Supporting PTP-1588 in BMv2: A Proposed Ingress and Egress Timestamping Scheme

Bill Pontikakis, François-Raymond Boyer

Agenda

- **WHY** (Introduction & Problem Motivation)
  - 5G / ORAN / eCPRI
  - P4 and BMv2 (eCPRI as a new case-study for P4)
  - Timing Synchronization using PTP-1588 over IEEE 802.3 Ethernet
- **HOW** (Technical Implementation)
  - Challenges with the original timestamping implementation in BMv2
  - Details of our modifications to the BMv2 switch
  - How we implement Ingress and Egress timestamps
- **WHAT** (Results & Achievements we have so far)
  - Error resolution with respect to original implementation
  - Statistical results showcasing the time error reduction

Legacy Cell Towers & Cell-site Cabinets

- Legacy cell towers rely on long copper cabling and power-hungry amplifiers, requiring extensive infrastructure, such as large footprints, dedicated huts, power backups, and cooling systems.
- This setup results in high operational costs, low bandwidth, and limits scalability, making it unsuitable for future mobile network demands.

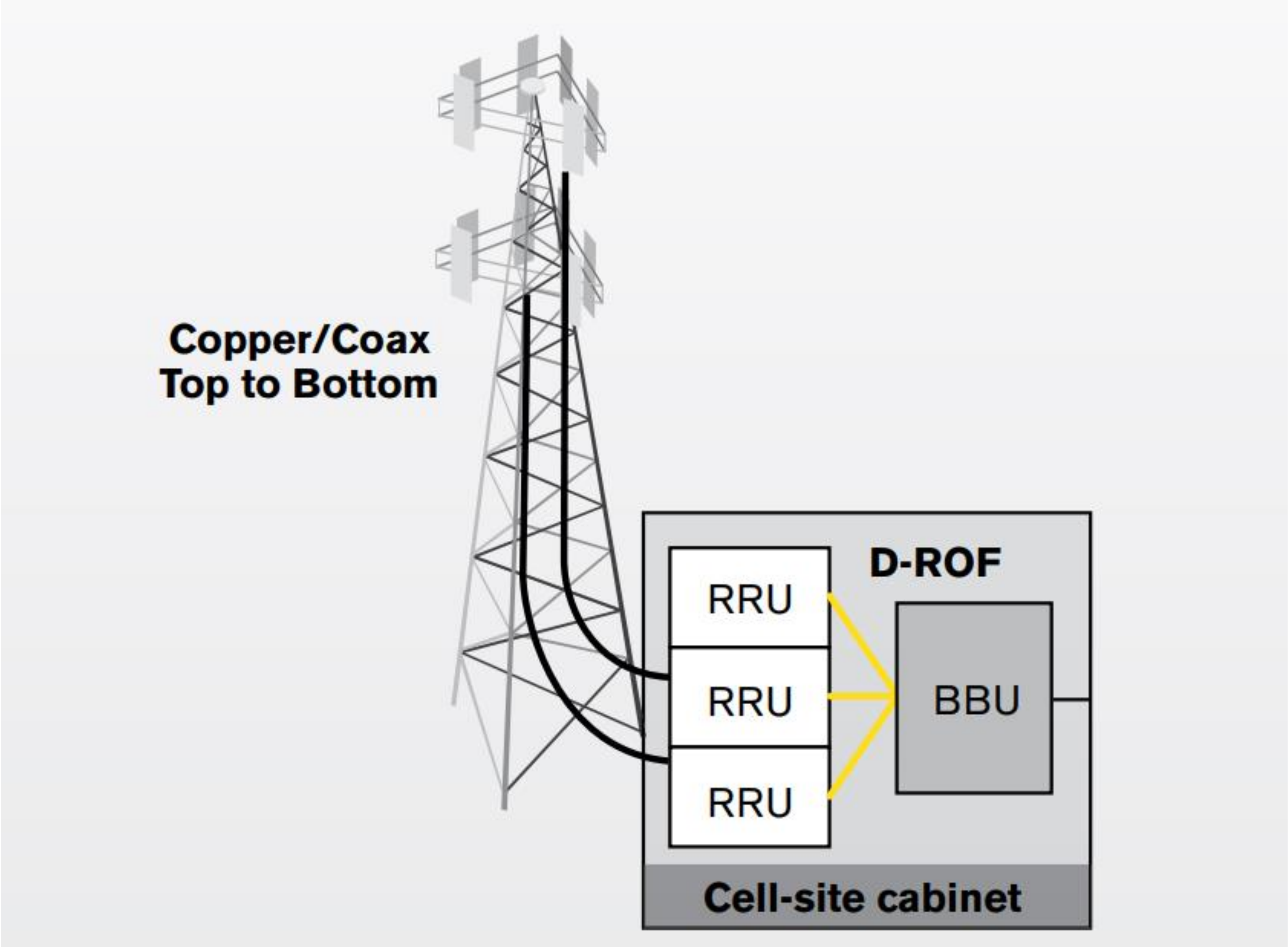
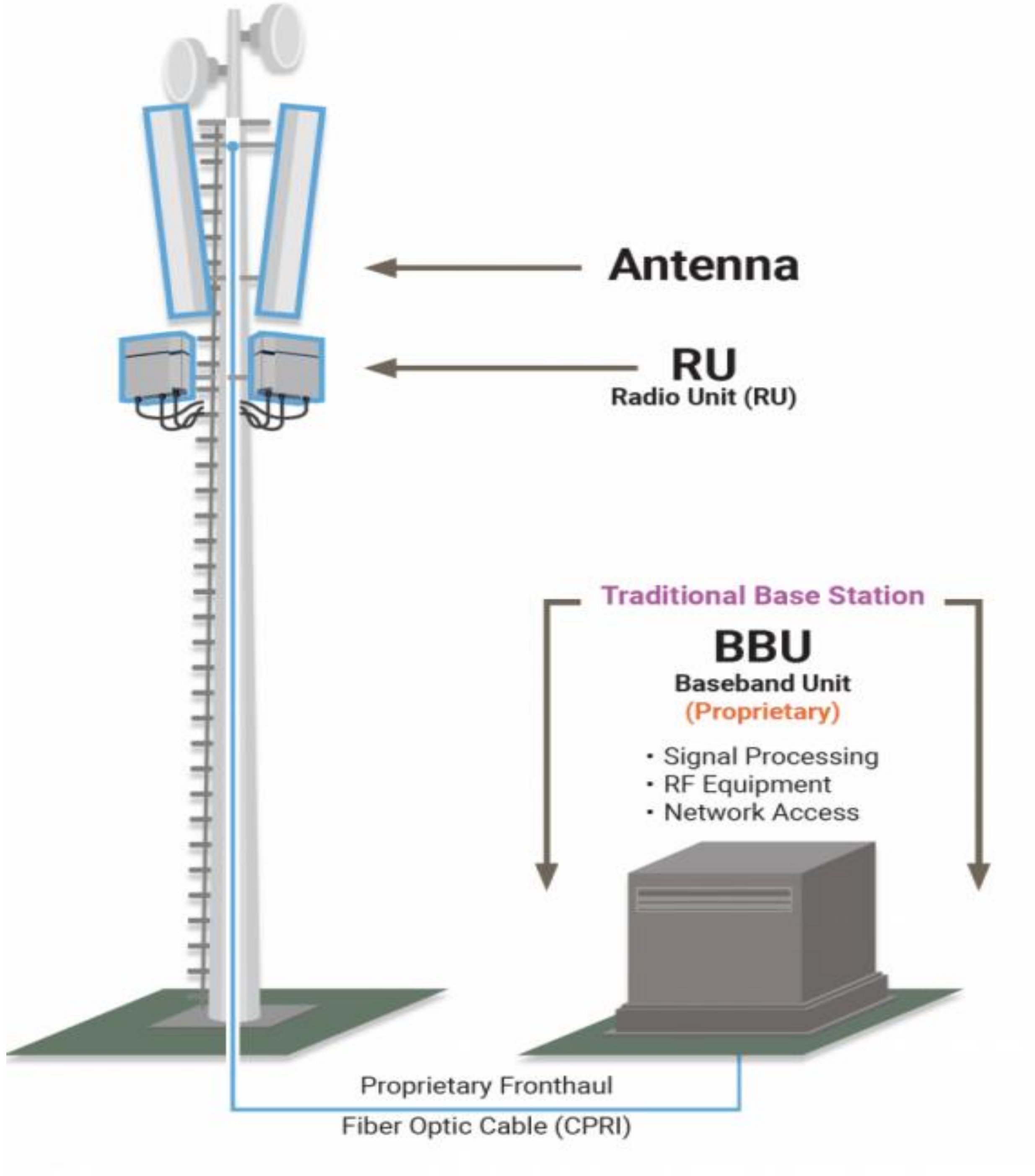


