INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Monitoring the Science, Technology and Society (RCTS) Network using perfSONAR

Edgar Cerqueira Inácio

Licenciado

Dissertação para obtenção do Grau de Mestre
em Engenharia de Eletrónica e Telecomunicações

Orientadores : Prof. Nuno Miguel Abreu Luís
Prof. Luis Miguel Pires

Júri:

Presidente: Prof. Rui António Policarpo Duarte

Arguente: Prof. Pedro António Marques Ribeiro

Julho, 2024

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Monitoring the Science, Technology and Society (RCTS) Network using perfSONAR

Edgar Cerqueira Inácio

Licenciado

Dissertação para obtenção do Grau de Mestre
em Engenharia de Eletrónica e Telecomunicações

Orientadores : Prof. Nuno Miguel Abreu Luís
Prof. Luis Miguel Pires

Júri:

Presidente: Prof. Rui António Policarpo Duarte

Arguente: Prof. Pedro António Marques Ribeiro

Julho, 2024

Aos meus ...

Acknowledgments

Ao refletir sobre a jornada que me trouxe até este momento, sinto uma profunda gratidão que transcende palavras. Cada pessoa que cruzou o meu caminho e cada experiência vivida contribuíram para moldar não apenas esta dissertação, mas em quem me tornei. Permitam-me expressar minha mais sincera apreciação a todos que os que foram uma parte essencial desta viagem.

Inicialmente, quero expressar a minha sincera gratidão ao Instituto Superior de Engenharia de Lisboa (ISEL), que não só me acolheu mas também proporcionou um ambiente enriquecedor para o meu crescimento acadêmico. Um agradecimento especial aos meus orientadores, Miguel Luís e Luís Pires, cuja sabedoria, paciência e orientação foram fundamentais durante toda a jornada deste projeto.

À Unidade FCCN da FCT, o meu sincero agradecimento por me terem permitido levar a cabo este projeto. A oportunidade de trabalhar num ambiente tão estimulante e inovador foi, sem dúvida, um privilégio.

Não posso deixar de mencionar o inestimável apoio da minha equipa de ASR na FCCN. A todos os membros que me acolheram e apoiaram no meu desenvolvimento tanto pessoal quanto profissional e que se tornaram uma segunda família para mim, Pedro Lorga, Clayton Verlique, Sergio Gonçalves, Gonçalo Lopes, Tiago Antunes, Esmeralda Pires, João Guerreiro e Filipe Santana, o meu mais caloroso agradecimento. Um reconhecimento especial ao Emanuel Massano e ao João Silva, pela sua assistência técnica inestimável e por sempre estarem disponíveis quando mais necessitei. À nossa diretora, Ana Pinto, agradeço especialmente pela liberdade e confiança depositadas em mim, permitindo-me aceder aos recursos necessários sempre que necessário.

Aos meus amigos e companheiros de faculdade - Curinha, Shiko, Soares, Mafalda, Guilherme, Nuno, Caracóis e, particularmente, ao meu caro André Alves, o meu braço

direito neste Mestrado - a nossa irmandade foi o pilar que sustentou este desafio. As longas noites de trabalho conjunto são momentos que guardarei eternamente.

Aos meus amigos mais próximos, que se tornaram a família que escolhi para mim, Gonçalo Silva, Rodrigo Moreira, Hugo Gomes, Daniela Calado e Beatriz Freitas o meu eterno agradecimento pela força e apoio constantes. Ao Gonçalo Moreira, a tua irmandade, amizade e lealdade inabalavel foram um fator decisivo nos momentos mais desafiadores da minha vida. A todos vocês um muito obrigado por demonstrarem que a verdadeira amizade é inabalável.

Contudo, acima de todos, o meu agradecimento mais profundo e sincero vai para a minha família. Aos meus pais Fernando e Vanda Inácio ao meu tio João Inácio e à minha querida avó Fatima Cerqueira, o vosso amor, sacrifício e apoio foram a minha fortaleza. Palavras falham em expressar a minha gratidão e amor. Vocês são a fundação da minha existência, e cada sacrifício, cada gesto de amor e apoio incondicional foram a força motriz por trás de cada passo que dei. À minha avó, que não está mais entre nós, sabe que o teu espírito e amor foram a minha luz guia mesmo nos momentos mais escuros. A Deus, agradeço por me ter dado a força necessária para superar cada desafio, iluminando o meu caminho com fé e esperança de forma a me tornar no homem que sou hoje.

Esta dissertação é mais do que um trabalho académico, é um mosaico de todas as pessoas que tocaram a minha vida, cada uma deixando a sua marca eterna no meu coração. A vocês, a minha mais profunda e sincera gratidão. Que possamos continuar a cruzar caminhos e a construir novas memórias juntos. Obrigado a todos por tornarem esta minha jornada inesquecível.

Abstract

Effective network monitoring is crucial for maintaining the integrity, performance, and reliability of any advanced networking infrastructure. As network demands grow and the complexity of network systems increases, continuous monitoring becomes a essential step to ensure seamless operations and service quality. Network monitoring enables real-time detection and troubleshooting of issues, helping to prevent potential disruptions that could impact users. By providing detailed insights into network performance, monitoring tools like Performance focused Service Oriented Network monitoring ARchitecture (perfSONAR) play a pivotal role in upholding Service Level Agreement (SLA) and ensuring that users receive consistent and reliable network services.

This dissertation details the implementation of the perfSONAR network across Portugal's higher education institutions, focusing on enhancing network performance monitoring and optimization. The project was structured in phases, starting with a Proof Of Concept (POC) to validate perfSONAR tests in a controlled environment, followed by an intermediate phase involving strategic placement of test-point servers in different geographical locations. The final phase expanded the network nationwide, offering comprehensive coverage for performance analysis.

Significant challenges were encountered and overcome, particularly in database management. The OpenSearch database initially struggled with shard capacity due to daily index creation. This issue was resolved by transitioning to datastreams, improving scalability and performance. Additionally, the integration of Grafana for data visualization played a key role in simplifying the interpretation of complex network data, facilitating more effective troubleshooting.

The project's potential for future enhancements includes expanding the range of perfSONAR tests and collaborating with entities like Gigabit European Academic Network

(GÉANT). One area of focus is developing tests for monitoring eduroam authentication processes. These advancements are expected to refine network performance monitoring and contribute to the field of network and data communications.

The successful deployment of the perfSONAR network marks a significant advancement in network infrastructure for Portugal's educational and research institutions. It enhances current capabilities and provides a scalable foundation for future technological developments. The project highlights the importance of strategic planning, innovative problem-solving, and adaptability in evolving network performance and optimization.

Keywords: perfSONAR, network monitorization, data visualization, network infrastructure, network diagnostics, network management.

Resumo

A implementação da rede perfSONAR nas instituições de ensino superior de Portugal representa um marco significativo no avanço da infraestrutura digital do país. Esta dissertação descreve de forma abrangente o processo de implementação da rede perfSONAR, um projeto que se destaca na interseção da inovação tecnológica, do planeamento estratégico e das necessidades dinâmicas da paisagem digital. O objetivo do projeto não foi apenas melhorar a monitorização e otimização do desempenho da rede, mas também estabelecer um precedente nos domínios académico e de pesquisa sobre como a infraestrutura digital pode ser eficazmente alavancada.

