UDP Handshake v2.1

This is the documentation to a task to perform a three-way handshake between a server and client, in ns-3, version 2.1 of this project.

Since the packets used for handshake are defined differently from the UDP specification, we can designate these special packets as **H-UDP (Handshake UDP).**

Implementation attempt on Python with ns-3 bindings

```
from ns import ns

# Define the UDP packet structure
class UDPPacket:
    def __init__(self, source_port, dest_port, seq_num, ack_num, flags, checksum, data):
        self.source_port = source_port
        self.dest_port = dest_port
        self.seq_num = seq_num
        self.ack_num = ack_num
        self.flags = flags
        self.checksum = checksum
        self.data = data

    def calculate_checksum(self):
        # Calculate the checksum based on packet fields (excluding the checksum itself)
        packet_str = f"{self.source_port}{self.dest_port}{self.seq_num}{self.ack_num}{self.flags}{self.data}"
        # Implement your checksum algorithm here
        # Example: Use CRC32
        checksum = ns.Crc32.Calculate(packet_str.encode())
        return checksum

# Initialize simulation
simulation = ns.Simulator

# Create nodes for client and server
client_node = ns.Node()
server_node = ns.Node()

# Install internet stack on the nodes
internet_stack_helper = ns.InternetStackHelper()
internet_stack_helper.Install(client_node)
internet_stack_helper.Install(server_node)

# Create network devices and install them
point_to_point_helper = ns.PointToPointHelper()
point_to_point_helper.SetDeviceAttribute("DataRate", ns.StringValue("100Mbps"))
point_to_point_helper.SetChannelAttribute("Delay", ns.StringValue("2ms"))

# Connect the client and server nodes with a point-to-point link
device = point_to_point_helper.Install(client_node, server_node)

# Assign IP addresses to the network devices
address_helper = ns.Ipv4AddressHelper()
address_helper.SetBase(ns.Ipv4Address("10.0.0.0"), ns.Ipv4Mask("255.255.255.0"))
interface = address_helper.Assign(device)

# Create UDP applications for client and server
client_app = ns.OnOffHelper("ns3::UdpSocketFactory", ns.Address(ns.InetSocketAddress(interface.GetAddress(1), 9)))
client_app.SetAttribute("PacketSize", ns.UintegerValue(1024))
client_app.SetAttribute("DataRate", ns.DataRateValue(ns.DataRate("5Mbps")))
client_app.SetAttribute("StartTime", ns.TimeValue(ns.Seconds(1.0)))
client_app.Install(client_node)

server_app = ns.PacketSinkHelper("ns3::UdpSocketFactory", ns.Address(ns.InetSocketAddress(ns.Ipv4Address.GetAny(), 9)))
server_app.Install(server_node)

# Enable logging
ns.LogComponentEnable("UdpHandshake", ns.LOG_LEVEL_INFO)

# Run the simulation
```

```
simulation.Run()

# Perform three-way handshake
seq_num = 1001
ack_num = 0
flags = "SYN"

# Generate SYN packet
syn_packet = UDPPacket(8888, 8888, seq_num, ack_num, flags, 0, "")
syn_packet.checksum = syn_packet.calculate_checksum()

# Send SYN packet from client to server
client_socket.Start(ns.Seconds(1.0))
client_socket.Stop(ns.Seconds(5.0))
client_socket = client_app.Get(0)
client_socket.SendTo(syn_packet, server_node.GetDevice(1).GetAddress(), 8888)
NS_LOG_INFO("Sent SYN to server")

# Receive SYN packet at server
server_socket = server_app.Get(0)
received_packet = server_socket.Recv()

# Verify SYN packet checksum
if received_packet.checksum == received_packet.packet.calculate_checksum():
    NS_LOG_INFO("Received SYN packet. Handshake in progress.")
    seq_num = received_packet.packet.ack_num
    ack_num = received_packet.packet.seq_num + 1
    flags = "SYN-ACK"

    # Generate SYN-ACK packet
    syn_ack_packet = UDPPacket(8888, 8888, seq_num, ack_num, flags, 0, "")
    syn_ack_packet.checksum = syn_ack_packet.calculate_checksum()

    # Send SYN-ACK packet from server to client
    server_socket.Send(syn_ack_packet, received_packet.remoteAddress)

    # Receive SYN-ACK packet at client
    received_packet = client_socket.GetPacket()

    # Verify SYN-ACK packet checksum
    if received_packet.packet.checksum == received_packet.packet.calculate_checksum():
        NS_LOG_INFO("Received SYN-ACK packet. Handshake in progress.")
        seq_num = received_packet.packet.ack_num
        ack_num = received_packet.packet.seq_num + 1
        flags = "ACK"

        # Generate ACK packet
        ack_packet = UDPPacket(8888, 8888, seq_num, ack_num, flags, 0, "")
        ack_packet.checksum = ack_packet.calculate_checksum()

        # Send ACK packet from client to server
        client_socket.Start(ns.Seconds(2.0))
        client_socket.SendTo(ack_packet, server_node.GetDevice(1).GetAddress(), 8888)
          NS_LOG_INFO("Sent ACK to server")
        NS_LOG_INFO("Handshake complete. Connection established.")
    else:
        NS_LOG_INFO("Invalid SYN-ACK packet. Handshake failed.")

# Cleanup and shutdown the simulation
simulation.Stop()
simulation.Destroy()
```

