

Homework

Antonio Nappa, Celestino Santagata

3 maggio 2020

Introduzione

L'obiettivo di questo homework è la progettazione di un'applicazione che consenta lo scambio di messaggi tra utenti (come in una chat) facendo uso delle socket. Il primo step consiste nel simulare l'interazione tra un client ed un server; nello specifico, tale passaggio è stato svolto prima sulla stessa macchina e poi su macchine diverse (collegate tramite rete Internet), notando di volta in volta le correzioni da dover apportare. Il livello successivo, invece, prevede la realizzazione di una chat room e, quindi, l'interazione tra più client. Tutto lo scambio di messaggi è stato seguito tramite il software *WireShark*, che permette una costante supervisione dell'operato.

Una prima decisione nello sviluppo dell'applicazione riguarda la scelta del protocollo da adoperare: TCP o UDP. Tra le due abbiamo preferito portare avanti il discorso con TCP.

Tra le impostazioni preliminari è importante la scelta della porta ed in particolare è importante evitare il conflitto con i numeri di porta più noti definiti negli RFC (Request For Comments); abbiamo scelto la porta 15102.

La realizzazione degli script prevede un ampio uso del modulo **socket**, che fornisce l'accesso all'interfaccia socket BSD (Berkeley Software Distribution)¹. Inoltre, per gestire la concorrenza dei processi abbiamo fatto ricorso al modulo **threading**.

Gli script realizzati dal professore hanno costituito il punto di partenza dell'elaborato e l'utilizzo dei moduli di Python è stato ridotto all'essenziale; abbiamo cercato di usare funzioni quanto più basilari possibile, dedicando maggior impegno nel rendere stabile il server e la comunicazione con esso.

¹Alcuni comportamenti possono dipendere dalla piattaforma, poiché le chiamate vengono effettuate alle socket API del sistema operativo.

Client-Server

Partiamo col descrivere gli script. Nello specifico, trattiamo prima il Server (*TCP-Server.py*) e, poi, il Client (*TCPClient.py*).

Server

Dopo la fase di inizializzazione della socket: *serverPort*, parametri *AF_INET* (uso di indirizzi IPv4) e *SOCK_STREAM* (socket di tipo TCP), *bind*, abbiamo inserito la possibilità di decidere il numero di connessioni da accettare in coda e, successivamente, abbiamo impostato un blocco di codice dedicato alla connessione.

La fase di *accept* è seguita dallo scambio di messaggi di benvenuto e dall'avvio di due thread: uno per la ricezione e uno per l'invio di messaggi. Per gestire in modo opportuno gli errori che si possono verificare durante la creazione della connessione e lo scambio di messaggi (ad esempio, chiusura brusca del client e/o del server), abbiamo deciso di inserire una successione di costrutti *try/except*. Inoltre, per regolare il flusso del ciclo *while* abbiamo inserito anche una variabile globale **status** che dà al gestore del server una sorta di controllo da superuser per chiudere il server stesso o, come vedremo a breve, chiudere la connessione con il client.

Nel caso in cui non si presentino errori, la connessione viene stabilita² ed ha inizio l'esecuzione dei thread *rcv_thread* e *snd_thread*. In entrambi i thread è presente un *try/except* per controllare le eccezioni ed il flusso è regolato da costrutti condizionali: la variabile globale **status** interrompe il ciclo *while* nel caso in cui si voglia chiudere la connessione con il client ('stop') o spegnere il server ('close'), **snd_counter** e **rcv_counter** permettono ai thread di interagire tra di loro accertandosi che il numero di errori che si verificano nello scambio di messaggi (in entrata e/o in uscita) non superi una certa soglia. Noi abbiamo deciso di chiudere la connessione dopo 3 errori.

²*connectSocket, addr = serverSocket.accept()*: quando un nuovo client "bussa" alla porta del server, nel riprendere l'esecuzione viene creata una nuova socket chiamata *connectSocket*. Questa nuova socket (socket di connessione) è diversa dalla socket di benvenuto ed è dedicata allo specifico client che ha bussato; inoltre, *serverSocket.accept()* ritorna anche una tupla con l'indirizzo del client.

La scelta di inserire tali variabili di check è nata durante i vari test di interoperabilità, in cui abbiamo cercato di "stressare" il più possibile la connessione, forzando a più riprese e in diversi modi la chiusura della stessa, non solo attraverso il metodo convenzionale. È proprio dai risultati ottenuti con questi tentativi che abbiamo deciso di impostare una verifica all'interno di entrambe le funzioni *receive_message* e *send_message*. Se si susseguono 3 o più errori durante la ricezione o l'invio di messaggi, la connessione viene automaticamente chiusa.

Questa gestione delle eccezioni ci ha permesso di raggiungere una certa stabilità nell'interazione tra server e client.

Restano da fare altre due considerazioni. È presente un check sulla lunghezza del messaggio da inviare; nel caso in cui tale messaggio sia vuoto, non viene inviato al client ed il ciclo viene ripetuto. Le *escape-words* per gestire convenzionalmente la chiusura della connessione sono *quit* ed *exit*.

Client

La struttura dello script dedicato al client è sostanzialmente la stessa di quella del server, a meno di piccoli accorgimenti e modifiche.

La creazione della socket è seguita dal tentativo di stabilire la connessione; nel caso in cui questa vada a buon fine, vengono avviati i thread dedicati allo scambio di messaggi. Anche qui abbiamo sfruttato il costrutto *try/except* per gestire le eventuali eccezioni; in particolare, abbiamo distinto l'errore di tipo *ConnectionRefusedError*³ da tutti gli altri.

