

Laboratorio 1: Redes de Computadores

Profesor: Viktor Tapia

Ayudantes de Laboratorio: Luciano Yevenes

1 de septiembre de 2025

1. Objetivos del Laboratorio

- Aprender a utilizar sockets UDP y TCP en un entorno local
- Entender las diferencias básicas entre TCP y UDP
- Implementar comunicación cliente-servidor
- Comprender el manejo básico de datos en red

2. Introducción

Los sockets son la interfaz básica para la comunicación en red. TCP garantiza que los datos lleguen en orden y sin errores, mientras que UDP es más rápido pero no garantiza la entrega.

En este laboratorio trabajaremos solo en modo local (localhost) para evitar complicaciones de red y enfocarnos en aprender los conceptos básicos.

3. Contexto: Centro de Comando Espacial

3.1. La Misión

La Estación Espacial Internacional necesita un nuevo sistema de comunicaciones de respaldo después de que una tormenta solar dañara sus sistemas principales. Como ingenieros junior de la NASA, han sido asignados para crear un prototipo de sistema de comunicaciones que permita:

1. **Comunicación confiable** entre la estación espacial y el centro de control terrestre
2. **Alertas rápidas** para emergencias que requieren respuesta inmediata

El sistema debe ser simple pero robusto, capaz de funcionar con recursos limitados en el espacio.

3.2. Arquitectura del Sistema

Centro de Control (Servidor TCP):

- Recibe y almacena todos los reportes de la estación espacial
- Mantiene un registro histórico de las comunicaciones
- Permite consultar mensajes anteriores para análisis

Sistema de Alertas (Servidor UDP):

- Envía confirmaciones inmediatas de mensajes recibidos
- No requiere almacenamiento permanente (para ahorrar memoria)
- Respuesta rápida para situaciones críticas

Estación Espacial (Clientes):

- Se comunica con ambos sistemas según la naturaleza del mensaje
- Reportes detallados van por TCP (confiable)
- Confirmaciones rápidas van por UDP (velocidad)

3.3. Escenario de Uso

Día típico en el espacio:

...

Astronauta: "Control, aquí Estación. Reporte matutino del sistema de oxígeno"

Centro de Control: "Mensaje recibido y almacenado. Todo nominal"

Astronauta: "¡ALERTA! Detectamos basura espacial aproximándose"

Sistema de Alertas: "Confirmación inmediata - Alerta recibida"

...

Su misión es implementar este sistema de comunicaciones que podría salvar vidas en el espacio.
¡La tripulación cuenta con ustedes!

4. Implementación

4.1. Preparación

- Python 3.6 o superior
- Todo en localhost (127.0.0.1)
- Puertos sugeridos: TCP=8001, UDP=8002

4.2. Parte A: Comunicación TCP

-
- **Centro de Control (Servidor TCP)**
 - **Archivo:** centro_control.py

Requisitos:

- Escuchar en puerto 8001 esperando comunicaciones de la estación espacial
- Aceptar **solo una conexión** a la vez (recursos limitados en el espacio)
- Procesar reportes de los astronautas y almacenarlos de forma segura

- Mantener un registro histórico de todas las comunicaciones
- Comandos espaciales a implementar:
 - REPORTE:mensaje → Almacena el reporte y confirma recepción
 - CONSULTAR → Devuelve historial de reportes para análisis
 - MISION_COMPLETA → Finaliza la sesión de comunicación

- **Estación Espacial (Cliente TCP)**
- **Archivo:** estacion_espacial.py

Requisitos:

- Conectar al Centro de Control en puerto 8001
- Permitir a los astronautas enviar reportes y consultas
- Mostrar las confirmaciones del Centro de Control
- Mantener comunicación hasta completar la misión

Ejemplo de comunicación espacial:

```
...
Astronauta> REPORTE:Sistema de oxígeno funcionando nominalmente
Centro de Control: Reporte almacenado. Todo bajo control, Estación.
Astronauta> REPORTE:Avistamiento de aurora boreal espectacular
Centro de Control: Reporte almacenado. Todo bajo control, Estación.
Astronauta> CONSULTAR
Centro de Control: === HISTORIAL DE COMUNICACIONES ===
                    1. Sistema de oxígeno funcionando nominalmente
                    2. Avistamiento de aurora boreal espectacular
Astronauta> MISION_COMPLETA
Centro de Control: Comunicación finalizada. Buen trabajo, astronautas.
...
```

4.3. Parte B: Comunicación UDP

- **Sistema de Alertas (Servidor UDP)**
- **Archivo:** sistema_alertas.py

Requisitos:

- Escuchar en puerto 8002 para alertas críticas
- Responder inmediatamente sin almacenar (optimizado para velocidad)
- No mantener historial entre mensajes (memoria limitada)
- Para cada alerta recibida, confirmar con: CONFIRMADO: [alerta recibida]"

