Comprehensive Guide to Network Intrusion Detection Feature Collection

Jayavardhan Govvada

April 11, 2025

1 Introduction

This document details the methodology for collecting 41 network features essential for intrusion detection systems (IDS). Features are categorized and mapped to specific collection techniques.

2 Feature Taxonomy

2.1 Basic Connection Features (Direct from Packets)

Feature	Protocol Layer	Extraction Method
duration	Transport	TCP session teardown time - handshake time
protocol_type	IP	IP header Protocol field (hex value)
src_bytes	Transport	TCP payload length from source
flag	Transport	Bitwise OR of TCP flags (SYN=0x02, ACK=0x10)

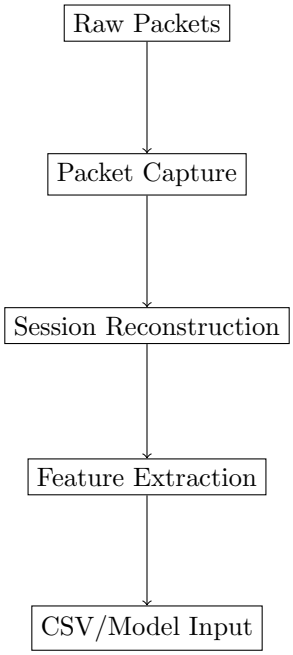
2.2 Time-Window Statistics (2-second Window)

Feature	Calculation
count	$\sum connections$ to same destination in window
srv_count	$\sum connections$ to same service (port)
serror_rate	$\frac{SYN\_errors}{total\_connections}$

2.3 Host Behavior Features

Feature	Data Source
dst_host_count	Flow records aggregated by destination IP
dst_host_srv_serror_rate	SYN errors per service per host

3 Implementation Architecture



4 Detailed Feature Collection Workflow

4.1 Packet Capture Requirements

- Capture full packet payloads (-s 0 in tcpdump)
- Minimum 1Gbps throughput capacity
- Promiscuous mode for network taps

Listing 1: Packet Capture Command

```
tcpdump -i eth0 -s 0 -w full_capture.pcap
```

4.2 Session Reconstruction with Zeek

Listing 2: Zeek Script for Features

```
event connection_state_remove(c: connection) {
    Log::write(IDS_Features, {
        ts = c$start_time,
        duration = c$duration,
        src_bytes = c$orig$num_bytes_ip,
        dst_bytes = c$resp$num_bytes_ip,
        # ... (38 additional features)
    });
}
```

5 Feature Validation Checklist

- Verify numerical ranges:
  - duration: 0-58,000 seconds (16-bit counter rollover)

  - src_bytes: 0-4,294,967,295 (32-bit unsigned)

- Check categorical values:
  - protocol_type: tcp, udp, icmp
  - flag: 10 possible combinations (SF, S0, etc.)

6 Complete Feature Matrix

Feature	Type	Collection Method
num_compromised	count	Correlate with vulnerability scanner results
root_shell	binary	Monitor /bin/bash execution with root EUID
num_file_creations	count	Audit kernel file creation syscalls

7 Feature Extraction Details

7.1 Network Layer Features

7.1.1 A. Protocol Type

- **Source:** IP headers in each packet (TCP, UDP, ICMP)
- **Extraction:** Direct read from network layer packet header

7.1.2 B. Service Type

- **Source:** Transport layer (TCP/UDP) header
- **Extraction:** Port number mapping to services (HTTP, FTP, SMTP)

7.1.3 C. Flag Values

- **Source:** TCP packet headers
- **Extraction:** State indicators (SYN, ACK, FIN)

7.2 Packet Metrics

7.2.1 D. Byte Counts

- **Source:** Packet payload and length fields
- **src_bytes:** Source to destination data volume
- **dst_bytes:** Destination to source data volume

7.2.2 E. Packet Indicators

- **land:** Source/destination address match check
- **wrong_fragment:** Fragmentation field analysis
- **urgent:** TCP urgent pointer field check

7.3 Feature Validation

- Source and destination address verification
- Protocol conformance checking
- Packet integrity validation

8 Data Storage Recommendations

- Short-term: Apache Parquet files (columnar storage)
- Long-term: TimeSeries DB (InfluxDB)
- Real-time: Kafka streams with Spark Structured Streaming

9 Appendix: Full Feature List

1. duration - Connection duration in seconds
2. protocol_type - Transport layer protocol
3. service - Destination service (port number)

...