Lo scambio di messaggi con il server avviene secondo i soliti metodi *send* e *rcv*: per terminare in modo corretto la connessione è necessario che la socket venga chiusa sia dal client che dal server, il trigger di tale chiusura è l'invio di un messaggio del tipo *quit* o *exit* (così come richiesto dal server stesso); il tutto viene supervisionato grazie al solito costrutto *try/except*. Il flusso dei thread è regolato dalle variabili globali che abbiamo già incontrato in precedenza, l'unica differenza è che ora la variabile **status** ha il compito di notificare a *snd_thread* e *rcv_thread* della chiusura della connessione ad opera del server, in modo che non si creino conflitti dovuti all'asincronia, appunto, dei due processi.

³No connection could be made because the target machine actively refused it.

Chat room

Analizziamo ora la chatroom: discutiamo i passaggi più importanti (e che differiscono notevolmente dagli script visti in precedenza) prima del Server (*TCPServer_multi-thread.py*) e, poi, del Client (*TCPClient_multi-thread.py*).

Server

Dopo la fase di inizializzazione della socket, abbiamo creato due thread: uno per la supervisione dello stato del server (una sorta di controllo concesso al super-user) e un altro per la connessione tra client e server.

Il primo, *serverStatus*, permette di introdurre una variabile globale **check** che controlli il flow del codice; in particolare, se impostata a 1 blocca l'avvio di nuove connessioni, a 2 chiude il server (per far sì che un inserimento errato impedisca di scrivere nuovamente, abbiamo settato un ciclo *while* in cui l'aggiunta di un'altra variabile (*status*) permette proprio di evitare questo problema). L'altro, che abbiamo indicato genericamente come *thread*, avvia la funzione di connessione e viene di volta in volta aggiunto alla lista *threads*; come vedremo più avanti, tale lista ci permette di tenere traccia dei thread avviati (senza sapere a priori quanti ne siano⁴) e di introdurre separatamente il metodo *join*. Un primo punto di grande rilievo è proprio questo: per gestire la concorrenza delle connessioni, rendendo indipendenti i diversi task del programma, è necessario impostare *start* e *join* dei thread in due cicli separati (mettendoli nello stesso ciclo si ritorna al caso precedente 1:1, che non è ciò che vogliamo).

Passiamo ad analizzare più approfonditamente la funzione *connection*. Dato l'utilizzo del *while* per creare i thread, abbiamo deciso di sfruttare una variabile globale **flag** per far sì che il programma resti in attesa della connessione tramite *serverSocket.accept()* e nel frattempo non apra altre socket indesiderate (finché non si connette un nuovo client, non vengono avviati ulteriori thread). Tutta la funzione segue una serie di costrutti *try/except* che abbiamo implementato a seguito dei vari

⁴Infatti, si trova all'interno di un costrutto *while* che consente di accettare un numero non definito a priori di connessioni.

test di interoperabilità (sia in locale sia in rete Internet) e degli errori sorti durante l'esecuzione. *Try/except* permette di gestire le eccezioni che si possono presentare e di differenziare il proseguimento del programma a seconda del tipo e del livello del problema (in fase di accettazione della connessione, in fase di inizializzazione con *username*, scambio di messaggi vero e proprio); inoltre, per rendere più semplice la fase di debugging abbiamo aggiunto una serie di *print* differenziati.

Una volta stabilita la connessione, il server dà il benvenuto al client chiedendogli di inserire uno username ed avviando successivamente lo scambio di messaggi attraverso la funzione *message_exchange(connectSocket, addr, username)*. Il client viene aggiunto ad una lista, *client_list*, che sfruttiamo ogniqualvolta c'è necessità che il server "distribuisca" un messaggio: a *client_list* viene aggiunta una tupla del tipo (*connectSocket, addr, username*); rendendo *client_list* accessibile a livello globale, per "distribuire" un messaggio all'intera chatroom basta far riferimento alla prima componente di ogni tupla ed avere così a disposizione tutte le connection socket degli utenti connessi.

Dedichiamoci ora alla funzione *message_exchange*. Dato che il server svolge solo un ruolo di intermediario, non c'è un vero bisogno di scindere in funzioni separate le fasi di ricezione e invio di un messaggio; quindi, decodificato il messaggio proveniente dal client considerato, si fa una breve verifica per controllare che non sia un messaggio di uscita dalla chatroom, dopodiché si procede ad inviare il messaggio a tutti gli altri componenti della stanza. Nel caso in cui si riceva un messaggio di abbandono, abbiamo inserito una condizione tale che il server non invii il messaggio anche al client stesso. Inoltre, un motivo di conflitto potrebbe essere la mancata rimozione da *client_list* del client che ha deciso di uscire ed è a tal proposito che abbiamo aggiunto *client_list.remove()*.

Uno dei problemi che abbiamo riscontrato riguarda la chiusura del server (*status='close'* e *check=2*) e la conseguente terminazione dei thread attivi. Abbiamo provato a sfruttare l'attributo *daemon* dei thread⁵ per gestire la chiusura delle connessioni, ma a quanto pare si incorre in una sorta di conflitto qualora si usi una console interattiva ed il programma non venga avviato invece da terminale⁶; pur inserendo esplicitamente *sys.exit()* il problema sussiste. È per questo motivo che in *message_exchange* è presente una condizione di controllo riguardante proprio la variabile globale **check**.

