Programmazione di Reti - Progetto 3

Uhalid Abou EL Kheir

14 giugno 2024

Indice

1 Introduzione

Utilità di monitoraggio della rete scritta in Python. L'utilità utilizza richieste ICMP Echo per verificare la raggiungibilità degli host e segnala il loro stato come online o offline.

2 Requisiti

- Python 3
- Privilegi amministrativi (per l'accesso ai socket raw)
- File di configurazione (`config.json`)

3 Panoramica del Codice

3.1 Calcolo del Checksum

La funzione `checksum` calcola il checksum per il pacchetto ICMP.

```
def checksum(data):
    """Calcola il checksum ICMP."""
    s = 0
    n = len(data) % 2
    for i in range(0, len(data) - n, 2):
        s += (data[i] << 8) + data[i + 1]
    if n:
        s += (data[-1] << 8)
    while s >> 16:
        s = (s & 0xFFFF) + (s >> 16)
    return ~s & 0xFFFF
```

3.2 Creazione e Invio del Pacchetto ICMP

La funzione `ping_host` crea e invia un pacchetto di richiesta ICMP Echo e attende la risposta Echo.

```
def ping_host(host, timeout):
    """
    Invia un pacchetto ICMP ECHO_REQUEST e attende la risposta ECHO_REPLY.

    Args:
        host (str): L'indirizzo IP o il nome host da raggiungere.
        timeout (int): Timeout in secondi.

    Returns:
        bool: True se l'host risponde entro il timeout, False altrimenti.
    """
    try:
        dest_addr = socket.gethostbyname(host)
```

```
except socket.gaierror:
    print(f"Impossibile risolvere l'host: {host}")
    return False

icmp = socket.getprotobyname("icmp")
try:
    icmp_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp)
except socket.error as e:
    if e.errno == 1:
        e.msg += " - Prova ad eseguire come amministratore."
    print(f"Errore nella creazione del socket: {e}")
    return False

icmp_socket.settimeout(timeout)

icmp_type = 8  # ECHO_REQUEST
icmp_code = 0
icmp_checksum = 0
icmp_id = os.getpid() & 0xFFFF  #Usa il PID processo come ID
icmp_seq = 1
header = struct.pack('!BBHHH', icmp_type, icmp_code, \
    icmp_checksum, icmp_id, icmp_seq)

# Aggiungo il payload fisso
payload = b'A' * 41
packet = header + payload
icmp_checksum = checksum(packet)
header = struct.pack('!BBHHH', icmp_type, icmp_code, \
    icmp_checksum, icmp_id, icmp_seq)
packet = header + payload

try:
    icmp_socket.sendto(packet, (dest_addr, 0))
except socket.error as e:
    print(f"Errore nell'invio del pacchetto: {e}")
    return False

try:
    packet, _ = icmp_socket.recvfrom(1024)
    icmp_header = packet[20:28]
    _type, _code, _checksum, _id, _seq = \
        struct.unpack('!BBHHH', icmp_header)
    if _type == 0 and _id == icmp_id:  # ECHO_REPLY
        return True
except socket.timeout:
    return False
except socket.error as e:
    print(f"Errore nella ricezione del pacchetto: {e}")
```

```
            return False
        finally:
            icmp_socket.close()

    return False
```

3.3 Monitoraggio degli Host

La funzione `monitor_hosts` monitora continuamente una lista di host eseguendo il ping a intervalli regolari.

```
def monitor_hosts(hosts, sleep_time, timeout):
    while True:
        for host in hosts:
            if ping_host(host, timeout):
                print(f"{host} e' online")
            else:
                print(f"{host} e' offline")
        time.sleep(sleep_time)
```

3.4 Caricamento della Configurazione

La funzione `load_config` carica i parametri di configurazione da un file JSON.

```
def load_config(config_file):
    try:
        with open(config_file, 'r') as f:
            return json.load(f)
    except FileNotFoundError:
        print(f"File di configurazione non trovato: \
            {config_file}")
        raise
    except json.JSONDecodeError:
        print(f"Errore nel parsing del file di configurazione: \
            {config_file}")
        raise
```

