import threading

from queue import Queue

from dash import Dash, dcc, html, dash\_table

from dash.dependencies import Output, Input, State

from scapy.all import sniff

import pandas as pd

# Queue for sharing data between threads

packet\_queue = Queue()

packet\_history = []  # Store captured packets

# Define a function to handle packet sniffing (using Scapy)

def packet\_callback(packet):

    if packet.haslayer('IP'):

        src\_ip = packet['IP'].src

        dest\_ip = packet['IP'].dst

        protocol = packet['IP'].proto

        ttl = packet['IP'].ttl

        src\_mac = packet.src

        dest\_mac = packet.dst

        # Store packet data in the queue and history

        packet\_data = {

            'Source MAC': src\_mac,

            'Destination MAC': dest\_mac,

            'Source IP': src\_ip,

            'Destination IP': dest\_ip,

            'Protocol': protocol,

            'TTL': ttl

        }

        packet\_queue.put(packet\_data)

        packet\_history.append(packet\_data)

# Function to start sniffing packets

def start\_sniffing():

    print("Packet sniffer started using Scapy.")

    sniff(prn=packet\_callback, store=0, filter="ip", iface="Wi-Fi", timeout=60)

# Start sniffing in a separate thread

threading.Thread(target=start\_sniffing, daemon=True).start()

# Initialize Dash app

app = Dash(\_\_name\_\_)

# App layout

app.layout = html.Div([

    html.H1("Network Sniffer Dashboard"),

    # Filter Inputs

    html.Div([

        dcc.Input(id='filter-src-ip', type='text', placeholder='Source IP'),

        dcc.Input(id='filter-dest-ip', type='text', placeholder='Destination IP'),

        dcc.Input(id='filter-protocol', type='text', placeholder='Protocol (e.g., 6 for TCP, 17 for UDP)'),

        html.Button('Apply Filters', id='apply-filters', n\_clicks=0),

    ], style={'marginBottom': '20px'}),

    # Save to CSV Button

    html.Button('Save to CSV', id='save-csv', n\_clicks=0, style={'marginBottom': '20px'}),

    # Data Table

    dash\_table.DataTable(

        id='packet-table',

        columns=[

            {"name": "Source MAC", "id": "Source MAC"},

            {"name": "Destination MAC", "id": "Destination MAC"},

            {"name": "Source IP", "id": "Source IP"},

            {"name": "Destination IP", "id": "Destination IP"},

            {"name": "Protocol", "id": "Protocol"},

            {"name": "TTL", "id": "TTL"}

        ],

        style\_table={'overflowX': 'auto'},

        style\_cell={'textAlign': 'left'},

    ),

    # Interval for auto-refresh

    dcc.Interval(id='interval', interval=1000, n\_intervals=0)

])

# Callback to update the table with filtered data

@app.callback(

    Output('packet-table', 'data'),

    [Input('interval', 'n\_intervals'),

     Input('apply-filters', 'n\_clicks')],

    [State('filter-src-ip', 'value'),

     State('filter-dest-ip', 'value'),

     State('filter-protocol', 'value')]

)

def update\_table(n, n\_clicks, src\_ip\_filter, dest\_ip\_filter, protocol\_filter):

    packets = []

    while not packet\_queue.empty():

        packets.append(packet\_queue.get())

    # Apply filters

    filtered\_packets = packet\_history

    if src\_ip\_filter:

        filtered\_packets = [p for p in filtered\_packets if p['Source IP'] == src\_ip\_filter]

    if dest\_ip\_filter:

        filtered\_packets = [p for p in filtered\_packets if p['Destination IP'] == dest\_ip\_filter]

    if protocol\_filter:

        try:

            protocol\_filter = int(protocol\_filter)

            filtered\_packets = [p for p in filtered\_packets if p['Protocol'] == protocol\_filter]

        except ValueError:

            pass  # Ignore invalid protocol filter

    return filtered\_packets

# Callback to save data to CSV

@app.callback(

    Output('save-csv', 'children'),

    [Input('save-csv', 'n\_clicks')]

)

def save\_to\_csv(n\_clicks):

    if n\_clicks > 0:

        df = pd.DataFrame(packet\_history)

        df.to\_csv('captured\_packets.csv', index=False)

        return "Saved!"

    return "Save to CSV"

if \_\_name\_\_ == '\_\_main\_\_':

    print("Starting Dash app...")

    app.run\_server(debug=True)