CODICE SERVER

```
"""
Implementazione del Server per il protocollo Go-Back-N ARQ
Versione modificata con gestione migliorata dei file di log
"""

import socket
import json
import time
import os
import glob
from datetime import datetime
from typing import Dict, List, Optional


class GBNServer:
    """
    Server per il protocollo Go-Back-N ARQ che accetta solo pacchetti in ordine
    e risponde con ACK appropriati.
    """

    def __init__(self, host: str = "localhost", port: int = 12345, loss_probability: float = 0.0):
        """
        Inizializza il server Go-Back-N.
        """
        # Configurazione socket UDP
        self.host = host
        self.port = port
        self.socket = None
        self._create_socket()

        # Reset stato
        self.reset_state()

        # Simulazione perdita ACK
        self.loss_probability = loss_probability

        self.running = False

        # Gestione file di log
        self.log_file = self._get_log_filename()
        self._setup_log_file()

    def _get_log_filename(self):
        """Genera il nome del file di log con numerazione incrementale basata su sessione."""
        # Crea la cartella logs se non esiste
        if not os.path.exists('logs'):
            os.makedirs('logs')
```

```
    # Trova il prossimo numero di sessione disponibile
    session_number = self._get_next_session_number()

    if session_number == 0:
        return os.path.join('logs', 'demo_run.log')
    else:
        return os.path.join('logs', f'demo_run{session_number}.log')

def _get_next_session_number(self):
    """Trova il prossimo numero di sessione disponibile controllando tutti i possibili file."""
    # Pattern per trovare tutti i file demo_run*.log
    pattern = os.path.join('logs', 'demo_run*.log')
    existing_files = glob.glob(pattern)

    if not existing_files:
        return 0

    # Estrai tutti i numeri dai nomi dei file esistenti
    numbers = []
    for file_path in existing_files:
        filename = os.path.basename(file_path)
        if filename == 'demo_run.log':
            numbers.append(0)
        elif filename.startswith('demo_run') and filename.endswith('.log'):
            try:
                # Estrai il numero tra 'demo_run' e '.log'
                number_str = filename[8:-4]  # Rimuovi 'demo_run' e '.log'
                if number_str.isdigit():
                    numbers.append(int(number_str))
            except:
                continue

    if not numbers:
        return 0

    # Restituisci il prossimo numero disponibile
    return max(numbers) + 1

def _setup_log_file(self):
    """Inizializza il file di log con header."""
    try:
        with open(self.log_file, 'w', encoding='utf-8') as f:
            f.write(f"=== LOG SESSIONE GO-BACK-N SERVER ===\n")
            f.write(f"Avvio: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
            f.write(f"Server: {self.host}:{self.port}\n")
            f.write(f"Loss Probability: {self.loss_probability}\n")
            f.write("="*60 + "\n\n")
```

```
        except Exception as e:
            print(f"Errore creazione file log: {e}")
            self.log_file = None

    def _create_socket(self):
        """Crea e configura il socket UDP"""
        try:
            if self.socket:
                self.socket.close()
        except:
            pass

        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        # Permetti riuso dell'indirizzo
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        # Timeout per evitare blocchi
        self.socket.settimeout(1.0)

        try:
            self.socket.bind((self.host, self.port))
        except OSError as e:
            if "Address already in use" in str(e) or "10048" in str(e):
                print(f"⚠️  Porta {self.port} in uso, tentativo con porta alternativa...")
                # Prova con una porta diversa
                for port in range(12346, 12356):
                    try:
                        self.port = port
                        self.socket.bind((self.host, self.port))
                        print(f"✅ Server spostato su porta {self.port}")
                        break
                    except:
                        continue
                else:
                    raise OSError("Nessuna porta disponibile")
            else:
                raise

    def reset_state(self):
        """Reset dello stato del server per nuovo test"""
        # Gestione sequenza pacchetti
        self.expected_seq_num = 0  # Prossimo numero di sequenza atteso
        self.last_ack_sent = -1    # Ultimo ACK inviato

        # Statistiche e logging
        self.stats = {
            'packets_received': 0,
            'packets_in_order': 0,
            'packets_out_of_order': 0,
```

```
            'acks_sent': 0,
            'acks_lost': 0
        }
        self.log_entries = []
        self.received_messages = []  # Messaggi ricevuti in ordine

    def log_event(self, event_type: str, seq_num: int = None, message: str = ""):
        """Registra un evento nel log con timestamp dettagliato."""