Figure 1 - Application Note 310 - Understanding the Basics of CPRI Fronthaul Technology - Gary Macknofsky, Product Manager, EXFO

Baseband Unit & Remote Radio Head Architectures in Traditional RAN

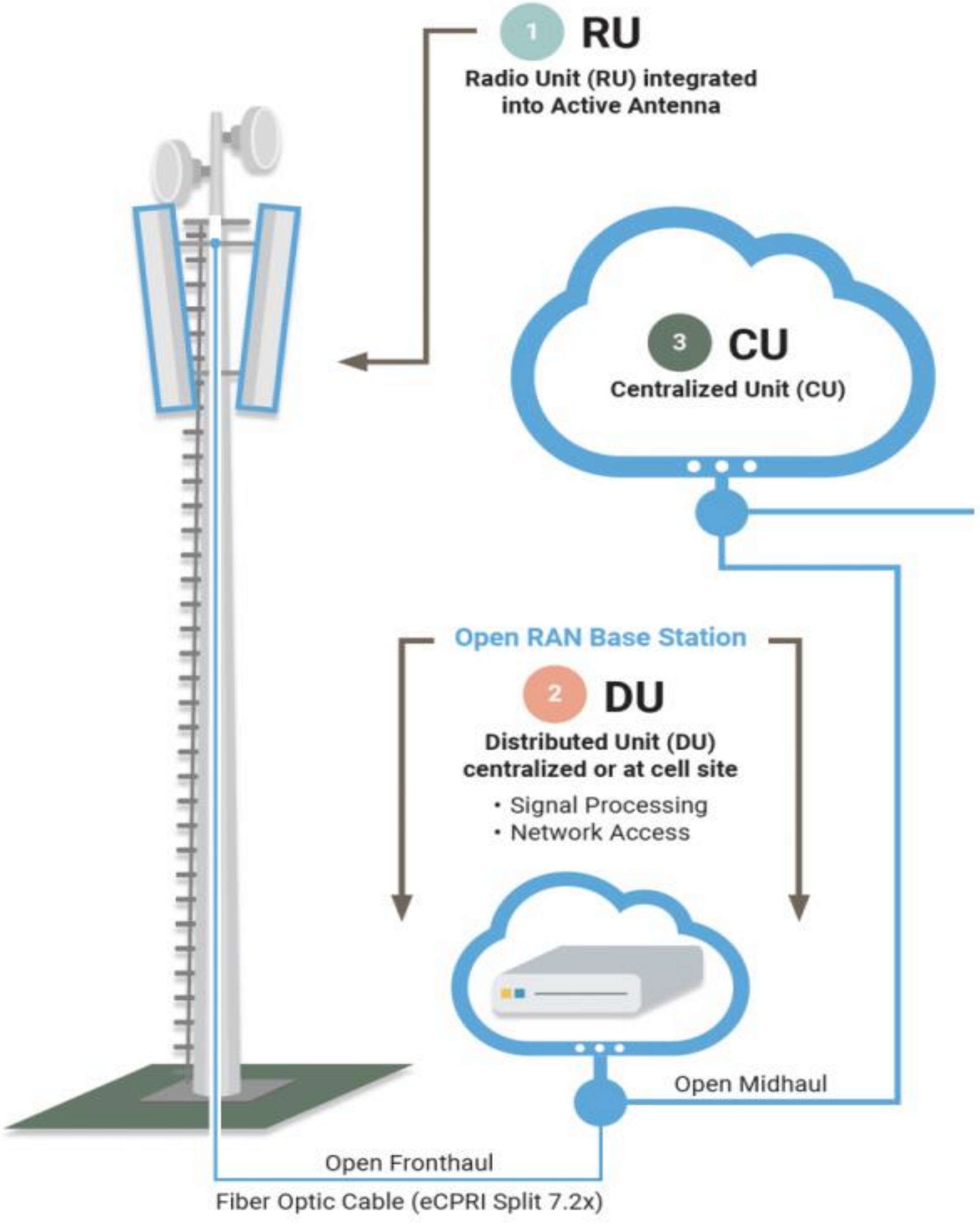
- In upgraded cell sites Fiber replaces copper cabling, reducing noise, power needs, and increasing bandwidth for C-RAN.
- Remote radio unit moved to the top of the tower, co-located with antennae.
- Communication between the baseband unit (BBU) and remote radio head (RRU) now uses the Common Public Radio Interface (CPRI) protocol
- BBU is proprietary and from a single vendor.



https://www.mavenir.com/portfolio/mavair/radio-access/openran/

Open RAN (O-RAN)

- Open RAN enables modular RAN components from multiple vendors by standardizing interfaces.
- Main elements: Radio Unit (RU), Distributed Unit (DU), Centralized Unit (CU).
- Communication between RU and DU now occurs over Open Fronthaul using protocols like eCPRI.
- **Key benefits:** Increased agility, innovation, cost savings, and support for virtualization on vendor-neutral hardware.