Diversos aspectos técnicos e de configuração foram fundamentais na implementação deste projeto. Adicionalmente, a integração do Grafana como ferramenta de visualização marcou um significativo avanço técnico. Esta plataforma open-source foi utilizada para criar dashboards interativos, melhorando a visualização e interpretação das métricas de desempenho do perfSONAR. O processo de implementação envolveu a configuração da base de dados OpenSearch e integração de um plugin de visualização Matrix no sistema Grafana existente na FCCN. Estes passos conduziram ao desenvolvimento de dashboards dinâmicos que proporcionaram uma interface amigável para os administradores de rede, simplificando a interpretação de dados de rede complexos.

A implementação do perfSONAR em Portugal, desde o seu conceito até à implementação nacional, serve como um marco importante para o avanço da qualidade da rede de inovação e tecnologia Portuguesa oferecida pela FCCN. O projeto foi meticulosamente executado em fases distintas, cada uma com o seu foco específico e conjunto de desafios. A fase inicial, a Prova de Conceito (POC), estabeleceu as bases para o sistema de monitorização de desempenho da rede. Esta fase foi crucial na demonstração da viabilidade e impacto potencial da rede perfSONAR, validando os conceitos subjacentes ao projeto.

Posteriormente, o projeto avançou para uma fase intermédia, marcada pela distribuição estratégica de servidores de pontos de teste em várias localizações geográficas. Esta etapa foi instrumental para ampliar o escopo do projeto, permitindo uma avaliação mais abrangente do desempenho da rede em diferentes ambientes e configurações.

A fase final, a implementação nacional, foi um empreendimento monumental que estendeu o alcance da rede para cobrir o espectro educacional de Portugal. Esta fase caracterizou-se não só pelos seus feitos técnicos, mas também pela demonstração da escalabilidade e adaptabilidade do projeto. Permitiu uma coleta extensiva de dados e percepções, proporcionando um entendimento profundo da eficiência e saúde da rede em todo o país.

Ao longo do projeto, vários desafios foram encontrados, especialmente na gestão de bases de dados e coordenação logística. Os desafios iniciais com o sistema de base de dados OpenSearch, particularmente as limitações de capacidade dos seus fragmentos, apresentaram obstáculos técnicos significativos. A solução inovadora, envolvendo uma mudança para uma metodologia de fluxos de dados mais eficiente, melhorou significativamente o desempenho e a escalabilidade da base de dados.

Em termos de desafios logísticos, a implementação nacional exigiu planeamento e execução cuidadosos. Cada implementação de servidores de pontos de teste perfSONAR em várias regiões teve de considerar desafios geográficos, infraestrutura de rede local e requisitos institucionais. A execução meticulosa de cada configuração de ponto de teste foi um testemunho do compromisso do projeto com a precisão e eficiência.

A dissertação fornece uma análise comparativa do perfSONAR com outros sistemas de monitorização implementados na FCCN. Destaca as vantagens do perfSONAR em fornecer medições ativas detalhadas de métricas de desempenho de rede, como latência, perda de pacotes e taxa de transferência, e sua capacidade de automatizar testes de desempenho de rede. Isso representou um avanço significativo em relação às configurações manuais tradicionais de equipamentos de rede. A integração da Telemetria em Fluxo com o perfSONAR criou um quadro abrangente para monitorização de desempenho de rede, melhorando as capacidades gerais de detecção de falhas e mitigação da rede.

O impacto e os resultados da implementação da rede perfSONAR foram significativos, melhorando a monitorização e otimização do desempenho da rede nos setores educacional e de investigação em Portugal. A implementação bem-sucedida do projeto levou a uma infraestrutura de rede robusta e resiliente, crucial para as comunidades académicas e de pesquisa. A implementação nacional, em particular, não foi apenas uma conquista técnica, mas também um testemunho da capacidade do projeto de se

adaptar a ambientes e demandas de rede diversos.

Olhando para o futuro, o potencial de expansão dos testes perfSONAR é considerável. As colaborações com organizações como a GÉANT são antecipadas para liderar o desenvolvimento de novos e inovadores testes. Um foco notável é o monitoramento e validação dos processos de autenticação do Eduroam. A integração desses novos testes com ferramentas avançadas como o Grafana espera-se que melhore ainda mais as capacidades de monitorização da rede. Isso permitiria a criação de visualizações detalhadas e personalizáveis que auxiliam na tomada de decisões e eficaz nas resoluções de problemas. A integração destes novos testes com ferramentas avançadas como o Grafana é esperada para melhorar ainda mais as capacidades de monitorização da rede, permitindo a criação de visualizações detalhadas e personalizáveis que auxiliam na tomada de decisões eficazes e na resolução de problemas.

A implementação bem-sucedida da rede perfSONAR em Portugal representa um avanço significativo na infraestrutura de rede para as instituições educacionais e de pesquisa do país. Este projeto sublinhou a importância do planeamento estratégico, da inovação na resolução de problemas e da adaptabilidade num domínio em rápida evolução como o da otimização e desempenho de redes. Refletindo sobre esta jornada, a dissertação destaca o poder transformativo da tecnologia, realçando as capacidades inerentes das comunidades académicas e de pesquisa. A implementação da rede perfSONAR, desde a sua concepção até à implementação em todo o país, serve como um testemunho do espírito colaborativo, da competência técnica e da abordagem proativa que são essenciais para o avanço da infraestrutura digital na era moderna.

Palavras-chave: perfSONAR, monitorização de rede, visualização de dados, infraestrutura de rede, diagnóstico de redes, gestão de redes.

Contents

List of Figures

List of Tables

Acronyms

BMCA Best Master Clock Algorithm. 23

CDMA Code Division Multiple Access. 18, 20

E2E End-to-End. 8, 23, 24, 25, 27, 28
ESnet Energy Sciences Network. 31, 82
EVPN Ethernet Virtual Private Network. 92

FCT Foundation for Science and Technology. 1
FLR Frame Loss Ratio. 12

GÉANT Gigabit European Academic Network. ix, xiii, 4, 25, 31, 49, 56, 81, 98
GNSS Global Navigation Satellite System. 16, 18, 19, 20, 23
GUI Graphical User Interface. 25, 26, 32, 33

HTTP Hypertext Transfer Protocol. 8, 29, 72, 79

ICMP Internet Control Message Protocol. 29, 39, 74, 90
IEEE Institute of Electrical and Electronics Engineers. 15, 20, 21
IETF Internet Engineering Task Force. 11
ILO Integrated Lights-Out. 45, 46, 77, 83
IPDV Inter-Packet Delay Variation. 12
iPerf Internet Performance Working Group. 26, 27, 38, 65
IPPM IP Performance Metrics. xv, 11, 14, 75, 76

ISEL Instituto Superior de Engenharia de Lisboa. 4, 75

KPI Key Performance Indicator. 8, 11, 32

LAN Local Area Network. 19, 25, 46, 47
LSR Lookup Service Registration. 34, 40

MA Measurement Archive. 33
MP Measurement Point. 39
MPO Multifiber Push On. 46
MTU Maximum Transmission Unit. 39

NLNOG Netherlands Network Operators Group. 25, 27, 28
NOC Network Operations Center. 25
NREN National Research and Education Network. 25, 31
NTP Network Time Protocol. 4, 14, 16, 17, 18, 19, 20, 24, 25

OSI Open System Interconnection. 20
OWAMP One-Way Active Measurement Protocol. 26, 39, 65

P2P Peer-to-Peer. 23, 24
perfSONAR Performance focused Service Oriented Network monitoring ARchitecture. ix, x, xi, xii, xiii, xvii, xix, xx, xxi, 2, 3, 4, 5, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 63, 64, 65, 67, 68, 70, 71, 72, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 97, 98, 99
POC Proof Of Concept. ix, xi, 4, 41, 46, 48, 75, 97
pScheduler perfSonar Scheduler. 32, 38, 53, 54, 65, 71
pSConfig perfSonar Config. 33, 34, 36, 37, 38, 52, 54, 77, 78, 84
PTP Precision Time Protocol. 4, 14, 20, 21, 22, 23, 24, 25