        timestamp = datetime.now().strftime("%H:%M:%S.%f")[:-3]
        log_entry = f"[{timestamp}] SERVER - {event_type}"

        if seq_num is not None:
            log_entry += f" SEQ:{seq_num}"

        if message:
            log_entry += f" - {message}"

        self.log_entries.append(log_entry)
        print(log_entry)

        # Salva nel file di log
        if self.log_file:
            try:
                with open(self.log_file, 'a', encoding='utf-8') as f:
                    f.write(log_entry + '\n')
            except Exception as e:
                print(f"Errore scrittura log: {e}")

    def create_ack(self, ack_num: int) -> bytes:
        """Crea un pacchetto ACK."""
        ack_packet = {
            'ack_num': ack_num,
            'timestamp': time.time()
        }
        return json.dumps(ack_packet).encode('utf-8')

    def simulate_ack_loss(self) -> bool:
        """Simula la perdita casuale di un ACK."""
        if self.loss_probability > 0:
            import random
            lost = random.random() < self.loss_probability
            if lost:
                self.stats['acks_lost'] += 1
            return lost
        return False

    def send_ack(self, ack_num: int, client_addr: tuple):
        """Invia un ACK al client con possibile simulazione di perdita."""
```

```
        # Simula perdita ACK
        if self.simulate_ack_loss():
            self.log_event("ACK_LOSS", ack_num, f"ACK simulato come perso verso
{client_addr}")
            return

        try:
            ack_packet = self.create_ack(ack_num)
            self.socket.sendto(ack_packet, client_addr)
            self.stats['acks_sent'] += 1
            self.last_ack_sent = ack_num
            self.log_event("ACK_SENT", ack_num, f"ACK inviato a {client_addr}")

        except Exception as e:
            self.log_event("ERROR", ack_num, f"Errore invio ACK: {e}")

    def process_packet(self, packet_data: bytes, client_addr: tuple):
        """Processa un pacchetto ricevuto dal client."""
        try:
            packet = json.loads(packet_data.decode('utf-8'))
            seq_num = packet['seq_num']
            data = packet['data']

            self.stats['packets_received'] += 1
            self.log_event("RECV", seq_num, f"Ricevuto da {client_addr}: {data}")

            # Verifica se il pacchetto è quello atteso (in ordine)
            if seq_num == self.expected_seq_num:
                # Pacchetto in ordine: accetta e memorizza
                self.stats['packets_in_order'] += 1
                self.received_messages.append({
                    'seq_num': seq_num,
                    'data': data,
                    'timestamp': datetime.now()
                })

                self.log_event("ACCEPT", seq_num, "Pacchetto accettato (in ordine)")

                # Aggiorna il numero di sequenza atteso
                self.expected_seq_num += 1

                # Invia ACK per questo pacchetto
                self.send_ack(seq_num, client_addr)

            else:
                # Pacchetto fuori ordine: ignora ma invia ACK duplicato
                self.stats['packets_out_of_order'] += 1
```

```
            if seq_num < self.expected_seq_num:
                # Pacchetto duplicato o già ricevuto
                self.log_event("DUPLICATE", seq_num,
                               f"Pacchetto duplicato (atteso: {self.expected_seq_num})")
            else:
                # Pacchetto futuro (gap nella sequenza)
                self.log_event("OUT_OF_ORDER", seq_num,
                               f"Pacchetto fuori ordine (atteso: {self.expected_seq_num})")

            # Invia ACK per l'ultimo pacchetto ricevuto in ordine
            if self.last_ack_sent >= 0:
                self.send_ack(self.last_ack_sent, client_addr)
                self.log_event("DUP_ACK", self.last_ack_sent,
                               "ACK duplicato per pacchetto fuori ordine")

        except json.JSONDecodeError:
            self.log_event("ERROR", None, "Pacchetto con formato JSON non valido")
        except KeyError as e:
            self.log_event("ERROR", None, f"Campo mancante nel pacchetto: {e}")
        except Exception as e:
            self.log_event("ERROR", None, f"Errore processing pacchetto: {e}")

    def start_server(self):
        """Avvia il server in modalità di ascolto continuo."""
        self.running = True
        self.log_event("SERVER_START", None, f"Server avviato su {self.host}:{self.port}")

        try:
            while self.running:
                try:
                    # Ricevi pacchetto dal client
                    data, client_addr = self.socket.recvfrom(1024)
                    self.process_packet(data, client_addr)

                except socket.timeout:
                    # Timeout normale, continua il loop
                    continue
                except Exception as e:
                    if self.running:
                        self.log_event("ERROR", None, f"Errore ricezione: {e}")

        except Exception as e:
            if self.running:
                self.log_event("ERROR", None, f"Errore server: {e}")
        finally:
            self.stop_server()

    def stop_server(self):
```

```
    """Ferma il server e chiude il socket."""