https://www.mavenir.com/portfolio/mavair/radio-access/openran/

High-level E2E Network Architecture of 5G ORAN Deployment

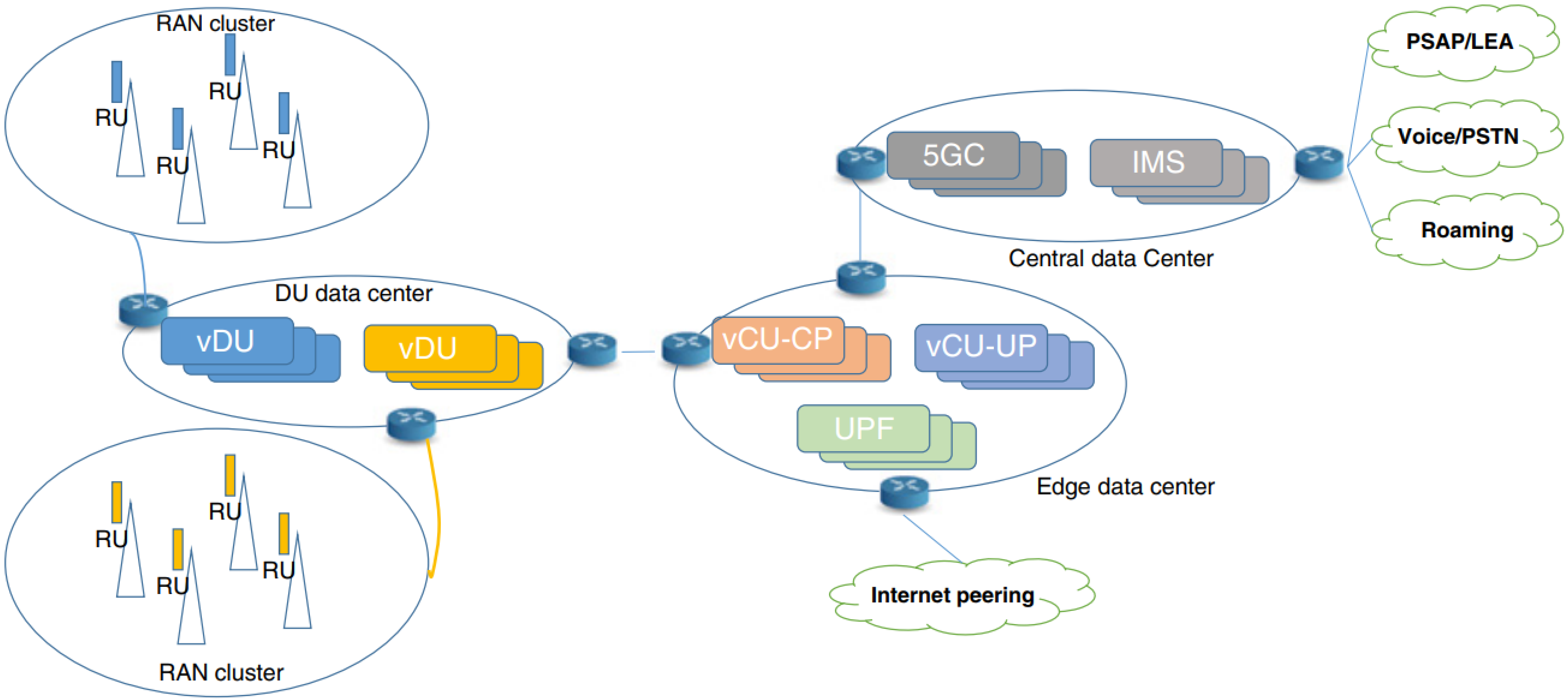


Figure 10.1 Chenumolu, S., 2023. Open RAN Deployments. Open RAN: The Definitive Guide, pp.145-171.

eCPRI (enhanced Common Public Radio Interface)

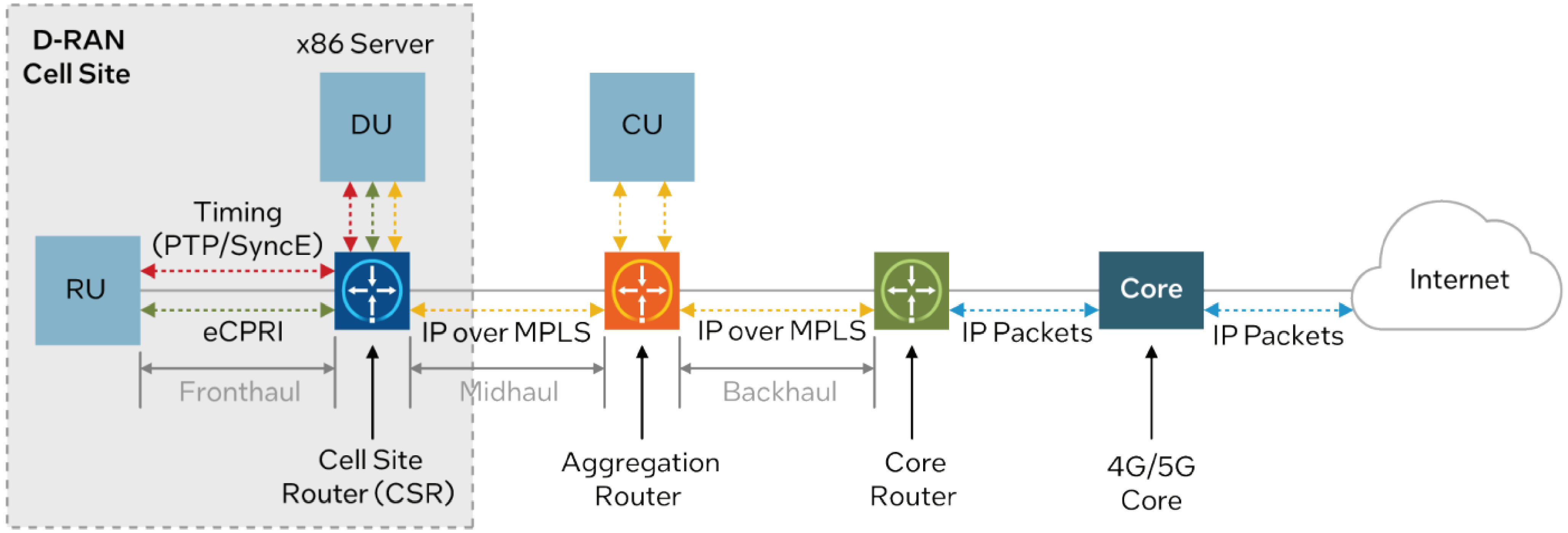


Figure 2 from: Solution Brief | Intel's Accelerated Virtual Cell Site Router Solution on an Intel® FPGA-Based SmartNIC N6000-PL Platform Helps Communication, 2023

- 5G networks rely on eCPRI to transfer user and control data between Distributed Units (DUs) and Radio Units (RUs) in the fronthaul.
- eCPRI is susceptible to delays and packet loss, especially in congested networks.
- 5G fronthaul networks have strict end-to-end timing requirements.
- PTP used to maintain precise synchronization.