4. dst_host_srv_rerror_rate - % of REJ errors

1. Traffic Capture and Session Reconstruction Packet Capture: Use tools such as tcpdump, Wireshark, or a dedicated network tap to capture raw packets from your network interface. These tools record packets in formats (like pcap) that you can process later.

Session/Flow Reconstruction: To derive a "connection" or "flow" (a series of packets between two endpoints), you must reassemble these packets. Tools like Zeek (formerly Bro), Suricata, or specialized libraries (e.g., in Python: Scapy) can help reconstruct sessions based on the tuple (source IP, source port, destination IP, destination port, protocol). This reconstruction is the foundation for most of your features.

2. Direct Feature Extraction Some features can be computed directly from the metadata of a single connection:

Duration:

What: The time difference between the first and the last packet of the connection.

How: Store the timestamp of the initial packet and subtract it from the timestamp of the final packet for that session.

Protocol Type:

What: The protocol used in the connection (e.g., TCP, UDP, ICMP).

How: Extract this directly from the IP header in each packet.

Service:

What: The type of network service (e.g., http, ftp, smtp).

How: Derive it from the destination port and/or deep packet inspection if necessary. Maintain a lookup table (often using IANA port registries or protocol-specific libraries) to map ports to service names.

Flag:

What: TCP flags (e.g., SYN, ACK, FIN, etc.) that describe the state of the connection.

How: Extract these from the TCP header of packets during the session.

$\mathrm{src}_bytesanddst_bytes$:

What: The number of data bytes sent from the source to the destination and vice versa.

How: Sum the payload sizes (excluding headers) for the packets in each direction.

Indicators like 'land': $\mathrm{'wrong}_fragment', 'urgent'$:

What: These are either binary indicators or counters. For example, 'land' indicates if the source and destination addresses/ports are identical, $\mathrm{'wrong}_fragment' canindicatemalformedpackets, and 'urgent' countsthe$

How: Check corresponding fields and conditions in packet headers as the session is reconstructed.

3. Aggregated and Stateful Feature Calculations Many features (especially the ones related to counts and rates) are computed over a window of connections rather than a single connection:

Rate and Count Features: Examples include $\mathrm{count}, \mathrm{srv}_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate,$

Maintain a Historical Buffer: For each new connection, you might want to maintain a rolling window of previous connections (e.g., within the last 2 seconds or the last N connections) to calculate these counts and rates.

Implement Windowed Aggregations: Use programming constructs (or streaming analytics frameworks like Apache Flink, Spark Streaming, or custom Python scripts) to compute:

Count Features: Total number of connections from the source IP or to the same service, etc.

Rates: The fraction of connections exhibiting error flags or other unusual behavior over the total connections in the window.

Examples:

$\text{same}_s rv_r ate$: $Ratio of connections that use the same service as the current connection.$

$\text{diff}_s rv_r ate$: $Ratio of connections with a dif ferent service.$

Connection Behavioral Features: Features like $\text{num}_f ailed_l ogins, logged_i n, num_c ompromised, root_s hell, su_a ttempte$

Deep Packet Inspection: To detect login attempts, failed authentications, or commands, you may need to parse application-layer data (e.g., SSH or Telnet protocol details).

Protocol-Specific Parsers: Use IDS frameworks (e.g., Zeek, Suricata) that come with built-in signatures and scripts to detect such events.

Event Correlation: Track session state and correlate multiple packets/events to decide if a login was attempted or succeeded, tallying the count accordingly.

4. Implementing the Feature Extraction Pipeline Automation with Scripting or Stream Processing: Develop scripts (using Python, for example) to automate the above steps. A common approach might involve:

Packet Capture: Using a library like pyshark (a Python wrapper for tshark) to read packet captures in real time.

Session Tracking: Implement logic that aggregates packets into sessions.

Feature Computation: For each session, compute the direct features and update rolling aggregates for the statistical features.