QoS Quality of Service. 11

RCTS Science, Technology and Society Network. 1, 2, 3, 4, 5, 40, 93
RIPE Réseaux IP Européens. 26, 27, 28

RIPE NCC Réseaux IP Européens Network Coordination Centre. 25
RNP Rede Nacional de Ensino e Pesquisa. 31
RTT Round-Trip Time. xx, 16, 20, 29, 39, 86, 90

SDO Standards Development Organizations. 24
SFP Small Form-factor Pluggable. 45, 46, 77
SLA Service Level Agreement. ix, 2, 3, 11
SNMP Simple Network Management Protocol. 20, 90, 91
SNTP Simple Network Time Protocol. 15
SR Short Reach. 46
SSH Secure Shell. 20, 29
SyncE Synchronous Ethernet. 15

TCP Transport Control Protocol. 15, 16, 29, 38, 39
TIG Telegraph, InfluxDB & Grafana. 92
TIME Time Protocol. 15
TSP Time Synchronization Protocol. 7, 14, 15, 16
TTL Time to Live. 39
TWAMP Two Wire Active Measurement Protocol. 39, 65

UDP User Datagram Protocol. 15, 16, 20, 38, 39
UTC Universal Time Coordinated. 17, 18, 19

WAN Wide Area Network. 19, 25

1

Introduction

The Science, Technology and Society Network (RCTS), operated by the FCCN (Computer Science Foundation) that is the scientific computing unit of the Foundation for Science and Technology (FCT), plays a crucial role in providing advanced networking services to the research and education communities in Portugal. As the demand for high-speed network services increases, ensuring the quality of the services provided by RCTS becomes increasingly important. This has led to the need for a comprehensive and continuous monitoring system that can validate the quality of services provided in real-time and enable network engineers to troubleshoot issues as they arise.

Modern digital infrastructure management and optimization rely heavily on network monitoring. In addition to giving an overview of the network's performance right now, it also offers a historical data trail very helpful to analyze trends and make long-term plans. Today's networks are complex webs of devices, services, and connections that require constant attention to maintain efficiency and operational continuity.

Picture a bustling metropolis, where traffic lights, cameras, and sensors work tirelessly to regulate the flow of vehicles and ensure the safety and efficiency of the roadways. In much the same way, network monitoring tools stand as the digital sentinels of our interconnected world, meticulously observing the flow of data packets as they traverse the vast expanse of the network.

One of the main advantages of network monitoring is its capacity to anticipate problems before they become ones that could negatively affect services, especially in research and education networks like the RCTS.

Just as sudden traffic bottlenecks can disrupt the flow of vehicles on city streets, anomalies in network traffic – such as unexpected spikes in usage or unusual patterns of data transmission – can signal potential issues or security threats. Network monitoring tools excel at identifying these anomalies, allowing administrators to investigate and address them promptly.

This preventive approach is crucial in settings where service degradation or outages can seriously impair research projects and vital communications. Moreover, network monitoring makes resource optimization easier, guaranteeing that the network's infrastructure is not only reliable but also reasonably priced and expandable to accommodate future needs.

Moreover, comprehensive network monitoring supports the development of a knowledge base regarding network performance and issues. This repository of information becomes a powerful tool for network engineers and decision-makers, enabling them to make informed choices about upgrades, expansions, and policy changes. It also supports academic research into network technologies and methodologies, fostering innovation and progress.

This dissertation focuses on the development and implementation of a monitoring system for the RCTS network using perfSONAR. The main objective is to validate the efficacy of the perfSONAR tool for monitoring the RCTS network. The research focuses on assessing the capabilities of perfSONAR in simulating client traffic and detecting potential issues that could impact network performance. By continuously monitoring the network using perfSONAR, network engineers can quickly detect and troubleshoot issues, ensuring that the SLAs are met. The research will also investigate the challenges associated with the implementation of a perfSONAR-based monitoring system for a complex network like RCTS.

The licensed perfSONAR logo removed from the official website can be seen in the image displayed in the cover.

1.1 Objectives

The main objective of this dissertation is to deploy a robust network monitoring solution for the RCTS using perfSONAR. This objective is driven by the critical need to enhance the network's performance and reliability to support the diverse requirements of Portugal's research and education communities. The envisioned monitoring system aims to offer a comprehensive, real-time view of the network's health, performance metrics, and potential bottlenecks, thereby enabling network engineers to promptly

identify and rectify issues. By simulating client traffic and analyzing the network's response, the system seeks to validate the quality of services provided by RCTS continuously. This proactive approach to network management is expected to not only ensure adherence to SLA but also to contribute significantly to the optimization of network resources, thereby facilitating a seamless experience for end-users. Furthermore, this dissertation will explore the technical and operational challenges encountered during the implementation of a perfSONAR-based monitoring solution in a complex and dynamic network environment like RCTS, offering valuable insights into best practices and innovative solutions.

1.2 Contributions

The implementation of the perfSONAR monitoring system within the FCCNs RCTS network infrastructure has yielded significant contributions to both the network's operational efficiency and the broader network monitoring domain:

1. **Enhanced Network Performance and Reliability:** Through the deployment of perfSONAR, this project has directly contributed to the improvement of network performance and reliability for the research and education communities in Portugal. The monitoring system's ability to detect, analyze, and respond to network issues in real-time has dramatically reduced downtime and service disruptions, ensuring a more reliable and consistent network experience.

2. **Scalability and Future-Proofing:** The architecture and implementation strategies developed for the perfSONAR monitoring system are inherently scalable and adaptable. This contribution is particularly valuable as it not only meets the current demands of RCTS but also accommodates future expansions and technological advancements, thereby future-proofing the network's monitoring capabilities.

3. **Innovative Solutions to Monitoring Challenges:** The project has identified and addressed numerous challenges while deploying a monitoring system across a complex national network. These challenges include optimizing database performance under high data volumes and integrating advanced visualization tools like Grafana for more intuitive data analysis. The solutions developed in response to these challenges represent significant contributions to the field of network monitoring, offering a reference model for similar future initiatives.

1.3 Methodology

In order to solve the problems listed above, we started by putting together a POC composed of two probes equipped with 10 Gbit/s network cards and two with 100 Gbit/s network cards. This was done in order to validate and test the hardware compatibility, find and fix any problems, and come up with baseline metrics that we can use to compare to future results. After the initial testing were concluded, we began the implementation of the perfSONAR system. Following the advice of GÉANT experts, the new version 5 was used for the core system, while the older version 4 was still used for the software on the probes.

Due to the limited number of probes, the mass distribution of probes to client institutions will be organized by institution priority and size, beginning with the layer three nodes of the FCCN network, which are locations where the main routers can be encountered. Secondly, the public education institutes, and thirdly, the government laboratories. However, even before the mass distribution, an intermediate step was taken in which three probes were implemented, one in a FCCN location in the city of Lisbon, other inside Instituto Superior de Engenharia de Lisboa (ISEL) and the third one in a FCCN location in the city of Porto, in order to test and validate the chosen solution.

1.4 Document Structure

This document is divided into multiple chapters that combine together in order to paint the final picture of the implementation of perfSONAR within the RCTS network.

- **Introduction:** The opening chapter, sets the stage for the investigation ahead. Within this section, the objectives of the study are explained. The methodology section outlines the strategic approach employed, monitoring strategies, and troubleshooting methodologies essential to achieving the study's objectives.

