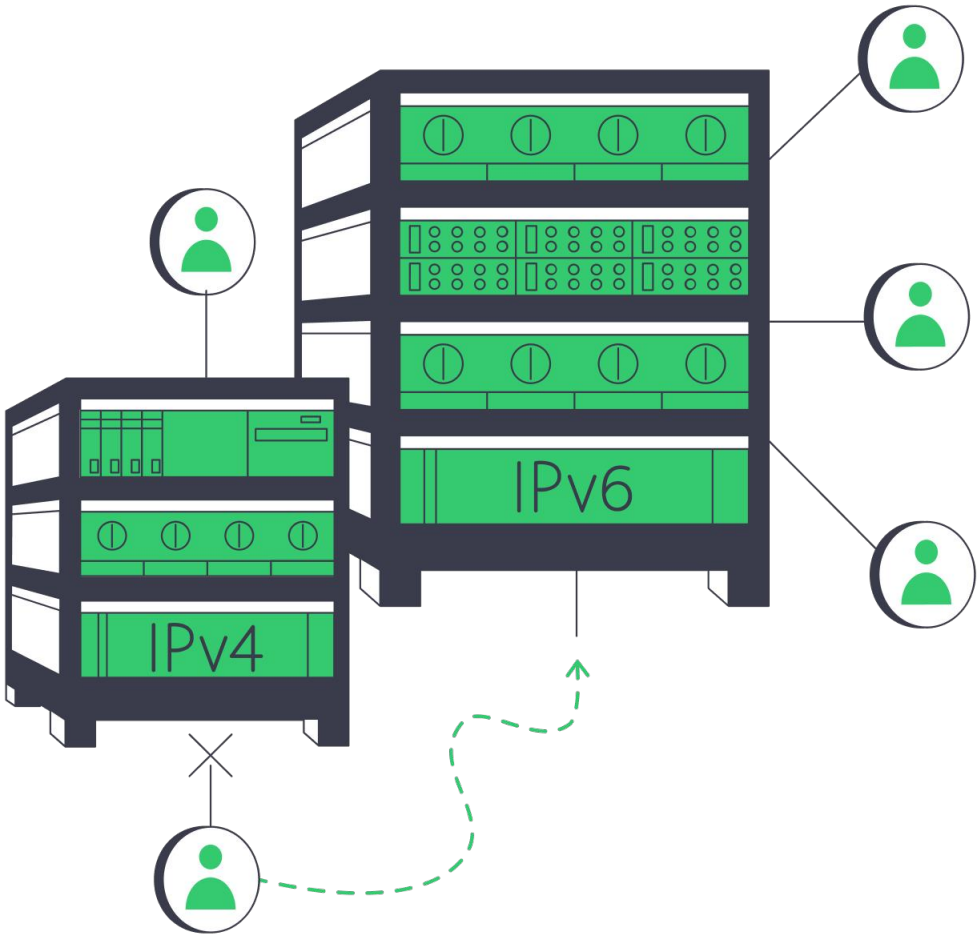
# Performance and Diagnostic Metric

## IPv6 Extension Header - Destination Option



# Abstract

The implementation of the Performance and Diagnostic Metrics (PDM) extension header within the IPv6 framework marks a significant advancement in the monitoring and diagnostic capabilities of network communications. The PDM extension header introduces a standardized method for collecting essential performance-related metrics, including Round Trip Time, Round Trip Delay, etc. These metrics are critical for effective network management and optimization, enabling network administrators to gain deeper insights into network performance and to promptly address issues.

This report presents an in-depth analysis of the PDM extension header, detailing its design, implementation, and integration with UDP DNS packets. It elaborates on the structure of the PDM extension header, its fields, and the rationale behind its inclusion in the IPv6 protocol suite. Further sections cover the development environment, the process of incorporating the PDM header into UDP DNS packets, and the necessary modifications to existing packet handling routines.

The report also discusses the challenges encountered during the implementation and testing phases, along with the solutions employed to overcome these challenges. In conclusion, the findings underscore the impact of the PDM extension header on enhancing network performance and diagnostics. The report concludes with suggestions for future research and development to further optimize and expand the use of the PDM extension header in various network scenarios.

# Table of Content

# Introduction

Effective network management hinges on the ability to diagnose performance issues accurately. Embedding optional sequence numbers and timing measurements within packets provides real-time and retrospective insights into network behavior. However, the current IPv6 specification lacks timing provisions in both its main header and existing extension headers. The Performance and Diagnostic Metrics (PDM) destination option, theorized in **[RFC8250]** fills this gap by introducing fields tailored for these measurements.

