente dei due

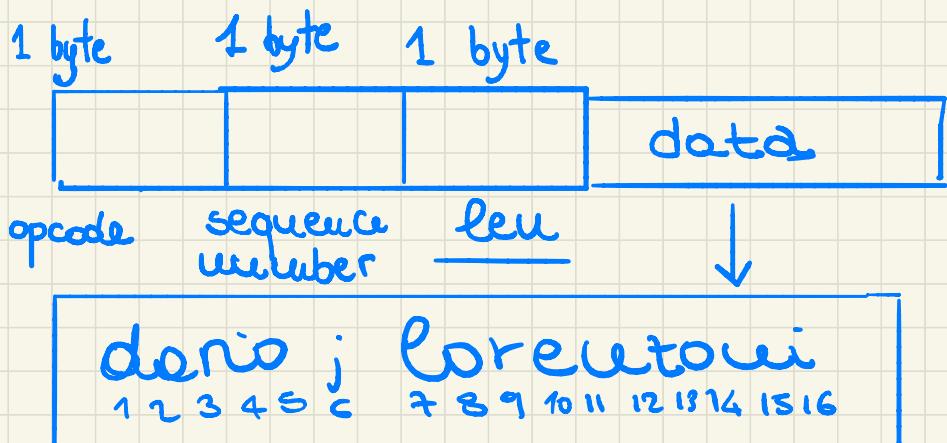


terza fase delle partite o crash



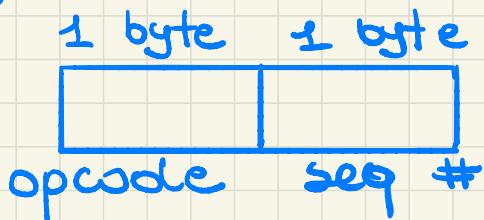
- login

255



memcpy (dest, src + 3, len)

- login - OK / - NO



client

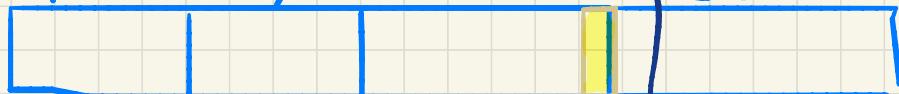
$$Y = \frac{\text{seg } \#}{\text{.256}}$$

seg # = rand()



- available_usr_list

opcode seq # len ↗ data



1 byte 1 byte 4 byte

last flag

Piemo/nuance
ancora
qualcosa

1. dario(j) raffaele; riccardo

- challenge - req

opcode seq # len ↗ data



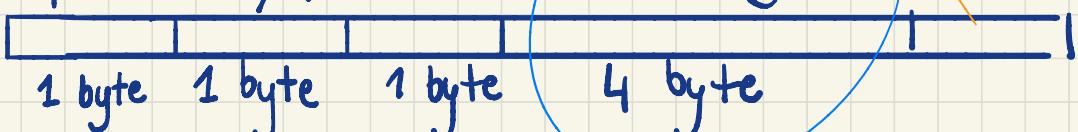
1 byte 1 byte 1 byte

sempre
così
attende l'
lp inizializz.
tene' a
()