- **State of the Art:** In the "State of the Art" chapter some of the existing research and products of network monitoring are presented and explained. Here, we explore the fundamental distinctions between active and passive monitoring, while also dissecting critical time synchronization protocols like Network Time Protocol (NTP) and Precision Time Protocol (PTP). Additionally, a review of related work provides context and insights garnered from prior research efforts in perfSONAR and network monitoring systems.

- **perfSONAR:** Advancing into the core of this dissertation, this chapter offers a deep dive into the architecture and functionalities of the perfSONAR tool. We navigate through its various application layers, from visualization and archiving to configuration, scheduling, measurement tools, and discovery.
- **perfSONAR Implementation:** The subsequent chapter "perfSONAR Implementation" elaborates on the practical implementation of perfSONAR within the RCTS network environment. Detailed descriptions of the network architecture, hardware configurations, and connectivity solutions provide a better understanding of the technical aspects.
- **Conclusions and Future Work:** This document then culminates with a "Conclusions and Future Work" that as the name suggests reflect upon the work developed and in what can be done in the future.

2

State of the Art

The following chapter of this dissertation provides an overview of the current state of research related to network performance measurement and analysis. It also examines the two most commonly used Time Synchronization Protocols (TSPs) and their importance in performance metrics. It also discusses some related work that has been done in the field of network performance measurement and analysis, focusing on research papers, tools, and technologies. The goal of this chapter is to lay the groundwork for understanding the different approaches and techniques used in network performance measurement, analysis, and diagnosis by looking at the current state of the art. This will help guide the rest of the project's implementation.

2.1 Metrics

In the context of networks, metrics are quantifiable criteria used to rank the quality of various parts of a network and the efficiency of its activities. Latency, throughput, packet loss, network availability, congestion, jitter, are some possible examples of such measures.

Metrics are a straightforward method of quantifying and analyzing network behavior to establish whether or not it is up to par with the desired standards. They aid in problem detection and diagnosis, enhancement of network performance, and the making of educated infrastructure management and design decisions for networks.

Network monitoring software, traffic analysis tools, network analyzers, and performance testing tools are just some of the tools and approaches that may be used to assess and keep an eye on Key Performance Indicator (KPI). They are crucial for keeping the network running smoothly and making sure it is accommodating users and the company's requirements.

2.1.1 Active vs Passive Monitoring

Active and passive network monitoring serve distinct functions. Active monitoring revolves around anticipating network performance and issues by simulating End-to-End (E2E) network behavior, assisting network administrators in predicting the behavior of actual network users. The executions can happen instantly or at predetermined intervals set by the system administrator [1].

The goal of active monitoring is to get a full view of the network in real time so that problems can be found before they affect active users. Active analysis works best when a specific statistic, like packet loss, jitter, Hypertext Transfer Protocol (HTTP) response time, or latency, is being looked at.

The primary advantage of active monitoring is total network awareness and the ability to eradicate blind spots immediately. In addition, you may measure the effect of newly-integrated hardware devices on network performance and focus your study on problematic network segments [2].

As active monitoring is based on predictive data, it does not always provide an accurate accounting of network performance. It is most effective when evaluating a single measure, but it cannot examine every component of your network simultaneously. As a result of the ongoing data analysis, it can also deplete network resources [1].

Pros of active monitoring:

- Predictive network troubleshooting methodology;
- The simulation of user behavior;
- Full visibility in real-time into the network;
- Allows you to identify possible problems before they impact users;

Cons of active monitoring:

- Resource-intensive;

- Based on predictive data, so not always accurate;

Passive monitoring, on the other hand, collects and analyzes actual user's data over certain time periods by using specific network connections. The data can also provide a more comprehensive perspective on a network's performance and encompass a broad range of measures. This technique ends up gathering and generating even more performance data than active monitoring.

As real-time user data is collected through passive monitoring, network administrators are made aware of problems that affect end users directly. Instead of making changes based on predictions, administrators get alerts about situations that need immediate attention. This helps them figure out what is really affecting customers and the quality of the user experience. It is especially useful for post-incident analysis and communication to determine what went wrong.

Since there is more time between tests with passive monitoring, it puts less strain on the network than active monitoring. Passive monitors often analyze traffic to and from a particular device, necessitating specialized hardware.

Pros of passive monitoring:

- Retrieves actual user data from selected network nodes;
- Collects voluminous performance data;
- Provides a comprehensive perspective of the performance of a network using a variety of measures.

Cons of passive monitoring:

- Identified issues affect genuine users and must be resolved promptly;
- As the complexity of your network increases, you must continually upgrade your passive monitors to reflect their evolving nature.

2.1.2 Active Metrics and Methods

Active metrics are metrics that incorporate one or more of the aspects of what are called "active methods", which are responsible for generating new packet streams. Typically, the packet stream generated serves as the measuring foundation and may have a known source and destination. Due to this fact, the adjective "synthetic" can

be employed a different number of times to classify active measurement streams. The created streams may be used only as "fill-in traffic" in order to enhance the total traffic load present on the network.

Since measurement typically requires detecting the relevant packets at different measurement locations, the packets in the intended stream usually have measurement-specific fields or field values (or are augmented or modified to include measurement-specific fields or field values). The sequence number is the most common added field to encounter, and it is often paired with a timestamp.

When using active methods and metrics, normally at least at the source, the properties of the packet stream of interest created are known. This information may or may not be conveyed to the destination as part of the technique. Since new traffic is being created, Active Methods impact the overall load on the network, which should be taken into account when performing tests where such extra load could be a detriment to the results obtained.

2.1.3 Passive Metrics and Methods

Passive metrics are all metrics that can be described by using one or more characteristics of passive methods, just like active metrics can be described by using one or more characteristics of active methods. These passive methods are based on watching the packet streams that are already on the network. This means that packets should not be added, taken away, or changed in any way during transmission. As a result, we are always reliant on the existence of packet streams on the network and for the packets to pass through the observation points in order to proceed with the measurements [3].

It is usual practice to undertake Passive Methods at one or more Observation Locations. Passive methods to evaluate performance metrics frequently require many observation points, for example, to evaluate the latency of packet transfer along a network path between two observation points. In this instance, the observed packets must contain sufficient information to identify the packets at each observation point.

For passive methods, the transmission of observations made to a collector is of the utmost importance. In some cases, sending (or exporting) the passive method results to a collector may increase the amount of traffic on the network being measured, creating an interference in the results. However, as mentioned with active methods and metrics, the collection of results is not exclusive to passive techniques, and the load from administration and operations of measurement equipment must always be considered as a possible influence on the observed values.

Some passive methods simply monitor and collect information on all packets that pass the observation point(s), whilst others filter the packets first and only collect information on packets that fit the filter criteria, narrowing the stream of interest.

2.1.4 QoS and IPPM

Quality of Service (QoS) and IPPM measurements play crucial roles in network design and maintenance. QoS [4] metrics are used to make sure that predetermined benchmarks for network performance are fulfilled, whereas IPPM is a collection of protocols and standards developed by the Internet Engineering Task Force (IETF) IPPM Working Group for measuring network quality, performance, and reliability. These measurements are intended to be used by network administrators, end users, and independent testing organizations. The data they generate is intended to provide objective and quantitative performance measurements [5].

The goals of the IPPM Working Group's and the metrics being developed were first defined an can be reviewed on the RFC 2330 (Framework for IP Performance Measurements, May 1998) [6]. This RFC provides guidelines for IPPM and fundamental ideas, including metrics, measurement methodology, and other concerns. Further explanations on how to monitor fundamental Internet connectivity can be found on the RFC 2498 (IPPM Metrics for Assessing Connectivity, September 1999) [7].