eCPRI Intel FPGA IP High-Level System Overview

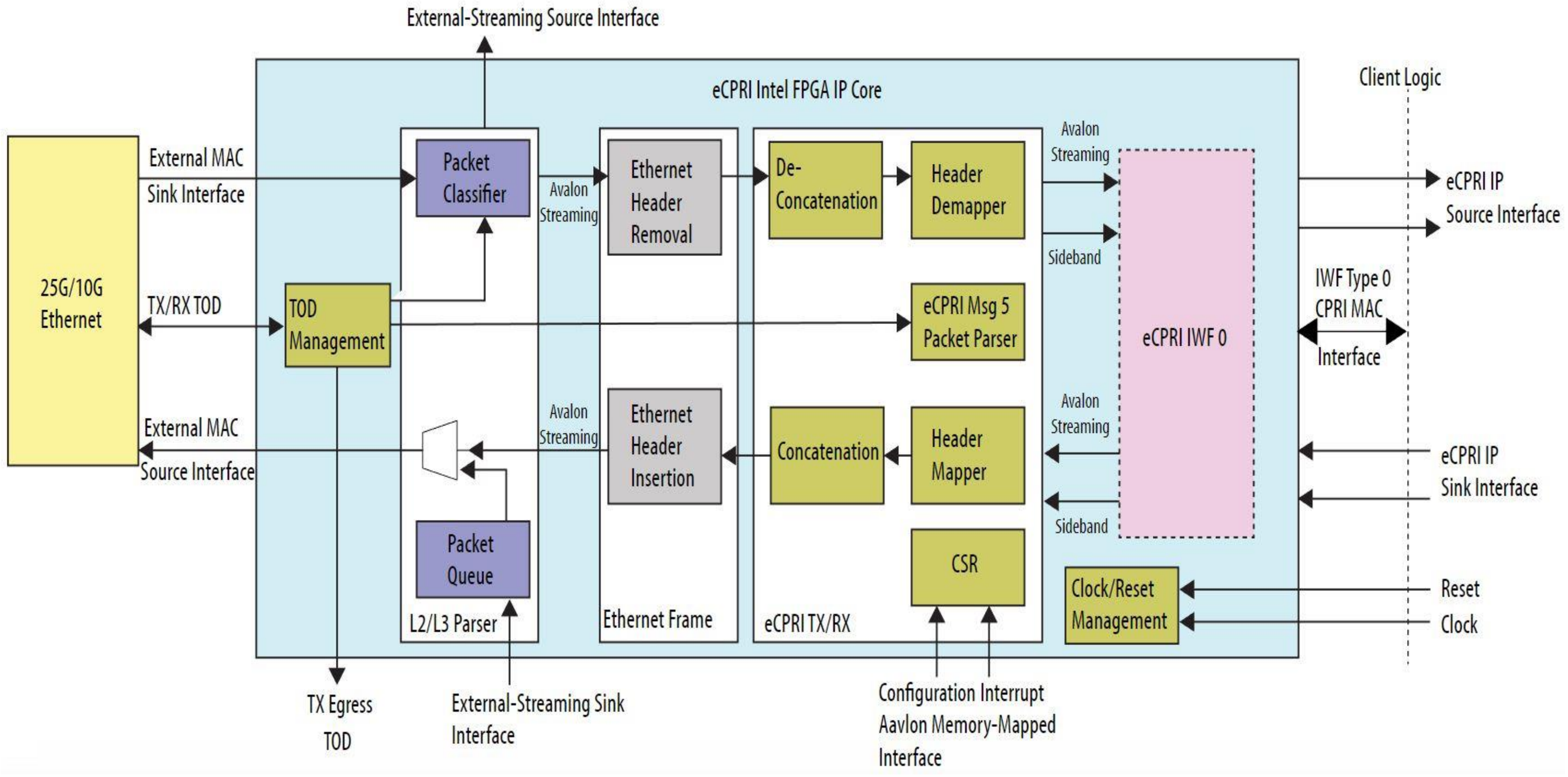


Figure 6 from: eCPRI Intel® FPGA IP User Guide, IP Version 2.0.2, 2023

PTPv2-1588 over IEEE 802.3 Ethernet implementation in P4

- We implemented eCPRI in P4, leveraging Intel's eCPRI FPGA IP as a reference design.
- We modified our initial P4 eCPRI implementation to enhance modularity and flexibility for future extensions.
- To validate our design, we modified BMv2 to support precise ingress and egress timestamps necessary for accurate PTP protocol testing.
- Our PTP implementation is a minimal, proof-of-concept (POC) unit designed to demonstrate the P4 language's ability to implement time synchronization protocols.

Implementing High-Precision Timestamps in BMv2

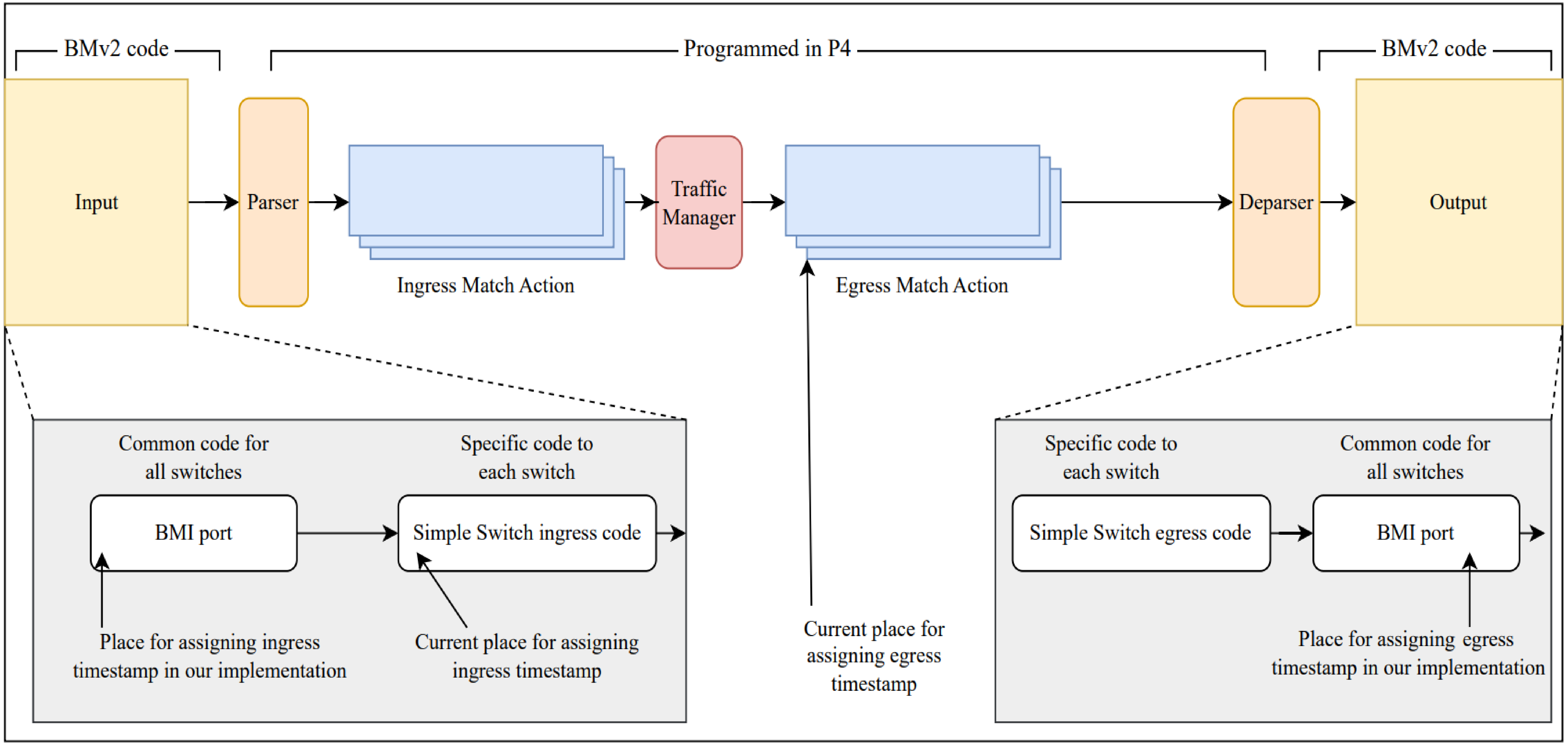
Limitations of Current BMv2 Standard Metadata Timestamps:

- Standard BMv2 timestamps lack precision due to delays introduced by additional processing stages before Ingress and after Egress.

Implementation of Precise Ingress and Egress Timestamps:

- Ingress timestamp is captured as early as possible when a packet enters the switch to reduce latency inaccuracies.
- Egress timestamp is taken at the last possible moment, just before the packet leaves the switch.
- Timestamps are represented in a 64-bit truncated format, aligning with PTPv2-1588 requirements and improving compatibility with time synchronization protocols.

Enhanced Timestamp Precision in BMv2 for Time-Sensitive Networking



Precise Ingress Timestamping for BMv2: Architecture and Code Modifications

- The ingress timestamp is a 64-bit truncated Unix format for improved precision.
- Unix timestamp measures time by the number of non-leap seconds that have elapsed since 00:00:00 UTC on 1 January 1970, and it is widely used in synchronization standards.
- Timestamping is integrated directly into the BMI port to capture ingress timestamps as early as possible in the packet processing pipeline.
- The BMI port code was converted from C to C++ to enhance code flexibility and maintainability.

Precise Ingress Timestamping for BMv2: Architecture and Code Modifications (cont'd)

- BMv2 was restructured to use input structs for function handlers, simplifying the addition of metadata to input packets.
- The restructuring extended to parent classes of device managers and switches, supporting future metadata and functionality enhancements.
- All changes are backward compatible and pass existing tests, ensuring smooth integration with previous systems.
- New tests were developed to validate the added components and ensure their functionality.