FROM ↓ TO ↓
dario(j) raffaele

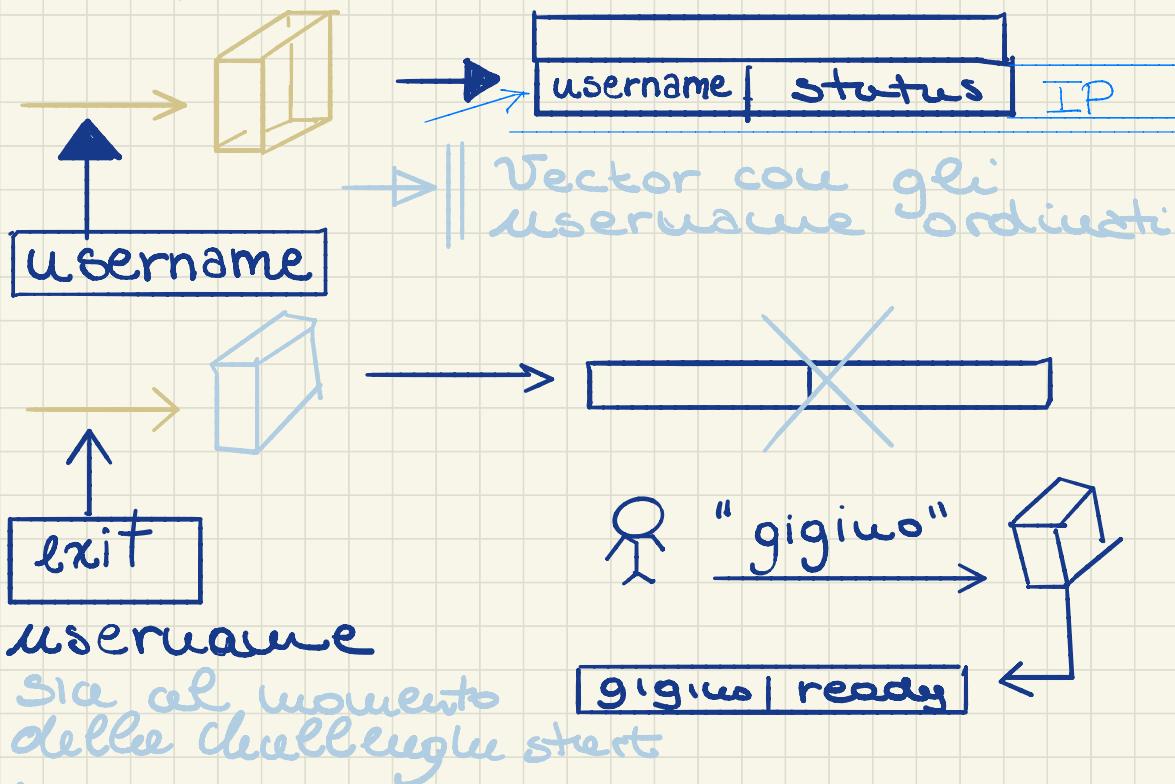
- .