QoS indicators, such as latency, throughput, and packet loss, are employed to guarantee that the network delivers as promised. Between service providers and customers, these KPI are typically outlined in SLA. Network performance is measured by IPPM measures like latency, packet loss, and jitter, and these metrics can be used to check if certain QoS metrics are being satisfied.

For instance, if a QoS metric specifies a maximum delay of 50 milliseconds, network engineers can utilize IPPM metrics to measure network latency and verify it is within the threshold. Similarly, if a QoS metric specifies a maximum packet loss of 0.1%, network engineers can utilize IPPM metrics to measure packet loss and ensure it falls below the threshold [8].

Listed below are some of the key IPPM metrics that have been established by the IETF:

1. Availability: This metric measures the amount of time a network or service is accessible. Network unavailability, equipment failure, and maintenance windows can impact availability.

2. Burst loss: This statistic is used to determine the number of data packet transmission bursts and gaps.

3. Data corruption ratio: This metric indicates the proportion of transmitted data that is corrupted.

4. Data loss ratio: This metric quantifies the proportion of transmitted data that is lost.

5. Duplicate packet ratio: This metric indicates the proportion of duplicate packets that are received at the destination.

6. Error rate: This metric indicates the proportion of sent packets that contain errors. Many variables, including network congestion, equipment failure, and packet errors, can contribute to error rate.

7. Frame Loss Ratio (FLR): This metric measures the percentage of frames or packets that are lost during transmission.

8. Inter-Packet Delay Variation (IPDV): This metric quantifies the delay variance between packets over a certain time interval.

9. One-way Available Bandwidth: Measures the amount of bandwidth that is available between the source and destination for one-way traffic.

10. One-way Delay: Measures the time it takes for a packet to travel from its source to its destination.

11. One-way Delay Variation (jitter): Measures the variation in delay between packets as they travel from the source to the destination.

12. One-way Loss Pattern: Measures the pattern of packet loss for one-way traffic.

13. One-way Delay and Loss Distribution: Measures the distribution of delay and loss for one-way traffic.

14. One-way Packet Loss: Measures the percentage of packets that are lost during transmission from the source to the destination.

15. Packet reordering: This metric indicates the degree to which packets are received out of sequence at the destination.

16. Round-Trip Available Bandwidth: Measures the amount of bandwidth that is available between the source and destination for round-trip traffic.

17. Round-Trip Delay: Measures the time it takes for a packet to travel from its source to its destination and back again.

18. Round-Trip Delay Variation (jitter): Measures the variation in delay between packets as they travel from the source to the destination and back again.

19. Round-Trip Loss Pattern: Measures the pattern of packet loss for round-trip traffic.

20. Round-Trip Delay and Loss Distribution: Measures the distribution of delay and loss for round-trip traffic.

21. Round-Trip Packet Loss: Measures the percentage of packets that are lost during transmission from the source to the destination and back again.

22. Throughput: This statistic indicates the amount of data that can be sent over a network in a given time period. Throughput can be affected by network congestion, transmission errors, and equipment constraints.

23. Utilization: This metric measures the percentage of available network capacity that is being used. Utilization can be affected by network traffic, equipment limitations, and network design. This indicator measures the proportion of network capacity that is being utilized. Network traffic, equipment limits, and network design can impact utilization.

A different number of solutions can perform network measurements, however a more advanced solution like the one used in this project is necessary when the goal is to perform correct one way delay measurements. One-way metrics are essential for network monitoring and diagnostics as they provide particular information about the network's performance in a single direction, whereas round trip measurements provide information about the network's performance after the package travels both ways. Some of the key benefits of using one way metrics are:

- More Accurate Measurements: One-way metrics provide more precise measurements of network performance since they do not account for the time required for a packet to return to its source. Delays and congestion in the opposite direction might impair round trip measures, resulting to erroneous results.

- Identify One-Way Issues: One-way metrics enable network administrators to spot issues that affect only one direction of the network, such as congestion or

latency problems. Metrics based on round trips can obscure one-way difficulties, making it more difficult to identify and diagnose the fundamental cause of a problem.

- Optimizing Network Performance: One-way metrics offer network administrators precise information about the network's performance in one direction, allowing them to optimize network performance in that direction. This can help enhance the performance and user experience of an application.
- Troubleshooting: One-way metrics can assist network managers in more efficiently solving network issues by providing precise information about the network's performance in one direction. This can reduce the amount of time required to identify and remedy network performance issues.

Time synchronization protocols play an essential part in IPPM by guaranteeing that precise and consistent timestamps are used to measure various network performance indicators. For measuring measures like as packet delay, jitter, and packet loss, which are vital for evaluating network performance, precise timestamps are required. If the clocks on various network devices are not synced, for instance, the time-stamps on packets will not precisely reflect the time they were sent or received. This importance is augmented especially when one way metrics are being used, caused by the fact that one way metrics start in the source device and end in a different destination device, since this source and destination diverge, without a proper TSP to synchronize the clocks of the different machines involved, the measurements results will become incorrect.

2.2 Time Synchronization Protocol

Protocols regarding time synchronization are an integral component of many networking and communication systems. By guaranteeing that all devices share a consistent notion of the present time, these techniques enable devices to coordinate their actions and exchange data. There are a variety of time synchronization protocols in use today, each with its own set of distinct characteristics and capabilities.

NTP and PTP are examples of some of the most extensively used time synchronization systems. In addition to NTP and PTP, a variety of other time synchronization protocols are utilized in certain applications or contexts. Specifically developed to synchronize the clocks of network devices via telecommunication networks is the Protocol for the

Synchronous Ethernet (SyncE) [9], which is defined by the Institute of Electrical and Electronics Engineers (IEEE) 1588 standard. In simpler networking setups, protocols such as the Simple Simple Network Time Protocol (SNTP) [10] and the Time Protocol (TIME) [11] are used for basic time synchronization. Example of a SNTP communication:

1. A client sends a request to a server for the current time.
2. The server receives the request and sends a response back to the client.
3. The response contains the current time according to the server's clock.
4. The client receives the response and adjusts its clock to match the server's clock.

Time synchronization techniques are an integral part of many networking and communication systems since they enable devices to coordinate their actions and transmit information consistently and reliably, becoming essential for ensuring the appropriate operation of a vast array of applications.

2.2.1 Time Synchronization Protocol UDP or TCP

In numerous applications, including telecommunications, financial transactions, and industrial control systems, accurate timekeeping is crucial. The choice of transport protocol for time synchronization messages is one of the important architectural considerations made during the creation of TSP. For this purpose, TSP primarily employ User Datagram Protocol (UDP) [12] rather than Transport Control Protocol (TCP) [13].

UDP is a connectionless protocol, while TCP is a connection-oriented protocol. This is one of the primary reasons for this choice. Before data can be shared in a connection-oriented protocol, a link must be established between the two devices. This raises the communication process's overhead and the system's delay. UDP, on the other hand, does not establish a connection prior to data exchange, making it a faster and more effective protocol for time synchronization.

TSPs require low-latency and high-precision time synchronization, which is another essential factor in the use of UDP. TCP, on the other hand, is intended to guarantee dependable data delivery, and it contains techniques such as flow control and retransmission that can increase latency and decrease time synchronization precision. In addition, TSPs frequently use multicast or broadcast messages to simultaneously synchronize many devices, which is not possible with TCP. UDP supports multicast and

broadcast, making it a more appropriate protocol for TSPs. TSPs are frequently utilized in circumstances where the network is unstable or lossy. In such instances, TCPs retransmission methods can result in additional delays and diminished performance. UDP, on the other hand, lacks retransmission capabilities, making it more resilient in such situations.