⁵In Python, un qualsiasi thread non-daemon ancora attivo impedisce al programma principale di uscire. D'altro canto, i thread di tipo daemon vengono "uccisi" non appena il programma principale termina.

⁶Discussioni su StackOverflow: <https://stackoverflow.com/questions/21843916/python-daemon-thread-does-not-exit-when-parent-thread-exits>, <https://stackoverflow.com/questions/57467403/python-daemon-threads-are-not-exiting-on-windows>.

Client

Abbiamo lasciato lo script lato client praticamente invariato, a riprova del fatto che nel cambiare modalità da Client-Server a Chat Room ciò che realmente subisce delle modifiche è il server, che ha il ruolo di intermediario tra i diversi client connessi (l'interazione resta 1:1 tra client e server, ma lo scambio di messaggi diventa condiviso con gli utenti all'interno della stanza).

Client con GUI

Nello script *TCPClient_gui.py* abbiamo voluto implementare un'interfaccia grafica molto rudimentale e, per fare ciò, abbiamo utilizzato il modulo *tkinter* (modulo già presente nella libreria standard di Python).

A livello concettuale l'impostazione è molto simile a quella dello script *TCPClient_multi-thread.py*, la differenza sostanziale (oltre al fatto che i messaggi ora vengono inseriti e stampati in una finestra dedicata) riguarda la fase di invio: finora abbiamo sfruttato i thread e i metodi del modulo *thread* per raggiungere l'asincronia del programma, tuttavia, il modulo *tkinter* ed in particolare l'introduzione di widget come *Button* permettono, tramite il metodo *mainloop*, di ottenere lo stesso risultato. È per questo motivo che il thread *snd_thread* è stato rimosso ed al suo posto è stato introdotto *send_button = tk.Button(window, text="Send", command=lambda: send_message(clientSocket))*⁷.

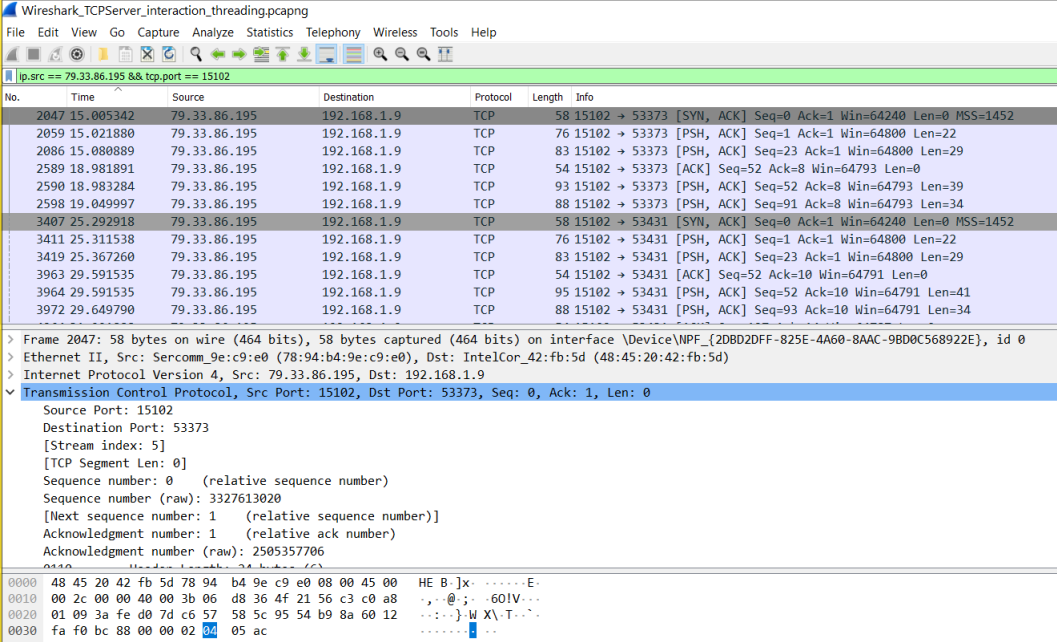
Test

1. Client-Server in locale (uno e più client)
2. Client-Server su macchine diverse
3. Client in locale e su macchina esterna
4. Client e Server di autori diversi
5. Client da terminale e client con GUI
6. Server contattato da più IP diversi (tentativo con i colleghi del corso), fig.5-6

Di seguito riportiamo alcuni screenshot di Wireshark relativi ad alcuni dei test di interoperabilità sopracitati.

⁷È da notare che *Button* presenta problemi quando si fa riferimento ad una funzione con parametri; per ovviare a questa difficoltà, abbiamo fatto ricorso alla funzione *lambda*.

