Flowsight-Smart Testing Insights/Validations

A

Major Project Report

Submitted to Manipal University, Jaipur

Towards the partial fulfillment for the Award of the Degree of

MASTER OF COMPUTER APPLICATIONS

2023-2025

By

Muskan Satwani

23FS20MCA00073

Under the guidance of

Dr. Devershi Pallavi Bhatt

Department of Computer Applications

School of AIML, IoT&IS, CCE, DS and Computer Applications

Faculty of Science, Technology and Architecture

Manipal University Jaipur

Jaipur, Rajasthan

May 2025

Department of Computer Applications
School of AIML, IoT&IS, CCE, DS and Computer Applications
Faculty of Science, Technology and Architecture
Manipal University Jaipur, Dehmi Kalan, Jaipur, Rajasthan, India- 303007

STUDENT DECLARATION

I hereby declare that this project ***Flowsight – Smart Insights/Validations*** *is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the University or other Institute, except where due acknowledgements has been made in the text.*

Place: Jaipur
Date: 14.05.2025

Muskan Satwani
23FS20MCA00073
MCA

Department of Computer Applications
School of AIML, IoT&IS, CCE, DS and Computer Applications
Faculty of Science, Technology and Architecture
Manipal University Jaipur, Dehmi Kalan, Jaipur, Rajasthan, India- 303007

Date:14.05.2025

CERTIFICATE FROM SUPERVISOR

*This is to certify that the work entitled "**Flowsight – Smart Insights/Validations**" submitted by **Muskan Satwani**(23FS20MCA00073) to **Manipal University Jaipur** for the award of the degree of **Master of Computer Applications** is a bonafide record of the work carried out by her under my supervision and guidance from 16th January 2025 to 24th May 2025.*

Dr. Devershi Pallavi Bhatt
Professor,
Department of Computer Applications
Manipal University Jaipur

Project Completion Certificate

ACKNOWLEDGEMENT

I would like to extend my heartfelt gratitude to my internal guide, **Dr. Devershi Pallavi Bhatt** for her invaluable guidance and support throughout this project. Her insights and encouragement have been instrumental in shaping the direction and outcome of this work.

I am profoundly grateful to my Nokia manager, **Mr. Saurabh Joshi**, for his unwavering support and belief in my abilities. Special thanks **to Mr. Ramji Yadav**, the software architect of our team, whose technical expertise and strategic vision were crucial to the success of this project.

A sincere thank you to my teammate, **Mr. Naman Sharma**, for his continuous help and collaboration. I am also deeply appreciative of the guidance provided by our team's Principal Solution Architect, **Mr. Vivek Talwar**, whose wisdom and experience greatly benefited this endeavor.

I would like to express my heartfelt appreciation to the **Department of Computer Applications** at my college for providing a robust educational foundation and fostering an environment conducive to research and innovation.

I am sincerely grateful to the HR team and the support staff at Nokia for their assistance and encouragement, which have been vital to the completion of this project. Additionally, I extend my gratitude to the support staff at my college for their help and support.

I am **forever indebted to my parents** for their unwavering support, love, and encouragement throughout my academic journey. Their belief in me has been a constant source of strength.

Finally, I want to thank everyone who has supported me in this project and throughout my MCA journey. Your encouragement and assistance have been truly appreciated, and I am deeply grateful for all the support I have received.

Any omission from this brief acknowledgement does not mean lack of gratitude.

ABSTRACT

FlowSight is an automated PCAP analysis software designed to streamline and standardize multi-protocol call-flow extraction for modern telecommunications networks. Built on top of the Pyshark library, FlowSight ingests packet captures (PCAP files) and processes signaling protocols—namely SIP, Diameter, DNS, GTPv2, and S1AP—extracting key information into a structured Excel report. The report includes:

Protocol Data: Detailed per-packet rows combining protocol names, message types, code mappings, timestamps, and relevant identifiers (e.g., IMSI, TEID).

- **Summary**: SIP call-level success criteria, showing for each session whether the INVITE→200 OK handshake succeeded, whether the REGISTER transaction completed, and whether session teardown (BYE→200 OK) was clean.
- **Stats**: Bar charts visualizing the distribution of SIP response-code categories (1xx, 2xx, 3xx, 4xx, 5xx) and other protocol outcomes over time.
- **Optional Flow Diagrams**: Graphical SIP call-flow sequences rendered from the same Excel data.

FlowSight's modular architecture separates PCAP loading, protocol extractors, data aggregation, and reporting into clear components, enabling easy extension to new protocols or output formats. Configuration-driven mapping dictionaries (for command codes, result codes, flags) and a Flask-based front end allow both command-line and web-based usage. The tool automates cleanup of temporary files and enforces authentication for secure operation

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 History of Nokia

Nokia has a long and rich history, dating back to the late 19th century. Here's a brief overview:

- **1865:** Fredrik Idestam establishes a wood pulp mill in Nokia, Finland, marking the beginning of the company.
- **1871:** The company is renamed Nokia Ab.
- **1912:** Nokia merges with Finnish Rubber Works and Finnish Cable Works, forming Nokia Corporation.
- **1960s:** Nokia expands into electronics, producing televisions and other consumer goods.
- **1980s:** Nokia enters the mobile phone market with the Mobira Senator, one of the first commercially available mobile phones.
- **1990s:** Nokia becomes a dominant player in the mobile phone market, with its iconic "brick" phones and later, the popular Nokia 3310.
- **2000s:** Nokia faces increasing competition from smartphones, particularly from Apple and Samsung.
- **2011:** Nokia sells its mobile phone business to Microsoft.
- **2014:** Nokia acquires Alcatel-Lucent, a telecommunications equipment company.
- **Present:** Nokia focuses on network infrastructure, including 5G technology, and digital transformation solutions.

Nokia's history is marked by innovation and adaptation, from its early days as a paper mill to its dominance in the mobile phone market and its current focus on network infrastructure.

1.2 Transition to Networking

Nokia has been a major player in the telecommunications industry for decades, initially known for its mobile phones. However, the company has successfully transitioned to become a leading provider of network infrastructure and solutions.

Here are some key points about Nokia's transition to networks:

- **Shifting Focus:** Nokia recognized the changing landscape of the mobile industry and the growing demand for advanced network technologies. They strategically shifted their focus from consumer electronics to network infrastructure, investing heavily in research and development.
- **Acquisitions and Partnerships:** Nokia acquired several companies, including Alcatel-Lucent in 2016, to strengthen its portfolio of network solutions and expand its global reach. They also formed strategic partnerships with other technology companies to develop innovative solutions.
- **5G Leadership:** Nokia has been a pioneer in 5G technology, playing a key role in the development and deployment of 5G networks worldwide. They offer a comprehensive range of 5G solutions, including core networks, radio access networks, and transport networks.

- **Focus on Innovation:** Nokia continues to invest in research and development to stay ahead of the curve in network technologies. They are actively involved in developing next-generation technologies like 6G and edge computing.
- **Global Presence:** Nokia has a strong global presence, with operations in over 100 countries. This allows them to provide network solutions to a wide range of customers, including telecommunications operators, enterprises, and governments.

Nokia's transition to networks has been a strategic success, positioning them as a leading provider of network infrastructure and solutions for the digital age. They are well-equipped to meet the growing demand for advanced network technologies and continue to play a vital role in shaping the future of connectivity.

1.3 Nokia CNS

During my internship, I had the privilege of working in the **Cloud and Network Services at Nokia**. It is a suite of cloud-based services that help businesses manage and optimize their network infrastructure. These services are designed to help businesses:

- **Improve network performance and reliability:** By providing real-time insights into network performance and identifying potential issues before they impact users.
- **Reduce operational costs:** By automating tasks and simplifying network management.
- **Increase agility and flexibility:** By providing a cloud-based platform that allows businesses to quickly deploy and scale their network infrastructure.

Nokia CNS offers a range of services, including:

- **Network Performance Management (NPM):** Provides real-time visibility into network performance and helps identify and resolve issues.
- **Network Configuration Management (NCM):** Automates network configuration tasks and helps ensure consistency across the network.
- **Network Security Management (NSM):** Provides tools for managing and securing the network against threats.
- **Network Analytics:** Provides insights into network usage and performance trends.

Nokia CNS is a comprehensive solution that can help businesses of all sizes improve their network operations.

1.4 Integration and Automation Team

While doing my internship I was part of Integration and Automation team under CNS R&D Software Excellence at Nokia, Noida. Our team's main goal was to develop automation scripts to streamline the testing of software defects, ensuring the robustness and reliability of Nokia's cloud and networking services. Essentially, our job was to ensure that the product being developed is not only functional but also meets the highest standards of quality.

1.4.1 Team Structure and Roles

- **Manager:** Our manager **Mr. Saurabh Joshi** was the driving force behind the team, providing strategic direction and ensuring that our projects aligned with the overall goals of the department. He facilitated communication between upper management and our team, making sure we had the resources we needed to succeed.

- **Software Architect:** The software architect in our case **Mr. Ramji Yadav** was responsible for designing and overseeing the architecture of our automation framework. He ensured that our solutions were scalable, efficient, and integrated seamlessly with existing systems. His expertise in software design was crucial for tackling complex technical challenges.

- **Solution Architect: Mr. Vivek Talwar** the team's solution architect played a pivotal role in ensuring our agile practices. He organized and facilitated our sprint meetings, ensured that we adhered to agile principles, and helped remove any obstacles that impeded our progress. He kept the team focused and productive.

- **Automation Engineer:** The automation engineers, including myself, were tasked with writing and maintaining the automation scripts. We worked closely with the software architect to implement automated tests that identified defects and verified fixes. Our work was critical for improving the efficiency and accuracy of the testing process.

- **Integration Engineers:** The integration engineers focused on ensuring that different components of the Network software worked together seamlessly. They tested the integration points and resolved any issues that arose from combining various modules and features.

1.4.2 Agile and Sprint Practices

Our team followed agile practices, a project management methodology that emphasizes iterative development, collaboration, and flexibility.

- **Agile Principles:** Agile is a type of software development practice in which adaptability and continuous improvement are of paramount importance. Customer feedback and making iterative changes to our processes and products are highly prioritized. This approach allows the teams to respond quickly to changes and deliver high-quality software.
- **Sprint Planning:** By following Agile principles we ensured that our work was organized into sprints, which are short, time-boxed periods (usually two weeks) where specific tasks and goals are set. Before the start of any sprint, we had a sprint planning meeting. During these meetings, we used to decide which tasks would be tackled in the upcoming sprint. Each task was assigned a point value based on its complexity and effort required.

- **Daily Discussions:** On every working day, we held a brief meeting in the morning where each team member shared what they worked on the previous day, what they planned to work on that day, and any blockers they were facing. These meetings kept everyone on the same page and allowed us to address issues promptly.

1.5 Problem Statement

Existing open-source solutions focus on single protocols or raw decoding. There is a gap for a unified software framework that can:

1. **Handle Multiple Protocols**: Simultaneously parse SIP, Diameter, DNS, GTPv2, and S1AP from the same PCAP.
2. **Map Codes to Human-Readable Meanings**: Translate numeric command-codes, result-codes, cause values, and flags into descriptive labels.
3. **Group Packets by Session Identifiers**: Consistently derive an Identifier per session (preferring IMSI, falling back to TEID or TMSI) for session-level aggregation.
4. **Produce Structured Excel Reports**: Generate per-packet detail, session summaries, and statistical charts in a single workbook.
5. **Support Both CLI and Web Interfaces**: Cater to power users and non-technical stakeholders via a Flask front end that accepts PCAP uploads and returns ready-to-read reports.

FlowSight software addresses each of these requirements, offering a cohesive solution for automated, reproducible call-flow analysis.

2. MOTIVATION

Telecommunications networks rely on a variety of signaling protocols to set up, manage, and tear down sessions. Network engineers and researchers commonly use Wireshark or Tshark to inspect PCAP files, manually filtering and decoding messages such as SIP INVITEs, Diameter authentication requests, or GTPv2 session management procedures. This manual process is:

- **Time-Consuming**: Sorting through large packet captures to identify specific AVPs, message types, and error codes can take hours per capture.
- **Error-Prone**: Manual extraction risks overlooking edge cases—missing a single "Auth-Answer" message or misreading a result code can invalidate a troubleshooting session.
- **Non-Reproducible**: Different analysts may apply inconsistent filters or formatting, making cross-comparison of results difficult.

Automating this workflow with dedicated software not only accelerates analysis but also ensures consistency across captures, enabling large-scale studies, regression testing, and simplified reporting for academic or operational teams.

2.1 Empowering Network engineers:

Network engineers are responsible for translating high-level troubleshooting or test objectives into precise packet-level analyses. Traditionally, they must manually craft tshark or Wireshark filters, search through hundreds of thousands of packets, and stitch together disparate protocol messages (SIP INVITEs, GTPv2 Create Session Answers, Diameter Authentication Answers, etc.). This manual process is both error-prone and slow. FlowSight empowers these engineers by automatically generating per-session call-flow data in a unified Excel format—with one row per packet, fully decoded "Protocol Message" strings, and mapped codes (e.g., result codes, cause values, AVP meanings). By taking ambiguous textual troubleshooting goals (e.g., "verify that the MME successfully updates the location and triggers the SGW Create Session") and producing structured call-flow steps, FlowSight reduces reliance on free-form filters and accelerates root-cause analysis.

2.2 Reduction in Manual Packet Inspection:

In conventional workflows, engineers spend significant time writing and refining capture filters, extracting fields into CSVs, and hand-assembling session summaries—often repeating the same steps across multiple PCAPs. FlowSight automates the full pipeline:

1. **PCAP Ingestion**: Automatically detects and loads all relevant PCAPs in a directory.
2. **Multi-Protocol Extraction**: Simultaneously parses SIP, Diameter, DNS, GTPv2, S1AP, PFCP, NGAP, and HTTP/2-SBI messages without manual filter changes.
3. **Mapping & Decoding**: Translates every numeric code into human-readable descriptions via mappings.py.
4. **Excel Reporting**: Emits a ready-to-use workbook with conditional formatting, summary flags, and charts.

By eliminating manual scripting for each protocol and each new PCAP, FlowSight frees engineers to focus on interpreting results and designing corrective actions rather than on repetitive extraction tasks.

2.3 On-Demand, Incident-Driven Analysis: Network incidents and test failures often arise unexpectedly, and time is of the essence. FlowSight supports on-demand analysis directly from user-provided PCAPs—even in the midst of a live outage. With the Flask web interface, a network engineer or SOC analyst can upload a failed call capture, select the relevant mapping and configuration files, and receive a complete end-to-end call-flow report within minutes. This reactive capability transforms troubleshooting from a manual, ad-hoc exercise into a streamlined, consistent process:

- **Rapid Turnaround**: Immediate Excel reports identify where the SIP INVITE failed, which GTPv2 cause code was returned, or whether the PFCP Heartbeat timed out.
- **Customization**: Engineers can dynamically adjust mappings or enable 5G protocol parsers (NGAP, PFCP) without touching the core code.

- **Proactive Follow-Up**: Summary flags (e.g., “GTPv2 Create Session Answer missing,” “SIP 2xx not received”) help decide next steps—packet captures can be re-analyzed with different display filters or forwarded to vendors for deeper inspection.

2.4 Enhancing Coverage, Fidelity, and Accuracy:

FlowSight’s end-to-end automation not only speeds analysis but measurably improves the completeness and correctness of each investigation:

- **Comprehensive Protocol Coverage**: By supporting both legacy EPC protocols (Diameter, GTPv2-C, S1AP) and modern 5G interfaces (PFCP, NGAP, HTTP/2 SBIs), FlowSight ensures no signaling path is overlooked.
- **Edge-Case Detection**: Conditional formatting and summary checks highlight uncommon failure modes—missing AVPs, unexpected cause codes, or mismatched identifier groupings—drawing attention to corner-case behaviors.
- **Consistency Across Runs**: Every PCAP, whether from live networks or test benches, is processed through the same logic, guaranteeing reproducible results and simplifying regression analyses.
- **Accuracy via Mappings**: Centralized mapping dictionaries eliminate misinterpretation of numeric codes, ensuring that “2001” is always shown as “DIAMETER_SUCCESS” and “16” as “REQUEST_ACCEPTED” for GTPv2 causes.

By systematically capturing every relevant field, mapping it to a human-readable form, and aggregating sessions into a single workbook, FlowSight delivers higher test coverage and greater confidence in both manual and automated troubleshooting workflows.

3. OBJECTIVES

Our objectives focus on eliminating the manual toil of packet-level call-flow debugging and harnessing live network traces (PCAPs) to drive proactive, data-driven troubleshooting.

3.1 Automated Call-Flow Extraction from PCAPs: Traditionally, network automation engineers spend hours crafting and refining Wireshark/Tshark filters, exporting CSVs, and stitching together multi-protocol call flows by hand. Our first objective is to deliver a software that ingests raw PCAP files and automatically generates end-to-end call-flow steps for all key signaling protocols (SIP, Diameter, GTPv2-C, S1AP, PFCP, NGAP, HTTP/2 SBIs). By parsing each packet in turn, mapping every numeric code to a human-readable label (via mappings.py), and grouping packets by session identifier (IMSI → TEID → PFCP Session ID), FlowSight eliminates manual filter-writing and error-prone field extraction. The result is a structured Excel workbook—with a "Protocol Data" sheet showing one row per packet and a unified "Protocol Message" column—ready for consumption by engineers or downstream automation scripts. This automated pipeline reduces hands-on effort, ensures consistency across captures, and delivers repeatable, auditable call-flow reports in minutes instead of hours.