While it builds with no errors, running this in ns-3 gives us an unknown error with a stack dump. ns-3 bindings with Python are correctly configured, with sample files tested. I could not find any way to resolve this error since it is not clear to me what changes to make.

Output

Description

UDP Packet definition

- `UDPPacket` defines a class representing a UDP packet, with the attributes:
  - source port
  - destination port
  - sequence number - implemented to differentiate between packets used for handshake.
  - acknowledgment number
  - flags
  - data.
- It also includes a method to calculate the checksum of the packet, for handshake verification
- These specialized packets can be used to verify identity and connection between client and server, after which default datagrams can be sent as per the UDP specification.

Client configuration

- `client_node = ns.Node()` creates a client node.
- `internet_stack_helper = ns.InternetStackHelper()`: Creates an InternetStackHelper object.
- `internet_stack_helper.Install(client_node)`: Installs the internet stack on the client node.
- `client_app = ns.OnOffHelper("ns3::UdpSocketFactory", ns.Address(ns.InetSocketAddress(interface.GetAddress(1), 9)))`: Creates a client application using OnOffHelper(), which generates UDP traffic.
- `client_app.SetAttribute("PacketSize", ns.UintegerValue(1024))`: Sets the packet size attribute.
- `client_app.SetAttribute("DataRate", ns.DataRateValue(ns.DataRate("5Mbps")))`: Sets the data rate attribute.
- `client_app.SetAttribute("StartTime", ns.TimeValue(ns.Seconds(1.0)))`: Sets the start time attribute.
- `client_app.Install(client_node)`: Installs the client application on the client node.

Server configuration

- `server_node = ns.Node()`: Creates a server node.

- `internet_stack_helper.Install(server_node)`: Installs the internet stack, an object created from the class InternetStackHelper, on the server node.
- `server_app = ns.PacketSinkHelper("ns3::UdpSocketFactory", ns.Address(ns.InetSocketAddress(ns.Ipv4Address.GetAny(), 9)))`: Creates a server application using PacketSinkHelper, which receives and processes UDP packets.
- `server_app.Install(server_node)`: Installs the server application on the server node.