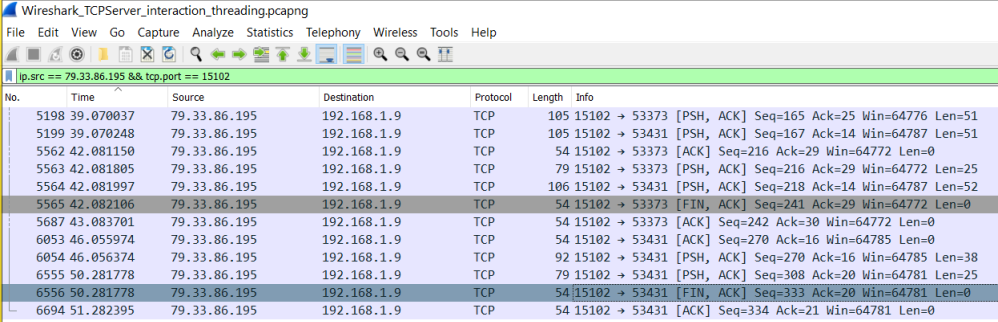
Client-Server 1:1



No.	Time	Source	Destination	Protocol	Length	Info
2047	15.005342	79.33.86.195	192.168.1.9	TCP	58	15102 → 53373 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1452
2059	15.021880	79.33.86.195	192.168.1.9	TCP	76	15102 → 53373 [PSH, ACK] Seq=1 Ack=1 Win=64800 Len=22
2086	15.080889	79.33.86.195	192.168.1.9	TCP	83	15102 → 53373 [PSH, ACK] Seq=23 Ack=1 Win=64800 Len=29
2589	18.981891	79.33.86.195	192.168.1.9	TCP	54	15102 → 53373 [ACK] Seq=52 Ack=8 Win=64793 Len=0
2590	18.983284	79.33.86.195	192.168.1.9	TCP	93	15102 → 53373 [PSH, ACK] Seq=52 Ack=8 Win=64793 Len=39
2598	19.049997	79.33.86.195	192.168.1.9	TCP	88	15102 → 53373 [PSH, ACK] Seq=91 Ack=8 Win=64793 Len=34
3407	25.292918	79.33.86.195	192.168.1.9	TCP	58	15102 → 53431 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1452
3411	25.311538	79.33.86.195	192.168.1.9	TCP	76	15102 → 53431 [PSH, ACK] Seq=1 Ack=1 Win=64800 Len=22
3419	25.367260	79.33.86.195	192.168.1.9	TCP	83	15102 → 53431 [PSH, ACK] Seq=23 Ack=1 Win=64800 Len=29
3963	29.591535	79.33.86.195	192.168.1.9	TCP	54	15102 → 53431 [ACK] Seq=52 Ack=10 Win=64791 Len=0
3964	29.591535	79.33.86.195	192.168.1.9	TCP	95	15102 → 53431 [PSH, ACK] Seq=52 Ack=10 Win=64791 Len=41
3972	29.649790	79.33.86.195	192.168.1.9	TCP	88	15102 → 53431 [PSH, ACK] Seq=93 Ack=10 Win=64791 Len=34

Frame 2047: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device\NPF_{20BD2DFF-825E-4A60-8AAC-9BD0C568922E}, id 0
 Ethernet II, Src: Sercomm_9e:c9:e0 (78:94:b4:9e:c9:e0), Dst: IntelCor_42:fb:5d (48:45:20:42:fb:5d)
 Internet Protocol Version 4, Src: 79.33.86.195, Dst: 192.168.1.9
 Transmission Control Protocol, Src Port: 15102, Dst Port: 53373, Seq: 0, Ack: 1, Len: 0
 Source Port: 15102
 Destination Port: 53373
 [Stream index: 5]
 [TCP Segment Len: 0]
 Sequence number: 0 (relative sequence number)
 Sequence number (raw): 3327613020
 [Next sequence number: 1 (relative sequence number)]
 Acknowledgment number: 1 (relative ack number)
 Acknowledgment number (raw): 2505357706
 Header length: 24 bytes (6)
 0000 48 45 20 42 fb 5d 78 94 b4 9e c9 e0 08 00 45 00 HE B :]x.E-
 0010 00 2c 00 00 40 00 3b 06 d8 36 4f 21 56 c3 c0 a8 ., .@. . 60IV...
 0020 01 09 3a fe d0 7d c6 57 58 5c 95 54 b9 8a 60 12 .: .} W X\ T...
 0030 fa f0 bc 88 00 00 02 05 ac

Figura 1: Wireshark - TCP interoperability test (vista lato Client), i pacchetti di tipo *SYN* individuano l'avvio della connessione TCP.



No.	Time	Source	Destination	Protocol	Length	Info
5198	39.070037	79.33.86.195	192.168.1.9	TCP	105	15102 → 53373 [PSH, ACK] Seq=165 Ack=25 Win=64776 Len=51
5199	39.070248	79.33.86.195	192.168.1.9	TCP	105	15102 → 53431 [PSH, ACK] Seq=167 Ack=14 Win=64787 Len=51
5562	42.081150	79.33.86.195	192.168.1.9	TCP	54	15102 → 53373 [ACK] Seq=216 Ack=29 Win=64772 Len=0
5563	42.081805	79.33.86.195	192.168.1.9	TCP	79	15102 → 53373 [PSH, ACK] Seq=216 Ack=29 Win=64772 Len=25
5564	42.081997	79.33.86.195	192.168.1.9	TCP	106	15102 → 53431 [PSH, ACK] Seq=218 Ack=14 Win=64787 Len=52
5565	42.082106	79.33.86.195	192.168.1.9	TCP	54	15102 → 53373 [FIN, ACK] Seq=241 Ack=29 Win=64772 Len=0
5687	43.083701	79.33.86.195	192.168.1.9	TCP	54	15102 → 53373 [ACK] Seq=242 Ack=30 Win=64772 Len=0
6053	46.055974	79.33.86.195	192.168.1.9	TCP	54	15102 → 53431 [ACK] Seq=270 Ack=16 Win=64785 Len=0
6054	46.056374	79.33.86.195	192.168.1.9	TCP	92	15102 → 53431 [PSH, ACK] Seq=270 Ack=16 Win=64785 Len=38
6555	50.281778	79.33.86.195	192.168.1.9	TCP	79	15102 → 53431 [PSH, ACK] Seq=308 Ack=20 Win=64781 Len=25
6556	50.281778	79.33.86.195	192.168.1.9	TCP	54	15102 → 53431 [FIN, ACK] Seq=333 Ack=20 Win=64781 Len=0
6694	51.282395	79.33.86.195	192.168.1.9	TCP	54	15102 → 53431 [ACK] Seq=334 Ack=21 Win=64781 Len=0

Figura 2: Wireshark - TCP interoperability test (vista lato Client), i pacchetti di tipo *FIN* individuano la fine della connessione TCP.