Enhancing BMv2 Egress Timestamps

- The egress timestamp was added at the last possible moment, just before the packet leaves the switch.
  - **Effect**: It improves precision.
  - **Consequence:** The egress timestamps cannot be directly accessed from the P4 pipeline.
  - **Resolution:** … NEXT SLIDE

Enhancing BMv2 Egress Timestamps (cont'd)

Two metadata fields were added, accessible from P4:

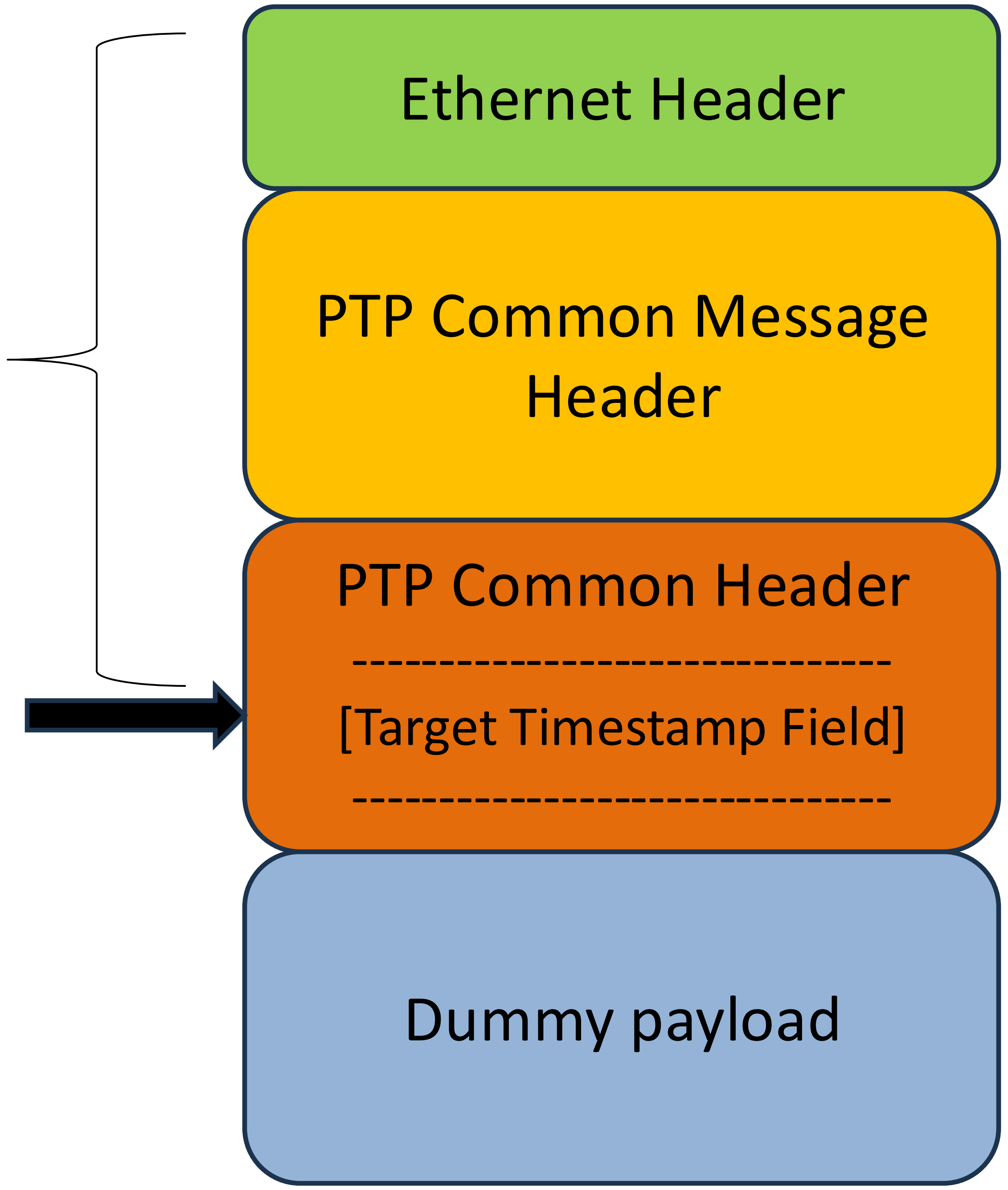
- A **1-bit field enables or disables** the automatic truncated egress timestamp.
- A **32-bit field sets the offset** in the packet where the egress timestamp is injected.

- All changes are backward compatible and passed all existing tests:
  - **Effect**: Ensures smooth integration with existing designs.
- New tests were developed:
  - **Goal**: To validate the added functionality and ensure proper operation.

Enhancing BMv2 Egress Timestamps (cont'd)

Timestamp insertion offset is set inside P4 pipeline

Automatic egress timestamp injection after the P4 pipeline is finished processing



Software Interface Modification: New Packet Handler

Old Packet Handler – Hardwired Variables

- **Hardwired to specific parameters, limiting flexibility and making it difficult to extend functionality**

```
using PacketHandler = std::function<void(int port_num, const char *buffer, int len, void* cookie)>;
```

New Packet Handler – Flexible Structs

- **Encapsulates all parameters a struct, improves flexibility, simplifies maintenance, enables extensibility:**

```
struct PacketInfo {
   int port_num; const char* buffer; int len;
   MyMetadata metadata;
};
using PacketHandlerWithPacketInfo = std::function<void(const PacketInfo *packetInfo)>;
```

Software Interface Modification: New Set_Packet_Handler function

Old set_packet_handler function – Void Pointer

- **Passes a cookie (void pointer) to the SwitchWContexts object to access its specific receive function in bmi_port.c. However, using a void pointer limits type safety and flexibility:**

ReturnCode set_packet_handler(const PacketHandler &handler, void *cookie);

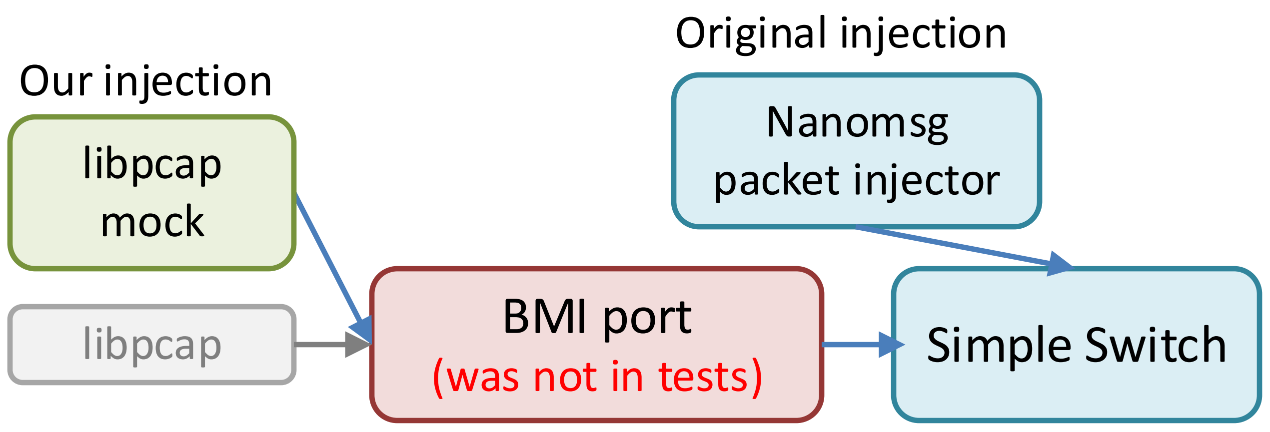
New set_packet_handler function – Lambda based access to Metadata

- **Replaces the Void Pointer with a lambda function, providing bmi_port.cpp direct access to the "receive with metadata" function. This approaches enhances type safety and simplifies access to Metadata:**

ReturnCode set_packet_handler_with_packet_info(const PacketHandlerWithPacketInfo &handler);

Implementation and Validation of BMI Port Testing

Background:
- ***Original Testing Limitation:***
  - bmi_port code was not tested
  - nanomsg-based injection into switch



Testing Update:
- ***Reason for New Tests:***
  - Had to put new functionality in bmi_port.
  - Are we breaking something? Is new functionality correct?
- ***Solution:*** libpcap mock – link-time replacement of library.

