# Firecell_5GNetwork_Test

July 3, 2024

ajdfjsdajf

## 1 Analisi delle Prestazioni dell'infrastruttura di Rete Firecell 5G

### 1.1 Firecell - Labkit:

Il Firecell-Labkit è un'infrastruttura che permette di distribuire e utilizzare una rete di comunicazione basata su 4G/5G sia in modalità stand-alone (senza l'utilizzo della rete 4G) sia in modalità non-stand-alone in cui vengono sfruttati sia i nodi appartenenti alla rete 4G (enB) sia in nodi appartenenti alla rete 5G (gnB). Il Firecell-Labkit è fornito sia di componenti software sia di tools necessari alla distribuzione della rete e alla validazione del sistema. Tra essi troviamo: * **OAI 5GCN(AUSF,UDM,UDR,AMF,SMF,UPF)** * **MME** * **OAI EPC(HSS, SPGWC,SPGWU)** * **OAI RAN(eNodeB,gNodeB)** * **Wireshark** * **UHD(USRP HW DRIVERS)** * **SCRCPY(Remote access to Android UE)** Tutte queste componenti, sia hardware sia software, cooperano al fine di interconnettere i vari **User Equipement(UE)** che avranno necessità di comunicare all'interno della rete. ### 5GCN: 5G Core Network Si tratta di una rete progettata con un'architettura **orientata ai servizi (SBA)** definita dal 3GPP. In questo tipo di architettura si sfruttano una serie di componenti appartenenti al 5GC, detti anche **Funzioni di Rete (NF)**, che interagiscono in modo da fornire e richiedere servizi ad altri **NF** autorizzati ad accedere ai propri. Tale interazione avviene con la logica **produttore-consumatore**. Le funzioni di rete legate al **Control Plane (CP)** sono separate rispetto a quelle appartenti allo **User Plane (UP)** in modo da renderle scalabili in maniera indipendente. * **User-Plane(Data plane):** trasporta il traffico degli utenti della rete. * **Contol-Plane:** controlla come i **pacchetti** dati sono trasferiti. Il processo di **Routing Table** è considerato parte del CP. La **Core Network** è schematicamentente rappresentata dall'**AMF** e **UPF**: * **AMF(Access & Mobility Management Funcion):** accede sia allo **User Equipement** e al **RAN**. * **UPF(User Plane Function):** gestisce i dati dell'utente. ### MME: Mobility Management Entity Si tratta del principale responsabile della gestione della mobilità all'interno della rete LTE e 5G. Tale gestione comprende il tracciamento e la mobilità dello UE. Il MME performa funzioni di gestione della sessione, tra cui: creazione, modifica e terminazione di sessioni di comunicazione tra UE e la rete. Inoltre supervisiona la creazione, modifica e rilascio di portanti (canali logici che trasportano dati utente tra UE e rete 5G). ### RAN: Radio Access Network La principale entità dell'NG-RAN è il **gNodeB (gNB)**. Quest'ultimo può essere suddivisono in una **Central Unit** e uno o più **Distribuited Unit(s)** connesse all'interfaccia radio. Il nodo radio gnB condente all'UE di conttersi al 5GCN NG (punto di riferimento tra l'accesso e il core network 5G, ed è costituito da parecchie interfacce, maggiormente N2 e N3). Nel caso NSA (non-stand-alone) in cui viene sfruttata anche l'interfacce aerea 4G LTE, il nodo radio di riferimento è il **eNodeB (enB)** che fornisce terminazioni E-UTRAN UP e CP verso l'UE. Si collega anch'esso al NG-CORE tramite l'interfaccia NG. ### UE: User