The PDM extension header offers significant advantage by providing precise transaction measurements and consistent instrumentation across different nodes without requiring time synchronization between session partners. Additionally, it does not interfere with the existing protocol stack, which means that all the other protocols like TCP, UDP, etc, will work as intended.

The PDM extension header allows for the measurement of critical metrics such as round-trip delay and server delay. Round-trip delay measures the time taken for a packet to travel from a source host to a destination host and back, while server delay indicates the time between a packet's receipt by a device and the sending of the first corresponding response. Round-trip delay highlights network transfer latency, whereas server delay sheds light on server performance, encompassing both application processing times and network stack delays. If server processing time is problematic, additional client-based measurements may be necessary to distinguish between application and stack-related delays.

# Background

The current IPv6 protocol, despite its significant advancements over IPv4, does not inherently support timing measurements in its main header or any of its extension headers. This limitation is a critical oversight, especially considering the increasing complexity and performance demands of modern networks. Traditional network performance monitoring often relies on external tools and post-event analysis, which can be cumbersome and less accurate.

The Performance and Diagnostic Metrics (PDM) extension header was drafted in **[RFC8250]** to fill this gap by embedding optional sequence numbers and timing information directly within IPv6 packets. This provides a standardized method for collecting real-time performance data.

The necessity for the PDM extension header stems from several key factors: the current IPv6 protocol lacks built-in timing fields, which hinders real-time performance analysis and troubleshooting; traditional methods relying on external tools and post-event analysis introduce

delays and potential inaccuracies; modern networks' increasing complexity demands accurate, real-time performance metrics to maintain service levels and quickly diagnose issues; and PDM's ability to provide precise transaction measurements, eliminate time synchronization requirements, and offer consistent instrumentation across different organizations and protocols ensures comprehensive and efficient network management.

# Motivation

The motivation for the development of the PDM extension header is closely tied to the extensibility features inherent in the IPv6 protocol. IPv6 was designed with a flexible architecture that allows for the incorporation of extension headers, which can be used to introduce new functionalities without disrupting the core protocol. This extensibility is a key advantage of IPv6, enabling it to adapt to evolving network requirements and technologies.

The PDM extension header leverages this feature to address a significant gap in the current IPv6 specification: the lack of built-in timing fields and performance monitoring capabilities. Traditional network monitoring tools, which often require external setups and post-event analysis, are not only cumbersome but also limited in their accuracy and timeliness. By embedding performance and diagnostic metrics directly within IPv6 packets, the PDM extension header provides real-time, precise insights into network behavior, for any and all protocols above IPv6.

This capability is crucial for several reasons. Modern networks are becoming increasingly complex, with a wide variety of applications and services that demand consistent and reliable performance. Accurate, real-time performance metrics are essential for maintaining service levels, diagnosing issues quickly, and optimizing network operations. The PDM extension header addresses these needs by providing a standardized method for collecting and interpreting performance data across different transport protocols and organizational boundaries.

Moreover, the elimination of the need for time synchronization between communicating devices simplifies deployment and reduces potential sources of error, making PDM an efficient tool for network administrators. Its ability to provide detailed metrics such as round-trip delay and server delay further enhances its value, enabling quick identification of whether performance issues are network-related or server-related.

# Design & Architecture

The fundamental principles guiding the PDM protocol's design are simplicity, robustness, flexibility, scalability, compatibility, efficiency, and transparency. Each principle ensures the protocol effectively meets its objectives while integrating seamlessly into existing systems. PDM's design adheres to straightforward principles, using the IPv6 Destination Options header to embed performance data, ensuring minimal complexity in implementation and ease of deployment. This simplicity helps in quick adoption and reduces the learning curve for network engineers. The protocol is designed to handle various network conditions, ensuring reliable performance metrics collection even in the presence of packet loss or delays. This robustness guarantees accurate diagnostics without being easily disrupted by network anomalies. PDM can be dynamically configured for all packets or specific protocols, ports, and IP addresses, allowing customized deployment based on specific diagnostic needs. This flexibility ensures the protocol can be tailored to diverse network environments and requirements. PDM supports a wide range of time differentials, from nanoseconds to extended delays in delay-tolerant networks, making it suitable for small-scale and large-scale network deployments. This scalability ensures the protocol can grow with expanding network infrastructures. The protocol seamlessly integrates with existing IPv6 frameworks and adheres to **[RFC8200]**, ensuring it can be used alongside other IPv6 features without causing conflicts. This compatibility guarantees smooth implementation in current network systems. By embedding performance data directly into packets, PDM minimizes additional overhead and maximizes the use of available bandwidth. This efficiency ensures the protocol delivers valuable diagnostics with minimal impact on network performance. PDM operates transparently within the network, providing diagnostic data without altering packet flow or requiring changes to intermediary devices. This transparency ensures that performance monitoring is unobtrusive and does not interfere with normal network operations.