No.	Time	Source	Destination	Protocol	Length	Info
3563	23.657969619	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=1 Ack=1 Win=64240 Len=0
3515	23.712695316	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=1 Ack=23 Win=64218 Len=0
3531	23.767970989	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=1 Ack=52 Win=64189 Len=0
4113	27.938268111	95.251.48.71	192.168.1.82	TCP	63	53431 → 15102 [PSH, ACK] Seq=1 Ack=52 Win=64189 Len=9
4121	27.99631139	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=10 Ack=93 Win=64148 Len=0
4131	28.056786074	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=10 Ack=127 Win=64114 Len=0
4422	30.338315728	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [PSH, ACK] Seq=10 Ack=127 Win=64114 Len=4
4438	30.396103695	95.251.48.71	192.168.1.82	TCP	60	53373 → 15102 [ACK] Seq=8 Ack=165 Win=64076 Len=0
4430	30.39611171	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=14 Ack=167 Win=64074 Len=0
5445	37.416556206	95.251.48.71	192.168.1.82	TCP	71	53373 → 15102 [PSH, ACK] Seq=8 Ack=165 Win=64076 Len=17
5455	37.475975893	95.251.48.71	192.168.1.82	TCP	60	53373 → 15102 [ACK] Seq=25 Ack=216 Win=64025 Len=0
5456	37.475984663	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=14 Ack=218 Win=64023 Len=0
5828	40.428997576	95.251.48.71	192.168.1.82	TCP	60	53373 → 15102 [PSH, ACK] Seq=25 Ack=216 Win=64025 Len=4
5834	40.440864478	95.251.48.71	192.168.1.82	TCP	60	53373 → 15102 [ACK] Seq=29 Ack=242 Win=64000 Len=0
5841	40.496858612	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=14 Ack=270 Win=63971 Len=0
5886	41.438929299	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [FIN, ACK] Seq=14 Ack=270 Win=64000 Len=0
6398	44.402450499	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [PSH, ACK] Seq=14 Ack=270 Win=63971 Len=2
6415	44.476410329	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=16 Ack=308 Win=63933 Len=0
6951	48.626405388	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [PSH, ACK] Seq=16 Ack=308 Win=63933 Len=4
6955	48.642379563	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [ACK] Seq=20 Ack=334 Win=63998 Len=0
7082	49.627754897	95.251.48.71	192.168.1.82	TCP	60	53431 → 15102 [FIN, ACK] Seq=20 Ack=334 Win=63968 Len=0

Frame 7082: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

- Ethernet II, Src: AvastSecureNet (38:10:05:be:6f:4f), Dst: AsrockIn71:23:9c (d0:50:99:71:23:9c)
- Internet Protocol Version 4, Src: 95.251.48.71, Dst: 192.168.1.82
- Transmission Control Protocol, Src Port: 53431, Dst Port: 15102, Seq: 20, Ack: 334, Len: 0
 - Source Port: 53431
 - Destination Port: 15102
 - [Stream index: 18]
 - [TCP Segment Len: 0]
 - Sequence number: 20 (relative sequence number)
 - [Next sequence number: 20 (relative sequence number)]
 - Acknowledgment number: 334 (relative ack number)
 - 0101 = Header Length: 20 bytes (5)
 - Flags: 0x011 (FIN, ACK)
 - Window size value: 63998
 - [Calculated window size: 63998]
 - [Window size scaling factor: -2 (no window scaling used)]
 - Checksum: 0x0e31 [Unverified]
 - [Checksum Status: Unverified]
 - Urgent pointer: 0
 - [Timestamps]

```

0000  d0 50 99 71 23 9c 38 10  d5 be 6f 4f 08 00 45 00  P  q#8 . . . 00 . E .
0008  00 28 04 e4 40 7b 06 9f  a4 5f f0 47 00 00 00 00  [ . { . . . . .
0020  01 52 d0 b7 3a fe c7 f7  7a 78 5e 42 d8 b6 59 11  R . . . . . z x ^ ( P .
0030  f9 a4 8e 31 00 00 00 00  00 00 00 00 00 00 00  . . . . .

```

Figura 3: Wireshark - TCP interoperability test (vista lato Server).