3.2 Proactive, On-Demand Network Analysis from Live Traces: Waiting for a test bench or manual trigger to capture signaling failures can delay root-cause diagnosis. Our second objective is to turn any network trace—whether from a live outage, regression suite, or field test—into immediate, actionable insights without human intervention. FlowSight's Flask web front-end lets engineers upload PCAPs on demand, select protocol-mapping files, and receive back an Excel report that not only highlights where the SIP INVITE failed or which GTPv2 cause code was returned, but also flags missing procedures (e.g., "PFCP Heartbeat Response not seen") and charts protocol-level response distributions over time. By extracting real-world signaling behavior directly from packet captures—rather than relying on static test scenarios—FlowSight empowers teams to dynamically generate targeted follow-up captures, automate downstream validation scripts against the Excel output, and continuously improve network reliability through data-driven feedback loops.

4. THEORY

Before delving into the methodology used by FlowSight software, it is essential to understand the theoretical underpinnings of packet-level protocol analysis and call-flow visualization. This section covers the key concepts that drive our design:

4.1 Packet Capture and PCAP Format :

- A PCAP file is a binary sequence of time-stamped packet records. Each record contains the raw bytes of the frame plus metadata (timestamp, captured length).
- Tools like Tshark/Wireshark and Pyshark parse this format to expose individual protocol layers.

Layered Dissection

- Pyshark uses the Wireshark dissector library to break each packet into OSI layers (Ethernet → IP → UDP/TCP → Application).

- Display filters (e.g., display_filter="sip") let us select only packets containing a given protocol layer.

4.2 Protocol Layering and Field Extraction:

- **OSI and TCP/IP Stacks**
- FlowSight targets both transport (UDP/TCP/SCTP) and application layers: SIP over UDP/TCP, Diameter over SCTP/TCP, GTP-U/C over UDP, DNS over UDP/TCP, PFCP over UDP, NGAP over SCTP, HTTP/2 SBIs over TLS/TCP.
- **Field Mapping**
- Each layer exposes named fields (e.g., packet.sip.Call_ID, packet.diameter.result_code).
- Mapping dictionaries (mappings.py) translate numeric values (AVP codes, result codes, message types) into human-readable labels.

4.3 IMS Registration and Session Flow:

- **IMS Architectural Elements**
  - **P-CSCF** (Proxy), **I-CSCF** (Interrogating), **S-CSCF** (Serving), **HSS** (Home Subscriber Server).
  - Registration uses SIP REGISTER via P-CSCF → I-CSCF → S-CSCF, which queries HSS over Diameter S6a/Cx.
- **Combined SIP & Diameter Flow**
  - **SIP REGISTER** triggers a Diameter Authentication/Location-Update sequence:
    1. S-CSCF → HSS (Diameter ULR)
    2. HSS → S-CSCF (Diameter ULA)
    3. S-CSCF → UE (SIP 200 OK)
- **Session Establishment**
  - **SIP INVITE** passes through P-CSCF/I-CSCF/S-CSCF, and S-CSCF uses Diameter Cx to fetch user profile, then responds to INVITE.

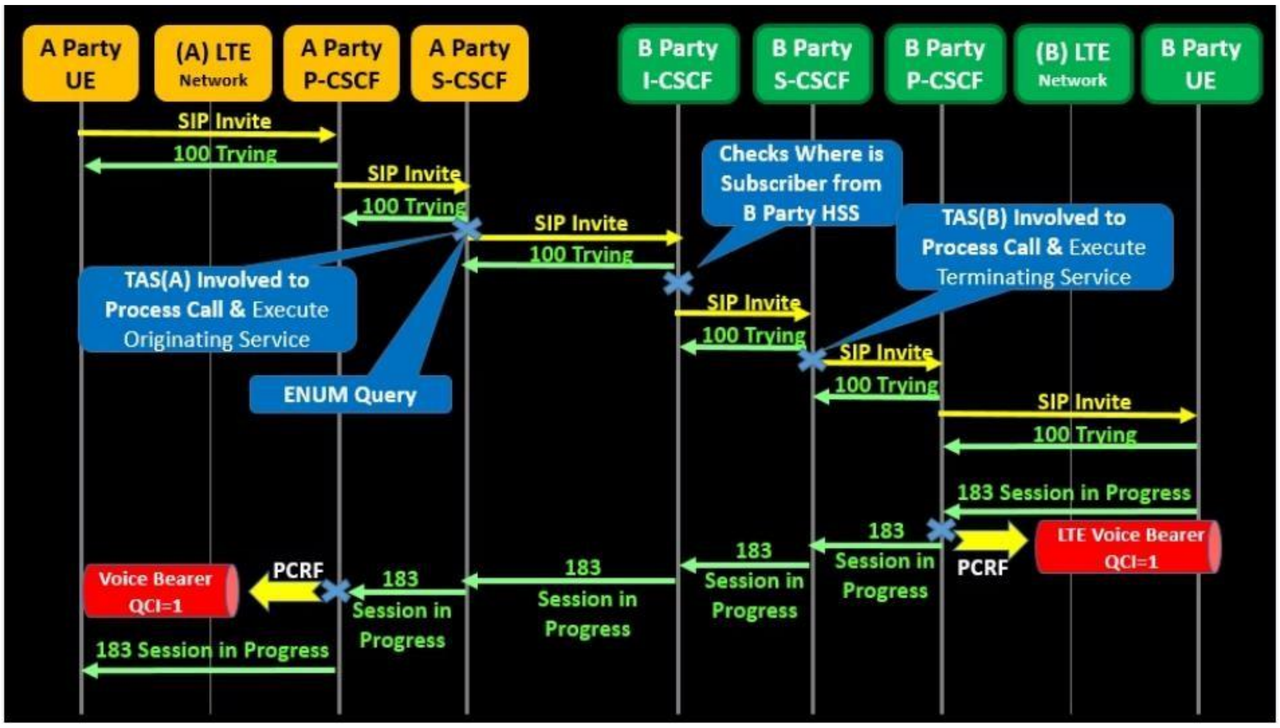


Fig. 1: *IMS Call Flow*

4.4 Session Initiation Protocol (SIP):

Overview and Purpose: SIP is an application-layer *signaling* protocol for creating, modifying, and terminating multimedia sessions (such as VoIP calls, video conferences, and messaging) over IP networksietf.org. It is widely used in telecommunications, especially VoIP and IMS networks. SIP establishes sessions by carrying session descriptions (typically in SDP) so that endpoints can agree on compatible media types. In essence, SIP initiates communication (INVITE), confirms it (ACK), and ends it (BYE). For example, a user's device (User Agent) sends an **INVITE** to establish a call; the callee replies with a "200 OK" to accept; and the caller sends **ACK** to confirm. When the call ends, one side sends a **BYE** message to terminate the session

Role in Telecommunications: SIP serves as the signaling layer in VoIP/IMS systems. It works alongside media transport protocols: while SIP sets up and tears down calls, the actual media streams (audio/video) are typically carried by RTP. SIP also provides user location (via registration), authentication, call-routing policies, and supports proxies, redirect servers, and registrars. It is the core protocol for call control in many IP-based telephony deployments. SIP can run over UDP, TCP, or TLS, using default port 5060 (UDP/TCP) and 5061 for secure TLS transport.

Common Messages and Operations: SIP uses a request–response model similar to HTTP, with several standard methods (requests) and corresponding responses. Common SIP **request methods** include:

- **INVITE:** Initiate a new session (call).
- **ACK:** Acknowledge receipt of a final response to INVITE.
- **BYE:** Terminate a session.
- **CANCEL:** Cancel a pending INVITE.

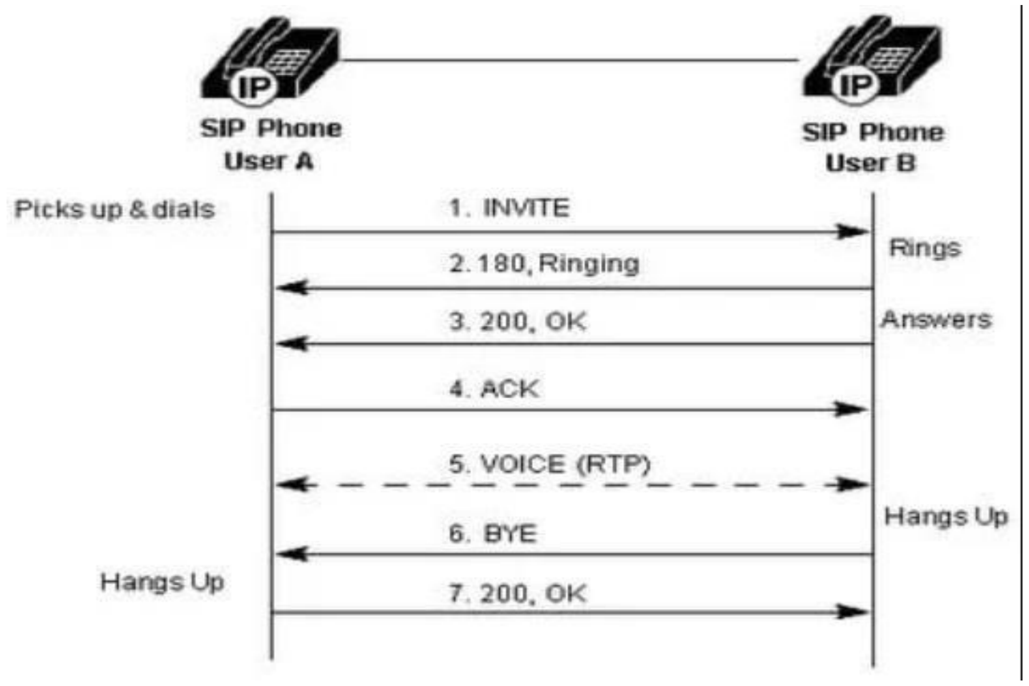
- **REGISTER:** Register a user’s current location with a SIP registrar.
- **OPTIONS:** Query capabilities of a peer.
- **PRACK, INFO, UPDATE, SUBSCRIBE/NOTIFY, MESSAGE** and others for supplementary features.

Responses use standard SIP/HTTP-like status codes (1xx Provisional, 2xx Success, 3xx Redirection, 4xx/5xx/6xx Error). For example, an **INVITE** request elicits provisional responses (e.g. “180 Ringing”) and a final “200 OK” when the call is accepted.

Protocol Structure and Key Fields: SIP messages are text-based and consist of a start-line plus header lines and an optional body. For requests, the start-line is like INVITE sip:bob@biloxi.com SIP/2.0; for responses, it is like SIP/2.0 200 OK. Common header fields include **Via** (proxy path), **From**, **To**, **Contact**, **Call-ID** (globally unique call identifier), **CSeq** (sequence number and method), **Max-Forwards**, and **Subject**, among others. The **Call-ID** (UUID per session) and tags in From/To allow correlating requests and responses and distinguishing dialogs. Message bodies typically carry the SDP session description. SIP’s textual format and explicit headers make it human-readable and easy to inspect in packet captures.

PCAP Analysis: In packet capture traces, SIP signaling can be identified by traffic on UDP/TCP port 5060 and by the “SIP” protocol indicator. PCAP tools like Wireshark can decode SIP messages, extract call flows, and display dialogs. An analyst can follow an INVITE transaction (requests and responses) to reconstruct a call. Fields like Call-ID, From/To tags, and CSeq allow linking messages in a call flow. SIP PCAP analysis is useful for VoIP troubleshooting: one can check for failed calls (e.g. missing final response), authentication errors, and correct SDP parameters. Detecting repeated INVITEs or retransmissions can indicate network issues. Thus, SIP analysis in PCAP provides insight into signaling-level problems in telephony networks.

Network Architecture (Stack): SIP is purely an application-layer protocol (in the IP suite). It typically runs over UDP or TCP (and over TLS for SIPS) on top of IP. SIP fits into the control (signaling) plane of the network architecture. It interacts with other VoIP protocols: for example, after SIP sets up a session, media flows using RTP (often with accompanying RTCP) over UDP. SIP may also use underlying DNS SRV records for server discovery. In summary, SIP sits at the top of the stack in telecommunications, coordinating session control in VoIP/IMS.



.

Fig. 2: SIP Call Flow Diagram

4.5 Diameter Protocol:

Overview and Purpose: Diameter is an *Authentication, Authorization, and Accounting (AAA)* protocol designed as the successor to RADIUS. It provides a flexible messaging framework for AAA services in networksf5.com. In telecom, Diameter is used for tasks like subscriber authentication, policy control, billing, and mobility management. It is especially prevalent in 3G/4G/5G networks (e.g. LTE/EPC and IMS) for interfaces such as S6a (HSS–MME), Cx (HSS–CSCF), and Gy (online charging). Diameter supports reliable transport (over TCP or SCTP) and has built-in features for capability negotiation and failover that RADIUS lacks.

Role in Telecommunications: Diameter forms the backbone of AAA interactions in modern mobile networks. For example, when a user attaches to an LTE network, the MME queries the HSS via the S6a interface using Diameter to retrieve subscriber data (location, subscription info) and authentication vectors. Other network elements like the PCRF (Policy and Charging Rules Function) and the Online Charging System (OCS) also communicate via Diameter to manage QoS and charging. Diameter messages carry AVPs (Attribute-Value Pairs) that encode subscriber identity, session context, and accounting data. As the F5 glossary notes, Diameter provides AAA messaging services for network access and data mobility applications in 3G, IMS, and LTE/4G networks, forming the basis for service administration (user services, QoS levels, etc.)

Common Messages and Operations: Diameter messages are organized as Request/Answer pairs with numeric *command codes*. The base protocol defines commands for general AAA, and various applications (e.g. NAS, credit control, SIP) add more. Common **base commands** include:

- **Capabilities-Exchange-Request/Answer (CER/CEA):** Used when two Diameter peers connect, to exchange capabilities.

- **Device-Watchdog-Request/Answer (DWR/DWA):** Heartbeat messages to check connectivity.
- **Disconnect-Peer-Request/Answer (DPR/DPA):** To tear down an existing connection.
- **Re-Auth-Request/Answer (RAR/RAA):** To prompt a peer to re-authenticate or re-authorize an ongoing session.
- **Abort-Session-Request/Answer (ASR/ASA):** To abruptly terminate a session.
- **Accounting-Request/Answer (ACR/ACA):** Used when accounting is needed (e.g. usage records).

Additional common commands (often in “credit-control” or 3GPP applications) include:

- **Credit-Control-Request/Answer (CCR/CCA):** Used by charging applications (e.g. Gy interface in LTE).
- **AA-Request/Answer (AAR/AAA):** Authentication/Authorization for NAS (defined in 3GPP NAS application).
- **Sh, Cx, S6a commands:** e.g. User-Data-Request/Answer, etc., specific to IMS/EPC.

Protocol Structure and Key Fields: A Diameter message has a fixed 20-byte header followed by zero or more AVPs (Attribute-Value Pairs)en.wikipedia.orgen.wikipedia.org. The header fields include (from [51], bit offsets):

- **Version (8 bits):** Protocol version (always 1 today).
- **Message Length (24 bits):** Total message size in bytes (header + AVPs).
- **Flags (8 bits):** Control bits; notably **R** (Request bit), **P** (Proxiable), **E** (Error), and **T** (Potentially re-transmitted message)en.wikipedia.org. The R-bit indicates request or answer.
- **Command-Code (24 bits):** Identifies the command (e.g. 257 for CER, 272 for CCR)en.wikipedia.org.
- **Application-ID (32 bits):** Indicates which application context (base protocol or extension) the message belongs to.
- **Hop-by-Hop ID (32 bits):** A transaction identifier used to match requests with answers along the pathen.wikipedia.orgen.wikipedia.org.
- **End-to-End ID (32 bits):** A globally unique identifier to detect duplicate transactions.

Each AVP (following the header) has its own header (code, length, flags) and carries a specific parameter (e.g. Session-Id, Origin-Host, User-Name, etc.). AVPs enable Diameter’s extensibility. The structure thus allows adding new AVPs or commands without redesigning the protocol.

PCAP Analysis: In packet captures, Diameter traffic typically appears on TCP or SCTP port 3868 (IANA-assigned). Tools like Wireshark can dissect Diameter: one can see CER/CER exchanges when a session is opened, and view the AVPs in each message. For network troubleshooting, one might inspect AAA exchanges (e.g. authentication errors in AAR/AAA responses), or view credit-control sequences (CCR/CCA) for billing issues. Because many 3GPP interfaces run on Diameter, capturing at the core network (e.g. between MME and HSS, or P-GW and PCRF) allows analysis of subscriber attach, detach, and charging procedures. For example, one could verify whether a Subscriber-Data-Answer (in

S6a) contains the expected subscriber profile. In summary, Diameter's standardized header and AVPs make it easy to filter PCAP traffic by port and decode the protocol for diagnosis.

Network Architecture (Stack): Diameter is an application-layer protocolen.wikipedia.org that runs directly over transport (TCP or SCTP over IP). It is generally part of the control plane in telecom networks. Within the OSI/Internet stack, Diameter resides above the transport layer. In 3GPP architecture, it is used on interfaces between core network functions (MME, HSS, PCRF, etc.), in contrast to protocols like GTP or S1AP that may run over UDP/SCTP. Diameter itself uses reliable transports (TCP, or SCTP for multi-homing) to ensure that messages (and their AVPs) are delivered in order.

4.6 Domain Name System (DNS):

Overview and Purpose: DNS is the Internet's naming system, a distributed database that translates human-friendly domain names (e.g. www.example.com) into IP addresses (e.g. 93.184.216.34)extrahop.com. This translation is essential because while users use names, network communications require numeric addresses. DNS also stores various other resource records (MX for mail servers, TXT, NS for delegations, etc.).

Role in Networking: DNS is ubiquitous in networking as the first step in accessing any Internet resource. Every time a client (e.g. a web browser) looks up a hostname, it generates a DNS query. DNS resolution typically involves a hierarchy: the client's resolver may contact a root server, then a TLD server, then an authoritative server to obtain the final answer (or rely on cached results). DNS is crucial for load balancing, service discovery, email routing, and any system that refers to hostnames rather than raw IPs. Enterprise networks often run internal DNS servers, and many advanced features (e.g. split-horizon DNS, DNSSEC for security, DNS over TLS/HTTPS for privacy) have emerged.