One significant design choice in the development of PDM was to embed performance metrics directly into the IPv6 Destination Options header, ensuring seamless integration with existing IPv6 infrastructure. This choice prioritizes compatibility but introduces the trade-off of potential packet drops when using IPv6 transition technologies or devices that don't handle multiple extension headers properly. Another critical decision was the use of 16-bit fields for time differentials, allowing for efficient data encapsulation while requiring scaling factors to represent a wide range of values. The protocol's reliance on end-host processing for calculating and storing metrics ensures minimal impact on intermediary devices, but this places additional memory and computation demands on the hosts. Additionally, the use of random initial packet sequence numbers enhances security but may complicate debugging and analysis. Balancing simplicity with robustness, PDM is designed to be both easy to implement and resilient under various network conditions, though this sometimes results in a trade-off between precision and resource consumption.

# IPv6 Extension Headers, Destination Option & Packet Structure

The main components of the PDM protocol include the IPv6 Destination Options header, packet sequence numbers, delta time fields, scaling factors, and dynamic configuration options. These components work together to provide comprehensive performance diagnostics while maintaining compatibility with existing IPv6 infrastructure.

**IPv6 Destination Options Header:** Destination Option is an Extension Header for IPv6 that is used here to carry the PDM information. It is identified by a Next Header value of 60 and follows the structure defined in **[RFC8200]**. This header ensures that PDM data is processed only by the destination node, avoiding unnecessary handling by intermediary routers. As shown in **Fig 1**. This Extension Header starts after IPv6 Packet and ends before the main payload.

**Packet Sequence Number This Packet (PSNTP):** A 16-bit unsigned integer that uniquely identifies each packet within a PDM transaction. It helps in tracking the sequence of packets sent, which is crucial for measuring round-trip times and identifying packet loss or reordering.

**Packet Sequence Number Last Received (PSNLR):** Another 16-bit unsigned integer that records the sequence number of the last packet received. This field, combined with PSNTP, helps in understanding the flow of packets between the source and destination, by linking one PDM packet to another.

**Delta Time Last Received (DELTATLR):** This 16-bit field measures the time difference between the reception of the current packet and the last packet. It provides insights into the latency experienced during packet transmission.

**Delta Time Last Sent (DELTATLS):** Similar to DELTATLR, this 16-bit field measures the time difference between the sending of the current packet and the last packet sent. It helps in analyzing the time taken by the sender to process and send packets.

**Scaling Factors (SCALEDTLR and SCALEDTLS):** These 8-bit fields adjust the time differentials to fit within the 16-bit delta fields. By scaling the time values, PDM can accommodate a wide range of time measurements, from very small to very large delays, with some precision loss.

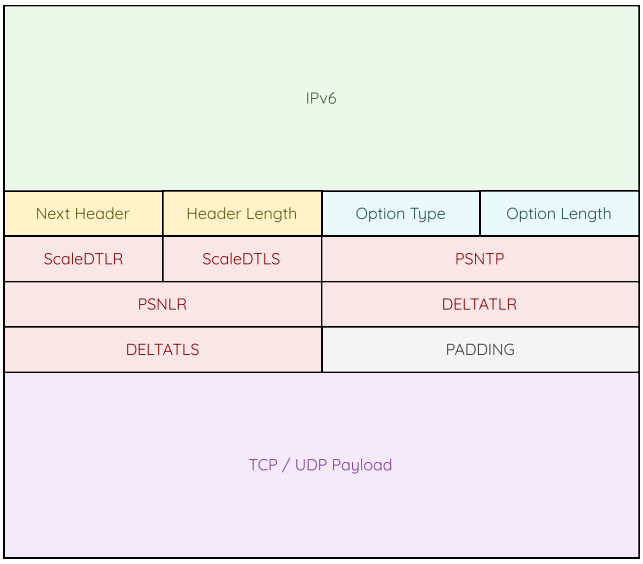



Fig 1. IPv6 Packet Structure with PDM Extension Header

Central to the PDM structure are the delta time and packet sequence number fields. "PSNTP" (Packet Sequence Number This Packet) and "PSNLR" (Packet Sequence Number Last Received) are 16-bit integers used to track packet order and correlate requests with responses, which is critical for diagnosing packet loss and reordering. The delta time fields, "DELTATLR" (Delta Time Last Received) and "DELTATLS" (Delta Time Last Sent), measure the latency of the last received packet and the last sent packet. These 16-bit fields, combined with their respective scaling factors, provide accurate measurements of network latency and processing delays.