A pseudocode outline could be:

python Copy Edit for packet in $\text{packet}_s tream$: $session_i d = get_s ession_i d(packet) e.g., based on 5-tuple(srcIP, srcport, dstIP, dstport, protocol) update_s ession(session_i d, packet)$

if $\text{is}_s ession_c omplete(session_i d)$: $features = features['duration'] = session[session_i d].end_t ime - session[session_i d].start_t ime features['protocol_t ype'] = packet.protocol features['service'] = map_p ort_t o_s ervice(packe extract_t cp_f lags(session[session_i d]) features['src_b ytes'] = sum(packet.src_b ytes for packet in session[session_i d]) featu sum(packet.dst_b ytes for packet in session[session_i d]) ... compute other direct features$

For windowed features: features['count'] = $\text{count}_c onnections(current_w indow, src = packet.src_i p) features['same_s compute_s ame_s rv_r ate(current_w indow, current_c onnection = features) ... compute additional aggregate features$

Now send features to your IDS model $\text{process}_f eatures(features) Optionally, clear state for session_i dor maintain for Consider leveraging open-source IDS frameworks like Zeek or Suricata. They have rich ecosystems and many built-in scripts or plugins to extract and log these kinds of features directly. From the resulting logs, you could further process to e$

5. Practical Considerations Performance: Real-time traffic processing at high network speeds may require a highly optimized implementation, possibly in a compiled language (like C/C++) or using specialized network hardware, coupled with parallel processing or hardware acceleration.

Memory and State Management: Proper handling of state (for rolling windows, aggregations, etc.) is crucial. You might need caching mechanisms or in-memory databases to track connection history and rapidly update statistical features.

Time Synchronization: Accurate timestamping is vital. Ensure that your packet capture tool and processing pipeline are synchronized (e.g., using NTP) to compute durations and rates correctly.

Handling Encrypted Traffic: With increasing use of encryption, not all features can be directly extracted (e.g., deep packet inspection for failed logins). In such cases, metadata and flow information (packet sizes, timing, etc.) become even more critical.

Summary In practice, you first capture network packets and reconstruct sessions using tools like tcpdump, Wireshark, or Zeek. Then, you extract direct features (such as protocol type, duration, src/dst bytes, TCP flags) from the packet headers. More complex features—especially those based on historical aggregation (like connection counts, error rates, and service rate ratios)—require maintaining state over a sliding window of past connections. This is done by setting up an automated pipeline (using scripting or specialized stream processing frameworks) that processes

each session in real time and computes the required 41 features. Finally, many open-source IDS frameworks have already solved portions of this problem, and integrating or adapting them can save you significant development time.

When building an intrusion detection system, most of these features are derived from analyzing individual network packets—but many require additional processing and aggregation over multiple packets in a connection. Here's a breakdown of where each type of feature comes from:

1. Direct Packet Attributes These features are directly available from the headers or metadata of individual network packets:

Protocol Type

Source: IP headers in each packet (e.g., TCP, UDP, ICMP).

Explanation: The protocol information is set by the network layer and can be read directly from the packet's header.

Service

Source: Often inferred from the destination port number found in the transport layer (TCP/UDP) header.

Explanation: By mapping common port numbers to services (HTTP, FTP, SMTP, etc.), the service type can be identified.

Flags

Source: TCP header of each packet.

Explanation: TCP flags (such as SYN, ACK, FIN) are contained within the header and indicate the state or type of the connection.

Bytes Count ($\text{src}_bytesand\text{dst}_bytes$)

Source: Payload size and overall packet length from each packet.

Explanation: By summing up the data carried in packets, you get the total bytes sent from source to destination and vice versa.

Indicators like 'land', '$\text{wrong}_fragment', and' urgent'$

Source: They are usually derived directly by checking header fields (e.g., verifying if source and destination IP/port are the same, or looking at the fragmentation and urgent pointer fields in packets).

Explanation: These are binary or counter-based checks that you can perform on each packet or during packet reassembly.

2. Aggregated or Session-Level Features While individual packets provide the basic building blocks, many of your features are computed by analyzing a complete connection or flow, which includes multiple packets:

Duration

Source: The timestamps of the first and last packet in a connection.

Explanation: By recording when a connection starts and ends, you can calculate the connection duration.

Count-based Features (e.g., $\text{num}_failed_logins, count, srv_count, etc.$)

Source: Aggregated logs or flow data generated by network monitoring tools over a specified time window or session.

Explanation: These features are not present in any single packet. They are computed by accumulating packet or session information over time. For example, a tool may maintain a buffer of recent connections to compute error rates or connection counts.

Behavioral and Application-Level Features (e.g., $\text{logged}_in, root_shell, su_attempted$)

Source: Information extracted using deep packet inspection (DPI) or application layer logs.

Explanation: These require analyzing the content beyond the headers, such as login attempts or command executions, which are often present in higher-level protocols (SSH, Telnet, etc.) and might be found in the payload rather than the header.