Network configuration

- P2P Link
  - `point_to_point_helper = ns.PointToPointHelper()`: Creates a PointToPointHelper object.
  - `point_to_point_helper.SetDeviceAttribute("DataRate", ns.StringValue("100Mbps"))`: Sets the data rate attribute for the devices.
  - `point_to_point_helper.SetChannelAttribute("Delay", ns.StringValue("2ms"))`: Sets the channel delay attribute.
  - `device = point_to_point_helper.Install(client_node, server_node)`: Creates a point-to-point link between the client and server nodes.
- IP Address Assignment
  - `address_helper = ns.Ipv4AddressHelper()`: Creates an Ipv4AddressHelper object.
  - `address_helper.SetBase(ns.Ipv4Address("10.0.0.0"), ns.Ipv4Mask("255.255.255.0"))`: Sets the IP address range and subnet mask.
  - `interface = address_helper.Assign(device)`: Assigns IP addresses to the network devices.

Mechanism of Handshake

- The code generates a SYN packet from the client to the server and sends it.
- The server receives the SYN packet, verifies its checksum, and if valid, generates a SYN-ACK packet and sends it back to the client.
- The client receives the SYN-ACK packet, verifies its checksum, and if valid, completes the handshake by sending an ACK packet to the server.
- The code logs whether the handshake is successful or not.

Implementation attempt on Python with socket programming

```
import socket # low level socket programming in Python
import struct # pack and unpack datagrams

# Function to calculate checksum using Internet Checksum algorithm
def calculate_checksum(data):
    checksum = 0
    for i in range(0, len(data), 2):
        if i + 1 < len(data):
            checksum += (data[i] << 8) + data[i + 1]
        elif i < len(data):
            checksum += data[i]
    checksum = (checksum >> 16) + (checksum & 0xffff)
    checksum += (checksum >> 16)
    return ~checksum & 0xffff

def send_packet(socket, packet):
    socket.sendto(packet, ("127.0.0.1", 8001))

def receive_packet(socket):
    data, addr = socket.recvfrom(1024)
    return data, addr

def server_handshake(socket):
```

```
    # Receive SYN from client
    data, addr = receive_packet(socket)
    syn_packet = struct.unpack("!50si", data)
    syn, seq_num = syn_packet

    # Verify checksum
    checksum = calculate_checksum(data)
    if checksum != 0:
        print("Invalid packet received. Discarding.")
        return

    print("Received SYN from client")

    # Send SYN-ACK to client
    syn_ack_packet = struct.pack("!50si", b"SYN-ACK", seq_num)
    send_packet(socket, syn_ack_packet)
    print("Sent SYN-ACK to client")

    # Receive ACK from client
    data, addr = receive_packet(socket)
    ack_packet = struct.unpack("!50si", data)
    ack, seq_num = ack_packet

    # Verify checksum
    checksum = calculate_checksum(data)
    if checksum != 0:
        print("Invalid packet received. Discarding.")
        return

    print("Received ACK from client")

    # Server connection established
    print("Server connection established with client")

def client_handshake(socket):
    seq_num = 1

    # Send SYN to server
    syn_packet = struct.pack("!50si", b"SYN", seq_num)
    send_packet(socket, syn_packet)
    print("Sent SYN to server")

    # Receive SYN-ACK from server
    data, addr = receive_packet(socket)
    syn_ack_packet = struct.unpack("!50si", data)
    syn_ack, seq_num = syn_ack_packet

    # Verify checksum
    checksum = calculate_checksum(data)
    if checksum != 0:
        print("Invalid packet received. Discarding.")
        return

    print("Received SYN-ACK from server")

    # Send ACK to server
    ack_packet = struct.pack("!50si", b"ACK", seq_num)
    send_packet(socket, ack_packet)
    print("Sent ACK to server")

    # Client connection established
    print("Client connection established with server")

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind(("127.0.0.1", 8000))

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.bind(("127.0.0.1", 8001))

    server_handshake(server_socket)
    client_handshake(client_socket)

    server_socket.close()
    client_socket.close()
```