Equipement Si tratta di un qualisasi dispositivo utilizzato direttamente dall'utilizzatore finale per comunicare. Esso si connette alla **base station node B (gnB/enB)** come specificato dal 3GPP. L'UE gestisce i seguenti compiti verso la rete centrale: * gestione **mobilità**; * gestione **chiamate**; * gestione **sessione**; * gestione **identità**. I protocolli corrispondenti bengono trasmessi in modo trasparente tramite un **NODE B** che non modifica/utilizza/comprende le informazioni. Questi protocolli sono detti **NON ACCESS STRATUM**. ## Test sulla rete Firecell 5G: L'obiettivo di questi test è la valutazione delle prestazioni della rete 5G implementata tramite un core Firecell. Le situazioni di test considerate riguardano la comunicazione tra un server centrale (sul nodo centrale Firecell) e un client, con l'aggiunta di vari livelli di traffico generato tramite iperf. Inoltre, al fine di verificare i time-stamp ottenuti durante la comunicazione client-server, viene utilizzato Wireshark per ottenere il log dettagliato dei pacchetti. ### Componenti del Test: 1. Client: implementato su dispositivo esterno con modem 5G dotato di SIM connesso alla rete Firecell. Ha la funzione di inviare pacchetti TCP al server variando la dimensione del payload. Una volta inviati, ha il compito di porsi in ascolto della risposta del server. Inoltre, esso ha il compito di registrare il time-stamp di invio e ricezione e salvarli in un file CSV in modo da effettuare il calcolo dell'RTT. 2. Server: posto sul nodo centrale della rete Firecell 5G e posto in ascolto dei messaggi inviati dal client. Ha il compito di effettuare un echo dei pacchetti TCP inviati dal client una volta ricevuti. 3. Iperf: permette di configurare un ulteriore server e uno o più client in modo da geneare del traffico aggiuntivo per osservare il comportamento della rete. 4. Wireshark: permette di monitorare e registrare i pacchetti TCP per verificare i time-stamp. ### Scenari di Test: 1. Comunicazione TCP tra Client e Server: * **Client:** invia pacchetti TCP di diverse dimensioni (64, 128, 256, 512, 1024, 2048, 4096, 8192, 16284, 32768, 65536 bytes) al server. Registra il time-stamp di invio e ricezione e salva i dati in un file CSV. * **Server:** avviato sul nodo centrale con indirizzo IP: XXXX. Riceve e rimanda indietro i pacchetti. * **Wireshark:** monitora e registra i pacchetti con gli indirizzi IP rispettivi di client e server. 2. Comunicazione TCP con traffico di rete generato da iperf: * Client, Server e Wireshark come nella situazione di test 1. * **Iperf:** genera traffico di rete con vari livelli di intensità. Quest'ultime sono: 1. 10 Mbps (basso traffico). 2. 100 Mbps (traffico moderato). 3. 1 Gbps (alto traffico). 4. 90% della capacità massima della rete (quasi saturazione).
Ogni test salva i time-stamp di invio e ricezione in un file CSV con la seguente struttura di **HEADER**: Payload;Timestamp Invio;Timestamp Ricezione;RTT. ### Implementazione programmi per il test: Di seguito sono riportati e descritti i programmi sviluppati sia per la realizzazione della comunicazione client-server di test, sia per la configurazione e l'avvio dei programmi iperf e wireshark necessari ai fini dei test di comunicazione. #### SERVER TCP:

[ ]:
```
import socket #importa la libreria contenente funzioni per la comunicazione␣
 ↪protocollare
import argparse #importa la libreria per passare argomenti alle funzioni da␣
 ↪riga di comando

def start_tcp_server(host, port): #funzione per l'avvio di un server TCP con␣
 ↪IP=host e sulla PORTA=port
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:␣
 ↪#definisce il tipo di protocollo
        server_socket.bind((host, port)) #connette il server all'indirizzo IP e␣
 ↪porta passati come argomento
        server_socket.listen() #pone il server in ascolto di richieste di␣
 ↪connessione dal client
```

```
        print(f"Server TCP in ascolto su {host}:{port}")

        while True:
            conn, addr = server_socket.accept() #accetta la connessione da␣
 ↪client
            with conn:
                print(f"Connected by {addr}")
                while True:
                    data = conn.recv(65536) #preleva il dati presenti nel␣
 ↪buffer connesso tra client-server
                    if not data: #se non ci sono messaggi o richieste
                        break #termina il ciclo interno e si ripone in ascolto␣
 ↪di una connessione
                    print(f"Ricevuti {len(data)} bytes da {addr}")
                    conn.sendall(data)  # Echo dei dati ricevuti

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="TCP Server") #Descrizione␣
 ↪della funzione realizzata
    parser.add_argument('--host', default='0.0.0.0', help='Host to listen on')␣
 ↪#Indirizzo IP
    parser.add_argument('--port', type=int, default=12345, help='Port to listen␣
 ↪on') #Port di ascolto
    args = parser.parse_args() #ricava gli argomenti passati da riga di comando

    start_tcp_server(args.host, args.port) #avvia il server TCP
```