    if not self.running:
        return

    self.running = False
    self.log_event("SERVER_STOP", None, "Server fermato")

    try:
        if self.socket:
            self.socket.close()
            self.socket = None
    except:
        pass

def print_statistics(self):
    """Stampa le statistiche del server in formato tabellare."""
    print("\n" + "="*60)
    print("STATISTICHE SERVER GO-BACK-N ARQ")
    print("="*60)
    print(f"{'Pacchetti ricevuti:':<25} {self.stats['packets_received']:>10}")
    print(f"{'Pacchetti in ordine:':<25} {self.stats['packets_in_order']:>10}")
    print(f"{'Pacchetti fuori ordine:':<25} {self.stats['packets_out_of_order']:>10}")
    print(f"{'ACK inviati:':<25} {self.stats['acks_sent']:>10}")
    print(f"{'ACK persi:':<25} {self.stats['acks_lost']:>10}")
    print(f"{'Messaggi accettati:':<25} {len(self.received_messages):>10}")

    if self.stats['packets_received'] > 0:
        in_order_rate = (self.stats['packets_in_order'] /
                         self.stats['packets_received']) * 100
        print(f"{'Tasso ordine corretto:':<25} {in_order_rate:>9.1f}%")

    if self.stats['acks_sent'] > 0:
        ack_loss_rate = (self.stats['acks_lost'] /
                         (self.stats['acks_sent'] + self.stats['acks_lost'])) * 100
        print(f"{'Tasso perdita ACK:':<25} {ack_loss_rate:>9.1f}%")

    print("="*60)

    # Salva le statistiche nel file di log
    if self.log_file:
        try:
            with open(self.log_file, 'a', encoding='utf-8') as f:
                f.write(f"\n{datetime.now().strftime('%H:%M:%S')} - STATISTICHE FINALI:\n")
                f.write(f"Pacchetti ricevuti: {self.stats['packets_received']}\n")
                f.write(f"Pacchetti in ordine: {self.stats['packets_in_order']}\n")
                f.write(f"Pacchetti fuori ordine: {self.stats['packets_out_of_order']}\n")
                f.write(f"ACK inviati: {self.stats['acks_sent']}\n")
                f.write(f"ACK persi: {self.stats['acks_lost']}\n")
```

```
                f.write(f"Messaggi accettati: {len(self.received_messages)}\n")
                if self.stats['packets_received'] > 0:
                    in_order_rate = (self.stats['packets_in_order'] /
                                     self.stats['packets_received']) * 100
                    f.write(f"Tasso ordine corretto: {in_order_rate:.1f}%\n")
                if self.stats['acks_sent'] > 0:
                    ack_loss_rate = (self.stats['acks_lost'] /
                                     (self.stats['acks_sent'] + self.stats['acks_lost'])) * 100
                    f.write(f"Tasso perdita ACK: {ack_loss_rate:.1f}%\n")
                f.write("="*60 + "\n")
            except Exception as e:
                print(f"Errore scrittura statistiche nel log: {e}")


# Esempio di utilizzo
if __name__ == "__main__":
    import sys

    # Parametri predefiniti
    host = "localhost"
    port = 12345
    loss_prob = 0.0

    # Gestione argomenti da riga di comando
    if len(sys.argv) > 1:
        try:
            port = int(sys.argv[1])
        except ValueError:
            print("Porta non valida, uso porta predefinita 12345")

    if len(sys.argv) > 2:
        try:
            loss_prob = float(sys.argv[2])
            if not 0.0 <= loss_prob <= 1.0:
                print("Probabilità di perdita deve essere tra 0.0 e 1.0")
                loss_prob = 0.0
        except ValueError:
            print("Probabilità di perdita non valida, uso 0.0")

    print(f"🚀 Avvio Server Go-Back-N ARQ")
    print(f"📍 Host: {host}")
    print(f"🔌 Porta: {port}")
    print(f"📉 Probabilità perdita ACK: {loss_prob}")
    print("="*50)

    # Crea e avvia il server
    server = GBNServer(host=host, port=port, loss_probability=loss_prob)
```

```
try:
    server.start_server()
except KeyboardInterrupt:
    print("\n🛑 Interruzione richiesta dall'utente")
except Exception as e:
    print(f"\n❌ Errore del server: {e}")
finally:
    server.stop_server()
    server.print_statistics()
    print(f"\n📋 Log salvato in: {server.log_file}")
    print("👋 Server terminato")
```