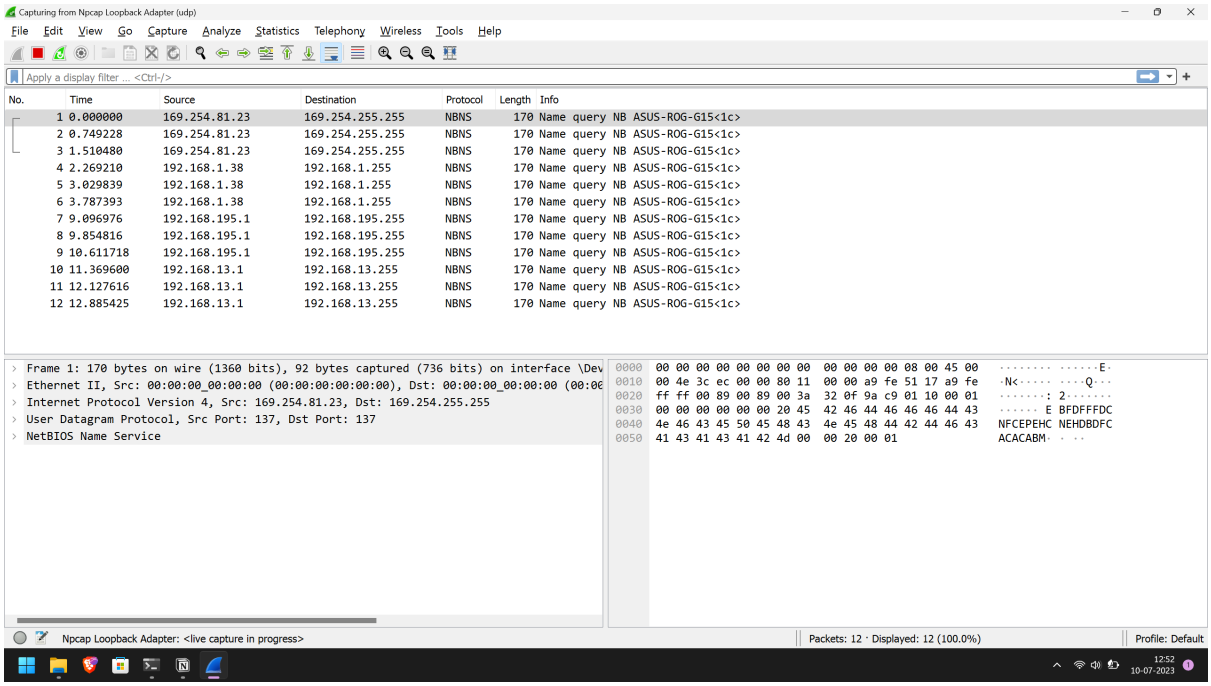
```
if __name__ == "__main__":
    main()
```

Description

- The server and client sockets are created and bound to their respective addresses.
- The server initiates the handshake by waiting for a SYN packet from the client.
- Upon receiving the SYN packet, the server verifies its checksum, sends a SYN-ACK packet back to the client, and waits for an ACK packet.
- The client initiates the handshake by sending a SYN packet to the server.
- Upon receiving the SYN packet, the server verifies its checksum, sends a SYN-ACK packet back to the client, and waits for an ACK packet.
- The client receives the SYN-ACK packet, verifies its checksum, sends an ACK packet to the server, and completes the handshake.
- If the handshake is successful, both the client and server print the connection established message.
- The sockets are closed.
- Verification between packets takes place with `calculate_checksum()` which uses Internet Checksum algorithm for its purpose.

Output

Unfortunately, no UDP packets originating from this script show up in Wireshark, so there is no way to verify whether the handshake is taking place. While the handshake procedure is upgraded from just echoes to verification using checksum, the lack of output is a regression from v1.