**CLIENT_TCP:**

[ ]:
```
#Programma Python per la realizzazione di un Client sfruttando il protocollo␣
 ↪TCP.
#Questo client ha la funzione di inviare verso il server specificato dai␣
 ↪parametri,
#un messaggio contenente il payload generato dalla dimensione passata tramite␣
 ↪parametro.
#Una volta inviato, tale Client si pone in attesa di una risposta dal server␣
 ↪che sarà
#realizzato in modo da emettere un echo del messaggio.
import socket
import argparse
import Utility

def tcp_client(server_host,server_port,payload_size,type_test):
    file=open("Istanti_temportali.csv","a") #apre il file csv in cui il client␣
 ↪andrà a salvare gli istanti temporali e le relative informazioni di test
    file.write("Inviato;Ricevuto;PackSize;Test;\n") #formato delle colonne del␣
 ↪file csv
```

```
    try:
        with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as client_socket:␣
↪#avvia il protocollo TCP
            client_socket.connect((server_host,server_port)) #Cerca la␣
↪connessione con il server TCP
        #   print(f"Connessione al server TCP {server_host}:{server_port}")
            payload=b'a'*payload_size #imposta la dimensione del payload come␣
↪messaggio da mandare in byte
            sec1,us1=Utility.time_stamp() #funzione definità nel file Utility␣
↪che estrae l'istante temporale dividendo i secondi dalle sue frazioni
            client_socket.sendall(payload) #invia il messaggio al server
        #   print(f"Inviati {payload_size} bytes")
            data=client_socket.recv(65536) #buffer impostato al massimo in modo␣
↪da effettuare test con variazione del payload
            sec2,us2=Utility.time_stamp() #salva l'istante immediatamente␣
↪successivo all'arrivo del messaggio da parte del server
            file.write(str(sec1)+'.'+str(us1)+';') #scrive le informazioni nel␣
↪file csv
            file.write(str(sec2)+'.'+str(us2)+';')
            file.write(str(payload_size)+';'+type_test+';')
            rtt_sec=sec2-sec1 #calcolo dell'RTT
            if rtt_sec >= 0: #controllo del segno dei secondi per verificare␣
↪che non sia a cavallo dei 60 s
                rtt_us=us2-us1 # nel caso sia corretto
            else:
                rtt_us=us1-us2 # nel caso per qualche motivo risulti negativo
            file.write(str(rtt_us)+';')
            file.write("\n")
            file.write("\n")
        #   print(f"Ricevuto {len(data)} bytes da {server_host}:{server_port}")
            file.close() #chiude il file csv
    except Exception as e:
        print(f"Eccezione {e} durante la connessione con il server.") #comunica␣
↪eventuali problematiche relative alla connessione con il server


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="TCP Client") #Argomenti␣
↪relativi alla chiamata del programma per l'avvio del client
    parser.add_argument('--server_host', default='127.0.0.1', help='Server␣
↪host')
    parser.add_argument('--server_port', type=int, default=12345, help='Server␣
↪port')
    parser.add_argument('--payload_size', type=int, default=1024, help='Payload␣
↪size in bytes')
```

```
    parser.add_argument('--type_test', default='no-traffico',help='Tipologia di␣
 ↪test')
    args = parser.parse_args()

    tcp_client(args.server_host, args.server_port, args.payload_size, args.
 ↪type_test)
```

**IPERF & WIRESHARK** Per l'avvio e la configurazione di client e server iperf e l'utilizzo del software wireshark sono stati sviluppati dei programmi in Shell Script che permettono la gestione di queste entità.

[ ]:
```
#!/bin/bash
#Funzione per l'avvio di un server IPERF passando IP, PORT e file di␣
 ↪destinazione dell'output
start_iperf_server(){
    local ip=$1 #primo parametro passato come argomento
    local port=$2 #secondo parametro passato come argomento
    echo "Avvio del server iperf su $ip : $port ..."
    iperf3 -s -B $ip -p $port & #avvio del server iperf in background␣
 ↪sull'indirizzo e port indicati
}
#Funzione per l'avvio di un client IPERF passando IP del server, PORT e durata␣
 ↪in secondi della comunicazione
start_iperf_client() {
    local ip=$1 #primo parametro passato come argomento: IP del server
    local port=$2 #secondo parametro passato come argomento: PORT del server
    local duration=$3 #terzo parametro passato come argomento: durata della␣
 ↪comunicazione
    local bitrate=$4 #quarto parametro passato come argomento: bitrate target␣
 ↪per la comunicazione
    iperf3 -c $ip -p $port -t $duration -b $bitrate & #avvio del client iperf␣
 ↪in background con i parametri indicati
    return $! #restituisce il PID del processo relativo all'avvio del client
}
#Funzione per l'avvio del programma wireshark in modo da catturare i pacchetti␣
 ↪utili per i test
avvio_tshark(){
    local interfaccia=$1 #primo parametro passato come argomento: interfaccia␣
 ↪di ascolto
    local durata=$2 #secondo parametro passato come argomento: durata␣
 ↪dell'ascolto
    local output_file=$3 #terzo parametro passato come argomento: file in cui␣
 ↪verrano salvati i risultati dell'ascolto
    echo "Avvio cattura dei pacchetti sull'interfaccia $interfaccia per $durata␣
 ↪s. I dati vengono salvati in $output_file ..."
```

```
    tshark -i $interfaccia -a duration:$durata -w $output_file & #avvio del␣
 ↪programma wireshark con i parametri indicati
    return $! #restituisce il PID del processo relativo all'avvio di wireshark
}
```