Benefits:
- ***Comprehensive Testing:*** This new testing setup allowed for thorough validation across all nine tests in the simple_switch test suite.
- ***Successful Outcomes:*** Using the libpcap mock replaced the nanomsg-based injector, ensuring that the bmi_port functionality was fully tested and verified.

Bug Discovery and Resolution in *bmi_port_destroy_mgr* Function

Bug in *bmi_port_destroy_mgr* Function:

Function Purpose: To destroy all active ports managed by BMI.

- **Problem:** Only half of the ports were being removed during the destruction process.
- **Cause:** Elements were being removed from the array while iterating over it, causing the array size to change dynamically.
- **Effect:** This led to skipped elements and some ports remaining undestroyed.

Resolution:

- ***Refactored Loop Logic:*** Modified the iteration approach to properly handle dynamic changes in the array, ensuring all ports are successfully destroyed.

Deadlock Resolution in BMI Port Interface

Issue in *_bmi_port_interface_remove* Function:

Problem:

- ***Deadlock:*** Occurred when _bmi_port_interface_remove performed a write operation and waited for a read response while holding port_mgr->lock.
- ***Conflict:*** The run_select thread also needed port_mgr->lock to read from or write to the same pipe during its loop, resulting in both threads being stuck.

Cause:

- ***Simultaneous Lock Acquisition:*** Both threads required the same lock at different times, leading to a deadlock where neither could proceed.

Resolution:

- ***Modified Locking Mechanism:*** Changed the locking strategy to allow non-blocking access, ensuring that both threads could proceed without waiting on each other indefinitely.

Bug Resolution in dev_mgr_bmi.cpp: Proper Thread and Resource Cleanup

Bug in *dev_mgr_bmi.cpp* Regarding *p_monitor*:

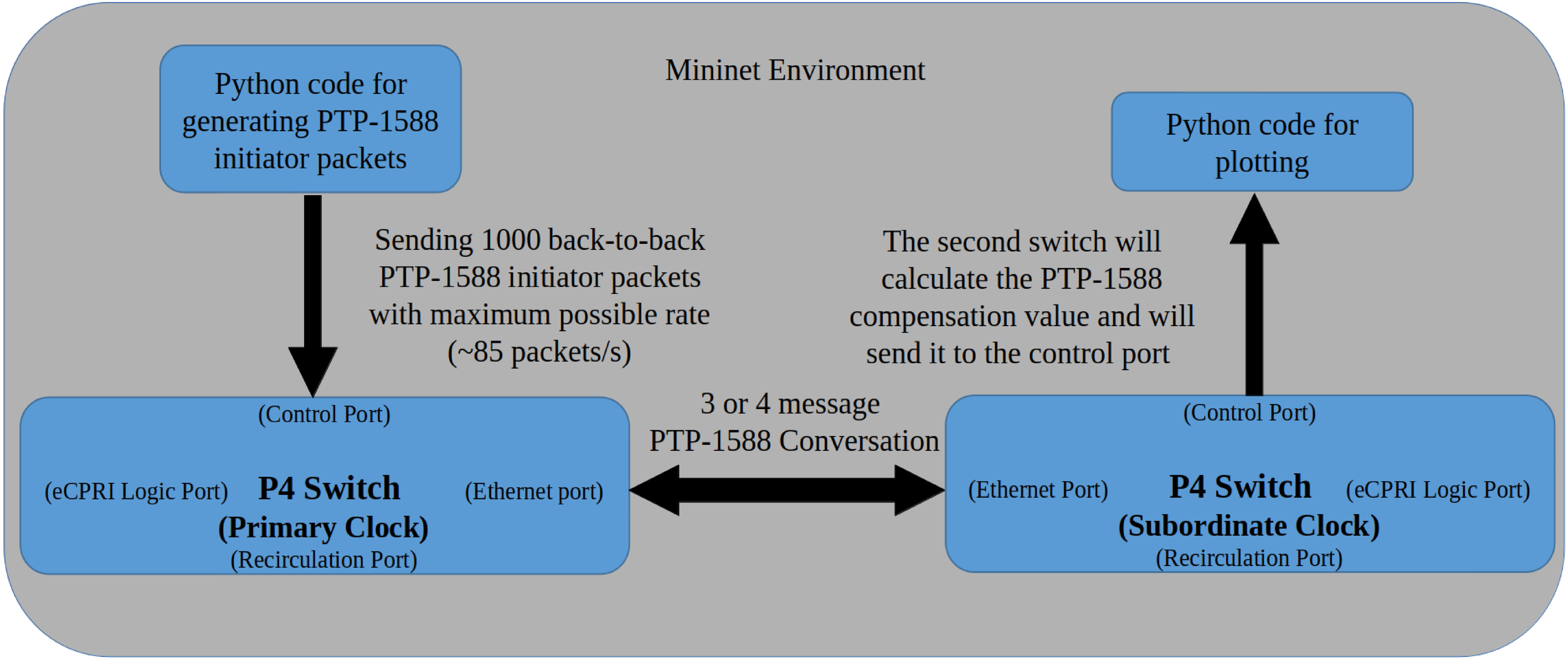
Problem:

- The p_monitor thread was not stopped at the correct time, leading to two potential issues:
  1. ***Pure Virtual Function Call:*** If p_monitor was not halted in the ~BmiDevMgrImp() destructor, it risked invoking a pure virtual function after the derived class was destroyed, leading to undefined behavior.
  2. ***Access to Destroyed Resource:*** The p_monitor thread needed to be stopped before destroying port_mgr. Failing to do so could result in the thread attempting to access an already destroyed port_mgr, causing potential crashes.

Resolution:

- ***Order of Destruction Modified:***
  - Ensured that p_monitor was properly stopped within the destructor of BmiDevMgrImp.
  - Ensured that p_monitor was stopped before port_mgr was destroyed, preventing access to invalid resources and avoiding crashes.