Wireshark_TCPServer_interaction.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 15102

No.	Time	Source	Destination	Protocol	Length	Info
38035	329.753810	167.86.124.211	192.168.1.9	TCP	56	15102 → 53035 [PSH, ACK] Seq=8 Ack=9 Win=64256 Len=2
38036	329.753810	167.86.124.211	192.168.1.9	TCP	54	15102 → 53035 [FIN, ACK] Seq=10 Ack=9 Win=64256 Len=0
38037	329.753894	192.168.1.9	167.86.124.211	TCP	54	53035 → 15102 [ACK] Seq=9 Ack=11 Win=132096 Len=0
38038	329.754416	192.168.1.9	167.86.124.211	TCP	54	52630 → 15102 [FIN, ACK] Seq=4 Ack=8 Win=132096 Len=0
38042	329.794400	167.86.124.211	192.168.1.9	TCP	54	15102 → 52630 [RST] Seq=8 Win=0 Len=0
1336.	1129.880651	192.168.1.9	167.86.124.211	TCP	54	53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1337.	1130.181341	192.168.1.9	167.86.124.211	TCP	54	[TCP Retransmission] 53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1337.	1130.781786	192.168.1.9	167.86.124.211	TCP	54	[TCP Retransmission] 53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1339.	1131.982943	192.168.1.9	167.86.124.211	TCP	54	[TCP Retransmission] 53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1341.	1134.391982	192.168.1.9	167.86.124.211	TCP	54	[TCP Retransmission] 53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1347.	1139.211742	192.168.1.9	167.86.124.211	TCP	54	[TCP Retransmission] 53035 → 15102 [FIN, ACK] Seq=9 Ack=11 Win=132096 Len=0
1358.	1148.811926	192.168.1.9	167.86.124.211	TCP	54	53035 → 15102 [RST, ACK] Seq=10 Ack=11 Win=0 Len=0

Window size value: 516
 [Calculated window size: 132096]
 [Window size scaling factor: 256]
 Checksum: 0x02a7 [unverified]
 [Checksum Status: Unverified]
 Urgent pointer: 0

▼ [SEQ/ACK analysis]
 [RTT: 0.042291000 seconds]
 ▼ [TCP Analysis Flags]
 > [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
 [The RTO for this segment was: 0.901135000 seconds]
 [RTO based on delta from frame: 133677]
 > [Timestamps]

```

0000  78 94 b4 9e c9 e0 48 45 20 42 fb 5d 08 00 45 00  x....HE B.]..
0010  00 28 14 32 40 00 80 06 00 c3 c0 a8 01 09 a7 56  -(2@....V
0020  7c d3 cf 2b 3a fe 3a 16 32 51 56 81 f8 3a 50 11  |...::2QV..P
0030  02 04 02 a7 00 00  .....

```

Figura 4: Wireshark - TCP interoperability test (vista lato Client), i pacchetti di tipo *RST* individuano una chiusura forzata della connessione TCP. L'indirizzo IP sorgente è diverso da quello degli esempi precedenti perché abbiamo fatto una prova anche su macchina virtuale hostata in rete presso un service provider, a cui è stato inviato lo script tramite protocollo FTP.

Chatroom

```

Il server è pronto a ricevere.
> Digitare 'stop' per bloccare l'avvio di nuove connessioni
   o 'close' per chiudere il server: Nuova connessione stabilita! Il client è: ('185.128.27.238', 55642)
Nuova connessione stabilita! Il client è: ('188.152.47.22', 51988)
Nuova connessione stabilita! Il client è: ('213.45.56.198', 57378)
Nuova connessione stabilita! Il client è: ('5.95.240.42', 52657)
> lele('188.152.47.22', 51988): celestino!!!!
> Antonio('213.45.56.198', 57378): we
> Celestino('185.128.27.238', 55642): Ciao ragazzi
> Italo('5.95.240.42', 52657): Celestiii
> Antonio('213.45.56.198', 57378): Che ve ne pare?
> lele('188.152.47.22', 51988): funge
> Italo('5.95.240.42', 52657): fatto molto bene
> Italo('5.95.240.42', 52657): (y)
> Celestino('185.128.27.238', 55642): Grazie <3
> Antonio('213.45.56.198', 57378): Appost
> lele('188.152.47.22', 51988): quit
Connessione con il client ('lele', ('188.152.47.22', 51988)) chiusa.
> Antonio('213.45.56.198', 57378): :)
Nuova connessione stabilita! Il client è: ('93.40.229.40', 4389)
> Italo('5.95.240.42', 52657): vamos
> Italo('5.95.240.42', 52657): exit
Connessione con il client ('Italo', ('5.95.240.42', 52657)) chiusa.
> Celestino('185.128.27.238', 55642): quit
Connessione con il client ('Celestino', ('185.128.27.238', 55642)) chiusa.
Nuova connessione stabilita! Il client è: ('79.49.62.147', 3611)
> Antonio('213.45.56.198', 57378): quit
Connessione con il client ('Antonio', ('213.45.56.198', 57378)) chiusa.
> Raffaele('93.40.229.40', 4389): ciao raga!
Nuova connessione stabilita! Il client è: ('188.152.47.22', 51990)
Nuova connessione stabilita! Il client è: ('213.45.56.198', 57473)
> lele('188.152.47.22', 51990): sono tornato
> chris('79.49.62.147', 3611): ciao
> Antonio('213.45.56.198', 57473): Chi c'è?
> chris('79.49.62.147', 3611): ciao guys
> chris('79.49.62.147', 3611): sono christian
> Antonio('213.45.56.198', 57473): Ciao Christian!
> chris('79.49.62.147', 3611): Ciao antonio <3
close> chris('79.49.62.147', 3611): lol a morte telegram così è più figo

```

Figura 5: Chatroom con i colleghi del corso (vista lato Server).