In conclusion, the need for low-latency, high-precision time synchronization, capability for multicast and broadcast, and robustness in unreliable networks motivates the usage of UDP in TSPs. TCP may be more suited for other types of applications, but it is not well suited for time synchronization.

2.2.2 Network Time Protocol

The NTP is one of the most popular time synchronization technologies. NTP is an extremely precise protocol designed to synchronize the clocks of devices on a network to within a few milliseconds [14]. It employs a hierarchical structure, with a small number of very accurate reference servers giving time information to all other servers and clients. NTP is highly adaptable and can function over a wide range of transport layers and network designs. Typically, these reference servers are linked to external time sources, such as Global Navigation Satellite System (GNSS) or atomic clocks, in order to maintain the highest level of precision.

Clients on a network synchronize their clocks with reference servers by transmitting and receiving NTP messages. The protocol can modify the client's clock based on the RTT of NTP messages.

NTP's adaptability to changing network conditions is one of its most important characteristics. It is capable of operating over a variety of transport levels, including IPv4 and IPv6, and can employ either unicast or multicast communication. It is also meant to be scalable, with the capacity to accommodate a high number of clients and servers without degrading performance significantly. NTP is a crucial tool for assuring the precise and dependable synchronization of network devices. It is essential to a vast array of applications, including financial systems, power grids, and telecommunications networks. Its continual evolution and improvement will ensure that it remains a vital resource in the years to come.

2.2.2.1 NTP Operation

However NTP is not designed to synchronize computers with one another. It is based on the idea that all machines should be as close as possible to the correct time - Universal Time Coordinated (UTC). A simple NTP network consists of a time server and clients (workstations, routers, and so on).

The purpose of a time server is to provide clients with accurate time. Each client executes a small background software that periodically requests the server for a precise UTC time reference. These queries are executed at predetermined intervals (usually every 15 minutes) to maintain the network's required synchronization precision. The primary function of the NTP is to time stamp data packets transferred between the server and the client.

Order of operations:

1. The client stamps the time when he transmits an NTP request packet to the server.
2. The server stamps the time when it receives the NTP request packet from the client.
3. When the server transmits the NTP reply packet back to the client, he stamps the time.
4. The client stamps the time when it receives this NTP reply packet.

Therefore, an NTP packet contains four timestamps. The client utilizes these timestamps to calculate the difference between its local time and the UTC time reference and then changes its local time to match the reference. When adjusting its internal time, the client can additionally determine the network delay and apply a correction factor. The ability to eliminate network latency results in more precise synchronization, typically within a few milliseconds. However, not all delay can be eliminated without symmetrical paths from and to the server.

2.2.2.2 NTP Hierarchy

The typical hierarchical structure of the NTP protocol prevents several clients from accessing the same primary time sources. It is advised to follow this hierarchy so that a high number of clients will not be configured to overload a busy stratum 1 time server.

NTP uses the stratum notion to define the number of NTP hops between a machine and an authoritative time source. As an example, a stratum 1 time server is directly connected to a radio or atomic clock. It then sends its time through NTP to a stratum 2 time server, and so forth.. A machine that utilizes NTP automatically selects as its time source the machine with the lowest stratum number with which it is configured to interact using NTP. This technique effectively creates an NTP speaker self-organizing tree. NTP is able to perform successfully over the non-deterministic path lengths of packet-switched networks because it is able to make accurate predictions of the next three important variables that are involved in the relationship between the client and the time server.

There are four options for administrators who are constructing an NTP-based time infrastructure:

1. Invest in an NTP Server appliance to place in the first tier of servers. This is the least difficult method of establishing a perfectly accurate, trustworthy, secure, and independently functioning UTC-synchronized network.

2. Obtain an external time source, such as GNSS or Code Division Multiple Access (CDMA), in order to create a server of stratum 1 status. This external time reference is then associated with an existing server to create a stratum 1 time server. This technique will produce an accurate, dependable, autonomous, and secure UTC-synchronized network despite being more difficult to implement and configure.

3. Synchronize an internal NTP server with Internet-accessible servers, transforming it into a stratum 2 or 3 server. As with any externally delivered service, it also serves as an access point for attackers. Additionally, time obtained through the Internet is less precise. In secure contexts where time synchronization is crucial, the usage of a public time server is inadvisable.

4. Designate a machine as the time authority and use its internal clock as the source of arbitrary time. Nonetheless, when this time source drifts, any NTP clients connected to it will drift as well. Despite the fact that the primary clock may be manually adjusted to the correct time on occasion, this would cause all clients to experience a momentary hiccup when the server adjusted. If a clock is modified by more than 17 minutes, all NTP client software will terminate due to the abrupt time shift. This technique can still create a synchronized network and may be acceptable in a few exceptional instances, but in any major installation, it is essential to keep the clocks synced with a maintained time standard.

The following Figure 2.1 illustrates an example of a possible distribution of NTP stratums using the first option mentioned above that makes use of a NTP server appliance. It is possible to observe that the stratum 1 NTP servers synchronize with the external source (GNSS in this case), and then it proceeds to synchronize the dependents on the following stratum.

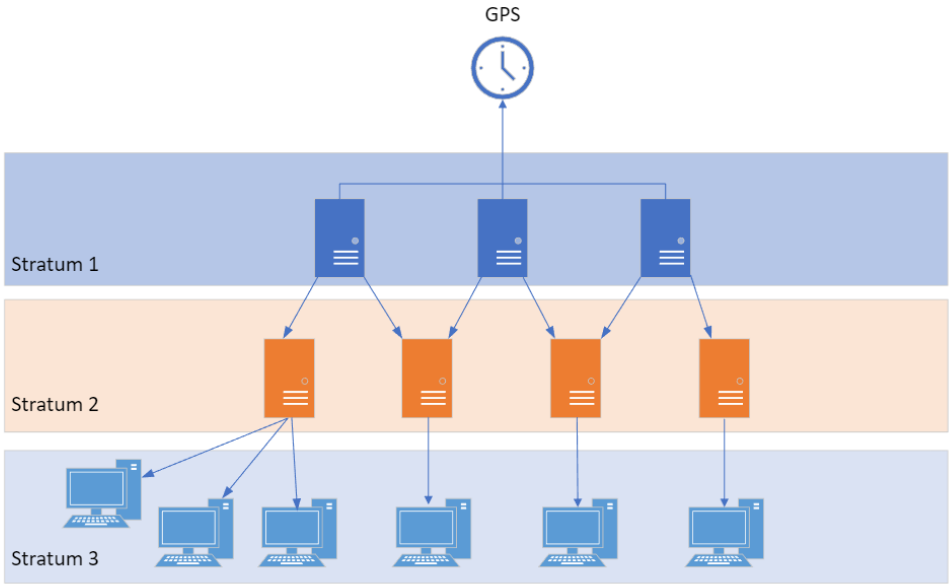

Figure 2.1: Diagram of a possible NTP Structure.

2.2.2.3 NTP Acccuracy

In terms of accuracy the degree of a server value depends on the option selected from the list above. The degree of synchronization between a client and a server is mostly determined by network delay. Everything that increases latency, such as hubs, switches, and routers, as well as network traffic, will impair accuracy. Under optimal conditions on a Local Area Network (LAN) with few sources of network delay, synchronization precision is normally within the range of a few milliseconds. In other words, all clients on the network will be synchronized to UTC and to one another to within half a millisecond and two milliseconds, respectively. Wide Area Network (WAN) synchronization precision typically falls between 10 and 100 milliseconds[15].