opcode seq # len ↗ challenge # data

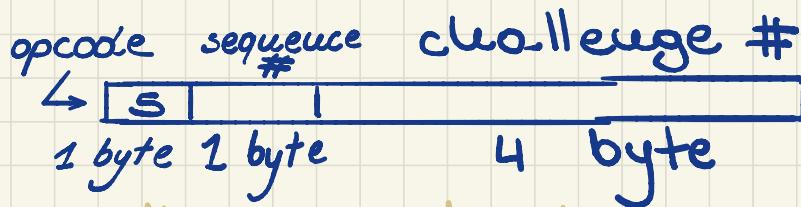


1 byte 1 byte 1 byte 4 byte

STRUTTURA DATI X UTENTI

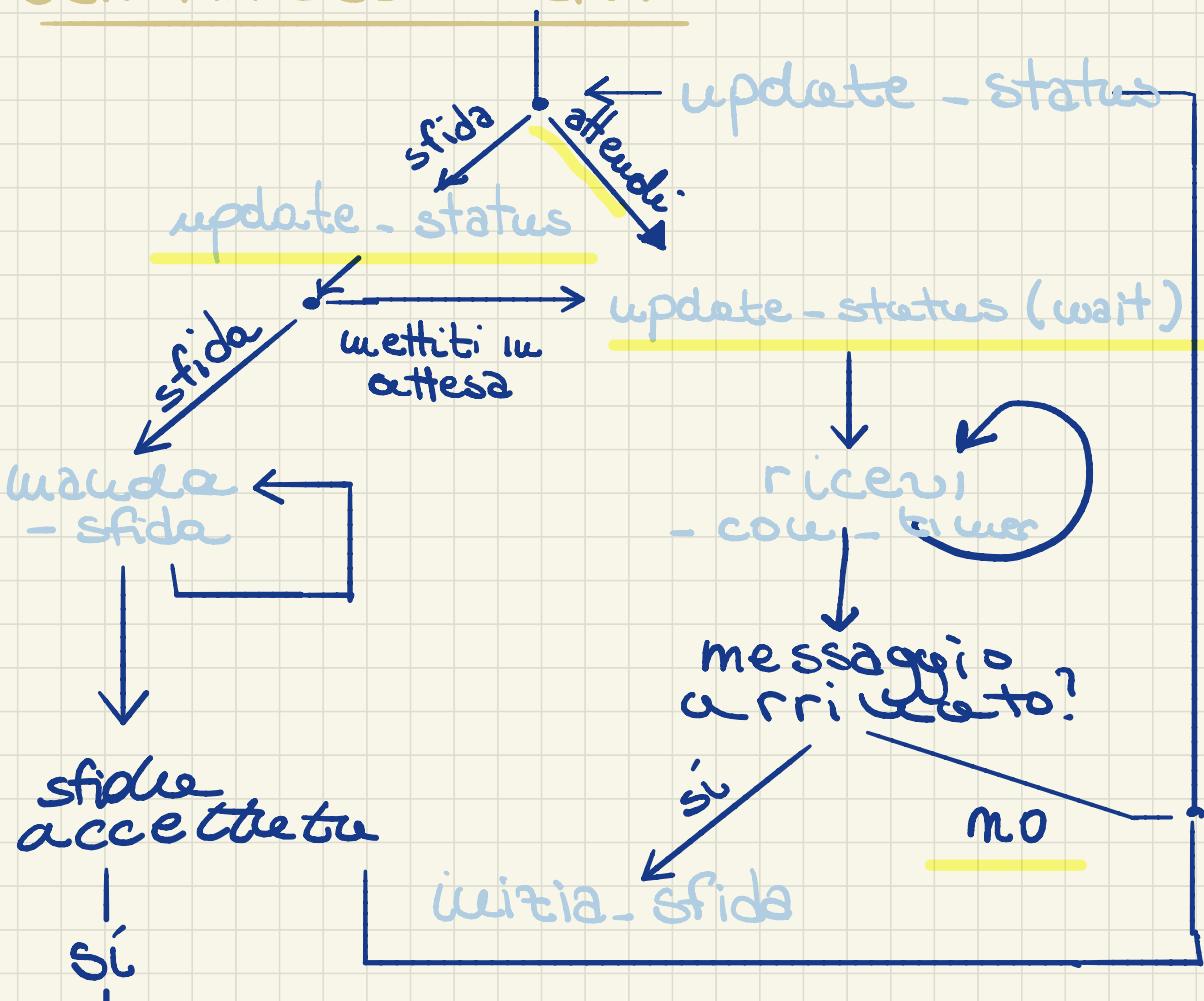


o challenge accepted:



• challenge start :

SCHEMA DEL CLIENT:



- update status
- opcode slot# user status-code

--	--	--	--

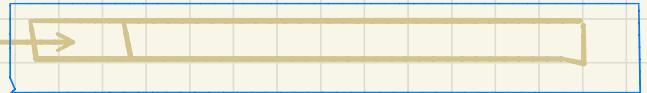
status code:

- 0 → idle
- 1 → challenge
- 2 → waiting

STRUTTURA DATI x CHALLENGE:

challenge:

challenge -
numero

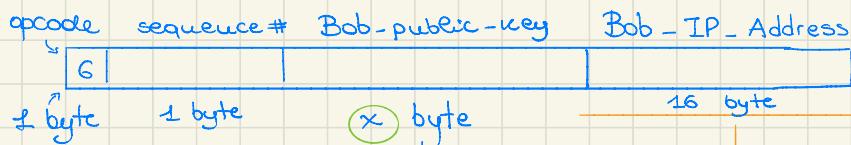


entry viene eliminata:

- dopo challenge - start
- dopo challenge - response
x Giggins (quindi utente a
nificato)
- quando arriva il max
timer expired.

challenge - start :