-
- **Estación Espacial (Cliente UDP)**
 - **Archivo:** estacion_espacial.py

Requisitos:

- Conectar al Sistema de Alertas para comunicaciones críticas
- Enviar alertas de emergencia y recibir confirmación inmediata
- Permitir múltiples alertas durante la sesión
- Terminar cuando el astronauta escriba "base_segura"

Ejemplo de uso en emergencia:

```
...
Emergencia> Detección de basura espacial en trayectoria de colisión
Alerta: CONFIRMADO: Detección de basura espacial en trayectoria de colisión
Emergencia> Maniobra evasiva completada exitosamente
Alerta: CONFIRMADO: Maniobra evasiva completada exitosamente
Emergencia> Todos los sistemas nominales, peligro evitado
Alerta: CONFIRMADO: Todos los sistemas nominales, peligro evitado
Emergencia> base_segura
Sistema: Modo emergencia desactivado. Mantente seguro allá arriba.
...
```

5. Especificaciones Técnicas

5.1. Formato de Datos

- **TCP:** Mensajes de texto terminados en \n
- **UDP:** Mensajes de texto simples (máximo 1024 bytes)
- **Codificación:** UTF-8

5.2. Manejo de Errores

- Capturar errores de conexión básicos
- Mostrar mensajes de error comprensibles
- Cerrar conexiones correctamente

5.3. Estructura Básica Sugerida

Para servidores:

```
import socket

# Crear socket
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # TCP
# o socket.SOCK_DGRAM para UDP

# Configurar y escuchar
servidor.bind(('localhost', PUERTO))
# ... resto de la lógica
...
```

Para cliente:

```
import socket

# Crear socket y conectar
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # TCP
cliente.connect(('localhost', PUERTO))
# ... resto de la lógica
...
```

6. Informe

Extensión: Máximo 3 páginas

Preguntas a Responder:

1. **Diferencias Observadas** (1 página):
 - ¿Qué diferencias notaron entre TCP y UDP al implementar?
 - ¿Cuál fue más fácil de programar y por qué?
2. **Problemas y Soluciones** (1 página):
 - ¿Qué problemas tuvieron durante la implementación?
 - ¿Cómo los resolvieron?
3. **Reflexión** (1 página):
 - ¿En qué casos usarían TCP vs UDP?
 - ¿Qué aprendieron sobre la programación con sockets?

7. Entregables

Deben entregar un archivo .zip en el cual vengan todos los archivos con el formato Lab1_Rut_USACH.

7.1. Readme

- Nombres y RUT del integrante
- Instrucciones para ejecutar cada programa
- Ejemplo de uso de cada programa

7.2. Estructura Básica Sugerida

```
Lab1_RUT_USACH.zip
|-- centro_control.py      # Servidor TCP - Centro de Control
|-- estacion_espacial.py   # Cliente - Estación Espacial
|-- sistema_alertas.py     # Servidor UDP - Sistema de Alertas
|-- Informe.pdf
'-- README.md
```

8. Consideraciones Generales

- Se deberá enviar al correo buzontareasluciano@gmail.com a mas tardar el día 29 de Septiembre a las 23:59 horas.
- Se descontarán 10 puntos por cada hora o fracción de atraso.
- Las copias serán evaluadas con nota 1,0 en el promedio de las tareas.
- Si se detecta uso excesivo de IA, se llamará al grupo a una interrogación presencial con el ayudante, de la cual también será parte el profesor.
- La tarea debe ser hecha en el lenguaje Python. Se asume que usted sabe programar en este lenguaje, ha tenido vivencias con el, o que aprende con rapidez, ademas solo podra usar las librerias estandar de python ademas de:
 - socket: Para comunicación de red
 - sys: Para argumentos del sistema
 - time: Para delays si es necesario
- Los códigos serán pasados por un software anti-plagio, en caso de ser detectada copia se pondrá nota 0, hasta que se realice una reunión con las personas involucrados.
- Pueden crear todas las funciones auxiliares que deseen, siempre y cuando estén debidamente comentadas.
- Las preguntas deben ser hechas a través del servidor de Discord. De esta forma los demás grupos pueden verse beneficiados también.
- Si no se entrega README o MAKE, o si su programa no funciona, la nota es 1,0 hasta la corrección.
- Se descontarán hasta 50 puntos por:

- No respetar el formato de entrega.
 - No respetar las especificaciones del README.
 - Solicitar edición de código al momento de revisar.
 - Código poco prolijo y mal estructurado (ausencia de indentación adecuada, falta de consistencia en el estilo, nombres de variables confusos o poco descriptivos, comentarios insuficientes o irrelevantes).
- **Una vez publicadas las notas tendrán 5 días para apelar con el corrector que les revisó, después de este plazo las notas no tendrán ningún tipo de cambio.**

Discord de la asignatura: <https://discord.gg/592v5TBJU9>