Common Messages and Operations: DNS uses two basic message types: **queries** and **responses**, both of which share the same basic formaten.wikipedia.org. A client (resolver) sends a **query** asking for one or more resource records (RRs) matching a name and type (e.g. type A for IPv4 address). The server returns a **response** with the answer RRs, and possibly authority and additional sections. Queries can be recursive (server does full resolution) or iterative (server points to other servers). Record types include A, AAAA (IPv6), CNAME (alias), MX, NS, TXT, SRV, and more. Common operations include:

- **Recursive Query:** Client asks local resolver for a name; if the resolver doesn't have it cached, it performs iterative queries up the chain.
- **Iterative Query:** Resolver queries root, then TLD, then authoritative servers as needed.
- **Zone Transfer (AXFR/IXFR):** Query for replicating DNS data between primary and secondary servers.
- **Update (RFC 2136):** Dynamic updates allow a client to add/remove records (e.g. DHCP client registering its hostname).
- **DNSSEC Lookups:** Queries for RRSIG, DNSKEY, etc., with additional validation steps.

Protocol Structure and Key Fields: A DNS message (query or response) has a fixed header (12 bytes) followed by four sections (Question, Answer RRs, Authority RRs, Additional RRs)en.wikipedia.org. The **header** fields include:

- **Transaction ID (16 bits):** Matches queries with responses.
- **Flags (16 bits):** Includes QR (query=0/response=1), OPCODE (standard query, inverse, status, etc.), AA (Authoritative Answer), TC (Truncated), RD (Recursion Desired), RA (Recursion Available), and RCODE (response code: NOERROR, NXDOMAIN, SERVFAIL, etc.)en.wikipedia.org.
- **QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT (16 bits each):** Number of records in Question, Answer, Authority, and Additional sections, respectivelyen.wikipedia.org.

Each **question** entry has a name (as a sequence of labels), a type (e.g. A, AAAA, MX), and a class (usually IN). Each **resource record** has a similar header (name, type, class, TTL, length) followed by its data (e.g. IP address for A record). The RCODE bits in the header indicate success (0) or errors like NXDOMAIN (3) if the name does not existen.wikipedia.org. Notably, DNS originally limited UDP messages to 512 bytes; larger responses set the TC bit or require using TCPietf.org. By default DNS runs over UDP port 53 (with TCP port 53 as fallback for large transfers)ietf.org.

PCAP Analysis: DNS traffic can be easily spotted on UDP port 53 in packet captures. Wireshark and similar tools will parse DNS packets and display queries and responses. Analyzing DNS in PCAP is useful for troubleshooting name resolution: one can check if the client query matches the response, if DNSSEC validations pass, or if there are timeouts. For example, missing responses or NXDOMAIN codes indicate lookup failures. Analysis may also reveal misconfigurations (wrong records, TTL issues) or malicious behavior (e.g. DNS spoofing attempts, DNS tunneling). Since DNS is often the first step for many protocols, correlating DNS queries with subsequent TCP/HTTP flows can help map hostnames to IPs observed in other traffic. Thus, understanding DNS content (query names, record types, etc.) is important when interpreting PCAPs, and many network analysis tools track DNS lookups to help identify endpoints in traces.

Network Architecture (Stack): DNS is an application-layer protocol in the Internet suite. It typically runs over UDP (port 53) and uses TCP (port 53) when responses exceed the UDP size limit or for zone transfersietf.org. In layered architecture, DNS sits above the transport layer. It can also be tunneled over other transports (DNS over TLS/HTTPS over TCP/UDP 443) for security. Functionally, DNS is a distributed database, so its “servers” act as application servers; resolvers and clients implement the client side. It is orthogonal to routing or switching layers, but essential for hostname-to-IP resolution in all higher-layer applications.

4.7 S1 Application Protocol (S1AP):

Overview and Purpose: S1AP (S1 Application Protocol) is a control-plane protocol defined by 3GPP for LTE (4G) networksblackduck.com. It provides signaling between the Evolved-UTRAN (E-UTRAN, i.e. eNodeBs/base stations) and the core network (Evolved Packet Core, EPC). In particular, S1AP carries NAS (Non-Access-Stratum) messages (like Attach, Paging, TAU) over the S1-MME interface and also supports control messages related to bearer setup, mobility (handover), and QoS. In short, S1AP relays control messages between the eNodeB and the MME (Mobility Management Entity), enabling functions such as initial user attach, handover, bearer creation, and paging.

Role in Telecommunications: S1AP is a critical LTE control-plane protocol. Whenever a user equipment (UE) connects, moves, or changes state, the eNodeB and MME exchange S1AP messages. For example, during an LTE Attach procedure, the eNodeB sends an **Initial UE Message** (containing the UE's NAS Attach Request) to the MME over S1AP. The MME may then issue an **Initial Context Setup Request** to the eNodeB via S1AP to establish resources and radio bearers. For inter-eNodeB handovers, S1AP carries **Handover Required/Request/Command** messages. The BlackDuck description states: "The S1 Application Protocol (S1AP) provides the control plane signalling between E-UTRAN and evolved packet core (EPC). The used interface is S1-MME between eNB and MME"blackduck.com. S1AP supports both *UE-associated* services (tied to a specific UE and bearer) and *non-UE-associated* services (e.g. MME status, global paging)itecspec.com.

Common Messages and Operations: S1AP messages are extensive. Some important UE-related procedures and their messages include:

- **Initial UE Message:** Sent by eNodeB to MME when a new UE initiates connection (includes NAS payload).
- **UE Context Setup/Release Request/Response:** MME requests eNodeB to configure or release a bearer for a UE.
- **Handover Notify/Required/Request:** Messages for intra-LTE handover (MME coordinates via S1AP).
- **Paging:** MME pages UE via the eNodeB when a downlink data arrives for an idle UE.
- **NAS Transport (uplink/downlink):** Carry NAS messages (Attach, Detach, TAU, Authentication, etc.) between UE (via eNodeB) and MME.
- **Error Indication, UE Context Release Indication, EnB Status Transfer, MME Status Transfer:** For various failure and status reporting.
- **Overload Start/Stop:** MME can signal eNodeB to reduce load.

The valid8.com list (partial) shows messages like "PAGING", "INITIAL UE MESSAGE", "UE CONTEXT RELEASE REQUEST", etc.valid8.com. S1AP also includes CDMA2000 interworking, trace, location reporting, and configuration transfer messages, but the core are those above.

Protocol Structure and Key Fields: S1AP messages are encoded using ASN.1 and transferred over SCTP. Each message has an **S1AP PDU** header with a procedure code and criticality, followed by Information Elements (IEs) specific to that message. Common IEs include *eNB UE S1AP ID*, *MME UE S1AP ID* (identifiers for the UE context in eNB and MME), *TMSI*, *Tracking Area ID*, *Bearer QoS parameters*, and embedded NAS messages. The ASN.1 definition (TS 36.413) specifies the structure. S1AP piggybacks NAS protocols, so NAS headers (like EPS attach request) are encapsulated as an IE within S1AP. Because it uses SCTP (typically port 36412), it supports multi-homing and ordered delivery. Unlike the other protocols discussed, S1AP is a purely binary, complex protocol specified by 3GPP and not simple to read by eye. Decoders in tools (Wireshark) interpret the ASN.1 to show the fields.

PCAP Analysis: In PCAP traces, S1AP can be identified as SCTP traffic (default destination port 36412 for eNB-MME S1AP). Wireshark decodes S1AP and displays the procedure and IEs. An analyst can follow S1AP procedures: for instance, seeing the **Initial Context Setup Request** and **Response** when a UE attaches. In mobile network debugging, capturing S1AP allows tracing attach/detach failures, unauthorized NAS requests, or handover issues. The two key IDs (MME UE S1AP ID and eNB UE S1AP ID) link signaling between eNB and MME. S1AP trace analysis is crucial for LTE signaling troubleshooting. With proper capture (e.g. on an MME interface), one can see which NAS messages were sent inside S1AP, and whether procedures succeeded.

Network Architecture (Stack): S1AP sits at the top of the control-plane in the LTE network architectureblackduck.com. It runs on top of SCTP, which in turn uses IP (layer 3). Thus, stack-wise: **Application (S1AP) / SCTP / IP / L2 (e.g. Ethernet/MPLS)**. S1AP is effectively part of the RAN–EPC interface. It is defined in 3GPP TS 36.413 (Release 17)blackduck.com. Because it carries NAS messages, it is connected logically above the NAS layer (NAS itself resides between UE and MME and is carried by S1AP). In network diagrams, S1AP appears on the S1-MME interface, under the umbrella of the 3GPP signaling plane for LTE.

4.8 GPRS Tunnelling Protocol (GTP):

Large Language Models (LLMs) are deep learning models, typically based on transformer architectures, that are pre-trained on large corpora of text data. LLMs, such as GPT-3 and T5, have demonstrated remarkable capabilities in natural language understanding, generation, and translation. These models can be fine-tuned on specific downstream tasks with relatively small amounts of taskspecific data, making them versatile and adaptable to a wide range of applications.

Overview and Purpose: GTP (GPRS Tunnelling Protocol) is a family of protocols used in mobile networks (2G/3G/4G/5G) to carry both user data and control signaling between core network nodesen.wikipedia.org. In 3GPP networks, GTP is used on interfaces such as Gn/Gp (between SGSN and GGSN in 3G), S1-U/S5/S8 (for user data between eNodeB and

P-GW/S-GW in LTE), and S11 (between MME and S-GW for control). GTP is split into multiple variants:

- **GTP-C (GTP-Control):** Used for control-plane signaling (session management). For example, creating, updating, or deleting a bearer (tunnel) uses GTP-C (e.g. "Create Session Request"/"Response").
- **GTP-U (GTP-User):** Used for user-plane (payload) tunneling. It encapsulates user IP packets (IPv4, IPv6, or PPP) inside GTP-U packets for transport between nodes (e.g. P-GW to eNodeB)en.wikipedia.org.
- **GTP′ (GTP-prime):** Used for charging data transport (between GSNs and charging gateways)en.wikipedia.org.

The purpose of GTP is to tunnel and route user traffic and to manage mobile session state (PDP contexts or EPS bearers). It allows a UE to maintain connectivity while moving by mapping its tunnel endpoint and user-IP at different locations.

Role in Telecommunications: GTP is fundamental to mobile core networks. When a user starts a data session (PDP/PDN connection), the network exchanges GTP-C messages to set up the tunnel (e.g. *Create Session Request* and *Response* on S11 and S5/S8 interfaces). Once set up, the user's IP packets are sent inside GTP-U packets between the RAN and the gateway. The Tunnel Endpoint Identifier (TEID) in the GTP header identifies which session/bearer the packet belongs to. For example, in LTE, when the MME tells the S-GW to create a bearer, the S-GW and eNodeB agree on a TEID and start encapsulating the UE's data in GTP-U towards the eNodeB. During mobility (handover), GTP-C is used to update endpoints, and GTP-U paths are switched. GTP′ is used by charging nodes to collect data usage records.

Common Messages and Operations: GTP messages are also organized as Request/Response or indications, but unlike Diameter, they are not "requests" to an application layer but rather control plane events. Examples include:

- **Create Session Request/Response:** (GTP-C) Establish a new user session/bearer.
- **Modify Bearer Request/Response:** (GTP-C) Update QoS or endpoints of an existing bearer.
- **Delete Session Request/Response:** (GTP-C) Tear down a bearer when a user detaches or ends session.
- **Echo Request/Response:** To check connectivity between GTP peers (like a heartbeat).
- **Version Not Supported Response:** If a GTP version mismatch occurs.
- **N-PDU Notification:** Indication about missing user data (rare).

For GTP-U (user plane), there are no "commands" per se aside from the tunnel header carrying data. However, the presence of a GTP-U header indicates a user data packet belonging to a TEID (tunnel). GTP′ (prime) uses a similar packet format to GTP-C but for charging.

Protocol Structure and Key Fields: A GTPv1 header is usually 8 bytes (minimum) and may be followed by optional fields (sequence number, N-PDU number, extension header). Key fields (GTPv1 as example)en.wikipedia.orgen.wikipedia.org:

- **Version (3 bits):** GTP version (1 for GTPv1).

- **Protocol Type (1 bit):** Distinguish GTP (1) vs GTP′ (0).
- **Flags (E, S, PN bits):** Indicate optional fields are present (Extension header, Sequence number, N-PDU number)en.wikipedia.org.
- **Message Type (8 bits):** Code for the GTP message (e.g. 32 = Create PDP Context Request in GTPv0, 32= Create Session Req in GTPv1). Different message type values are defined in 3GPP specs (e.g. 3GPP TS 29.281 for GTP).
- **Message Length (16 bits):** Length of payload (everything after the header).
- **TEID (32 bits):** Tunnel Endpoint Identifier, used to multiplex tunnels on the same UDP porten.wikipedia.org. This is the unique session identifier for the GTP tunnel at the receiving end.
- **Sequence Number (16 bits, optional):** Present if S-flag=1, for linking requests and responses.
- **N-PDU Number (8 bits, optional):** Present if PN-flag=1.
- **Next Extension Header Type (8 bits, optional):** If extensions are used (E-flag).

The TEID is especially important: it lets the receiver demultiplex user-plane packets to the correct bearer context. The Wikipedia header diagram [46] shows these fields in detail. GTPv2 (used in LTE control plane on S11/S5/S8) has a similar structure but with fields adapted for GTPv2 messages (e.g. no GTP′ flag, etc.).

PCAP Analysis: GTP traffic can be identified by UDP port numbers: by convention, **GTP-C** (control) uses UDP port 2123 and **GTP-U** (user) uses UDP port 2152en.wikipedia.org. Packet captures from network cores (e.g. between S-GW and eNodeB) will show GTP-U encapsulating user IP packets. Wireshark can decode GTP headers: one can see the TEID, message type, and various IE's. For control traces, one sees the Create Session, Modify/Delete messages. For user data, GTP-U frames can be "followed" to reveal the inner IP packet (Wireshark will decapsulate and show the inner Ethernet/IP/TCP etc). GTP analysis helps debug session setup and data path. For example, if a UE has no data connectivity, an analyst may check if the Create Session (GTP-C) succeeded and if GTP-U packets are flowing correctly. The TEID field in PCAP lets one track which tunnel each packet belongs to. GTP is therefore essential when analyzing mobile network packet captures.

Network Architecture (Stack): GTP is an IP-based protocol carried over UDP (and in GTP″'s case, sometimes TCP)en.wikipedia.org. For GTPv1 and v2, UDP is standard. The protocol stack is **GTP/UDP/IP/Ethernet** (or mobile-specific interfaces). In 3GPP architecture, GTP appears on interfaces within the core network (S1-U, S5/S8, S11) and between the core and RNC/SGSN (in 2G/3G). It is thus part of the GPRS core network layer, enabling mobility and tunneling of user-plane traffic. It is considered at the same layer as IPSec or GRE tunnels, but specialized for mobile networks. The Wikipedia stack diagram shows GTP under "application protocols" but above UDPen.wikipedia.org.

These core modules serve as the foundational building blocks of the FlowSight PCAP analysis tool, empowering it to dissect complex network captures, extract meaningful protocol insights, and visualize traffic flows with clarity. By combining scalable packet parsing, flexible protocol decoding, and intuitive report generation, FlowSight continuously evolves to streamline network diagnostics, accelerate troubleshooting, and inform performance optimization across diverse telecommunications environments.

5. METHODOLOGY

In our report, we employ a two-phase workflow that mirrors the real-world demands of network forensics and troubleshooting. In Phase 1 we begin by feeding raw PCAP files into FlowSight's parsing engine, where each packet is decoded, timestamped, and tagged according to protocol (SIP, Diameter, GTPv2, DNS, etc.). Custom filters eliminate noise and group related packets by identifiers (e.g., IMSI, Session-ID), ensuring every session is consistently labeled. This structured extraction lays the groundwork for deep analysis while preserving the fidelity of timing and context. In Phase 2, with structured data in hand, FlowSight applies a suite of analytic routines: it evaluates success criteria (such as registration, call setup, and error conditions), compiles statistics across sessions, and automatically renders flow diagrams and charts. Finally, all findings are assembled into a multi-tab Excel workbook—complete with summary tables, visualizations, and detailed logs—offering a clear, reproducible roadmap from raw captures to actionable network intelligence.

5.1 Phase 1: PCAP Data Extraction & Structuring

I collaborated with our network diagnostics team, whose daily task is to transform unprocessed PCAP files into insightful information. In its basic form, each capture is merely a mountain of packets, but it documents actual user calls, protocol handshakes, and error circumstances. For each release verification, the analysts' tasks include going through these captures, identifying session identifiers (IMSI, Call-ID, Session-ID), decoding protocols (SIP, Diameter, GTPv2, DNS), and manually creating filters and Excel reports.

Understanding the Problem:

Even with strong tools, the process was disjointed and manual: engineers had to spend hours manually mapping response codes, creating and fine-tuning TShark filters, and recording call start and finish timings in spreadsheets. Every new feature or vendor upgrade ran the danger of breaking old parsers, naming conventions shifted between teams, and packet fields came in vastly disparate forms. In addition to delaying release testing, this human effort brought minor mistakes that could be introduced into production, such as forgotten SIP headers, mislabeled IP subnets, or missed INVITE-200 sequences. Therefore, it was essential to automate this extraction workflow in order to ensure consistent, dependable validation of every network release.