531	11	642984328	185.128.27.238	192.168.1.82	TCP	74	55642	-15192	[SYN] Seq=0 Win=64240 Len=0 MSS=1326 SACK_PERM=1 TSval=2324243628 TSecr=0 WS=128
532	11	643989318	192.168.1.82	185.128.27.238	TCP	74	15192	-55642	[SYN, ACK] Seq=0 Ack=1 Win=65168 Len=0 MSS=1466 SACK_PERM=1 TSval=1447065594 TSecr=2324243628 WS=128
537	11	702938635	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2324243676 TSecr=1447065594
538	11	702444192	192.168.1.82	185.128.27.238	TCP	66	15192	-55642	[ACK] Seq=1 Ack=1 Win=65280 Len=2 TSval=1447065654 TSecr=2324243676
540	11	748123432	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=1 Ack=23 Win=64256 Len=0 TSval=1447065654 TSecr=1447065654
541	11	748139761	192.168.1.82	185.128.27.238	TCP	66	15192	-55642	[ACK] Seq=23 Ack=1 Win=65280 Len=2 TSval=1447065760 TSecr=2324243735
544	11	734989571	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=1 Ack=52 Win=64256 Len=0 TSval=2324243780 TSecr=1447065760
619	14	584885173	188.152.47.22	192.168.1.82	TCP	74	51988	-15192	[SYN] Seq=0 Win=64240 Len=0 MSS=1452 SACK_PERM=1 TSval=3059259461 TSecr=0 WS=128
620	14	584760744	192.168.1.82	188.152.47.22	TCP	74	15192	-51988	[SYN, ACK] Seq=0 Ack=1 Win=65168 Len=0 MSS=1460 SACK_PERM=1 TSval=1447065661 TSecr=3059259461 WS=128
623	14	621541171	188.152.47.22	192.168.1.82	TCP	66	51988	-15192	[ACK] Seq=1 Ack=1 Win=65280 Len=0 TSval=1447065661 TSecr=2324381601
624	14	621799997	192.168.1.82	188.152.47.22	TCP	88	15192	-51988	[PSH, ACK] Seq=1 Ack=1 Win=65280 Len=2 TSval=225301638 TSecr=3059259461
625	14	657974803	188.152.47.22	192.168.1.82	TCP	66	51988	-15192	[ACK] Seq=1 Ack=23 Win=64256 Len=0 TSval=3059259536 TSecr=225301638
626	14	657958514	192.168.1.82	188.152.47.22	TCP	95	15192	-51988	[PSH, ACK] Seq=23 Ack=1 Win=65280 Len=2 TSval=225301674 TSecr=3059259536
674	16	197380820	185.128.27.238	192.168.1.82	TCP	66	51988	-15192	[ACK] Seq=1 Ack=52 Win=64256 Len=0 TSval=1447065760 TSecr=1447065760
674	16	197384689	192.168.1.82	185.128.27.238	TCP	66	15192	-55642	[ACK] Seq=52 Ack=0 Win=65280 Len=0 TSval=1447061149 TSecr=2324248185
676	16	197949080	192.168.1.82	185.128.27.238	TCP	136	15192	-55642	[PSH, ACK] Seq=10 Ack=10 Win=65280 Len=78 TSval=1447061149 TSecr=2324248185
678	16	243765174	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=10 Ack=122 Win=64256 Len=0 TSval=2324248232 TSecr=1447061149
680	16	243759862	192.168.1.82	185.128.27.238	TCP	100	15192	-55642	[PSH, ACK] Seq=122 Ack=10 Win=65280 Len=34 TSval=1447061195 TSecr=2324248232
681	16	280913230	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=10 Ack=156 Win=64256 Len=0 TSval=2324248277 TSecr=1447061195
710	16	910986598	5.95.240.42	192.168.1.82	TCP	68	52657	-15192	[SYN] Seq=0 Win=64240 Len=0 MSS=1452 WS=256 SACK_PERM=1
711	16	917011243	192.168.1.82	5.95.240.42	TCP	66	15192	-52657	[SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1468 SACK_PERM=1 WS=128
714	16	909935806	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[SYN] Seq=0 Win=64240 Len=0 MSS=1452 WS=256 SACK_PERM=1
716	16	999873185	192.168.1.82	213.45.56.198	TCP	66	15192	-57378	[SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
717	16	910969171	192.168.1.82	213.45.56.198	TCP	76	15192	-57378	[PSH, ACK] Seq=1 Ack=1 Win=64256 Len=22
720	17	976547687	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=1 Ack=23 Win=132996 Len=0
721	17	976034444	192.168.1.82	213.45.56.198	TCP	66	15192	-57378	[PSH, ACK] Seq=1 Ack=52 Win=64256 Len=29
722	17	990703499	5.95.240.42	192.168.1.82	TCP	66	52657	-15192	[ACK] Seq=1 Ack=1 Win=68568 Len=0
726	17	134233661	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=1 Ack=52 Win=131840 Len=0
729	17	232988371	192.168.1.82	213.45.56.198	TCP	70	51988	-15192	[PSH, ACK] Seq=1 Ack=52 Win=64256 Len=4 TSval=3059262111 TSecr=225301674
730	17	233063888	192.168.1.82	188.152.47.22	TCP	66	15192	-51988	[ACK] Seq=52 Ack=5 Win=65280 Len=0 TSval=225304249 TSecr=3059262111
731	17	233102414	185.128.27.238	192.168.1.82	TCP	124	15192	-55642	[PSH, ACK] Seq=52 Ack=5 Win=65280 Len=5 TSval=1447062184 TSecr=2324248277
732	17	233116470	192.168.1.82	188.152.47.22	TCP	131	15192	-51988	[PSH, ACK] Seq=52 Ack=5 Win=65280 Len=65 TSval=225304249 TSecr=3059262111
735	17	268941225	188.152.47.22	192.168.1.82	TCP	66	51988	-15192	[ACK] Seq=52 Ack=117 Win=64256 Len=0 TSval=3059262148 TSecr=225304249
736	17	268951446	192.168.1.82	188.152.47.22	TCP	100	15192	-51988	[PSH, ACK] Seq=117 Ack=5 Win=65280 Len=34 TSval=225304249 TSecr=3059262148
737	17	270646164	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=10 Ack=215 Win=64256 Len=0 TSval=2324249267 TSecr=1447062184