Internet synchronization precision is unpredictable, necessitating extra care when configuring a client to use public NTP servers. The final possible accuracy of each client is contingent upon the precision of the time server used, the network latency, and the symmetry of the network paths to the time server. Clearly, a client cannot be more precise than its server.

Several variables may also impact timing accuracy and offset/delay calculations like changes in the network protocol stack between the application level and the physical connection, asymmetric delays in propagation between client and server, clock oscillator stability and timestamp measurement errors. Each NTP server keeps track of its local clock offset and round-trip delay relative to the primary reference source at the base of the synchronization tree. When a client sends a NTP request, prior to sending a response with timestamps, the server adds the errors it has accumulated since its clock was last updated. Consequently, the clock offset and RTT rise as the distance from the primary reference source grows.

2.2.2.4 NTP Security

NTP is built on top of the UDP protocol. Therefore, it is extremely vulnerable to IP spoofing. NTP makes use of UDP port 123, which must be blocked at the firewall as a minimum for network perimeter security. This prevents a client from retrieving time from an Internet-accessible public NTP server.

When the time server is put within the network firewall, security is maximized. The time server acquires time via an antenna from the GNSS or CDMA system without compromising network security. From within the firewall, it then distributes the time to the different clients via the network.

To increase protection against threats contained within the firewall, the time server should restrict access to the service using the NTP access control and authentication features. Only authenticated NTP packets should be permitted if feasible. Additionally, the server should only accept packets from known, authorized sources. A secure protocol should be used, such as Secure Shell (SSH) and/or Simple Network Management Protocol (SNMP) v3, when connecting with the time server for status and management (encrypted SNMP). SNMP v1 and v2 are not secure.

2.2.3 Precision Time Protocol

The protocol IEEE 1588, better known as PTP, is a networking protocol used to synchronize clocks in a computer network. This protocol is intended to enable precise time synchronization across various networked devices. It is especially helpful in areas like telecommunications, finance, and industrial control systems where keeping accurate track of time is of the utmost importance [16][17].

PTP runs at the data link layer of the Open System Interconnection (OSI) model and employs a master-slave architecture, in which one device on the network is designated

as the master clock and all other devices are slaves whose clocks are synchronized to the master. The master clock sends time messages to the slaves, who subsequently change their own clocks based on the time information received. In order to measure the latency, PTP sends messages between the master and slave devices [18]. After that the communication path delay is calculated by the PTP protocol using the exact message transmit and receive times. PTP then updates the current time information in the network data to account for the delay that was calculated, resulting in more accurate time information.

This delay measuring principle detects path delay between network devices, and local clocks are updated to account for this delay via a series of messages delivered between masters and slaves. We can figure out the one-way delay time by taking the average of how long it takes to send and receive a message. This logic, however, applies only when we have a symmetrical path, which is nearly impossible to find in modern networks due to buffering processes. Because of this fact, PTP provides a mechanism that temporarily renders the switches transparent to the master and slave nodes of the network in order to measure and account for the delay in a time-interval field in network timing packets.

As of today, we have two major types of different implementations of the IEEE 1588: software-only implementations and software implementations with hardware assistance. In a software-only implementation, every step of the protocol, including the processing of timestamps, is implemented at the application level. As a result, the timestamps decrease in terms of accuracy , as a great number of errors are inserted into the timestamp. On the other hand Hardware-assisted solutions create and process timestamps as near as feasible to the transmission medium.

Conceptually, the functioning of PTP consists of two stages. During the first phase, the PTP clocks arrange themselves into a hierarchy, with the grandmaster clock at the top and the slave clocks at the bottom. There are both boundary and transparent clocks in the center of the hierarchy. In the second stage, protocol messages are exchanged to ultimately synchronize all clocks with the master clock.

PTP describes protocol messages as either event messages or general communications [19]. An event message is a communication that requires precise time-stamping upon transmission, reception, or both, such as:

- Sync Messages: Sync messages are utilized to synchronize the master and slave clocks' times. The master clock transmits a sync message at predetermined intervals, which the slave clock uses to correct its time. The sync message includes a timestamp that indicates when the message was sent.

- Delay_Req: The messages are used to measure the delay between the master and slave clocks. The slave clock sends a delay request message to the master clock, which then sends a delay response message with a timestamp indicating when the message was received.
- Pdelay_Req: PTP uses Pdelay request messages to determine the route delay between two clocks. These messages are transmitted between network boundary clocks.

We also have general messages that do not need to be time stamped:

- Announce: Announce Messages: The master clock sends announce messages to all slave clocks to broadcast the current time. The announce message includes a timestamp indicating the current master clock time.
- Follow_Up: These type provides further information regarding a sync message. A follow-up message could indicate, for instance, that a sync message was delivered in response to a request from a specific slave clock.
- Delay_Resp: The use of Delay_Response messages is to measure the delay between the master and slave clocks. The slave clock sends a delay request message to the master clock, which then sends a delay response message with a timestamp indicating when the message was received.
- Pdelay_Resp: In response to a PDelay request message, PDelay response messages are transmitted. The PDelay response message includes a timestamp that indicates both the time the message was received and the time the PDelay request message was delivered.
- Signaling: Signaling messages are used to configure and decommission PTP connection between clocks. These messages are transmitted during protocol initialization and fault recovery.

The following is an example of a possible message communication in PTP:

1. The grandmaster clock broadcasts its presence and provides timing information via an Announce message sent to all networked devices.
2. A slave clock gets the Announce message and makes the decision to synchronize with the master clock.

3. The grandmaster clock gets the *Delay_Req* message and sends a Sync message, providing its timestamp to the slave clock.

4. The slave clock receives the Sync message and calculates the offset between its clock and the grandmaster's clock using the timestamp in the Sync message and the delay time from the *Delay_Req* and *Delay_Resp* messages.

5. The slave clock sends a Follow_Up message to acknowledge receipt of the Sync message and update its clock offset.

6. The grandmaster clock periodically sends new Announce messages to update timing information and allow new devices to synchronize to its clock.

In addition to this structural sequence, extra *Delay_Req* and *Pdelay_Req* messages could be transmitted between slave clocks in order to measure Peer-to-Peer (P2P) delay periods and calculate offsets, as well as Signaling messages used for management and setup. Nonetheless, the sequence described above is the essential process of PTP based clock synchronization between a master and slave clock [20].

The PTP also specifies four primary types of devices: standard clocks, transparent E2E clocks, transparent P2P clocks, and boundary clocks.

An ordinary clock is called a slave clock if it keeps time in sync with another device, like a grandmaster or boundary clock. An ordinary clock is referred to as a "grandmaster clock" if it serves time to the entire PTP network and is therefore the definitive source of time for all other network devices. The grandmaster clock is the major source of time for PTP clock synchronization inside a PTP domain. Most of the time, the grandmaster clock gets its time from a GNSS or an atomic clock and gets selected by using the Best Master Clock Algorithm (BMCA). This algorithm sets up a hierarchy of clocks and lets slave clocks use the most accurate time on the network.

With the exception of slave-only clocks, ports on ordinary clocks and boundary clocks transmit Announce messages determining the clock priority and quality. Each clock on the network can use the BMCA and the clock attributes it gets through Announce messages to figure out the PTP state of each port, which is usually master, slave, or passive, and choose the best clock to synchronize to.