5.1.1 Initializing the Extraction Loop

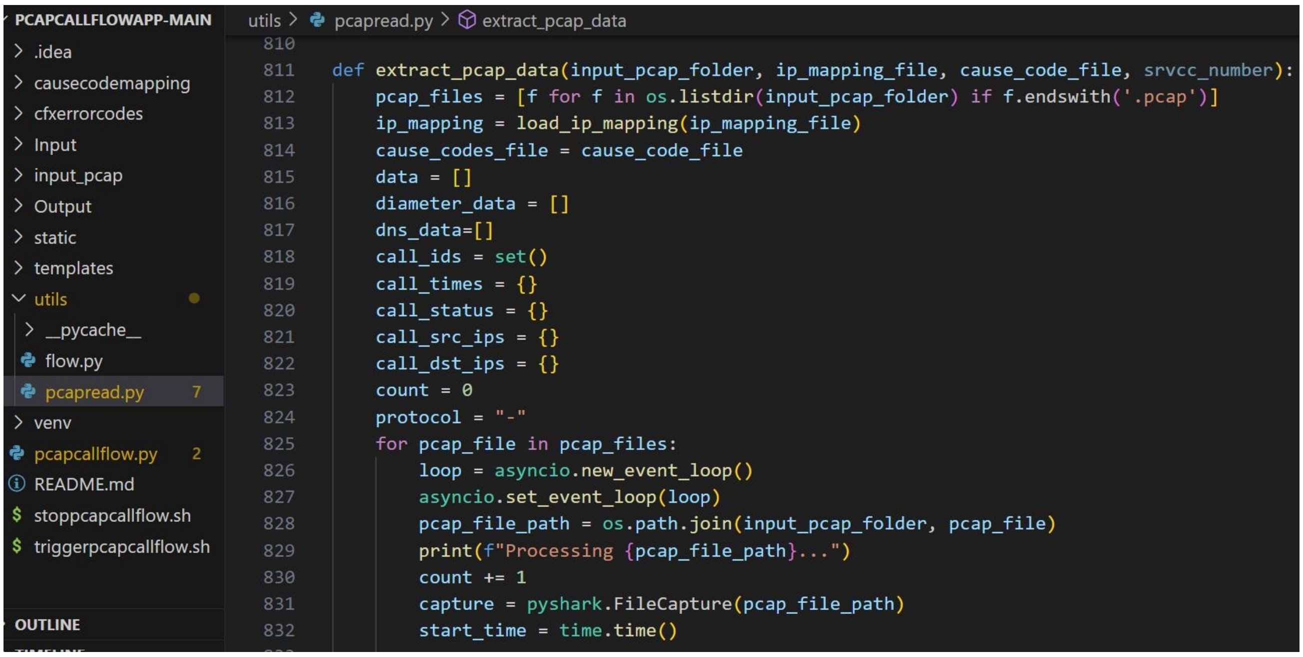
The extraction process begins by scanning through every .pcap file present in the specified input directory. This ensures that all packet captures—regardless of naming convention or file size—are included in the analysis. Each .pcap file typically contains raw network traffic data captured over a specific time period, and our goal is to extract relevant protocol information from each of them in a consistent, structured format.

Before diving into the packet data itself, the system first loads the IP-to-label mappings. These mappings are critical: they translate low-level technical identifiers like IP addresses or subnets into meaningful site names (e.g., converting 2401:4900:24:1400::/64 to something more readable like Site_A-TechLAN). This contextual labeling improves both traceability and human readability in the final reports.

Next, FlowSight prepares empty containers to hold extracted data. These include separate lists or dataframes for each supported protocol—SIP, Diameter, and DNS. This modular approach not only keeps the data organized but also allows different types of analysis to be applied based on protocol.

Alongside these data containers, a number of helper dictionaries and sets are initialized. These track unique call identifiers, time ranges for each session, IP addresses involved in the communication, and status outcomes. This setup stage ensures that once packet-level parsing begins, there is minimal overhead and no delay caused by late initialization or missing references.

By the time this step completes, the environment is fully primed: the .pcap files are identified, the metadata mappings are loaded, and all internal structures are initialized—allowing the extraction logic to proceed efficiently with clean access to all required references.



```python
def extract_pcap_data(input_pcap_folder, ip_mapping_file, cause_code_file, srvcc_number):
    pcap_files = [f for f in os.listdir(input_pcap_folder) if f.endswith('.pcap')]
    ip_mapping = load_ip_mapping(ip_mapping_file)
    cause_codes_file = cause_code_file
    data = []
    diameter_data = []
    dns_data=[]
    call_ids = set()
    call_times = {}
    call_status = {}
    call_src_ips = {}
    call_dst_ips = {}
    count = 0
    protocol = "-"
    for pcap_file in pcap_files:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)
        pcap_file_path = os.path.join(input_pcap_folder, pcap_file)
        print(f"Processing {pcap_file_path}...")
        count += 1
        capture = pyshark.FileCapture(pcap_file_path)
        start_time = time.time()
```

Fig. 3 Code Snippet for the same.

5.1.2 SIP Extraction:

Once we’ve bootstrapped our loop, every packet is checked for a SIP layer. Whenever we detect SIP traffic, FlowSight pulls out the call details we need to reconstruct each session:

Identify the Call-ID

- We grab packet.sip.call_id (or “–” if missing) and add it to our call_ids set. This lets us group all messages belonging to the same call.

Map Methods & Status Codes

- Using our pre-loaded cause_code_file, numeric status codes (200, 480, etc.) and SIP methods (INVITE, BYE) are translated into human-friendly descriptions (e.g. “200 OK → Invite Completed”).

Track Call Start & End Times

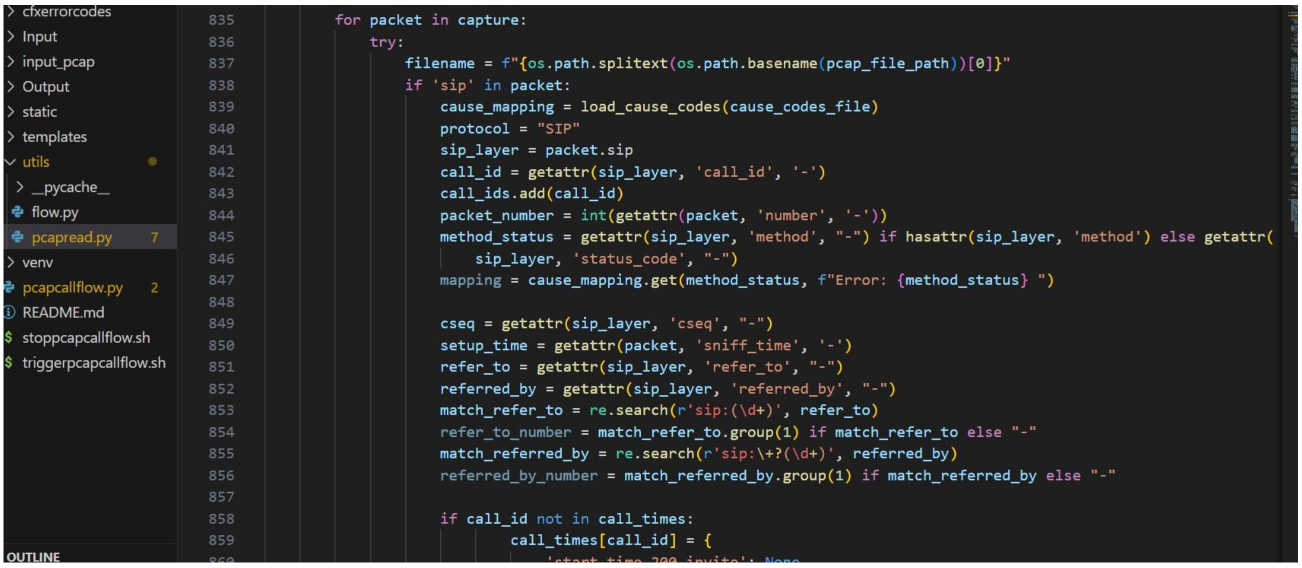
- The first time we see a 200 OK in response to an INVITE (via sip.cseq), we stamp that as the call’s start.
- The very first BYE we encounter marks the end time. From these two timestamps, downstream logic can compute total call duration.

Extract & Label IP Addresses

- Whether the packet carries IPv4 or IPv6, we take packet.ip.src/packet.ip.dst (or packet.ipv6.src/dst) and run them through get_ip_label() to turn raw addresses into sites like Mumbai-Core or Vendor-Edge.

Parse Key SIP Headers

- **Refer-To / Referred-By**: We regex out the dialed or forwarding number from URIs like sip:1234@domain.
- **Audio URIs**: For packets beginning with "INFO sip:msml@", we hunt for <audio uri="file://…"> in the message body.
- **SDP Media Types**: Any m= lines in the SDP get joined into a comma-separated list (audio, video, etc.).
- **Optional Headers**: Fields like warning and reason (if present) are captured too, giving extra context on call failures or provisional responses.



```python
for packet in capture:
    try:
        filename = f"{os.path.splitext(os.path.basename(pcap_file_path))[0]}"
        if 'sip' in packet:
            cause_mapping = load_cause_codes(cause_codes_file)
            protocol = "SIP"
            sip_layer = packet.sip
            call_id = getattr(sip_layer, 'call_id', '-')
            call_ids.add(call_id)
            packet_number = int(getattr(packet, 'number', '-'))
            method_status = getattr(sip_layer, 'method', "-") if hasattr(sip_layer, 'method') else getattr(
                sip_layer, 'status_code', "-")
            mapping = cause_mapping.get(method_status, f"Error: {method_status} ")

            cseq = getattr(sip_layer, 'cseq', "-")
            setup_time = getattr(packet, 'sniff_time', '-')
            refer_to = getattr(sip_layer, 'refer_to', "-")
            referred_by = getattr(sip_layer, 'referred_by', "-")
            match_refer_to = re.search(r'sip:(\d+)', refer_to)
            refer_to_number = match_refer_to.group(1) if match_refer_to else "-"
            match_referred_by = re.search(r'sip:\+?(\d+)', referred_by)
            referred_by_number = match_referred_by.group(1) if match_referred_by else "-"

            if call_id not in call_times:
                call_times[call_id] = {
```

Fig. 4 Code Snippet for the same.

5.1.3 Diameter & DNS Extraction:

For every packet carrying a Diameter layer, we:

Capture Core Metadata

- packet.number and packet.sniff_time give us packet sequence and timestamp.
- We label the protocol as “Diameter” so it’s easy to distinguish from SIP and DNS later.

Pull AVP Codes & Session Identifiers

- diameter.avp_code can be a list or single value; we join multiple codes with commas.
- Session context comes from diameter.session_id, while diameter.cmd_code and diameter.result_code tell us which command ran and whether it succeeded.

Extract Key AVPs

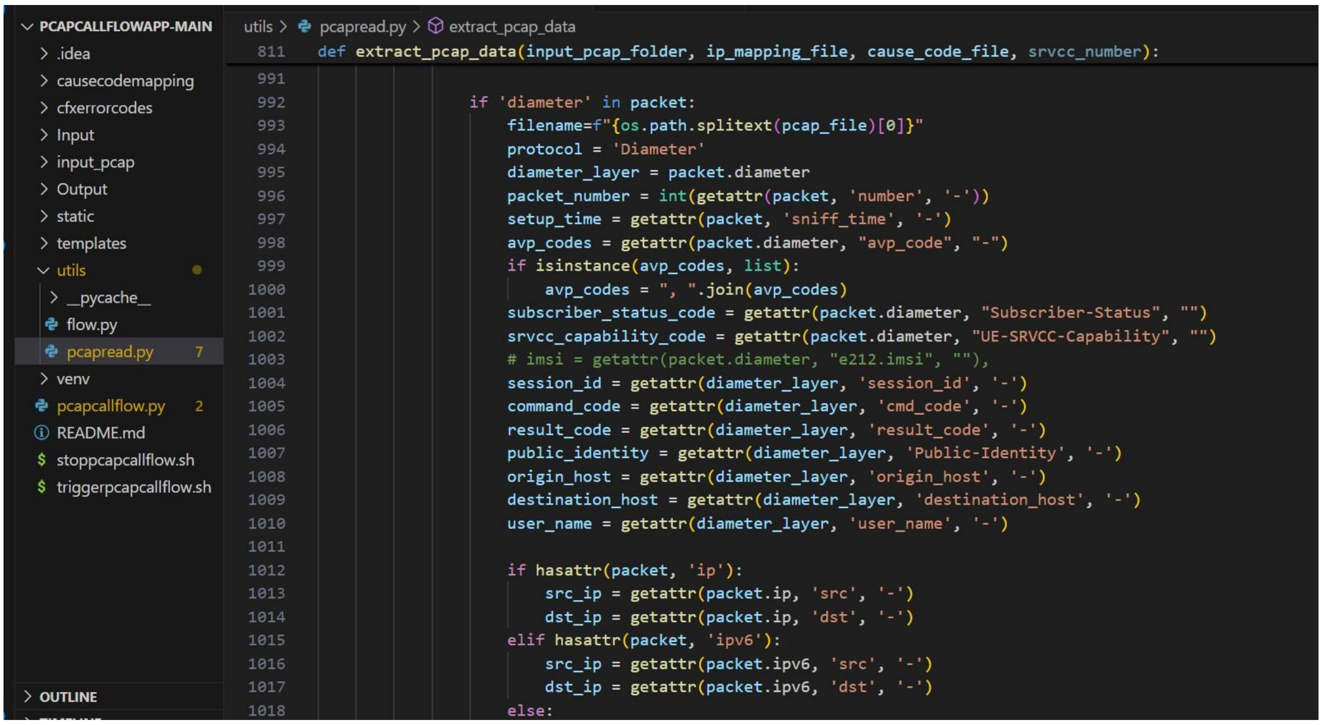
- **E212 IMSI** (if present) or fallback attributes like user_name.
- **UE-SRVCC-Capability** to track handover readiness.
- **Subscriber-Status** to know if the subscriber is active, barred, etc.

IP Address Labeling

- As with SIP, we normalize IPv4 vs. IPv6 and map raw addresses to site names for quick lookup.

Row Construction

- Every field—core metadata, AVPs, IP labels—gets appended to diameter_data as one row.



```python
def extract_pcap_data(input_pcap_folder, ip_mapping_file, cause_code_file, srvcc_number):

                if 'diameter' in packet:
                    filename=f"{os.path.splitext(pcap_file)[0]}"
                    protocol = 'Diameter'
                    diameter_layer = packet.diameter
                    packet_number = int(getattr(packet, 'number', '-'))
                    setup_time = getattr(packet, 'sniff_time', '-')
                    avp_codes = getattr(packet.diameter, "avp_code", "-")
                    if isinstance(avp_codes, list):
                        avp_codes = ", ".join(avp_codes)
                    subscriber_status_code = getattr(packet.diameter, "Subscriber-Status", "")
                    srvcc_capability_code = getattr(packet.diameter, "UE-SRVCC-Capability", "")
                    # imsi = getattr(packet.diameter, "e212.imsi", ""),
                    session_id = getattr(diameter_layer, 'session_id', '-')
                    command_code = getattr(diameter_layer, 'cmd_code', '-')
                    result_code = getattr(diameter_layer, 'result_code', '-')
                    public_identity = getattr(diameter_layer, 'Public-Identity', '-')
                    origin_host = getattr(diameter_layer, 'origin_host', '-')
                    destination_host = getattr(diameter_layer, 'destination_host', '-')
                    user_name = getattr(diameter_layer, 'user_name', '-')

                    if hasattr(packet, 'ip'):
                        src_ip = getattr(packet.ip, 'src', '-')
                        dst_ip = getattr(packet.ip, 'dst', '-')
                    elif hasattr(packet, 'ipv6'):
                        src_ip = getattr(packet.ipv6, 'src', '-')
                        dst_ip = getattr(packet.ipv6, 'dst', '-')
                    else:
```

Fig. 5 Code Snippet for the same.

Right after Diameter, we check for DNS records and extract:

Basic Packet Info

- Packet number, timestamp, and protocol label ("DNS").

Query Metadata

- dns.id for transaction tracking.
- dns.qry_name and dns.qry_type to know which domain was looked up and what record type (A, AAAA, SRV, etc.).

Aggregating Multi-Value Fields

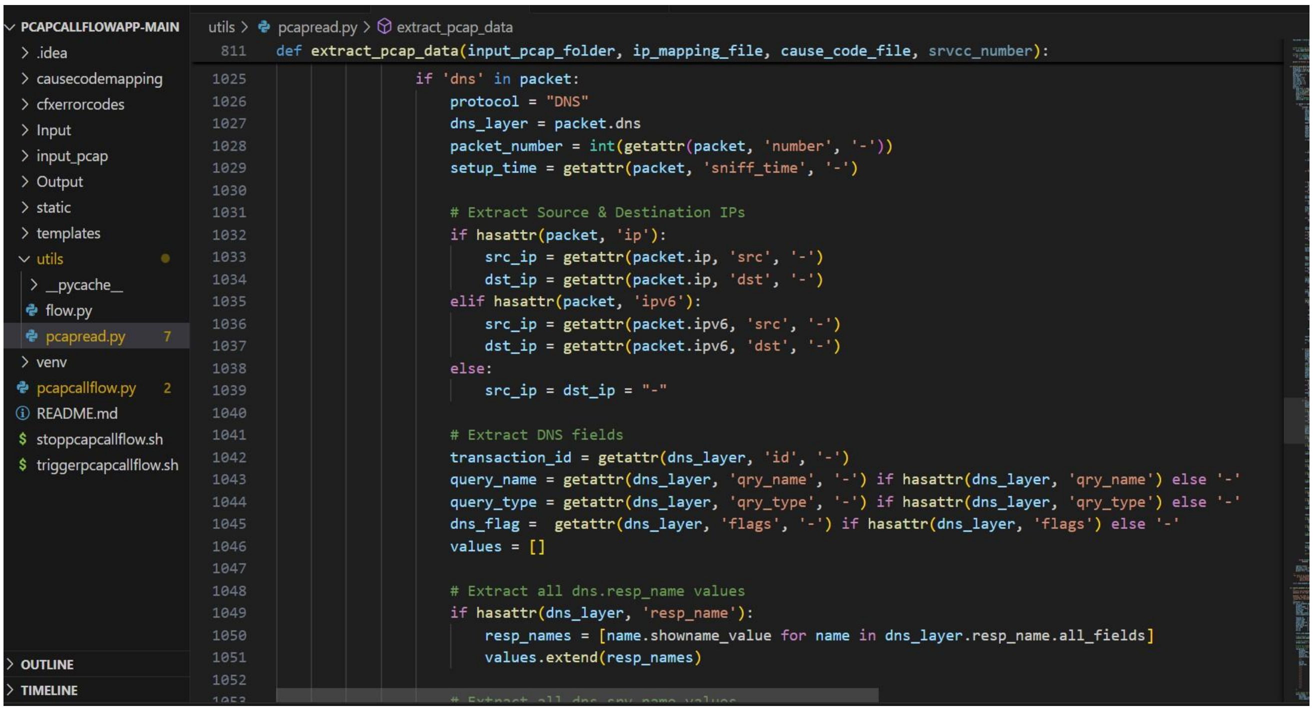
- **Answer Names**: all dns.resp_name entries are collected into combined_names.
- **SRV & NAPTR Services/Ports/Replacements**: we loop through .srv_name, .naptr_service, .srv_port, .naptr_replacement, etc., to build comma-separated strings.