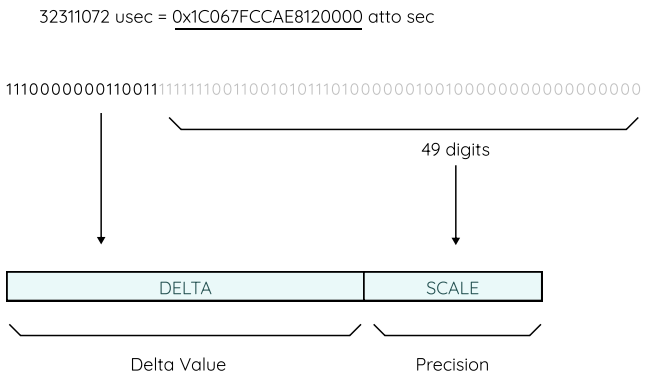



Fig 2. IPv6 Packet Structure with PDM Extension Header

The inclusion of padding is complementary as **[RFC8200]**, and ensures that the PDM option aligns correctly within the IPv6 header. This structured approach allows the PDM protocol to integrate seamlessly with existing network protocols, providing detailed performance diagnostics without significant overhead.

The interaction between delta time and scaling factors is essential for accurate performance measurement, when dealing with clock counters of different sizes and resolutions. Delta time fields alone cannot capture the wide range of possible time intervals in high-resolution networks due to size limitation of integers in network packets, OS kernels, hardware restrictions, etc. By introducing scaling factors, the protocol can represent a vast range of time values in a compact form. For instance, a scaling factor converts a time value from microseconds to attoseconds, fitting larger time intervals into the 16-bit delta time fields as shown in **Fig 2**. This method ensures that both, extremely small and extremely large differences in packet transmission and reception times are accurately recorded and transmitted, by losing some precision, enabling flexible, robust and adaptable latency measurements across diverse network conditions.

# Internal Working Mechanism

The Performance and Diagnostic Metrics (PDM) protocol is structured in a layered architecture, with a modular design that facilitates the integration with most of the protocols at the lower level. At its core, PDM utilizes the IPv6 Destination Options Extension Headers to embed performance-related information, ensuring minimal disruption to existing network protocols and infrastructure. This header serves as the primary conduit for transmitting PDM data, which includes packet sequence numbers, delta time fields, and scaling factors.

The client operates at the application layer, while the server implements the protocol at the kernel layer. This results in imprecise PDM info from the application layer compared to kernel-level measurements done on the server's end. On the server side, timestamps are recorded as soon as the PDM packet is detected, providing a more precise measurement of server latency. This discrepancy necessitates a more rigorous implementation on the client side to ensure the integrity, accuracy, and precision of the metric. A diagnostic metric for a blank TCP/UDP payload may help benchmark unintentional measurement noise. This discrepancy may significantly impact the accuracy of the recorded metrics, in certain cases like when both the client and server needs the accurate information of the metrics, although, in our scenario, where the server do not use the metric data, this discrepancy will not impact the functionality.

The Packet Sequence Number This Packet (PSNTP) and Packet Sequence Number Last Received (PSNLR), play a crucial role in tracking the flow of packets. These 16-bit integers are used to identify and correlate (or link) packets within the bulk of network packet transactions, facilitating the measurement of round-trip times and detection of packet loss or reordering. The PSNTP is incremented with each packet sent, while the PSNLR reflects the last received packet's sequence number. This interaction helps in maintaining a coherent sequence of packets, which is essential for accurate performance diagnostics.

Delta time fields, Delta Time Last Received (DELTATLR) and Delta Time Last Sent (DELTATLS), measure the time differences between the reception or sending of consecutive packets. These 16-bit fields, adjusted by their respective scaling factors (SCALEDTLR and SCALEDTLS), provide insights into the latency and processing times at both the client and server ends. The scaling factors convert the native time units (such as microseconds or nanoseconds) into a common unit of attoseconds, enabling uniform time measurement across different systems. This scaling is crucial for fitting large time differentials into the limited 16-bit fields.