- PORTA COSTANTE PER LA COMUNICAZIONE
PEER - TO - PEER

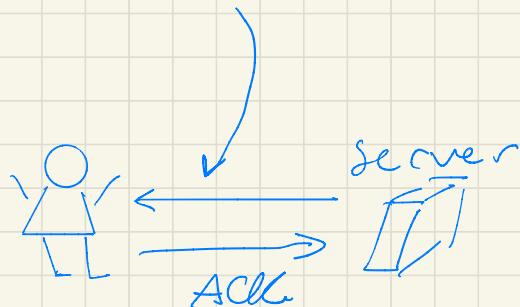


costanti

Indirizzo IPv4
sotto forma di
stringa " "

challenge - timer - expired :

opcode	sequence #
1	1 byte

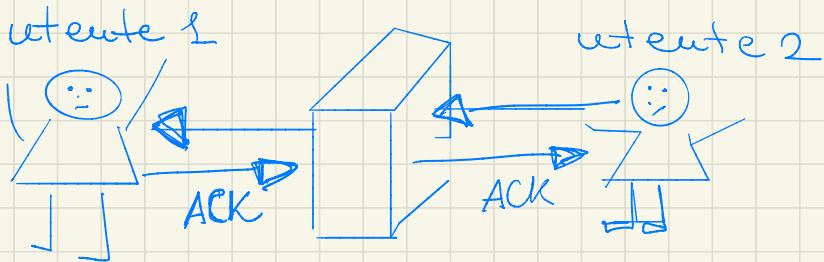


Server -recv from con timeout {

if scatta il timer

elimini le strutture
memorie mex a sette 1

O challenge - refuse



Server

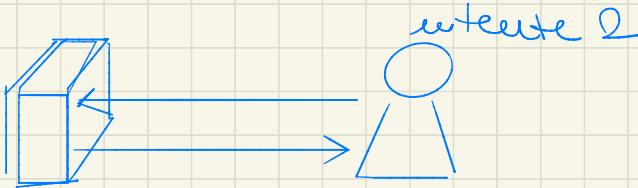
procedure

challenge# e elencare
le entry delle
strettura desti

opcode sequence# challenge#

8	.	.	4 bytes	1
1 byte	1 byte	4 bytes		

O challenge - unacceptable



challenge - unacceptable

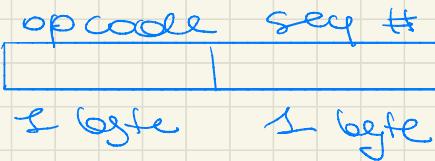
opcode seq#

	2 bytes	2 bytes
--	---------	---------

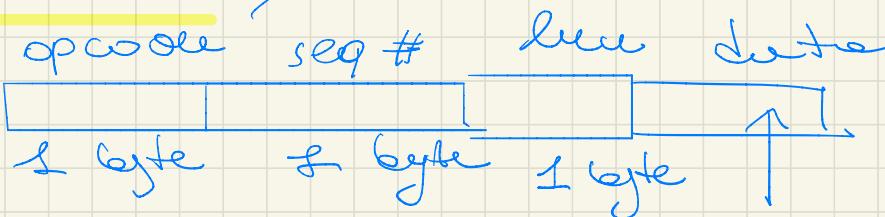
1 byte 2 bytes

- utente 1 sfida utente 2 con un messaggio disponibile
- utente 2 si sente troppo male

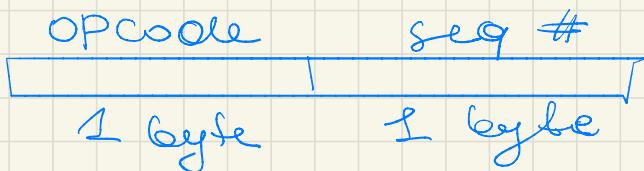
O ACK



O exit



O malformed message:



seq # = rand();

define SEQ_NUMBER_SIZE 1
define OP_CODE_SIZE

↑ ↑
malloc()

To do:

- codice in severale
- sistemi con il protocollo
- fare la documentazione
sui messaggi
- fare il protocollo x il
peer - to - peer