A-Record Extraction

- The .a field gives the resolved IP address—critical for correlating DNS lookups to actual endpoints.

Row Assembly

- We bundle every extracted attribute into one row in dns_data.



```python
def extract_pcap_data(input_pcap_folder, ip_mapping_file, cause_code_file, srvcc_number):
                if 'dns' in packet:
                    protocol = "DNS"
                    dns_layer = packet.dns
                    packet_number = int(getattr(packet, 'number', '-'))
                    setup_time = getattr(packet, 'sniff_time', '-')

                    # Extract Source & Destination IPs
                    if hasattr(packet, 'ip'):
                        src_ip = getattr(packet.ip, 'src', '-')
                        dst_ip = getattr(packet.ip, 'dst', '-')
                    elif hasattr(packet, 'ipv6'):
                        src_ip = getattr(packet.ipv6, 'src', '-')
                        dst_ip = getattr(packet.ipv6, 'dst', '-')
                    else:
                        src_ip = dst_ip = "-"

                    # Extract DNS fields
                    transaction_id = getattr(dns_layer, 'id', '-')
                    query_name = getattr(dns_layer, 'qry_name', '-') if hasattr(dns_layer, 'qry_name') else '-'
                    query_type = getattr(dns_layer, 'qry_type', '-') if hasattr(dns_layer, 'qry_type') else '-'
                    dns_flag =  getattr(dns_layer, 'flags', '-') if hasattr(dns_layer, 'flags') else '-'
                    values = []

                    # Extract all dns.resp_name values
                    if hasattr(dns_layer, 'resp_name'):
                        resp_names = [name.showname_value for name in dns_layer.resp_name.all_fields]
                        values.extend(resp_names)
```

Fig. 6 Code Snippet for the same.

5.2 Phase 2: Flow Construction, Success Criteria & Reporting

Phase 1 FlowSight turned raw PCAP captures into tidy, protocol-specific tables. From there, Phase 2 gathers those distinct rows into a cogent narrative of each conversation, automatically assessing whether sessions went as planned or failed, and compiles everything into a polished, easily absorbed Excel spreadsheet.

Phase 2 combines all protocol events into a single "Call Flow" timeline—color-coded for fast recognition of DNS requests, Diameter transactions, SIP messages, and error conditions—instead of allowing analysts to leap between SIP logs, Diameter AVPs, and DNS lookups. FlowSight detects successful registrations, completed call setups, MRF media announcements, or SRVCC handovers after that flow is carried up and flags any error codes along the way using a set of predetermined success-criteria rules (our "Automated Analysis").

Three well-structured Excel tabs are the end product:

Call-Flow

A multi-protocol, chronological log of each packet event that includes visual indicators for failures and protocol type.

Call-Summary

One line per session that summarizes the general conclusion from our Automated Analysis, start/end times, duration, and method chains (e.g., INVITE → 200 OK → BYE).

Stats-Dashboard

A collection of tables and charts that show patterns across captures, such as the sites with the highest 4xx failure rates or the frequency of handovers.

Phase 2 provides network teams with immediate insight into call performance and dependability by automating the hours-long process of manually adjusting filters and manipulating spreadsheets. This allows them to spend more time tackling actual issues rather than clicking.

5.2.1 Merging & Normalizing Protocol Rows

We take the three protocol-specific tables—data (SIP), diameter_data, and dns_data—and transform them into a unified structure that allows for seamless integration into a single, chronological "Call Flow" sheet. This process ensures that packets from different protocols can be meaningfully compared, ordered, and analyzed side by side.

Standardizing Columns Across Protocols:

Each protocol has its own native fields. For example:

- SIP focuses on methods like INVITE, BYE, and response codes (200 OK, 404 Not Found).
- Diameter includes Command Codes, Application IDs, and Result Codes (e.g., 2001 for success).
- DNS provides query types, response codes, and domain resolution results.

To align these, we create a **normalized column schema**, including:

- **Timestamp**
- **Call ID / Identifier** (Call-ID for SIP, IMSI for Diameter, Transaction ID or queried domain for DNS)
- **Source IP / Destination IP**
- **Protocol Type** (SIP / Diameter / DNS)
- **Protocol Message** (e.g., "INVITE", "Authentication-Info-Answer", "A-Record Query")
- **Result / Status** (e.g., SIP response code, Diameter result code, DNS reply code)
- **Additional Info** (custom field to hold method chains, error descriptions, or decoded fields)

- **Error Flag** (Boolean indicator for failure packets)

Fields that are not applicable to a certain protocol are filled with blanks or N/A to ensure format consistency.

Mapping and Renaming for Uniformity:

Each dataset undergoes a **mapping function** that renames protocol-specific fields into our common schema. Examples include:

- sip.Method becomes **Protocol Message**
- diameter.Command-Code becomes **Protocol Message**
- dns.qry_name becomes part of **Additional Info**

This abstraction allows downstream logic to treat packets uniformly during sorting, analysis, and visualization.

Adding Protocol Tags and Type Flags:

To make packet types easily distinguishable in the Call Flow:

- A **Protocol Type** column is added explicitly with values like "SIP", "Diameter", or "DNS".
- Visual tags may be used in the Excel sheet to quickly distinguish rows at a glance.

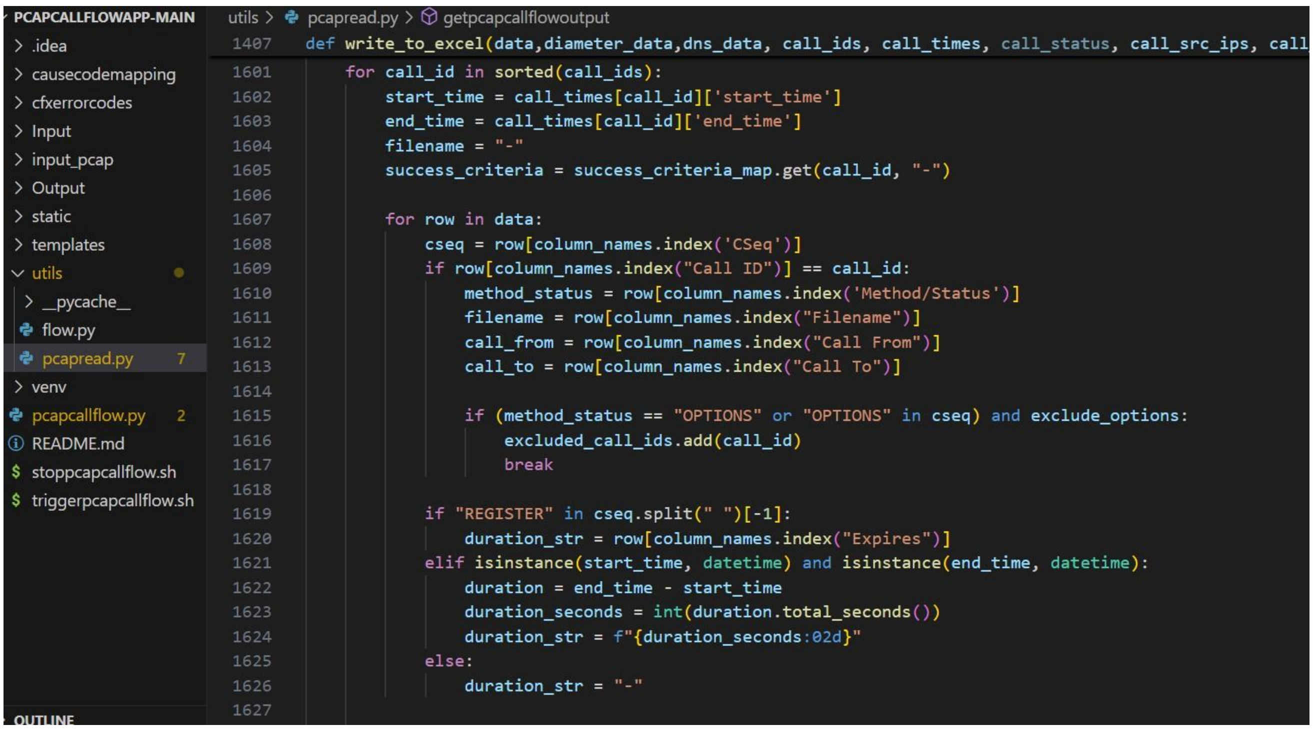
Merging Rows into a Single Table:

Once all protocol tables are normalized:

- They are concatenated into one DataFrame.
- The table is then **sorted by timestamp** to maintain chronological accuracy.

This merged table forms the backbone of the Call Flow tab, allowing users to view how different protocol events interplay during a call or session.

A single, unified table where SIP, Diameter, and DNS events live side-by-side, forming a rich, multi-protocol timeline of network activity. This table feeds directly into both **Flow Diagram** and **Call Summary** outputs in the Excel report.



```
def write_to_excel(data,diameter_data,dns_data, call_ids, call_times, call_status, call_src_ips, call
    for call_id in sorted(call_ids):
        start_time = call_times[call_id]['start_time']
        end_time = call_times[call_id]['end_time']
        filename = "-"
        success_criteria = success_criteria_map.get(call_id, "-")

        for row in data:
            cseq = row[column_names.index('CSeq')]
            if row[column_names.index("Call ID")] == call_id:
                method_status = row[column_names.index('Method/Status')]
                filename = row[column_names.index("Filename")]
                call_from = row[column_names.index("Call From")]
                call_to = row[column_names.index("Call To")]

                if (method_status == "OPTIONS" or "OPTIONS" in cseq) and exclude_options:
                    excluded_call_ids.add(call_id)
                    break

            if "REGISTER" in cseq.split(" ")[-1]:
                duration_str = row[column_names.index("Expires")]
            elif isinstance(start_time, datetime) and isinstance(end_time, datetime):
                duration = end_time - start_time
                duration_seconds = int(duration.total_seconds())
                duration_str = f"{duration_seconds:02d}"
            else:
                duration_str = "-"

```

Fig.7 Code Snippet for the same

5.2.2 Building the "Call Summary" Sheet:

We generate a concise, high-level summary for every unique session or call extracted from the full Call Flow. Each session is condensed into a single row, capturing its core details—such as source/destination endpoints, call timings, protocol activity, and most importantly, the outcome determined by our **Automated Analysis engine**.

This sheet enables teams to evaluate hundreds of calls at a glance without diving into packet-level data.

Session Grouping:

The Call Flow data is grouped by a session identifier:

- **SIP** calls are grouped by **Call-ID**.
- **Diameter** sessions are grouped by **IMSI** or **Session-ID**.
- **DNS** flows are grouped by **query name** or **transaction ID**.

Each group represents a single row in the summary sheet.

Extracting Timing Information:

For each group, we compute:

- **Start Time**: Timestamp of the first packet in the session.
- **End Time**: Timestamp of the last packet in the session.
- **Duration**: The difference between end and start time, in seconds.

These metrics help identify unusually short or long sessions.

Method Chain Construction:

From the packet sequence, we extract the main signaling flow:

- **SIP**: INVITE → 180 Ringing → 200 OK → ACK → BYE
- **Diameter**: Request → Answer pairs (e.g., ULR → ULA, AIR → AIA)
- **DNS**: Query → Response

These are concatenated into a **Method Chain** column, providing a snapshot of what protocol messages occurred.

Endpoint Extraction:

We capture:

- **Source IP**: From the first signaling packet (typically initiator).
- **Destination IP**: From the responding node.

For SIP calls, these are the caller and callee IPs. For Diameter, it reflects client and HSS/MME/SGSN. For DNS, it indicates resolver and authoritative server.

Automated Analysis Verdict:

This is the most critical column, generated using our custom rule engine. Based on patterns seen in the method chain, headers, result codes, and body content, we assign a verdict such as:

- ✅ **Successfully Connected** (SIP call completed: INVITE → 200 OK → ACK)
- Q **MRF Announcement Played** (INFO request contains audio URI, followed by 200 OK)
- 🔁 **SRVCC Successfully Connected** (Call routed to SRVCC number)

- ⚠ **Call Failed - 404 Not Found** (SIP failure response seen)
- ✕ **481 - Call/Transaction Does Not Exist**
- 🔒 **Authentication Success** (Diameter session with 2001 Result-Code for AIR → AIA)
- ▯ **ULR Completed** (Update Location Answer with 2001 result)
- ❓ **XRES vs RES Match Failure** (Authentication mismatch in Diameter AVPs)

Each rule is coded to match protocol-specific behaviors and include dynamic values (e.g., matched audio URIs or IMSIs).

The Call Summary sheet serves as a compact, actionable index of all sessions. It is particularly useful for QA engineers, testers, and network analysts who want a bird's-eye view of protocol health and call behavior across large packet captures.

5.2.3 Extracting Success Criteria from Call Flow

This step is crucial in deriving meaningful insights from the parsed SIP call flow by evaluating protocol interactions and metadata to assess the outcome of each session. The extract_success_criteria_from_flow() function is responsible for analyzing individual SIP transactions within the "Call Flow" worksheet of the Excel workbook and determining whether a call was successfully established, forwarded, held, resumed, registered, or failed, among other possible statuses. The success criteria are determined per Call-ID, which serves as a session identifier.

Core Processing Logic

The function loops through each row (representing a SIP message) and updates the success_criteria_map, a dictionary grouped by Call ID, containing various boolean flags and extracted values that indicate the outcome of the session. The following logic is applied:

General Session State Flags

- **INVITE 200 OK Received**: When an INVITE request is acknowledged with a 200 OK, the invite200 flag is set.
- **BYE Message Detected**: If a BYE request or response is found, the bye flag is set.
- If both invite200 and bye are True, then the session is marked as "Successfully Connected".

Call Failure Detection

- If the response code is 3xx, 4xx, or 5xx:
    - The cause code is looked up from the cause_code_file.
    - The reason or warning headers are appended, if present.

    - If a CFX error is detected in the request URI, it is mapped and appended.
    - These are combined into a "Call Failed" status.

SRVCC Call Detection

- If the destination (Call To) matches the srvcc_number, it is marked as an "SRVCC Successfully Connected" session.

WiFi Call Detection

- Based on the P-Access Network Info header, if values like IEEE-802, wlan-node, or 3GPP-WLAN are present, the session is flagged as a "WiFi Call".

Authentication Challenge

- If a 401 Unauthorized response is received for a REGISTER request, the session is marked as "Challenged for Authentication".

Registration Success

- If a 200 OK response is received for a REGISTER request and the Expires header is present, the session is marked as "Successfully Registered".

Call Forwarding

- By analyzing the History-Info header:
    - If it includes index=1.1, it extracts and stores the **Forwarded Number**.
    - If it includes index=1, it stores the **Original Number**.
- A response code 181 (Call is Being Forwarded) marks the session as "Call Forwarded".

Conference Call Detection

- Based on the presence of certain identifiers in the Request URI or Call To field (e.g., mmtel, or a match from conference_identifiers), the session is marked as a "Conference Call".
- Associated numbers are extracted from Refer-To and Referred-By headers and stored in a set.

Call Hold and Resume

- If Media Attribute contains sendonly or recvonly, the call is placed on hold.
- If sendrecv is observed after a hold state, it is flagged as "Call Resumed".