738	17	303200756	192.168.1.82	188.152.47.22	TCP	66	51988	-15192	[ACK] Seq=151 Win=64256 Len=0 TSval=3059262182 TSecr=225304249
772	17	953782819	192.168.1.82	5.95.240.42	TCP	76	15192	-52657	[PSH, ACK] Seq=1 Ack=1 Win=64256 Len=22
783	18	266654398	5.95.240.42	192.168.1.82	TCP	66	52657	-15192	[ACK] Seq=1 Ack=23 Win=66568 Len=0
784	18	266961213	192.168.1.82	5.95.240.42	TCP	83	15192	-52657	[ACK] Seq=23 Ack=1 Win=64256 Len=29
843	18	874414832	192.168.1.82	5.95.240.42	TCP	83	15192	-52657	[ACK] Seq=23 Ack=1 Win=64256 Len=29
844	18	810002355	5.95.240.42	192.168.1.82	TCP	66	52657	-15192	[ACK] Seq=23 Ack=1 Win=64256 Len=29
845	19	918302535	5.95.240.42	192.168.1.82	TCP	66	15192	-52657	[ACK] Seq=23 Ack=1 Win=64256 Len=29
903	20	581076312	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=1 Ack=52 Win=131840 Len=7
904	20	581090480	192.168.1.82	213.45.56.198	TCP	54	15192	-57378	[ACK] Seq=1 Ack=8 Win=64256 Len=0
905	20	581171411	192.168.1.82	185.128.27.238	TCP	128	15192	-55642	[PSH, ACK] Seq=215 Ack=10 Win=65280 Len=62 TSval=1447065593 TSecr=2324249267
906	20	581186561	192.168.1.82	185.128.27.238	TCP	128	15192	-51988	[PSH, ACK] Seq=215 Ack=10 Win=65280 Len=62 TSval=225307598 TSecr=3059262182
907	20	581260334	192.168.1.82	213.45.56.198	TCP	66	15192	-57378	[ACK] Seq=52 Ack=8 Win=64256 Len=0
908	20	617955574	185.128.27.238	192.168.1.82	TCP	66	51988	-15192	[ACK] Seq=5 Ack=213 Win=64256 Len=0 TSval=3059265497 TSecr=225307598
911	20	620086817	192.168.1.82	185.128.27.238	TCP	66	55642	-15192	[ACK] Seq=10 Ack=277 Win=64256 Len=0 TSval=2324252616 TSecr=1447065593
912	20	635217629	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=8 Ack=120 Win=131840 Len=0
913	20	635236368	192.168.1.82	213.45.56.198	TCP	66	15192	-57378	[ACK] Seq=120 Ack=8 Win=64256 Len=34
916	20	688729663	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=10 Ack=317 Win=131840 Len=0
925	21	347276636	5.95.240.42	192.168.1.82	TCP	66	52657	-15192	[PSH, ACK] Seq=1 Ack=52 Win=66568 Len=5
926	21	347291788	5.95.240.42	192.168.1.82	TCP	54	15192	-52657	[ACK] Seq=52 Ack=8 Win=64256 Len=0
927	21	347347784	192.168.1.82	185.128.27.238	TCP	124	15192	-55642	[PSH, ACK] Seq=277 Ack=10 Win=65280 Len=58 TSval=1447066299 TSecr=2324252616
928	21	347355732	192.168.1.82	185.128.27.238	TCP	124	15192	-51988	[PSH, ACK] Seq=277 Ack=10 Win=65280 Len=58 TSval=225308364 TSecr=3059265497
929	21	347360626	192.168.1.82	213.45.56.198	TCP	128	15192	-57378	[ACK] Seq=213 Ack=5 Win=65280 Len=58 TSval=225308364 TSecr=3059265497
930	21	347367031	192.168.1.82	5.95.240.42	TCP	120	15192	-52657	[PSH, ACK] Seq=52 Ack=8 Win=64256 Len=66
933	21	394144427	185.128.27.238	192.168.1.82	TCP	66	55642	-15192	[ACK] Seq=10 Ack=235 Win=64256 Len=0 TSval=2324253382 TSecr=1447066299
934	21	405796968	213.45.56.198	192.168.1.82	TCP	66	57378	-15192	[ACK] Seq=8 Ack=212 Win=131840 Len=0
937	21	581111193	192.168.1.82	185.128.27.238	TCP	124	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
939	21	630203630	185.128.27.238	192.168.1.82	TCP	66	52657	-15192	[ACK] Seq=212 Ack=8 Win=64256 Len=66
940	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
941	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
942	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
943	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
944	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
945	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
946	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
947	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
948	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
949	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
950	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
951	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
952	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
953	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
954	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
955	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
956	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
957	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
958	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
959	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
960	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
961	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
962	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
963	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
964	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66
965	21	630719688	185.128.27.238	192.168.1.82	TCP	76	15192	-52657	[ACK] Seq=212 Ack=8 Win=64256 Len=66