3.5 Lettura degli Host

Le funzioni `read_hosts_from_file` e `read_hosts_from_console` leggono le informazioni sugli host da un file o dall'input dell'utente, rispettivamente.

```
def read_hosts_from_file(file_path):
    """
    Legge gli host da un file.

    Args:
        file_path (str): Percorso del file contenente gli host.
```

```
    Returns:
        list: Lista degli host letti dal file.
    """
    try:
        with open(file_path, 'r') as f:
            return [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print(f"File degli host non trovato: {file_path}")
        raise
    except Exception as e:
        print(f"Errore nella lettura del file degli host: {e}")
        raise

def read_hosts_from_console():
    hosts = []
    while True:
        host = input("Inserisci un host (lascia vuoto per terminare): ")
        if not host:
            break
        hosts.append(host)
    return hosts
```

3.6 Blocco Principale

```
try:
    config = load_config('config.json')

    mode = config.get('mode', 'console')  # default e' 'console'
    sleep_time = config.get('sleep_time', 5)  # default e' 5 secondi
    timeout = config.get('timeout', 2)  # default e' 2 secondi

    if mode == 'file':
        hosts_da_monitorare = read_hosts_from_file(config['hosts_file'])
    else:
        hosts_da_monitorare = read_hosts_from_console()

    monitor_hosts(hosts_da_monitorare, sleep_time, timeout)
except Exception as e:
    print(f"Errore critico: {e}")
```

4 Configurazione

Il file `config.json` definisce i parametri per l'utilità di monitoraggio:

- **mode** (`string`): Specifica la modalità di lettura degli host. Può assumere i seguenti valori:

  - `"console"`: Gli host vengono letti dall'input dell'utente tramite la console.
  - `"file"`: Gli host vengono letti da un file specificato dalla chiave `hosts_file`.

- **hosts_file** (`string`): Il percorso del file che contiene la lista degli host da monitorare. Questa chiave è utilizzata solo se `mode` è impostato su `"file"`.
- **sleep_time** (`integer`): Il tempo di attesa in secondi tra un controllo e l'altro. Questo parametro definisce quanto spesso viene eseguito il ping sugli host.
- **timeout** (`integer`): Il timeout in secondi per ogni richiesta di ping. Se l'host non risponde entro questo intervallo di tempo, viene considerato offline.

```
{
    "mode": "console",
    "hosts_file": "hosts.txt",
    "sleep_time": 5,
    "timeout": 2
}
```

5 Utilizzo

1. **Preparare la Configurazione**:
   - Modificare il file `config.json` per impostare i parametri desiderati.
2. **Eseguire lo Script**:
   - Eseguire lo script con privilegi amministrativi:

     ```
     sudo python3 monitor.py
     ```

3. **Monitorare l'Output**:
   - Lo script stamperà lo stato di ciascun host a intervalli regolari.

6 Gestione degli Errori

Il codice include vari meccanismi di gestione degli errori:

- **Operazioni sui File**:
  - Gestisce `FileNotFoundError` e `json.JSONDecodeError` durante il caricamento del file di configurazione.
  - Gestisce `FileNotFoundError` ed eccezioni generiche durante la lettura del file degli host.
- **Operazioni sui Socket**:

  - Aggiunge la gestione degli errori per le eccezioni durante la creazione del socket e l'invio/ricezione dei pacchetti.

- **Eccezioni Generali**:
  - Il blocco principale è avvolto in un blocco try-except per catturare e stampare eventuali eccezioni non gestite.

7 Conclusione

Questa utilità di monitoraggio della rete fornisce un modo semplice ma efficace per monitorare la raggiungibilità degli host utilizzando richieste ICMP Echo. È configurabile tramite un file JSON e può leggere le informazioni sugli host da un file o dall'input della console.