At the kernel level on the server, the implementation is more complex due to the need for precise timestamping and the absence of a built-in mechanism to map response packets to their corresponding request packet at the network level. To address this issue, a segmented hashmap data structure is employed as shown in **Fig 3**. This data structure stores packet sequence numbers, recorded times, and protocol packet identifiers, enabling manual mapping of packets. This approach ensures that each response packet can be accurately matched to its request, allowing for precise measurement of server processing times and network latency.

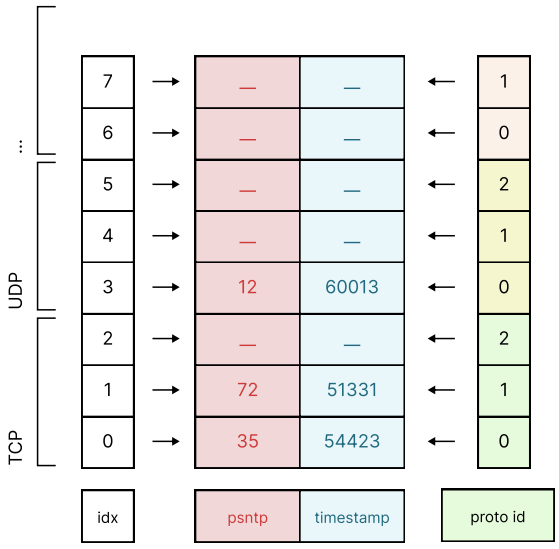



Fig 3. Segmented HashMap

# Segmented Hash Map

Segmented Hash Map is the core data structure used in the server to store the packet information. Segmented has been chosen to be the data structure of choice because of multiple reasons. According to the source RFC of the PDM protocol, the server needs to primarily store the PSNTP field, along with the timestamp, searchable through packet chain identifier. As a performance diagnostic tool, it necessitates low latency and fast data transaction from its respective data structure. A Queue like data structure would not suffice for this, as searching for a specific entry, when calculating the time delta, will itself, take a lot of time, and hence, will act as a noise in the final measurement. A Hashmap like data structure acts as a better alternative for this, as they are not required to traverse through the whole array. They also have a big advantage in this particular scenario as the key element, i.e. the packet chain identifier itself is an integer, and a separate calculation of hash is not necessary, discarding the computationally intensive part, making a hashmap, an excellent choice.

Additionally, the data structure also needs to search the packet chain identifier, within the group of it's particular protocol. This adds an additional requirement of either multiple hashmaps or segmentation in the central hashmap data structure. To achieve this, the hashmap array's indices have been divided into segments, where each segment represents the hashmap space for particular protocols, as used here and represented in **Fig 3**.

```
IPv6 {
    PDM ExtHdr (121, -, -, -, -, -)
    UDP {
        DNS (61, ..., ..., example.org)
    }
}
```

Fig 4. Dummy DNS Packet

This DNS packet warrants an addition of an entry at **( index 1, segment 1 )**, directly translating to **( raw index 4 )** of the hashmap array. The type of packet, in this case, a DNS type packet sets segment to 1, and the packet chain identifier, i.e. `Transaction Id` in this case sets the index to 1. Hence the DNS packet adds the structured object [121, 54113], a tuple representing the PDM Extension Header's packet sequence number, and the timestamp when the packet has been received, to the Segmented Hashmap.

Any incoming packet with a PDM Extension Header, say a DNS packet, will have some identification value, that gets mapped in the application layer, in this case a `Transaction Id` 61 in the DNS packet, as shown in **Fig 4**. This `Transaction Id` acts as key in the hashmap's UDP-DNS segment, say segment 1.