Rate Features (e.g., $\text{serror}_rate, srv_serror_rate, same_srv_rate, etc.$)

Source: Historical traffic data maintained either by network monitoring tools or custom logging solutions.

Explanation: These features are derived by calculating the ratio of specific types of events (for example, error responses) to the total number of events over a connection window or a sliding time frame.

3. Typical Workflow for Feature Extraction Packet Capture:

Tools such as tcpdump or Wireshark capture raw packets on the network.

These packets contain both header information and payload, offering raw data from which many features are extracted.

Session Reconstruction:

You reassemble packets into a connection or flow (using tools like Zeek, Suricata, or Scapy).

This process allows for the calculation of session-level features (duration, aggregates, etc.).

Feature Computation:

Direct Extraction: Using header fields for attributes such as protocol type, flags, and byte counts.

Aggregation: Using a time window or session logs to compute counts, rates, or behavioral features.

Deep Analysis: Examining payload data if needed (for example, to detect failed login patterns or application-specific events).

Conclusion Direct Extraction: Yes, many features such as protocol type, service, flags, and byte counts come directly from individual network packet headers or payload information.

Derived Features: However, more complex or context-aware features (like duration, rate calculations, and behavioral indicators) require aggregating data across multiple packets or sessions. You may also need to combine metadata with deeper inspection of packet contents when available.

In summary, while the raw data comes from network packets, you must perform additional processing—reassembling flows, aggregating statistics, and sometimes inspecting application-layer information—to fully extract all 41 features for your intrusion detection system.

You can capture and observe network packets—and therefore the connections and their behavior—by using packet capture and network analysis tools. Below are some practical methods and tools to get you started:

1. Packet Capture Tools These tools tap directly into your network interface (usually operating in promiscuous mode) and allow you to capture live traffic:

tcpdump:

What It Is: A command-line packet analyzer that can capture and display packet headers (or entire packets) in real time.

How It Works: Run tcpdump on your network interface (e.g., tcpdump -i eth0) to record packets. You can also apply filters (using BPF syntax) to capture only specific traffic such as TCP, UDP, or traffic from a particular IP address.

Example Command:

bash Copy Edit sudo tcpdump -i eth0 -w capture.pcap The -w option writes the packets to a file (here, capture.pcap) for later analysis.

Wireshark:

What It Is: A graphical network protocol analyzer that lets you capture and interactively browse the traffic running on a network.

How It Works: Launch Wireshark, select the network interface you want to monitor, and start capturing. Its powerful filtering and visualization options let you drill down into individual packets or flows to see headers, payloads, and protocol details.

Features: Real-time packet capturing, filtering based on protocols or IP addresses, and packet detail inspection.

Other Tools:

Zeek (formerly Bro): More than just a packet capture tool, Zeek can process network traffic to provide high-level logs and extract behavioral features (such as connection durations and counts) automatically.

Suricata: An IDS/IPS with built-in packet capture and analysis capabilities, often used in security monitoring.

2. Observing Connection Behavior Observing how connections behave typically involves capturing entire flows or sessions rather than just individual packets. Here's how you can do that:

Session Reconstruction:

Flow Creation: Tools like Wireshark and Zeek will group packets into flows (sessions) based on key identifiers such as source IP, destination IP, source/destination ports, and protocol type.

Behavior Analysis: Once grouped, you can analyze the flow for its duration, the number and type of packets exchanged, byte counts, and protocol flags that indicate connection behavior (for example, TCP flags showing the state of the connection).

Deep Packet Inspection (DPI):

This involves examining not only the header fields but also the packet payloads. It's useful for recognizing application layer events (like login failures or protocol-specific behavior). Tools like Wireshark allow you to inspect payload data, while Zeek's scripts may log higher-level events automatically.

Historical Data for Aggregation:

To observe behavior over time (e.g., computing rates like error rates or counting the number of connections), you can store captured data and aggregate it using custom scripts or a network monitoring framework.

Tools such as ELK stack (Elasticsearch, Logstash, Kibana) or Splunk are often used to analyze logs generated from packet captures or network sensors.

3. Practical Steps to Start Capturing and Analyzing Identify the Network Interface:

Determine which network interface you want to capture traffic on. On Linux, you might use ifconfig or ip addr to list interfaces.

Capture Packets:

Using tcpdump:

bash Copy Edit sudo tcpdump -i ¡interface¿ -w capture.pcap Replace ¡interface¿ with your network interface name (e.g., eth0 or wlan0).