Queste funzioni realizzate vengono utilizzate all'interno di ulteriori due programmi realizzati anch'essi in Shell Script che permettono il controllo e la realizzazione dei test di rete necessari ai fini dello studio. In particolare: * **Start_Server.sh:** questo programma, da avviare sul nodo centrale della rete Firecell 5G, permette l'avvio del server TCP, la configurazione e l'avvio del server Iperf e la configurazione e l'inizio dell'ascolto da parte del programma Wireshark.

[ ]:
```
#!/bin/bash
#Programma utilizzato per l'avvio dei server da lato core 5G posti sul nodo␣
 ↪centrale della rete 5G Firecell. Inoltre è anche configurato
#l'applicazione wireshark al fine di ottenere i valori dei time-stamp relativi␣
 ↪ai pacchetti scambiati nella fase di test della rete.
#-------------------------------------------------------------------------------
#Importo i file contenenti le funzioni utilizzate per avviare e controllare i␣
 ↪Client e
#server, la lettura del file di configurazione, la configurazione di client e␣
 ↪server iperf e
#la configurazione dell'ascolto effettuato da wireshark.
chmod +x Lettura_File_Config.sh #fornisce i permessi per eseguire i programmi
chmod +x Iperf.sh
chmod +x Wireshark.sh
chmod +x Analisi_Pacchetti.sh
source Lettura_File_Config.sh #Lettura del file di configurazione contenenti i␣
 ↪valori utilizzati per i test
. Iperf.sh #Funzioni per l'avvio di server il client iperf
. Wireshark.sh #Funzione per l'avvio dell'ascolto su wireshark
#-------------------------------------------------------------------------------
IP_SERVER=$(ini_get_value server ip) #indirizzo IP del server di echo
PORT_SERVER=$(ini_get_value server port) #port in cui si pone in ascolto il␣
 ↪server di echo
IPERF_SERVER_IP=$(ini_get_value iperf ip_server) #indirizzo IP del server iperf
IPERF_SERVER_PORT=$(ini_get_value iperf server_port) #porta su cui il server si␣
 ↪pone in ascolto
WIRESHARK_INTERFACCIA=$(ini_get_value wireshark interfaccia) #interfaccia di␣
 ↪ascolto di wireshark
WIRESHARK_DURATA=$(ini_get_value wireshark durata) #durata dell'ascolto di␣
 ↪wireshark
WIRESHARK_OUTPUT_FILE=$(ini_get_value wireshark output_file) #file in cui␣
 ↪vengono salvati i risultati ottenuti dall'ascolto di wireshark
FILE_CSV_WIRESHARK=$(ini_get_value wireshark out_csv) #file in formato csv␣
 ↪contenente i pacchetti di interesse dell'analisi di rete
#-------------------------------------------------------------------------------
```

```
python3 Server.py --host "$IP_SERVER" --port "$PORT_SERVER" & #avvio del server␣
 ↪con i parametri ottenuti da file di configurazione
start_iperf_server $IPERF_SERVER_IP $IPERF_SERVER_PORT & #avvia il server iperf␣
 ↪con i parametri ottenuti da file di configurazione
avvio_tshark $WIRESHARK_INTERFACCIA $WIRESHARK_DURATA $WIRESHARK_OUTPUT_FILE␣
 ↪#avvia l'ascolto di wireshark con i parametri ottenuti da file di␣
 ↪configurazione
wait $! #attende il termine del processo di ascolto di wireshark
if [ $? -eq 0 ]; then #controlla lo stato d'uscita dell'ascolto di wireshark␣
 ↪evidenziando eventuali errori
    echo "Cattura completata. File salvato in $WIRESHARK_OUTPUT_FILE"
  # Avvia Wireshark per analizzare il file di cattura
  #  wireshark $WIRESHARK_OUTPUT_FILE &
  #  kill $!
else
    echo "Errore nella cattura dei pacchetti."
    exit 1
fi
# Terminare il server iperf e python3
killall iperf3
killall python3
# Avvia l'analisi dei pacchetti ottenuti da wireshark in modo da estrarre␣
 ↪solamente quelli di interesse
./Analisi_Pacchetti.sh "$WIRESHARK_OUTPUT_FILE" "$IP_SERVER"␣
 ↪"$FILE_CSV_WIRESHARK"
```

- **Client_Test.sh:** questo programma permette l'avvio del client iperf con i valori di bitrate target impostati per i vari scenari di test. Inoltre, per ogni scenario di test, provvede ad avviare il client TCP con differenti dimensioni del payload.