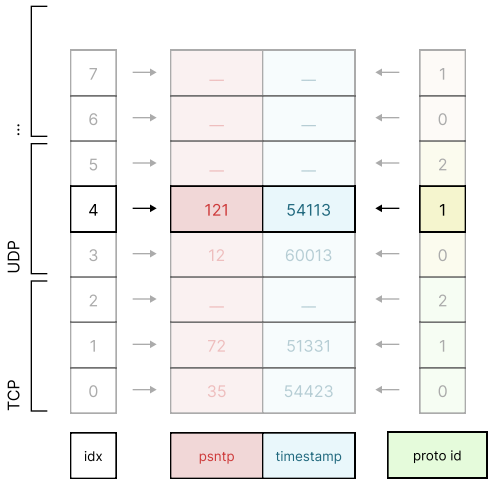



Fig 5. Entry of the DNS Packet in the Segmented Hashmap

# Server Module

The server module is implemented as a Kernel Module (*.ko) in Linux, giving more fundamental control and insight over the protocol processing. The Kernel Module registers **NF_INET_LOCAL_OUT** and **NF_INET_LOCAL_IN** hooks as TX and RX. When a packet in the RX, the module checks if the packet's timing needs to be diagnosed. This is determined by the presence of the PDM Extension Header in the IPv6 packet. In case, no PDM Extension Header is found, the packet is accepted by the Netfilter. When a PDM Extension Header is detected, the subroutine recording the time and mapping the packet-chain gets triggered. The subroutine fetches the PDM Extension Header and Packet Chain Identifier from the packet and records it in the Segmented Hash Map, in it's pre assigned protocol segment. Once the necessary information is successfully recorded, the RX accepts the whole packet, with it's PDM Extension Header inside the packet. This decision flow, along with the TX counter part, has been represented in **Fig 5**.

Once the whole packet is accepted by the Netfilter hook RX, the packet travels to the application layer, and the concerned application writes a response packet in the Netfilter buffer to be processed and transmitted. This brings the response packet to the Netfilter hook TX. The reason why RX maps each PDM request with it's packet chain identifier, is due to this architecture of network layers, where the network layer Netfiler hook TX will have no pre-mapped information of requests, for which it receives the response packet. Simply put, TX will receive all the packets, that are destined to go out from the system, and may or may not belong to a request chain with PDM Extension Header.

When the packet is received by TX, the packet's request chain identifier is determined and

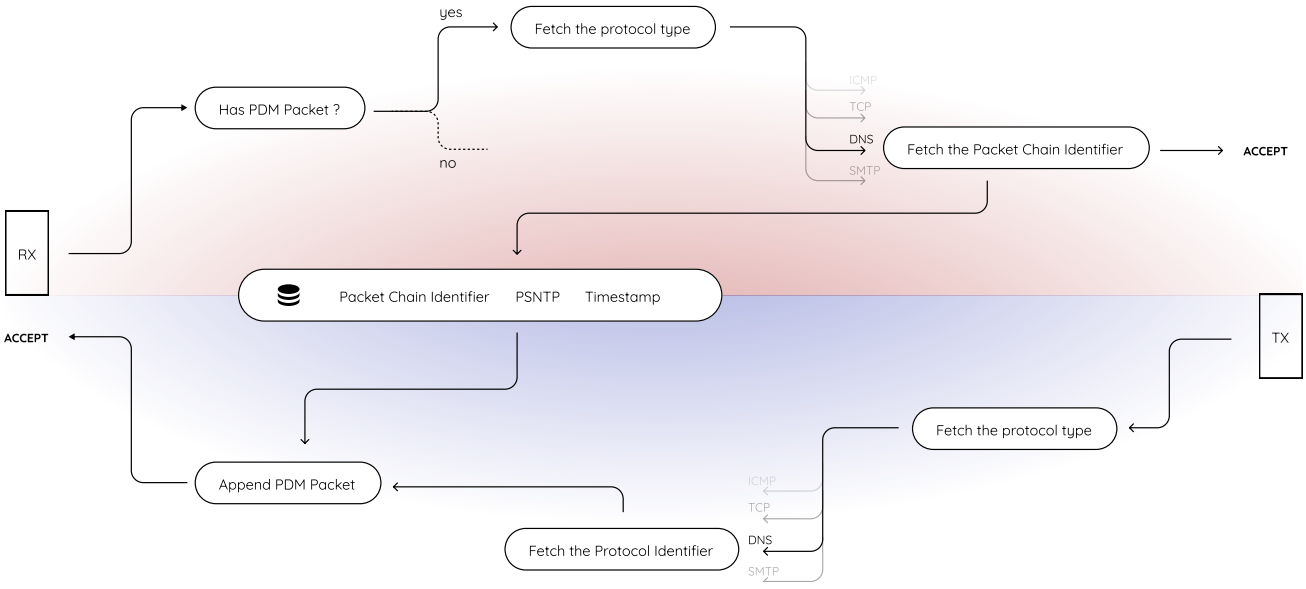



Fig 5. Logical flow in the Server Side (as Kernel Module)

cross-referenced with the Segmented Hash Map. If the request chain identifier is found in the hashmap, in it's designated segment, it asserts that the packet received by TX is a response to that particular request, and thus, needs a PDM Extension Header to be included. Following this, the PDM Extension Header record in the hashmap is utilised to form a new PDM Extension Header as a response, and attached to the packet, by modifying the socket buffer sk_buff.