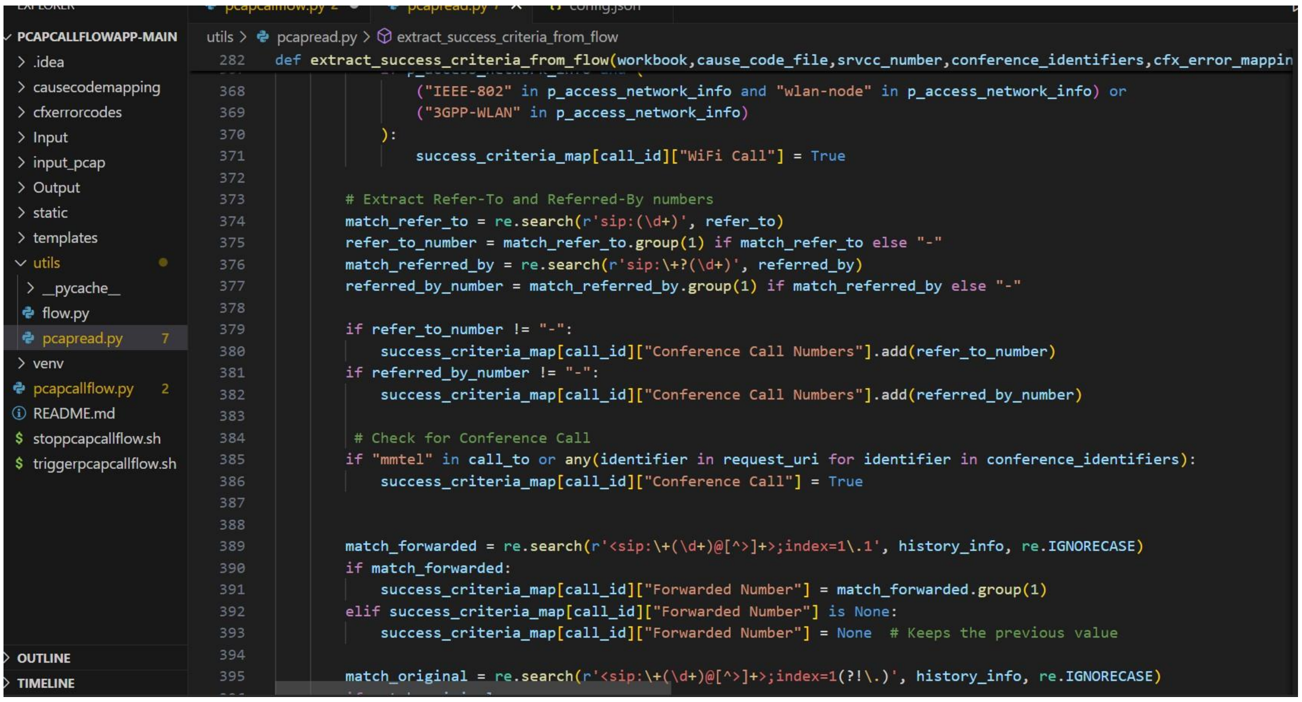
MRF (Media Resource Function) Announcement

- If the Call From or Call To includes msml, or the Request URI indicates an INVITE to annc@, the session is tagged as "MRF Announcement".

- Additional media information such as Audio URI or a play=file:// value is extracted and attached for more detailed reporting.

Video Call Detection

- If the Media Type column includes video, the call is marked as a "Video Call".



```
def extract_success_criteria_from_flow(workbook,cause_code_file,srvcc_number,conference_identifiers,cfx_error_mappin
                ("IEEE-802" in p_access_network_info and "wlan-node" in p_access_network_info) or
                ("3GPP-WLAN" in p_access_network_info)
            ):
                success_criteria_map[call_id]["WiFi Call"] = True

        # Extract Refer-To and Referred-By numbers
        match_refer_to = re.search(r'sip:(\d+)', refer_to)
        refer_to_number = match_refer_to.group(1) if match_refer_to else "-"
        match_referred_by = re.search(r'sip:\+?(\d+)', referred_by)
        referred_by_number = match_referred_by.group(1) if match_referred_by else "-"

        if refer_to_number != "-":
            success_criteria_map[call_id]["Conference Call Numbers"].add(refer_to_number)
        if referred_by_number != "-":
            success_criteria_map[call_id]["Conference Call Numbers"].add(referred_by_number)

         # Check for Conference Call
        if "mmtel" in call_to or any(identifier in request_uri for identifier in conference_identifiers):
            success_criteria_map[call_id]["Conference Call"] = True


        match_forwarded = re.search(r'<sip:\+(\d+)@[^>]+>;index=1\.1', history_info, re.IGNORECASE)
        if match_forwarded:
            success_criteria_map[call_id]["Forwarded Number"] = match_forwarded.group(1)
        elif success_criteria_map[call_id]["Forwarded Number"] is None:
            success_criteria_map[call_id]["Forwarded Number"] = None  # Keeps the previous value

        match_original = re.search(r'<sip:\+(\d+)@[^>]+>;index=1(?!\.)', history_info, re.IGNORECASE)
```

Fig. 8: Code Snippet for the same

5.2.4 Generation of SIP Method Statistics and IP Method Count Tables in the Stats Tab

To enhance the analytical capabilities of FlowSight, detailed SIP method-related statistics are generated and inserted into a dedicated worksheet named **"Stats"** within the Excel output.

First Occurrence Statistics Table

The function add_method_stats_to_stats_tab() generates a table that captures the **first occurrence** of selected SIP methods for a given file. These statistics provide insight into the initiation behavior of key SIP methods, such as INVITE and REGISTER, as well as error messages (e.g., 404, 487, 500).

Key Functionality:

- Extracts the **Call Flow** data into a DataFrame from the corresponding Excel sheet.
- Filters by **filename** if specified, ensuring only relevant rows are used.
- Builds a **table** where:

    - Rows represent **attributes** like SIP Method, Source IP, and Timestamp.
    - Columns represent **SIP Methods**, dynamically determined from both standard and error responses.
- Only the **first occurrence** of each SIP method is recorded based on appearance in the Call Flow tab.

SIP Method Count by Source IP Table

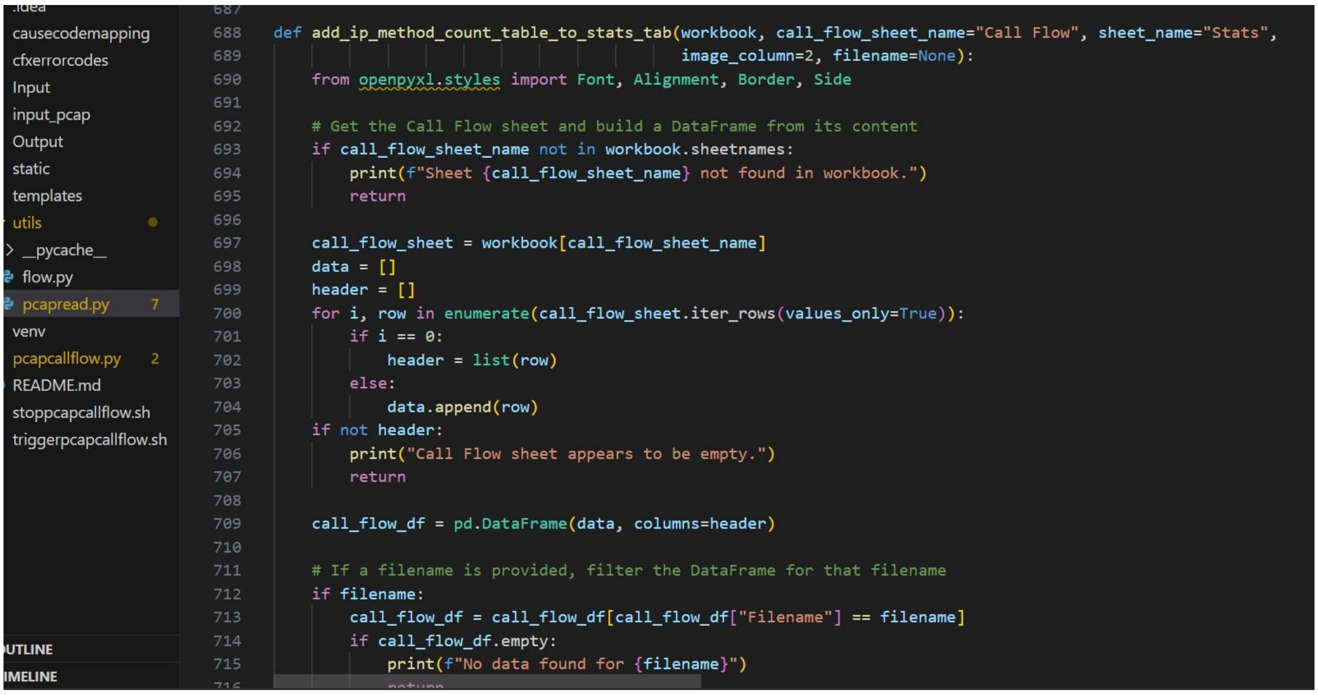
The function add_ip_method_count_table_to_stats_tab() adds a second statistical table that aggregates the **count of SIP messages** per method and per source IP address.

Key Functionality:

- Groups the filtered Call Flow data by Method/Status and Source IP.
- Computes the **total number of occurrences** of each unique (Method, IP) pair.
- Supports error methods from 4xx, 5xx, and 6xx classes (e.g., 487, 500, 603) in addition to core methods like INVITE and REGISTER.
- Sorts entries to ensure numeric error codes are placed before string methods.
- Presents the results in a horizontally expanding table format where each column corresponds to one unique (Method, IP) combination.

These statistics are invaluable in:

- **Debugging SIP call setups and failures** by identifying the source IPs triggering certain methods.
- **Load analysis**, observing which IPs generate the most signaling traffic.
- Highlighting **error conditions**, such as repeated 404s or 487s from specific network nodes.



```
def add_ip_method_count_table_to_stats_tab(workbook, call_flow_sheet_name="Call Flow", sheet_name="Stats",
                                           image_column=2, filename=None):
    from openpyxl.styles import Font, Alignment, Border, Side

    # Get the Call Flow sheet and build a DataFrame from its content
    if call_flow_sheet_name not in workbook.sheetnames:
        print(f"Sheet {call_flow_sheet_name} not found in workbook.")
        return

    call_flow_sheet = workbook[call_flow_sheet_name]
    data = []
    header = []
    for i, row in enumerate(call_flow_sheet.iter_rows(values_only=True)):
        if i == 0:
            header = list(row)
        else:
            data.append(row)
    if not header:
        print("Call Flow sheet appears to be empty.")
        return

    call_flow_df = pd.DataFrame(data, columns=header)

    # If a filename is provided, filter the DataFrame for that filename
    if filename:
        call_flow_df = call_flow_df[call_flow_df["Filename"] == filename]
        if call_flow_df.empty:
            print(f"No data found for {filename}")
```

Fig. 9: Code Snippet for the same

5.2.5 SIP Flow Diagram Generation and Visualization Tab:

In this phase, the tool creates SIP call flow diagrams that visually represent the communication between different IP addresses involved in SIP transactions. This provides a quick and intuitive understanding of call sequences, signaling directions, and protocol interactions, especially useful during troubleshooting or analysis of call behavior.

To translate SIP call flows captured in the Excel "Call Flow" tab into visual diagrams that highlight:

- Source and destination IPs
- SIP methods or response codes exchanged
- Call participants (e.g., Call From, Call To)
- Color-coded indicators for specific message types and errors

Implementation Details

- **Data Extraction**:
  The tool reads the "Call Flow" sheet from the specified Excel file using the pandas library.
- **IP Ordering and Mapping**:
  Unique IP addresses involved in the communication are collected to determine their sequence on the horizontal axis. The last octet (IPv4) or the last segment (IPv6) of each IP is extracted for cleaner display.

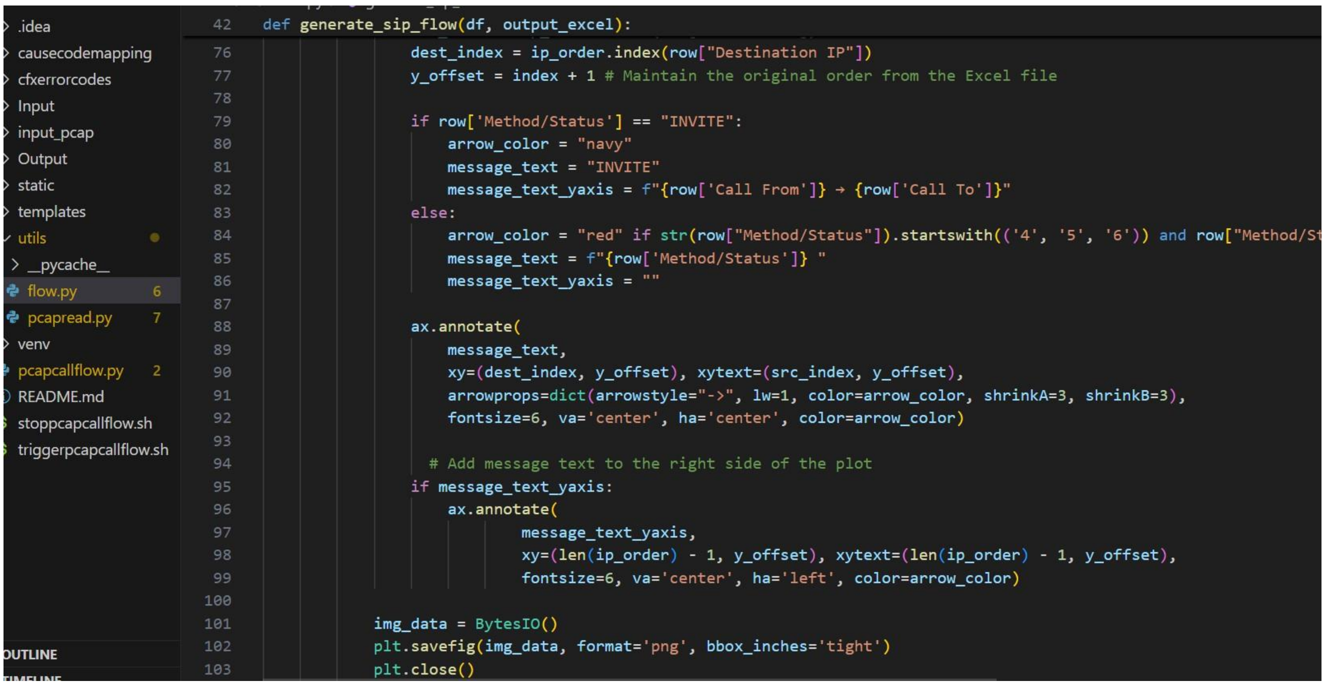
- **Method-Based Color and Annotation Logic**:
  SIP methods are plotted using arrows to represent direction (source → destination). INVITE requests are colored navy, while other methods or responses use colors determined by the associated Call ID. Errors (e.g., 4xx, 5xx, 6xx responses) are highlighted in red for immediate visibility.

- **Matplotlib Plotting**:
  Using matplotlib, each SIP message is visualized as a horizontal arrow. Additional annotations for Call From and Call To are rendered beside INVITE messages for clarity.

- **IP Mapping Row**:
  Below each diagram, an IP mapping row is added, correlating abbreviated IPs with their full addresses for easy reference.

- **Excel Integration**:
  The plotted diagrams are saved into memory and inserted into newly created sheets within the original Excel file using openpyxl. Each diagram is inserted into a dedicated sheet named after the corresponding filename (e.g., trace1 flow), and an IP legend is added below the diagram for reference.

- **Diagram Formatting**:
  The flow diagram tab includes a freeze pane on the first row and uses borders for visual clarity. This makes navigation and analysis more user-friendly.



```python
def generate_sip_flow(df, output_excel):
                dest_index = ip_order.index(row["Destination IP"])
                y_offset = index + 1 # Maintain the original order from the Excel file

                if row['Method/Status'] == "INVITE":
                    arrow_color = "navy"
                    message_text = "INVITE"
                    message_text_yaxis = f"{row['Call From']} → {row['Call To']}"
                else:
                    arrow_color = "red" if str(row["Method/Status"]).startswith(('4', '5', '6')) and row["Method/St
                    message_text = f"{row['Method/Status']} "
                    message_text_yaxis = ""

                ax.annotate(
                    message_text,
                    xy=(dest_index, y_offset), xytext=(src_index, y_offset),
                    arrowprops=dict(arrowstyle="->", lw=1, color=arrow_color, shrinkA=3, shrinkB=3),
                    fontsize=6, va='center', ha='center', color=arrow_color)

                  # Add message text to the right side of the plot
                if message_text_yaxis:
                    ax.annotate(
                            message_text_yaxis,
                            xy=(len(ip_order) - 1, y_offset), xytext=(len(ip_order) - 1, y_offset),
                            fontsize=6, va='center', ha='left', color=arrow_color)

            img_data = BytesIO()
            plt.savefig(img_data, format='png', bbox_inches='tight')
            plt.close()
```

Fig. 10: Code Snippet for the same

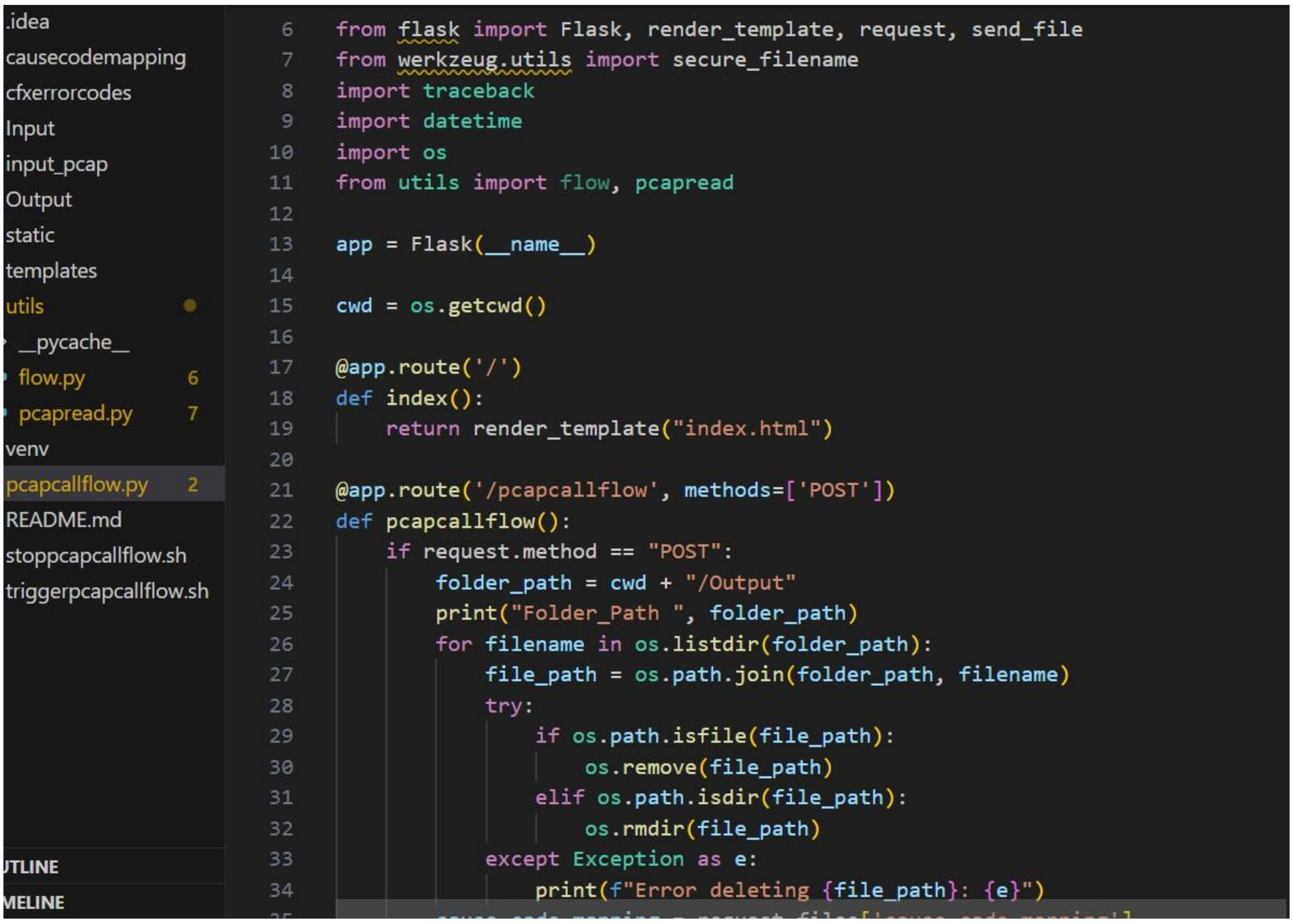
6 RESULTS AND DISCUSSION

6.1Overview of Deployment Modes:

FlowSight is available in two deployment configurations:

- **Cloud-Based Web Application**: Accessible via browser, hosted on a server.
- **Executable Desktop Application**: Standalone .exe file for local environments.