Mininet Testbench for the PTP-1588 Timestamp Implementation



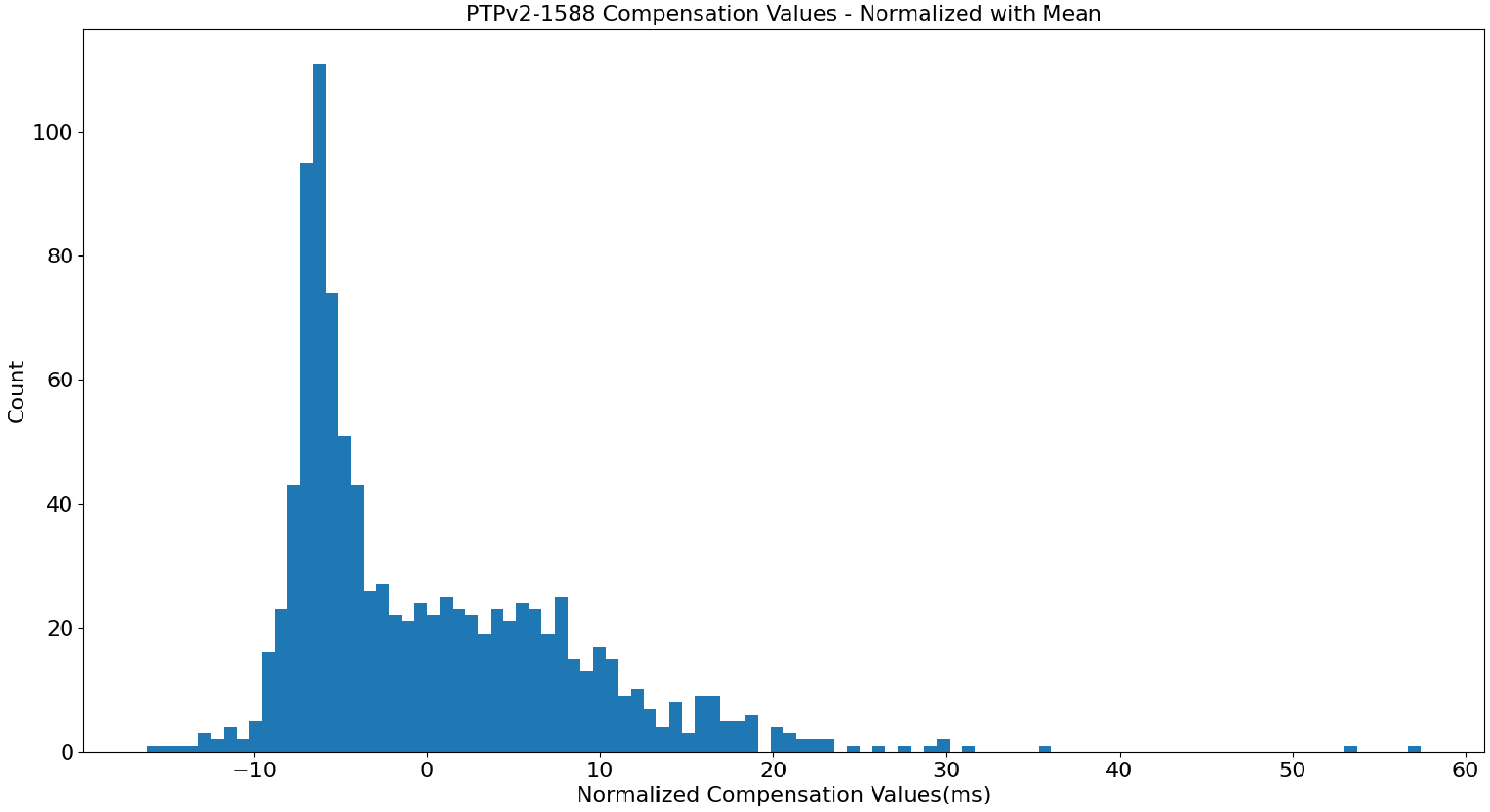
Distribution of Normalized Compensation Values for PTPv2-1588 Synchronization using Standard Timestamps

- All compensation values
- Normalized with mean

Max value = 57 ms
Min value = -16 ms

Range = 73 ms

Standard deviation = 12950

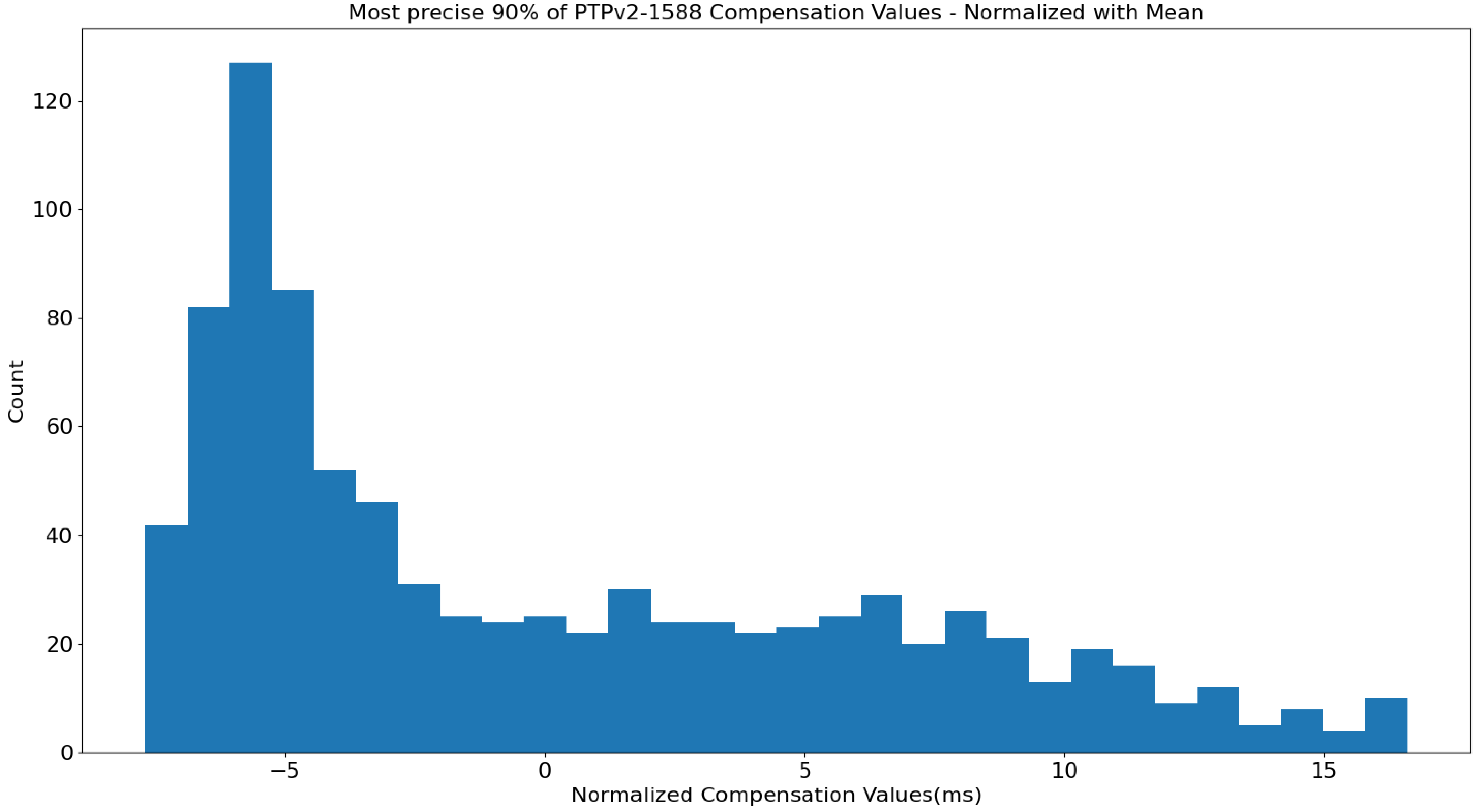


Detailed View of the Most Precise PTPv2-1588 Compensation Values using Standard Timestamps