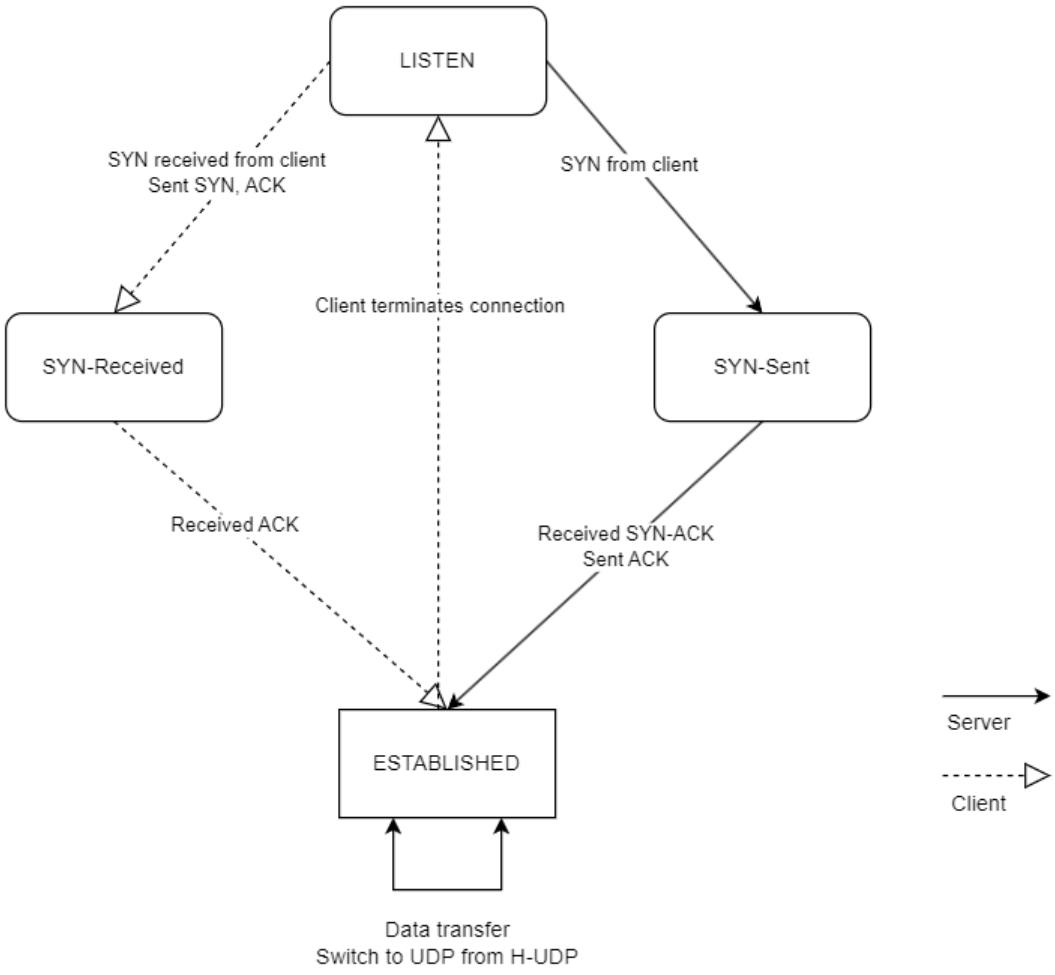


However, running two instances of the same script generates this error

```
PowerShell 7.3.5
PS C:\Users\Aneemesh> python3.11.exe C:\Stuff\udp-handshake.py
Traceback (most recent call last):
  File "C:\Stuff\udp-handshake.py", line 101, in <module>
    main()
  File "C:\Stuff\udp-handshake.py", line 89, in main
    server_socket.bind(("127.0.0.1", 8000))
OSError: [WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted
PS C:\Users\Aneemesh>
```

which suggests that the socket binding has taken place and that the client and server are indeed running on port 8000 and 8001 respectively. There seems to be an unidentifiable problem with the `server_handshake` and `client_handshake` functions.

UDP Handshake State Machine



Server states

- LISTEN
- SYN-Recieved
- ESTABLISHED

Client states

- SYN-Sent
- ESTABLISHED

References

- https://groups.google.com/g/ns-3-users
- https://www.geeksforgeeks.org/
- https://www.nsnam.org/docs/
- https://stackoverflow.com/
- https://docs.python.org/3/library/struct.html