Using Wireshark:

Open the application.

Select the desired interface.

Click "Start" to begin the capture session.

Analyze the Capture:

Open the saved capture file in Wireshark, where you can filter and inspect packet streams.

Look for sessions or flows by applying filters (like filtering TCP streams or specific protocol types).

Use Wireshark's "Follow TCP Stream" option to view the sequence of packets that make up an entire connection.

Extract Features or Log Data:

Use tools like Zeek or Suricata that automatically parse network flows and export logs with detailed features that you can then feed into your intrusion detection system.

Alternatively, write your own scripts (using libraries such as Scapy in Python) to parse the pcap file, reconstruct sessions, and compute features like durations or byte counts.

Summary Capturing Packets: You can obtain network packets directly from your network interface using tools like tcpdump and Wireshark.

Reconstructing Sessions: Once captured, you reconstruct connections (or flows) from these packets to observe the behavior of each connection.

Observation and Analysis: Packet analyzers (Wireshark for manual inspection or Zeek/Suricata for automated analysis) let you examine both individual packet details and aggregated session behavior.

By starting with these tools and steps, you can get a comprehensive view of your network's traffic, extract detailed features from both individual packets and reconstructed flows, and ultimately build a robust input pipeline for your intrusion detection system.

In practical, real-time network monitoring, a "connection" generally refers to a communication session between two endpoints—typically characterized as a network "flow." This flow is made up of multiple packets exchanged between the source and the destination. Here's a deeper look at what that means:

1. Definition of a Connection/Flow 5-Tuple Identification: A connection is usually identified by five key values:

Source IP address

Source port

Destination IP address

Destination port

Protocol (e.g., TCP, UDP, ICMP) This combination is often called the 5-tuple. Packets sharing the same 5-tuple are considered part of the same connection.

Session Establishment: For protocols like TCP, a connection starts with a handshake (e.g., the TCP three-way handshake: SYN, SYN-ACK, ACK) and continues with the exchange of data packets until one side terminates the session. In UDP communications, while there is no handshake, packets that share the same 5-tuple within a certain time window are still treated as part of a logical connection.

2. Practical Examples of Connections Web Browsing: When you load a webpage, your browser establishes one or more TCP connections with a web server. The packets involved in setting up (handshake), transferring HTTP data, and eventually closing the connection are all grouped into one logical connection.

Email Communication: Similar to web browsing, when your email client interacts with an SMTP or IMAP server, the series of packets related to sending or retrieving emails form a distinct connection.

Streaming or VoIP: In cases like streaming or Voice over IP (VoIP), the connection might involve numerous packets that represent an ongoing media stream, and even though there may not be a formal handshake, the packets are still grouped logically based on IP addresses, ports, and the protocol.

3. Observing and Analyzing Connections in Real Time Packet Capture and Analysis Tools: Using tools like Wireshark or tcpdump, you can watch these connections in real time.

Wireshark: Lets you filter and follow a specific TCP stream to see all the packets that belong to a connection.

tcpdump: Captures packets on an interface and, with appropriate filters, you can isolate packets that are part of a given connection.

Behavioral Analysis: In an intrusion detection system (IDS), the observed connection is analyzed not just by the individual packets, but also by the relationship between them. This includes:

Connection Duration: How long the flow exists.

Packet Count: The number of packets and their sizes in each direction.

Protocol Flags and Anomalies: For example, abnormal patterns in TCP flags (e.g., multiple SYN packets without corresponding ACKs) which might indicate an intrusion attempt.

Real-Time Monitoring: In real-time environments, systems continuously capture packets, reassemble them into connections (using the 5-tuple and timestamps), and then compute statistics for each connection. These statistics form the input features for your IDS, enabling it to detect unusual or malicious behavior.

4. Summary In a practical, real-time setting, the “packets” in a connection refer to the individual data units (packets) exchanged between two endpoints over a communication session, identified by the same 5-tuple. This collection of packets is analyzed together to understand the overall behavior of the connection, such as its duration, volume, and any anomalies in the data exchange. Tools like Wireshark and tcpdump make it possible to capture and inspect these flows, while systems like Zeek, Suricata, or custom scripts aggregate this information to provide insights into network behavior.

This contextual view of a ”connection” is fundamental for intrusion detection, as it allows the system to analyze the complete lifecycle of a communication session and spot suspicious patterns that might not be evident from a single packet alone.