- 90th percentile compensation values
- Normalized with mean

Max value = 16 ms
Min value = -8 ms

Range = 24 ms



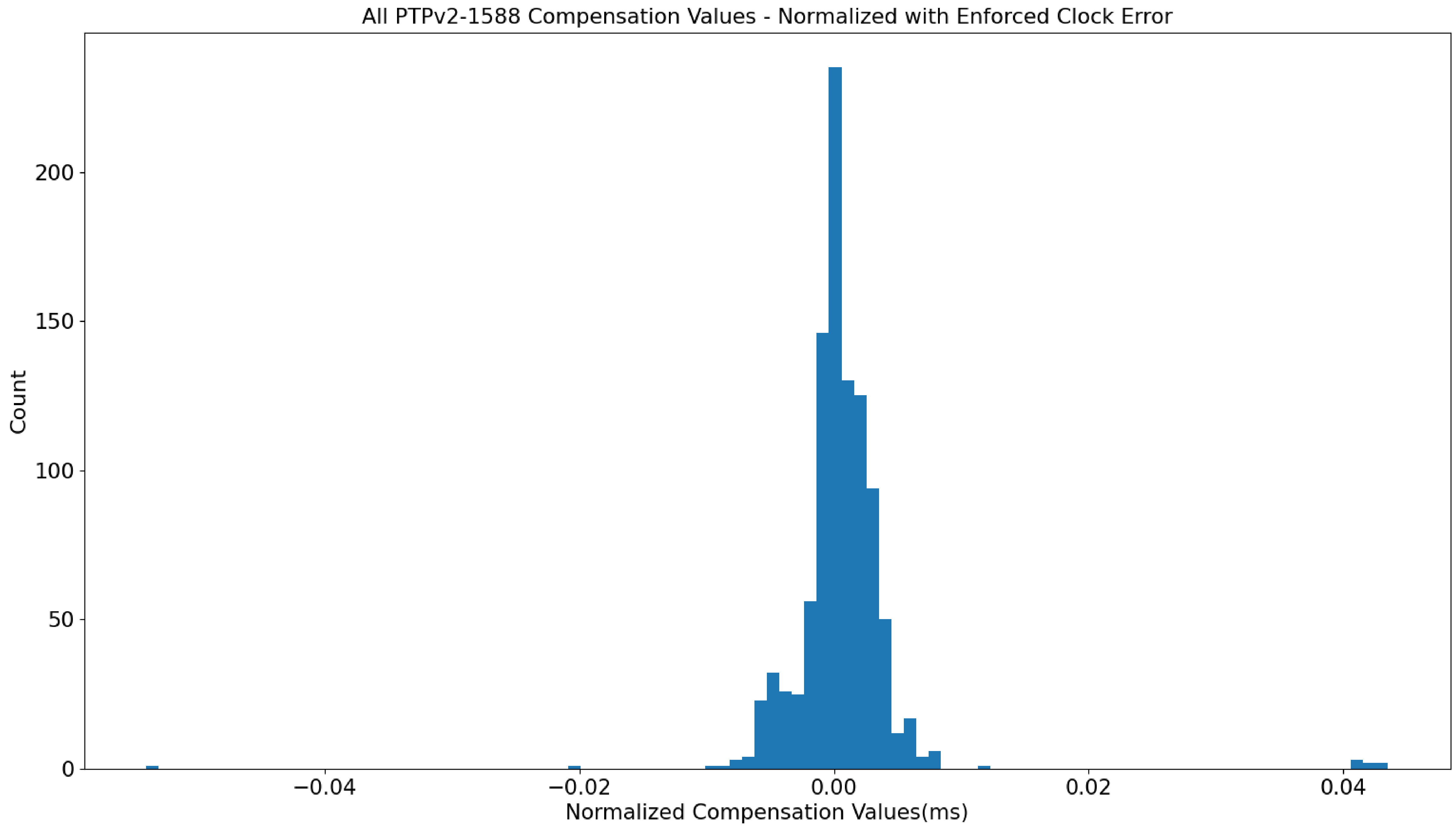
Distribution of Normalized Compensation Values for PTPv2-1588 Synchronization using New Timestamps

- All compensation values
- Normalized

Max value = 0.04 ms
Min value = -0.05 ms

Range = 0.09 ms = 90 µs

Standard deviation = 4723

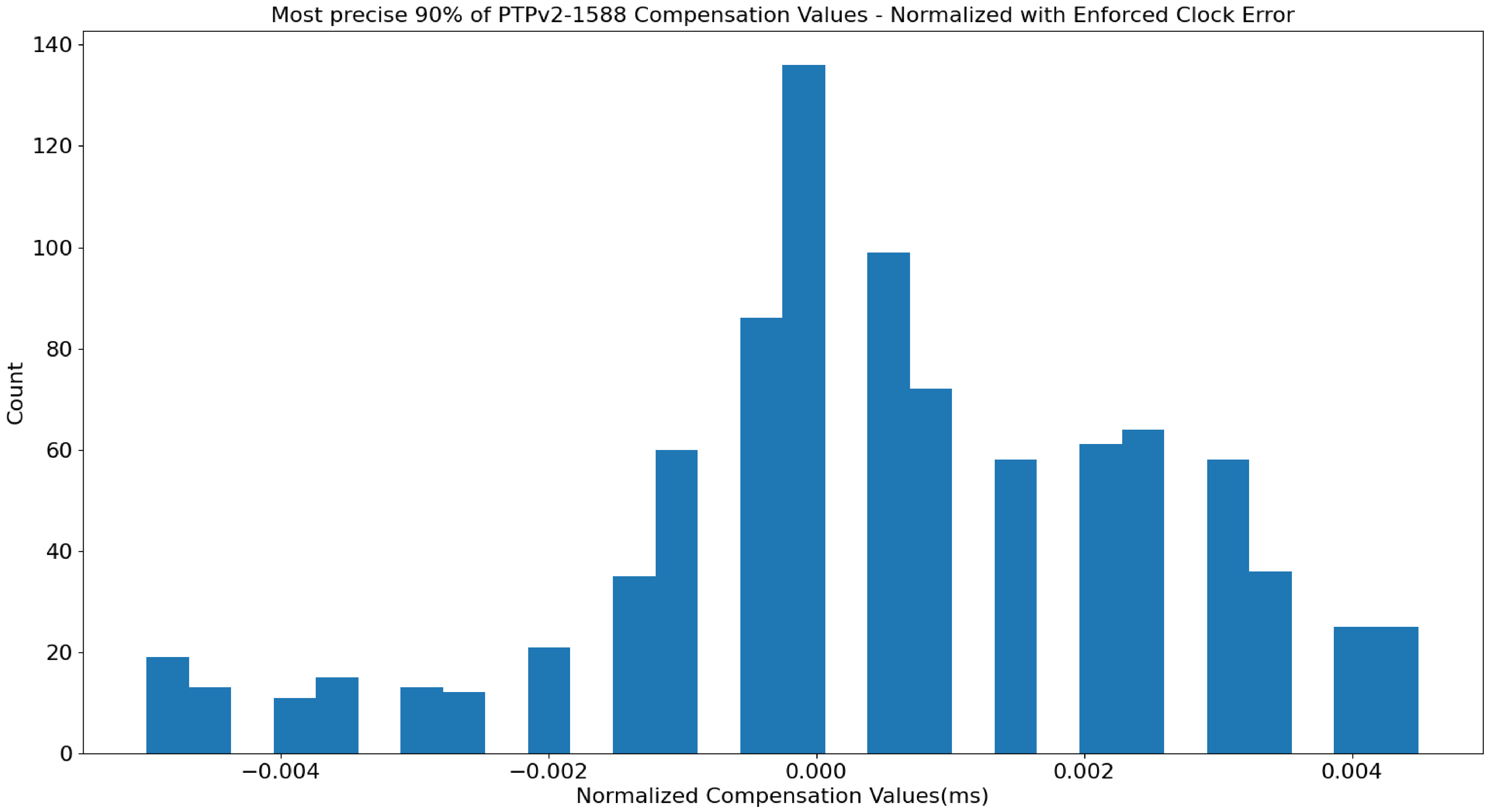


Detailed View of the Most Precise PTPv2-1588 Compensation Values using New Timestamps

- 90th percentile compensation values
- Normalized

Max value = 0.004 ms
Min value = -0.005 ms

Range = 0.009 ms = 9 μs



PTP-1588 New vs Standard Timestamp Implementation:

Standard Timestamp Implementation:

- Timestamps and measurements in µs.
- The 90th percentile of values lies within a **24 ms** range:
- Unable to enforce a specific clock difference value, which limits design verification.

New Timestamp Implementation:

- Timestamps and measurements in ns.
- The 90th percentile of values lies within a **9 µs** range.
- Allows enforcement of a specific clock difference value, enabling full design verification.

Conclusion

- **Improved Timestamp Precision in BMv2**: Enhanced the timestamp accuracy in the BMv2 switch, reducing timing errors from hundreds of milliseconds to tens of microseconds, critical for supporting time-sensitive applications.
- **Versatile Timestamping System**: The new timestamp system is flexible and can be applied to other synchronization protocols beyond PTP, broadening its utility across various network technologies.
- **Future-Proof BMv2 Structure**: Our modifications to BMv2 make the codebase more modular and adaptable, providing a robust foundation for future developments in precision timing and network innovations.

Thank You