Both versions share the same core processing engine but differ in user interface and deployment architecture.



```
.idea
causecodemapping
cfxerrorcodes
Input
input_pcap
Output
static
templates
utils
  _pycache_
  flow.py          6
  pcapread.py      7
venv
pcapcallflow.py    2
README.md
stoppcapcallflow.sh
triggerpcapcallflow.sh

TLINE
MELINE
```

```python
from flask import Flask, render_template, request, send_file
from werkzeug.utils import secure_filename
import traceback
import datetime
import os
from utils import flow, pcapread

app = Flask(__name__)

cwd = os.getcwd()

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/pcapcallflow', methods=['POST'])
def pcapcallflow():
    if request.method == "POST":
        folder_path = cwd + "/Output"
        print("Folder_Path ", folder_path)
        for filename in os.listdir(folder_path):
            file_path = os.path.join(folder_path, filename)
            try:
                if os.path.isfile(file_path):
                    os.remove(file_path)
                elif os.path.isdir(file_path):
                    os.rmdir(file_path)
            except Exception as e:
                print(f"Error deleting {file_path}: {e}")
```

Fig. 11: Code Snippet of cloud deployment

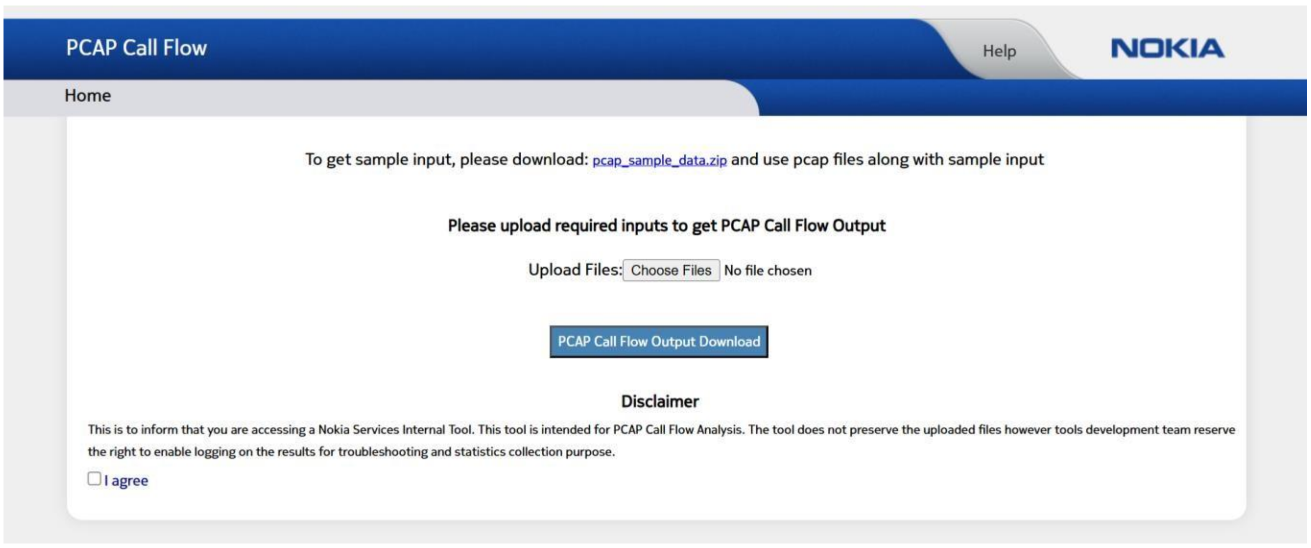
Fig. 24: Pipeline for creating automation scripts from the test case descriptions.

6.2 Cloud Deployment Results

The image below shows the main upload interface of the FlowSight cloud-based application. Users are prompted to upload four essential files:

- A **PCAP file** containing captured network packets
- A **cause code mapping** file
- An **IP mapping** file
- A **config.json** file that holds authentication and processing parameters

The intuitive design ensures that users can quickly provide all necessary inputs in a single step. The backend logic handles validation, processing, and dynamic folder creation. Once uploaded, the tool automatically initiates parsing and analysis.


PCAP Call Flow
Help
NOKIA
Home
Upload Files: Choose Files No file chosen
PCAP Call Flow Output Download
I agree

To get sample input, please download: pcap_sample_data.zip and use pcap files along with sample input

Please upload required inputs to get PCAP Call Flow Output

Disclaimer

This is to inform that you are accessing a Nokia Services Internal Tool. This tool is intended for PCAP Call Flow Analysis. The tool does not preserve the uploaded files however tools development team reserve the right to enable logging on the results for troubleshooting and statistics collection purpose.

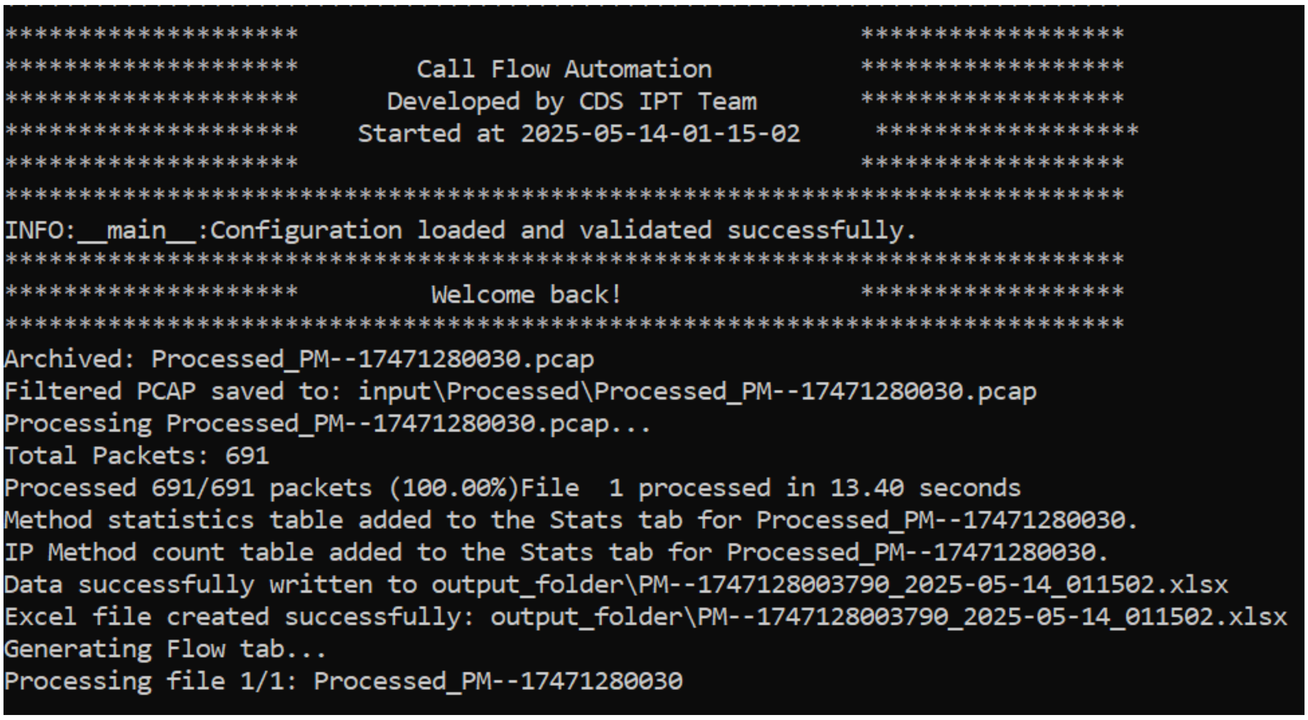
Fig. 12: Deployed on company's internal cloud

6.3 Executable (.exe) Version Results:

The screenshot below illustrates the graphical user interface (GUI) of the standalone .exe version of FlowSight. Users are provided with clearly labeled buttons or fields to select required input files from their local machine:

- **PCAP File** – The primary input containing network capture data
- **Config File** – Contains authentication credentials, mappings, and execution parameters
- **Cause Code Mapping** and **IP Mapping** – Used for decoding and contextualizing network messages

This interface is built to be lightweight and responsive, ensuring ease of use even for non-technical users. File selection is validated before proceeding, reducing input-related errors during execution.



```
*******************                                  ******************
*******************        Call Flow Automation      ******************
*******************      Developed by CDS IPT Team   ******************
*******************    Started at 2025-05-14-01-15-02  ******************
*******************                                  ******************
*************************************************************************
INFO:__main__:Configuration loaded and validated successfully.
*************************************************************************
*******************          Welcome back!           ******************
*************************************************************************
Archived: Processed_PM--17471280030.pcap
Filtered PCAP saved to: input\Processed\Processed_PM--17471280030.pcap
Processing Processed_PM--17471280030.pcap...
Total Packets: 691
Processed 691/691 packets (100.00%)File  1 processed in 13.40 seconds
Method statistics table added to the Stats tab for Processed_PM--17471280030.
IP Method count table added to the Stats tab for Processed_PM--17471280030.
Data successfully written to output_folder\PM--1747128003790_2025-05-14_011502.xlsx
Excel file created successfully: output_folder\PM--1747128003790_2025-05-14_011502.xlsx
Generating Flow tab...
Processing file 1/1: Processed_PM--17471280030
```

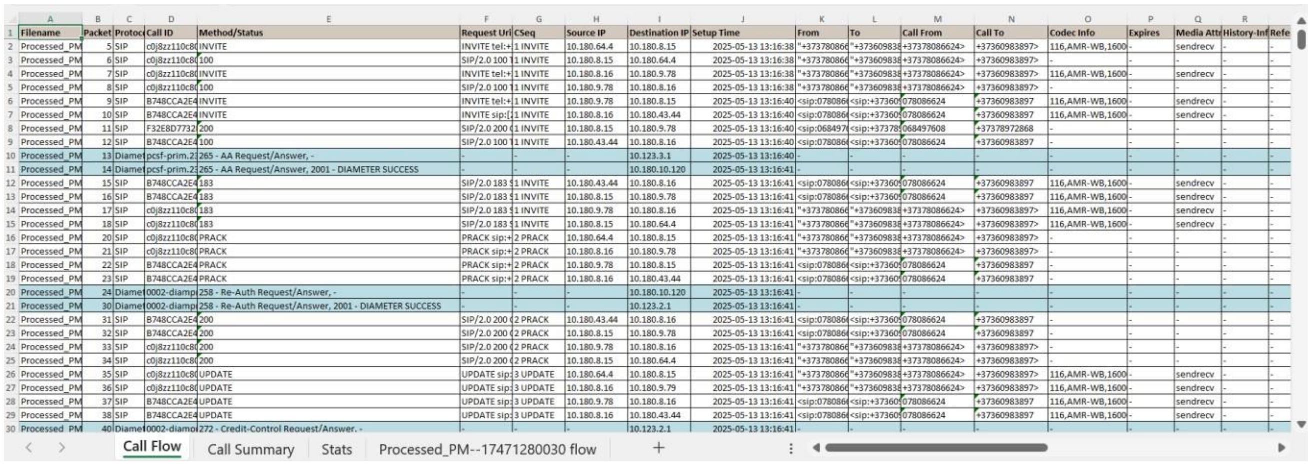
Fig. 13: Executable Version

6.4 Generated Excel Output:

The Excel file contains multiple well-organized tabs, each designed to capture a specific aspect of the analyzed session:

- **Call Flow:** A detailed packet-by-packet view of the communication, including timestamp, protocol, source/destination, and decoded message.
- **Summary:** A high-level overview grouped by Call-ID or Identifier, showing whether specific success criteria (e.g., Registration, Authentication, SRVCC) were met.
- **Diagram:** A graphical representation of the call signaling between endpoints, aiding quick visual analysis of call flow direction and methods.
- **Stats:** Aggregated success metrics and graphical charts for trend analysis across multiple sessions.

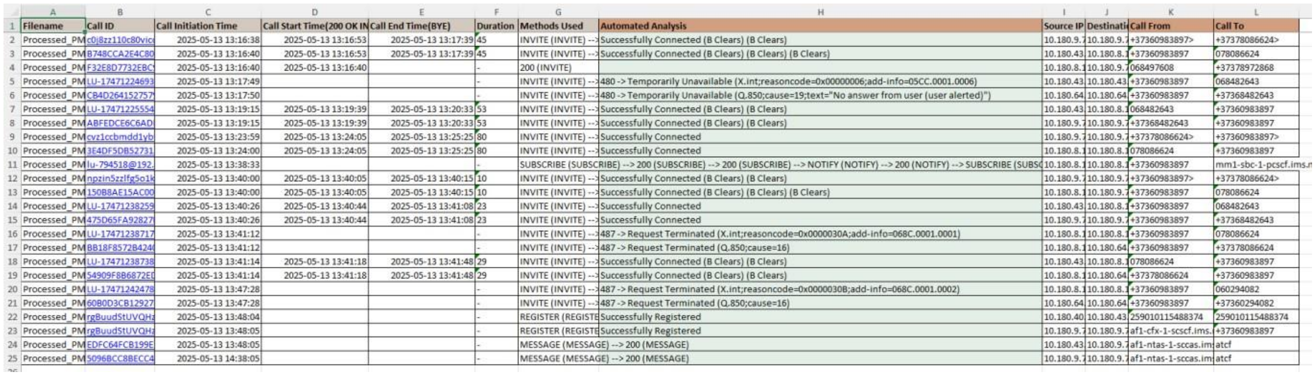
This consistency across both deployment modes ensures that users can transition between local and cloud platforms without adjusting their analysis or workflows.