The **sk_buff** is the data structure used by linux to process/manage network packets along with some other data like immediate sk_buff pointers, metadata, etc. The structure of sk_buff has a head and a tail, surrounding the main packet, as shown in **Fig 6.a**.

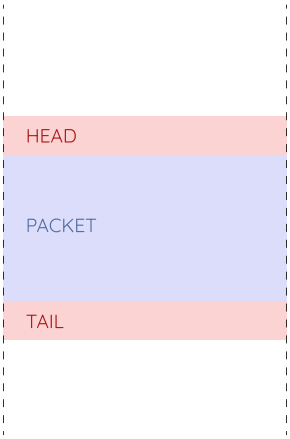



Fig 6.a Basic structure of sk_buff

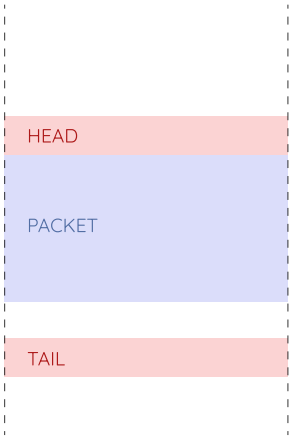



Fig 6.b Modification of packet in sk_buff

Modifying the packet means remapping the tail and thus needs additional steps. As shown in **Fig 6.b** the tail needs shifted by the exact amount of bytes, to be added or removed from the sk_buff. In case of adding the PDM Extension Header, the tail will be shifted by a specific amount, exactly equal to the size of the new extension header. Once the shifting is done, the packet will be dissected into 2 parts and similarly shifted. This is necessary, because the extension header is to be added in the middle of the packet, and not in the end. The PDM Extension Header is pushed right after the IPv6 Packet, and hence simplifies this process by a little. The dissection has been shown in the **Fig 6.c**. Once the dissection is done and the rest of the bytes have been shifted, the empty space created, can be addressed as the PDM Extension Header and be modified. Once the PDM Extension Header is successfully inserted and the sk_buff has been modified, the packet is accepted by the netfilter, and sent to the client

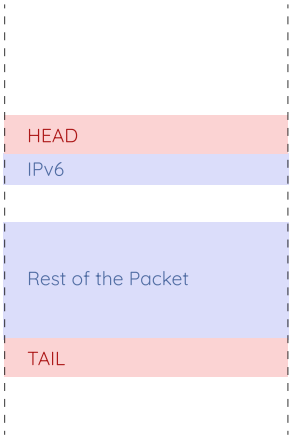



Fig 6.c Modification of packet in sk_buff

# Client Module

The process of the client is simple compared to that of the server and begins with considering a protocol, say DNS, and it's packet, say, taken as user input, the client frames it on IPv6 packet as a payload and adds a blank PDM Extension Header, with a PSNTP set to it's own unique identifier. This acts as the first packet to initiate the PDM transaction between client and server. The client works in the application layer, unlike the server and may incur additional noise in it's measurements due to internal latency. This has been represented in **Fig 7**.

Once the first packet is received by the server, it initiates the PDM transaction by sending the response packet with PDM Extension Header with its diagnostic metric embedded. This received packet is the second packet and its PDM Extension Header is used to embed its reply PDM Extension Header to the next packet with empty payload, referred to as packet three, is sent to the server. This ensures that the server gets the diagnostic data back.

The client records the time of sending each and every packet, and uses those to calculate the Round Trip Time and Round Trip Delay, and is presented in the console.

The current implementation of the client uses a python package scapy to perform network operations. Python being a comparatively slow runtime, on top of scapy serializing all parts of the packet makes the operations highly inefficient and time consuming.