[ ]:
```
#!/bin/bash
#Fornisce i permessi e le funzioni necessarie all'avvio delle entità client per␣
 ↪il test
chmod +x Lettura_File_Config.sh
. Iperf.sh
source Lettura_File_Config.sh
# Definisci i valori dimensione payload
dim_payload=("8" "16" "32" "64" "128" "256" "512" "1024" "2048" "4096" "8192"␣
 ↪"16384" "32768" "65536") #dimensioni di payload in bit
dim_bw=("10M" "100M" "1G" "9G") #Bitrate
IP_SERVER=$(ini_get_value server ip) #indirizzo IP del server TCP a cui il␣
 ↪client si connette
PORT_SERVER=$(ini_get_value server port) #port in cui il server TCP è in ascolto
IPERF_CLIENT_DURATION=$(ini_get_value iperf client_duration) #durata del test␣
 ↪realizzato tramite iperf
IPERF_CLIENT_PORT=$(ini_get_value iperf client_port) #port in cui si pone il␣
 ↪client iperf
```

```
IPERF_SERVER_IP=$(ini_get_value iperf ip_server) #indirizzo IP del server iperf
IPERF_SERVER_PORT=$(ini_get_value iperf server_port) #port del server iperf

for dim in "${dim_bw[@]}" #loop che varia il traffico generato dal client iperf
  do
    #chiamata a funzione di avvio del client con i parametri passati
    start_iperf_client $IPERF_SERVER_IP $IPERF_CLIENT_PORT␣
 ↪$IPERF_CLIENT_DURATION $dim #salvataggio del pid del processo client iperf␣
 ↪in modo da poter sapere quando è terminato
    pid=$!
    echo "Client iperf avvianto al: $IPERF_SERVER_IP:$IPERF_SERVER_PORT per␣
 ↪$IPERF_CLIENT_DURATION s con $dim di bitrate."
    for param1 in "${dim_payload[@]}" #loop che varia la dimensione dei payload␣
 ↪dei messaggi scambiati tra client e server
      do
        python3 Client.py --server_host "$IP_SERVER" --server_port␣
 ↪"$PORT_SERVER" --payload_size "$param1" --type_test "$dim" & #programma␣
 ↪python che avvia il client di test
        wait $! #aspetta che client e server abbiano comunicato
      done
    wait $pid #aspetta che iperf finisca
  done
```

Questi programmi prendono i loro parametri da un file di configurazione, chiamato **config.ini** che presenta la seguente struttura: [sezione] chiave = valore parametro … [sezione] chiave = valore parametro … [sezione] chiave = valore parametro … [sezione] chiave = valore parametro … Per la lettura da tale file è stato realizzato un programma, **Lettura_File_Config.sh**, che permette di aprire il file di configurazione e fornisce le funzioni per ricavare i parametri di interesse. In particolare: * **ini_get_section:** restituisce le sezioni del file di configurazione. * **ini_get_key_value:** restituisce le chiavi di una sezione passata come parametro. * **ini_get_value:** restituisce il valore relativo alla **sezione** e **chiave** passati come parametro. Infine, tramite il programma **Analisi_Pacchetti.sh** viene elaborato il file di output di wireshark permettendo di estrarre i pacchetti di interesse ai fini dei test sulla rete.

[ ]:
```
#!/bin/bash

# Verifica che siano stati passati i parametri necessari
if [ "$#" -ne 3 ]; then
    echo "Uso: $0 <file.pcapng> <indirizzo IP> <output.csv>"
    exit 1
fi

# Parametri
PCAP_FILE=$1 #argomento relativo al file di output di wireshark
IP_ADDRESS=$2 #argomento relativo all'indirizzo IP di interesse dei pacchetti
OUTPUT_CSV=$3 #argomento relativo al file di out in formato csv
```

```
# Estrazione dei pacchetti relativi all'indirizzo IP specificato
tshark -r "$PCAP_FILE" -Y "ip.src == $IP_ADDRESS || ip.dst == $IP_ADDRESS" -T␣
 ↪fields -e frame.number -e frame.time -e ip.src -e ip.dst -e ip.proto -e ip.
 ↪len -e frame.len -E header=y -E separator=, -E quote=d > "$OUTPUT_CSV"

echo "Estrazione completata. I dati sono stati salvati in $OUTPUT_CSV"
```