Filename	Packet	Protocc	Call ID	Method/Status	Request Uri	CSeq	Source IP	Destination IP	Setup Time	From	To	Call From	Call To	Codec Info	Expires	Media Attr	History-Inf	Refe
Processed_PM	5	SIP	c0j8zz110c80	INVITE	INVITE tel:+	1 INVITE	10.180.64.4	10.180.8.15	2025-05-13 13:16:38	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	6	SIP	c0j8zz110c80	100	SIP/2.0 100 T	1 INVITE	10.180.8.15	10.180.64.4	2025-05-13 13:16:38	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	7	SIP	c0j8zz110c80	INVITE	INVITE tel:+	1 INVITE	10.180.8.16	10.180.9.78	2025-05-13 13:16:38	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	8	SIP	c0j8zz110c80	100	SIP/2.0 100 T	1 INVITE	10.180.9.78	10.180.8.16	2025-05-13 13:16:38	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	9	SIP	B748CCA2E4	INVITE	INVITE tel:+	1 INVITE	10.180.9.78	10.180.8.15	2025-05-13 13:16:40	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	10	SIP	B748CCA2E4	INVITE	INVITE sip:[	1 INVITE	10.180.8.16	10.180.43.44	2025-05-13 13:16:40	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	11	SIP	F32E8D7732	200	SIP/2.0 200 (	1 INVITE	10.180.8.15	10.180.9.78	2025-05-13 13:16:40	<sip:068497	<sip:+37378	068497608	+37378972868	-	-	-	-	-
Processed_PM	12	SIP	B748CCA2E4	100	SIP/2.0 100 T	1 INVITE	10.180.43.44	10.180.8.16	2025-05-13 13:16:40	<sip:078086	<sip:+37360	078086624	+37360983897	-	-	-	-	-
Processed_PM	13	Diamet	pcsf-prim.2	265 - AA Request/Answer, -	-	-	-	10.123.3.1	2025-05-13 13:16:40	-	-	-	-	-	-	-	-	-
Processed_PM	14	Diamet	pcsf-prim.2	265 - AA Request/Answer, 2001 - DIAMETER SUCCESS	-	-	-	10.180.10.120	2025-05-13 13:16:41	-	-	-	-	-	-	-	-	-
Processed_PM	15	SIP	B748CCA2E4	183	SIP/2.0 183 S	1 INVITE	10.180.43.44	10.180.8.16	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	16	SIP	B748CCA2E4	183	SIP/2.0 183 S	1 INVITE	10.180.8.15	10.180.9.78	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	17	SIP	c0j8zz110c80	183	SIP/2.0 183 S	1 INVITE	10.180.9.78	10.180.8.16	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	18	SIP	c0j8zz110c80	183	SIP/2.0 183 S	1 INVITE	10.180.8.15	10.180.64.4	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	20	SIP	c0j8zz110c80	PRACK	PRACK sip:+	2 PRACK	10.180.64.4	10.180.8.15	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	21	SIP	c0j8zz110c80	PRACK	PRACK sip:+	2 PRACK	10.180.8.16	10.180.9.78	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	22	SIP	B748CCA2E4	PRACK	PRACK sip:+	2 PRACK	10.180.9.78	10.180.8.15	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	-	-	-	-	-
Processed_PM	23	SIP	B748CCA2E4	PRACK	PRACK sip:+	2 PRACK	10.180.8.16	10.180.43.44	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	-	-	-	-	-
Processed_PM	24	Diamet	0002-diamp	258 - Re-Auth Request/Answer, -	-	-	-	10.180.10.120	2025-05-13 13:16:41	-	-	-	-	-	-	-	-	-
Processed_PM	30	Diamet	0002-diamp	258 - Re-Auth Request/Answer, 2001 - DIAMETER SUCCESS	-	-	-	10.123.2.1	2025-05-13 13:16:41	-	-	-	-	-	-	-	-	-
Processed_PM	31	SIP	B748CCA2E4	200	SIP/2.0 200 (	2 PRACK	10.180.43.44	10.180.8.16	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	-	-	-	-	-
Processed_PM	32	SIP	B748CCA2E4	200	SIP/2.0 200 (	2 PRACK	10.180.8.15	10.180.9.78	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	-	-	-	-	-
Processed_PM	33	SIP	c0j8zz110c80	200	SIP/2.0 200 (	2 PRACK	10.180.9.78	10.180.8.16	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	34	SIP	c0j8zz110c80	200	SIP/2.0 200 (	2 PRACK	10.180.8.15	10.180.64.4	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	-	-	-	-	-
Processed_PM	35	SIP	c0j8zz110c80	UPDATE	UPDATE sip:	3 UPDATE	10.180.64.4	10.180.8.15	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	36	SIP	c0j8zz110c80	UPDATE	UPDATE sip:	3 UPDATE	10.180.8.16	10.180.9.79	2025-05-13 13:16:41	"+373780866	"+373609838	+37378086624>	+37360983897>	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	37	SIP	B748CCA2E4	UPDATE	UPDATE sip:	3 UPDATE	10.180.9.78	10.180.8.16	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	38	SIP	B748CCA2E4	UPDATE	UPDATE sip:	3 UPDATE	10.180.8.16	10.180.43.44	2025-05-13 13:16:41	<sip:078086	<sip:+37360	078086624	+37360983897	116,AMR-WB,1600	-	sendrecv	-	-
Processed_PM	40	Diamet	0002-diamp	272 - Credit-Control Request/Answer, -	-	-	-	10.123.2.1	2025-05-13 13:16:41	-	-	-	-	-	-	-	-	-

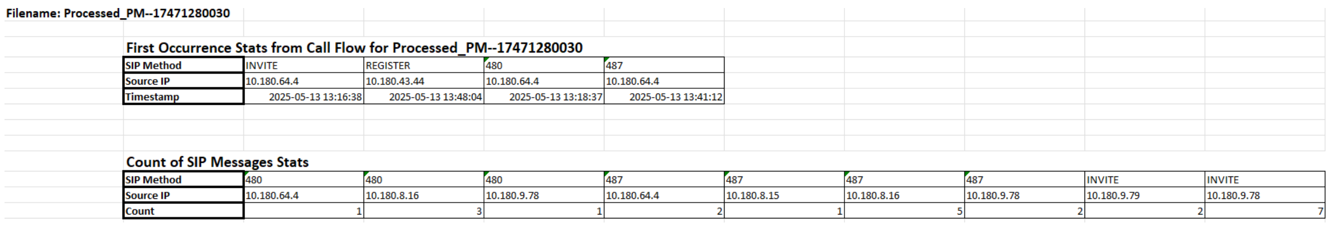
Call Flow | Call Summary | Stats | Processed_PM--17471280030 flow | +

Fig. 14: Sample Call Flow Report



Filename	Call ID	Call Initiation Time	Call Start Time(200 OK IN	Call End Time(BYE)	Duration	Methods Used	Automated Analysis	Source IP	Destinati	Call From	Call To
Processed_PM	c0j8zz110c80vic	2025-05-13 13:16:38	2025-05-13 13:16:53	2025-05-13 13:17:39	45	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.9.7	10.180.9.7	+37360983897>	+37378086624>
Processed_PM	B748CCA2E4C80	2025-05-13 13:16:40	2025-05-13 13:16:53	2025-05-13 13:17:39	45	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears) (B Clears)	10.180.43.	10.180.8.1	+37360983897	078086624
Processed_PM	F32E8D7732EBC	2025-05-13 13:16:40	2025-05-13 13:16:40		-	200 (INVITE)		10.180.8.1	10.180.9.7	068497608	+37378972868
Processed_PM	LU-17471224693	2025-05-13 13:17:49			-	INVITE (INVITE) -->	480 -> Temporarily Unavailable (X.int;reasoncode=0x00000006;add-info=05CC.0001.0006)	10.180.43.	10.180.43.	+37360983897	068482643
Processed_PM	CB4D264152757	2025-05-13 13:17:50			-	INVITE (INVITE) -->	480 -> Temporarily Unavailable (Q.850;cause=19;text="No answer from user (user alerted)")	10.180.64.	10.180.64.	+37360983897	+37368482643
Processed_PM	LU-17471225554	2025-05-13 13:19:15	2025-05-13 13:19:39	2025-05-13 13:20:33	53	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.43.	10.180.8.1	068482643	+37360983897
Processed_PM	ABFEDCE6C6AD	2025-05-13 13:19:15	2025-05-13 13:19:39	2025-05-13 13:20:33	53	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.9.7	10.180.9.7	+37368482643	+37360983897
Processed_PM	cvz1ccbmdd1yb	2025-05-13 13:23:59	2025-05-13 13:24:05	2025-05-13 13:25:25	80	INVITE (INVITE) -->	Successfully Connected	10.180.9.7	10.180.9.7	+37378086624>	+37360983897>
Processed_PM	3E4DF5DB52731	2025-05-13 13:24:00	2025-05-13 13:24:05	2025-05-13 13:25:25	80	INVITE (INVITE) -->	Successfully Connected	10.180.8.1	10.180.8.1	078086624	+37360983897
Processed_PM	lu-794518@192.	2025-05-13 13:38:33			-	SUBSCRIBE (SUBSCRIBE) --> 200 (SUBSCRIBE) --> 200 (SUBSCRIBE) --> NOTIFY (NOTIFY) --> 200 (NOTIFY) --> SUBSCRIBE (SUBSC		10.180.8.1	10.180.8.1	+37360983897	mm1-sbc-1-pcscf.ims.
Processed_PM	npzin5zzlfg5o1k	2025-05-13 13:40:00	2025-05-13 13:40:05	2025-05-13 13:40:15	10	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.9.7	10.180.9.7	+37360983897>	+37378086624>
Processed_PM	150B8AE15AC00	2025-05-13 13:40:00	2025-05-13 13:40:05	2025-05-13 13:40:15	10	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears) (B Clears)	10.180.8.1	10.180.9.7	+37360983897	078086624
Processed_PM	LU-17471238259	2025-05-13 13:40:26	2025-05-13 13:40:44	2025-05-13 13:41:08	23	INVITE (INVITE) -->	Successfully Connected	10.180.43.	10.180.8.1	+37360983897	068482643
Processed_PM	475D65FA92827	2025-05-13 13:40:26	2025-05-13 13:40:44	2025-05-13 13:41:08	23	INVITE (INVITE) -->	Successfully Connected	10.180.9.7	10.180.9.7	+37360983897	+37368482643
Processed_PM	LU-17471238717	2025-05-13 13:41:12			-	INVITE (INVITE) -->	487 -> Request Terminated (X.int;reasoncode=0x0000030A;add-info=068C.0001.0001)	10.180.8.1	10.180.8.1	+37360983897	078086624
Processed_PM	BB18F8572B424	2025-05-13 13:41:12			-	INVITE (INVITE) -->	487 -> Request Terminated (Q.850;cause=16)	10.180.8.1	10.180.64.	+37360983897	+37378086624
Processed_PM	LU-17471238738	2025-05-13 13:41:14	2025-05-13 13:41:18	2025-05-13 13:41:48	29	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.43.	10.180.8.1	078086624	+37360983897
Processed_PM	54909F8B6872ED	2025-05-13 13:41:14	2025-05-13 13:41:18	2025-05-13 13:41:48	29	INVITE (INVITE) -->	Successfully Connected (B Clears) (B Clears)	10.180.8.1	10.180.64.	+37378086624	+37360983897
Processed_PM	LU-17471242478	2025-05-13 13:47:28			-	INVITE (INVITE) -->	487 -> Request Terminated (X.int;reasoncode=0x0000030B;add-info=068C.0001.0002)	10.180.8.1	10.180.8.1	+37360983897	060294082
Processed_PM	60B0D3CB12927	2025-05-13 13:47:28			-	INVITE (INVITE) -->	487 -> Request Terminated (Q.850;cause=16)	10.180.64.	10.180.64.	+37360983897	+37360294082
Processed_PM	rgBuudStUVQHz	2025-05-13 13:48:04			-	REGISTER (REGISTE	Successfully Registered	10.180.40.	10.180.43.	259010115488374	259010115488374
Processed_PM	rgBuudStUVQHz	2025-05-13 13:48:05			-	REGISTER (REGISTE	Successfully Registered	10.180.9.7	10.180.9.7	af1-cfx-1-scscf.ims.	+37360983897
Processed_PM	EDFC64FCB199E	2025-05-13 13:48:05			-	MESSAGE (MESSAGE) --> 200 (MESSAGE)		10.180.43.	10.180.9.7	af1-ntas-1-sccas.im	atcf
Processed_PM	5096BCC8BECC4	2025-05-13 14:38:05			-	MESSAGE (MESSAGE) --> 200 (MESSAGE)		10.180.9.7	10.180.9.7	af1-ntas-1-sccas.im	atcf

Fig. 15: Sample Call Summary Report



Filename: Processed_PM--17471280030

First Occurrence Stats from Call Flow for Processed_PM--17471280030

SIP Method	INVITE	REGISTER	480	487
Source IP	10.180.64.4	10.180.43.44	10.180.64.4	10.180.64.4
Timestamp	2025-05-13 13:16:38	2025-05-13 13:48:04	2025-05-13 13:18:37	2025-05-13 13:41:12

Count of SIP Messages Stats

SIP Method	480	480	480	487	487	487	487	INVITE	INVITE
Source IP	10.180.64.4	10.180.8.16	10.180.9.78	10.180.64.4	10.180.8.15	10.180.8.16	10.180.9.78	10.180.9.79	10.180.9.78
Count	1	3	1	2	1	5	2	2	7

Fig. 16: Sample Stats Tab

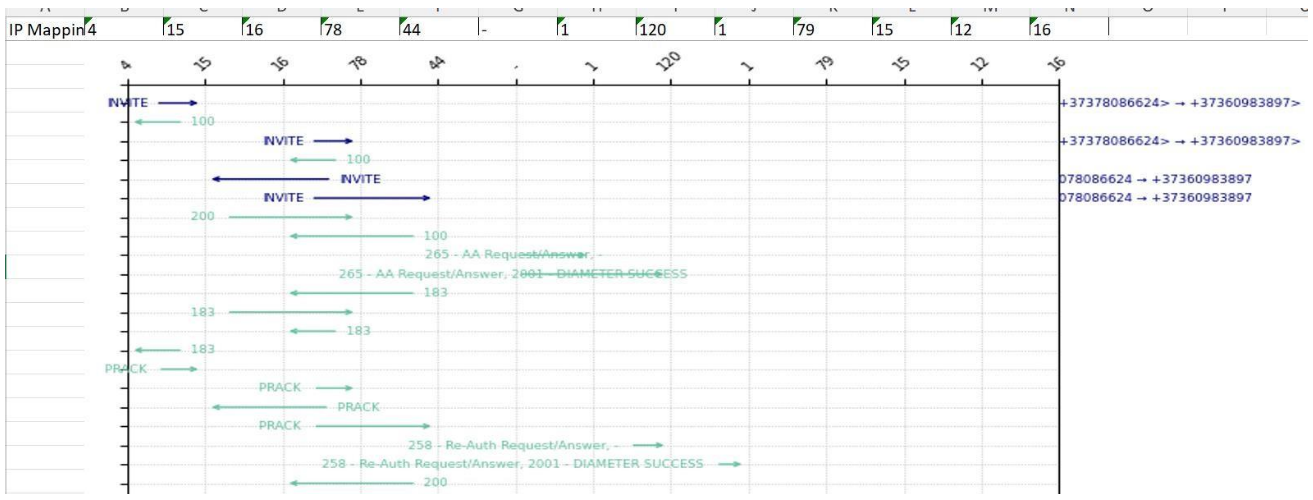


Fig. 17: Sample Flow Diagram Snippet

7 CONCLUSION AND FUTURE PROSPECTS

This report demonstrates the effectiveness of the FlowSight tool in analyzing PCAP (packet capture) files across telecom protocols such as SIP, Diameter, GTPv2, and DNS. FlowSight automates the extraction of session-level insights, call flow reconstruction, success criteria evaluation, and graphical flow visualization, enabling efficient network troubleshooting and protocol behavior validation.

During the development of FlowSight, several challenges were encountered. These included handling large volumes of packet data, ensuring protocol-specific parsing accuracy, maintaining high performance for large PCAP files, and dealing with cases of incomplete or malformed traffic. Additionally, the generation of accurate call flow diagrams and summaries required a consistent logic structure, especially when analyzing multi-protocol sessions.

The system is designed around a modular architecture that allows seamless integration of new protocol handlers, customizable success criteria definitions, and mapping-driven interpretation of field values using external configuration and mapping files. A key advantage of this design is the consistency between the cloud-based and standalone .exe versions, both of which utilize a common backend logic (pcapread.getpcapcallflowoutput()), ensuring a uniform output experience regardless of deployment mode.

FlowSight’s Excel output is designed for clarity and usability. It includes multiple tabs:

- **Call Flow:** Detailed logs of each captured packet with decoded fields.
- **Summary:** Grouped insights for each session or call, including pass/fail logic.
- **Stats:** Visual graphs and frequency charts based on extracted criteria.
- **Diagrams:** Visual representation of signaling paths between nodes.

As a practical tool, FlowSight significantly reduces manual effort in inspecting packet captures, especially in complex telecom environments. It enables testers, network engineers, and QA teams to quickly determine the outcome of call flows, identify failures, and gain visibility into network signaling.

Future Scope

Future enhancements to FlowSight may include:

- **Automated Defect Recognition:** Using heuristics or ML models to detect known failure patterns from packet behavior.
- **Integrated Log + PCAP Analysis:** Merging PCAP data with application or system logs to offer end-to-end visibility.
- **Chat-based Query System:** Allowing users to query sessions using natural language (e.g., "Show all failed calls involving IP X").
- **Anomaly Detection:** Using statistical or AI-based methods to detect outliers or unusual packet flows.
- **Support for Additional Protocols:** Extending current support to protocols such as HTTP/2, QUIC, or custom vendor-specific signaling.
- **Team Collaboration Tools:** Enabling annotation, tagging, and sharing of PCAP analysis within teams for faster debugging cycles.

By continuing to expand its capabilities and integrating with real-world workflows, FlowSight aims to become a comprehensive analysis solution that supports both operational and development teams in improving network reliability and software quality.

REFERENCES

1 Jacobson, V., Leres, C., & McCanne, S. (1989). tcpdump - Network Packet Analyzer. Developed at Lawrence Berkeley National Laboratory. Available at: - https://www.tcpdump.org

2. Wireshark: Retrieved from:- https://www.wireshark.org

3 Orebaugh, A., Ramirez, G., & Beale, J. (2006). https://www.oreilly.com/library/view/wireshark-ethereal/

4 Barford, P., Kline, J., Plonka, D., & Ron, A. (2002). https://dl.acm.org/doi/10.1145/637201.637210

5 Mockapetris, P. (1987). Available at: https://www.rfc-editor.org/rfc/rfc1034.html

6 Mockapetris, P. (1987). Available at-: https://www.rfc-editor.org/rfc/rfc1035.html

7 Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., & Schooler, E. 2002. SIP: Session Initiation Protocol. RFC 3261. IETF. Available - https://www.rfc-editor.org/rfc/rfc3261.html

8 Singh, K., & Schulzrinne, H. 2004. Peer-to-Peer Internet Telephony using SIP Technical Report, Columbia University.

9 Calhoun, P., Loughney , J., Guttman, E., Zorn , G., & Arkko, J. 2003 Diameter-base Protocol. RFC 3588. IETF. Available - https://www.rfc-editor.org/rfc/rfc3588.html

10 Fajardo, V., Arkko, J., Loughney, J., & Zorn, G. (2008). Diameter SIP Application. RFC 4740. IETF. Available - https://www.rfc-editor.org/rfc/rfc4740.html

11 3rd Generation Partnership Project (3GPP). (2023). (E-UTRAN); (S1AP). 3GPP TS 36.413. Available at- https://www.3gpp.org/ftp/Specs/archive/36_series/36.413/

12 3rd Generation Partnership Project (3GPP). 2023. 29.274. https://www.3gpp.org/ftp/Specs/archive/29_series/29.274/

13 3rd Generation Partnership Project (3GPP). -2023 (GTPv1-U). 3GPP TS 29.281 https://www.3gpp.org/ftp/Specs/archive/29_series/29.281/

14 Scapy Library. (n.d.). Packet-Manipulation Tool - https://scapy.net