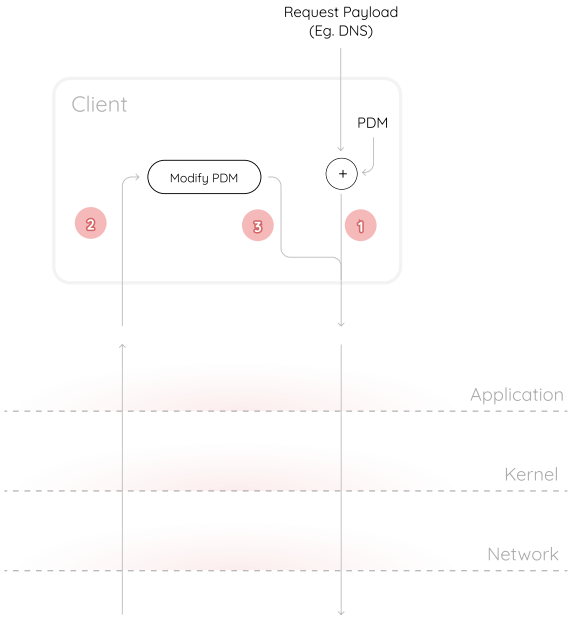



Fig 7. Logical flow in the Client Side (as a Python Client)

# Packet Transactions

The performance measurement between the client and server works with 3 packet transactions. The first packet (at 540413) is initiated by the client (Host A), announcing to the server (Host B) that it needs PDM Extension Header (represented by 6 tuple) attached with the response for the request. Suppose this PDM Extension Header looks like (16, 0, 0, 0, 0, 0). It means that the packet sequence number of the PDM Extension Header is 16. This is used to identify the exact PDM Extension Header from the pool packets, that corresponds to the request extension header. Once this PDM Extension Header reaches the server (at 681681), the PDM Extension Header is recorded and a response is sent back with server processing time for that particular packet. This is done by subtracting the receiving time from the transmission time (at 681705), i.e. 24 in this case, and embedding it on the response PDM Extension Header, responding back to the client. The client receives (at 540465) the packet and now knows the total Round Trip Time i.e. 52 and the time the server took to process the request, i.e 24. These 2 information can be utilized to calculate the the Round Trip Delay, i.e. 28. This Round Trip Delay can be considered as the network latency.

Till now, only the client has the information of the RTT and RTD of the server, where as the server knows nothing about the client. This is solved by sending a similar PDM Extension Header infused packet to the server (at 540485), with the timing information of client, as shown in **Fig 8**. When the server receives the final packet of the transaction (at 681753), it has enough data to calculate the Round Trip Time and Round Trip Delay. This lets the server inform of the client latency, as well as the network latency.

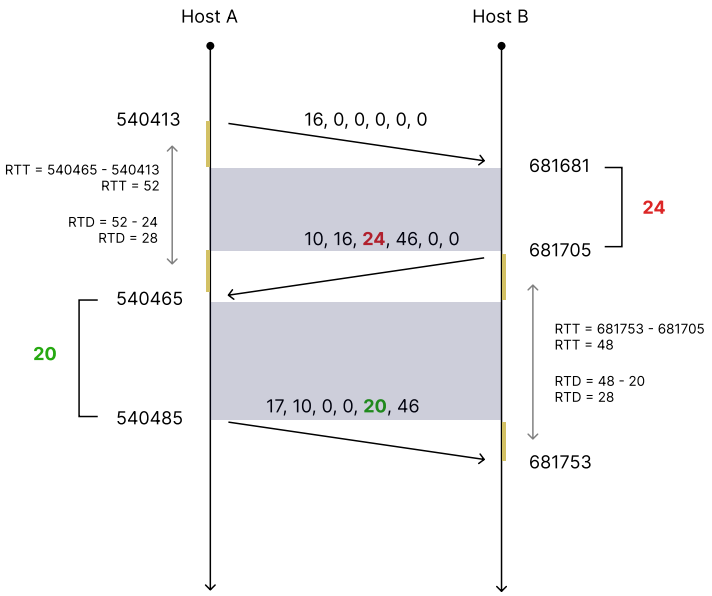



Fig 8. Packet transaction between 2 hosts (server-client)

# Known Issues with the Implementation

**[ISSUE 1]** Router / Proxy nodes in the middle may send response packets back to the client if PDM is enabled.

**[ISSUE 2]** Bruteforce attacks may be used to override the packet received time.

# References

**[RFC8250]** Elkins, N., Hamilton, R., & Ackermann, M. (n.d.). RFC 8250: IPv6 Performance and Diagnostic Metrics (PDM) destination option. IETF Datatracker. https://datatracker.ietf.org/doc/rfc8250/

**[RFC8200]** Deering, Dr. S. E., & Hinden, B. (n.d.). RFC 8200: Internet protocol, version 6 (ipv6) specification. IETF Datatracker. https://datatracker.ietf.org/doc/rfc8200/