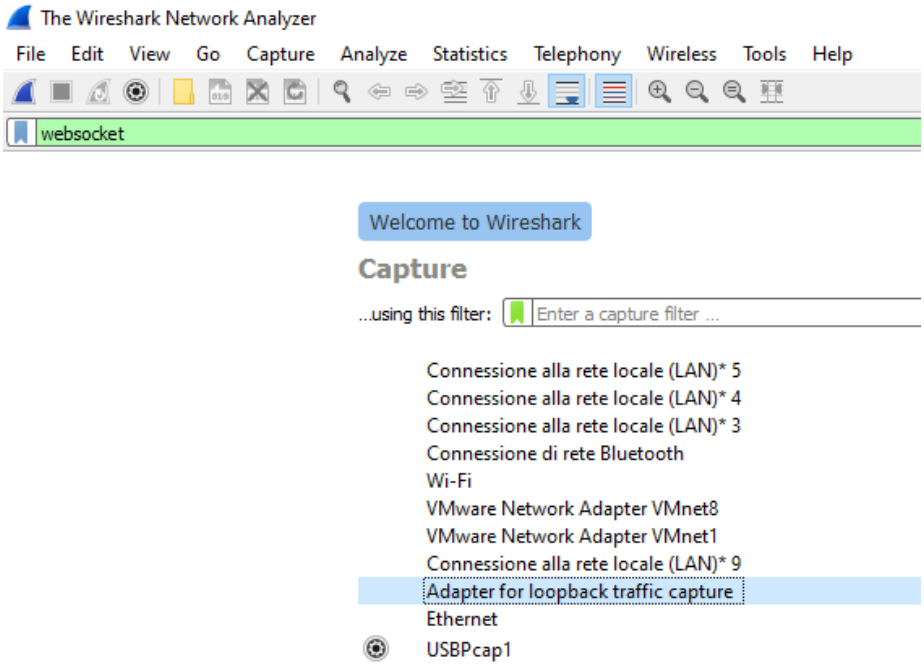
2.12.7 Esercizi

Esercizio 1

Lanciare l'applicazione dopo aver fatto partire l'ispezione del Network tramite la console di sviluppo del browser. Ogni quanto tempo il client fa sapere al server che è ancora connesso? È un'azione dovuta all'implementazione della chat o insita nel WebSocket? A cosa sere tale procedura?
Lanciare Wireshark e vedere cosa passa in rete sulla connessione TCP interessata.

Soluzione esercizio 1

Per analizzare la rete si utilizza il software Wireshark che consente di analizzare il flusso di pacchetti in entrata e in uscita. All'apertura del software, andando nella sezione "*Adapter for loopback traffic capture*" sarà possibile seguire tutti i pacchetti che riguardano il localhost. Per filtrare il risultato dei pacchetti, si inserisce la stringa "websocket" nella barra in alto, così da mostrare solamente quei pacchetti con protocollo WebSocket:



A questo punto, si aprono tre, quattro client. Quindi, si scrive l'URL localhost:4000 nel browser. Dato che il server non è in esecuzione, il browser non riesce a collegarsi al localhost:4000 poiché vede tale porta inutilizzata. Di conseguenza, il traffico catturato da Wireshark è inesistente.

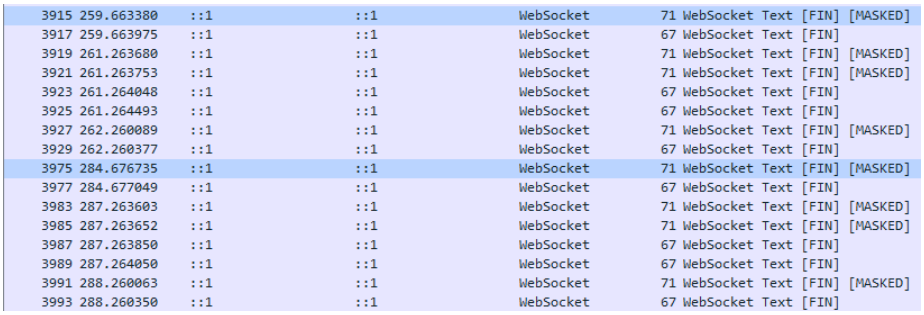
Avviando il server, in automatico vedrà il collegamento dei 3/4 client avviati precedentemente. Di conseguenza, su Wireshark appariranno dei pacchetti corrispondenti al collegamento dei client al server:

No.	Time	Source	Destination	Protocol	Length	Info
3777	234.656388	::1	::1	WebSocket	76	WebSocket Text [FIN] [MASKED]
3779	234.657653	::1	::1	WebSocket	72	WebSocket Text [FIN]
3786	234.763868	::1	::1	WebSocket	71	WebSocket Text [FIN] [MASKED]
3807	235.266748	::1	::1	WebSocket	76	WebSocket Text [FIN] [MASKED]
3809	235.267019	::1	::1	WebSocket	72	WebSocket Text [FIN]
3818	235.279132	::1	::1	WebSocket	76	WebSocket Text [FIN] [MASKED]
3820	235.279339	::1	::1	WebSocket	72	WebSocket Text [FIN]
3824	235.370623	::1	::1	WebSocket	71	WebSocket Text [FIN] [MASKED]
3828	235.390373	::1	::1	WebSocket	71	WebSocket Text [FIN] [MASKED]
3843	236.269003	::1	::1	WebSocket	76	WebSocket Text [FIN] [MASKED]
3845	236.269236	::1	::1	WebSocket	72	WebSocket Text [FIN]
3849	236.376232	::1	::1	WebSocket	71	WebSocket Text [FIN] [MASKED]

Cliccando su uno dei pacchetti con flag [MASKED] è possibile notare una cosa interessante riguardo il protocollo TCP. Ovvero, il numero di porta d'origine e destinazione. Per esempio, nell'immagine è possibile vedere come un client con porta 51021 (*Source Port*) stia comunicando con il server sulla sua porta 4000 (*Destination Port*):

```
▾ Transmission Control Protocol, Src Port: 51021, Dst Port: 4000, Seq: 633, Ack: 130, Len: 12
    Source Port: 51021
    Destination Port: 4000
    [Stream index: 327]
    [Conversation completeness: Incomplete, DATA (15)]
    [TCP Segment Len: 12]
    Sequence Number: 633    (relative sequence number)
    Sequence Number (raw): 286870038
    [Next Sequence Number: 645    (relative sequence number)]
    Acknowledgment Number: 130    (relative ack number)
    Acknowledgment number (raw): 1135635018
    0101 .... = Header Length: 20 bytes (5)
  ▸ Flags: 0x018 (PSH, ACK)
    Window: 10229
    [Calculated window size: 2618624]
    [Window size scaling factor: 256]
    Checksum: 0xfcd9 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  ▸ [Timestamps]
  ▸ [SEQ/ACK analysis]
    TCP payload (12 bytes)
    [PDU Size: 12]
```

Adesso che è chiaro quale siano i client (quelli "marchiati" con MASKED e il motivo per cui i messaggi sono mascherati è dovuto ad una questione di sicurezza) e quale il server, è possibile vedere sulla colonna (la terza) di sinistra qual'è il tempo in cui ogni client comunica al server che è ancora vivo:



3915	259.663380	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3917	259.663975	::1	::1	WebSocket	67 WebSocket Text [FIN]
3919	261.263680	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3921	261.263753	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3923	261.264048	::1	::1	WebSocket	67 WebSocket Text [FIN]
3925	261.264493	::1	::1	WebSocket	67 WebSocket Text [FIN]
3927	262.260089	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3929	262.260377	::1	::1	WebSocket	67 WebSocket Text [FIN]
3975	284.676735	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3977	284.677049	::1	::1	WebSocket	67 WebSocket Text [FIN]
3983	287.263603	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3985	287.263652	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3987	287.263850	::1	::1	WebSocket	67 WebSocket Text [FIN]
3989	287.264050	::1	::1	WebSocket	67 WebSocket Text [FIN]
3991	288.260063	::1	::1	WebSocket	71 WebSocket Text [FIN] [MASKED]
3993	288.260350	::1	::1	WebSocket	67 WebSocket Text [FIN]

In questo caso, al tempo 259 il client con porta 51021 ha comunicato al server che è ancora vivo. Ovviamente il server ha risposto con un ACK e successivamente gli altri 3 client hanno comunicato al server la loro presenza. Al tempo 284, nuovamente il client con porta 51021 ricomunica al server che è ancora vivo (idem per gli altri). Si deduce che il client fa sapere al server che è ancora connesso ogni 25 secondi circa ($284 - 259 = 25$).
Documentazione ufficiale riguardo al Keep-Alive nel protocollo WebSocket: `https://websockets.readthedocs.io/en/stable/topics/timeouts.html`
RFC documentation: `https://www.rfc-editor.org/rfc/rfc6455#page-36`