A boundary clock is a network device with more than one port that sends time to one or more ports and syncs with the time on one port. In other words, one of the ports is a slave port, while the rest are master ports. Essentially, boundary clocks stop and then begin time distribution. PTP-aware network components, such as switches, bridges, and routers, typically include this capability. For a PTP network to grow, boundary

clocks can be used to handle requests from slave clocks that the grandmaster clock would normally handle. This enables the network to accommodate a high number of slave clocks.

Transparent clocks are responsible for updating the time-interval field of PTP event messages. This update accounts for switch delay with an accuracy of one picosecond or less. In a PTP network, two types of transparent clocks can exist: E2E and P2P.

E2E transparent clocks measure the transit time of PTP event messages for SYNC and DELAY REQUEST messages [21]. This transit time is added to a data field (called the "correction field") in the SYNC message of the corresponding SYNC or the FOLLOW_UP message and is also added in the correction field of the DELAY_REQUEST message of the corresponding DELAY_RESPONSE message. This information is used by the slave to determine the time difference between the slave and master. The propagation delay of the link itself is not compensated for by E2E transparent clocks.

P2P transparent clocks track the time it takes for PTP event messages to get from one peer to another in the same way that E2E transparent clocks do. P2P transparent clocks additionally measure the time it is expected to take for a packet to travel from the upstream neighbor or P2P transparent clock to the P2P transparent clock being looked at, this value is called the upstream link delay. The message transit time and upstream link delay time are added to the correction field of the PTP event message, and the slave receives the sum of all link delays in the correction field of the message. Theoretically, this is the complete E2E delay of the SYNC packet (from master to slave).

This protocol also defines a variety of variable attributes and optional features that can be tuned to the needs of a particular network, however in an attempt to ease the configuration process the concept of “profiles” was created on PTP and a great number of different profiles were developed by Standards Development Organizations (SDO) [22]. These profiles let organizations or industry groups choose a range of features and options, as well as default values for protocol attributes, that meet the performance needs of apps in the domain and minimize device settings.

2.2.4 PTP vs NTP

PTP and NTP are two extensively employed time [23] synchronization protocols designed to synchronize the clocks of network devices to a common time reference. While both protocols serve the same goal, they have unique properties and are suited to various types of applications. Even though both protocols serve the same goal they are suited better for different kinds of applications [24].

NTP has a solid foundation and is utilized extensively across the Internet. Synchronizations in milliseconds for LANs and hundreds of milliseconds for WANs and the Internet are sufficient for many applications. However PTP is an emerging standard that has been widely adopted by the control and measurement sector. Early testing has shown that sub-microsecond synchronization performance is feasible over LANs.

There is a possibility that the optimal arrangement would involve a combination of the two timekeeping protocols, with NTP operating in the background of the Internet and PTP operating at the LAN level [25].

2.3 Related Work

This chapter compares perfSONAR [26] to well-known measuring tools like Réseaux IP Européens Network Coordination Centre (RIPE NCC) Atlas, Netherlands Network Operators Group (NLNOG) Ring, and SamKnows. The analysis will include brief descriptions of each tool's features and compare their capabilities. FCCN [27], as a National Research and Education Network (NREN), recognizes the importance of reliable network performance and availability for its user community. When it came time to select a measurement system, FCCN chose perfSONAR over other solutions because of its educational nature.

perfSONAR has a strong relationship with GÉANT [28], the pan-European research and education network that supports FCCN. GÉANT is a major user and supporter of perfSONAR, and the two organizations have worked closely together to improve network performance measurement and troubleshooting for the research and education communities. GÉANT has played a key role in the development and deployment of perfSONAR, and its Network Operations Center (NOC) uses perfSONAR extensively to monitor and troubleshoot network performance issues across the GÉANT network.

GÉANT and the perfSONAR work group also collaborate on training and education initiatives. GÉANT offers regular perfSONAR training sessions to its community members, and perfSONAR provides online training resources and documentation that are available to GÉANT users and the wider research and education community.

perfSONAR was developed by the research and education community and is specifically designed to support the needs of NRENs. It provides a platform for E2E network performance measurement, enabling network administrators to quickly identify and troubleshoot network issues.In addition, perfSONAR's web-based Graphical User Interface (GUI) and programmatic access make it easy for FCCN's user community to

access and analyze network performance data. This is especially important for an educational institution like FCCN, where students and researchers rely on high-quality network connectivity for their work.

perfSONAR offers a variety of features, such as:

- A GUI accessible through an website for creating and controlling measurements infrastructure;
- Automatic network performance testing and measurement, including delay, jitter, packet loss, and throughput;
- Many protocol support, including Internet Performance Working Group (iPerf), One-Way Active Measurement Protocol (OWAMP), and traceroute.
- Interoperability with additional network monitoring instruments as Nagios, Icinga and Zabbix;
- Support for federated measurement networks, enabling performance measurements across domains.

Réseaux IP Européens (RIPE) Atlas is a global probing network that actively measures Internet connection and reachability [29]. Distributed globally, RIPE Atlas probes are small hardware devices that detect parameters such as latency, jitter, and packet loss. In RIPE Atlas, we can also find what are known as "anchors", which are both enhanced RIPE Atlas probes with more measurement capacity as well as regional measurement targets within the greater RIPE Atlas network.

The global reach of RIPE Atlas is one of its primary strengths. RIPE Atlas can offer measurements of Internet connectivity and reachability from a broad variety of places, allowing network managers to identify and troubleshoot issues that may be affecting users in different regions. In addition, RIPE Atlas has a robust community of users and contributors who have built a vast array of tools and services on top of the platform's measurement data.

RIPE Atlas is a credit-based system in which users are allotted a specific number of credits to conduct measurements and access the associated data. Each measurement performed by a user costs a particular number of credits, depending on the type of measurement and the probe's position. RIPE Atlas gives users a set number of free credits when they create an account, and additional credits can be acquired in a number of ways, including by engaging in the RIPE Atlas community or by making a financial contribution to RIPE. While the credit-based approach may restrict the number

of measurements a user can perform, it ensures that the RIPE Atlas platform is sustainable and able to offer its users high-quality measurement data over the long run.

RIPE Atlas offers a variety of functionalities, such as:

- A vast network of globally dispersed probes;
- Several measurement kinds are supported, including ping, traceroute, and DNS queries;
- A web-based user interface for displaying measurement data;
- A protocol for programmatic data access;
- Detection instruments for network disruptions and misconfigurations.

Similar to perfSONAR and RIPE Atlas, NLNOG Ring consists of a community-driven network measurement platform composed of probes that are distributed across the Netherlands and other countries [30]. Distinct from the other solutions, SamKnows [31] is a measurement tool designed to measure Internet performance from the end-user perspective. SamKnows provides a small hardware device called the Whitebox that is placed in the user's home and measures various metrics.

One of the primary distinctions between these instruments is the scale at which they function. RIPE Atlas and NLNOG Ring operate on a worldwide scale, whereas perfSONAR is built for large-scale research and education networks. In contrast, SamKnows focuses on end-user measurements.

perfSONAR, RIPE Atlas, SamKnows, and NLNOG Ring all offer a web-based graphical user interface for visualizing and evaluating measurement data, as well as an API for programmatic access to the data. perfSONAR and RIPE Atlas offer integration with additional network monitoring tools, including Nagios and Zabbix. Several measurement kinds, including ping, traceroute, and DNS queries, are supported by all four tools. iPerf and OWAMP are among the advanced protocols supported by perfSONAR and NLNOG Ring.

One significant distinction between the instruments is the type of measurements they offer. perfSONAR focuses on measuring E2E network performance, whereas RIPE Atlas and NLNOG Ring measure Internet connectivity and reachability. SamKnows, on the other hand, offers measurements from the standpoint of the end user. perfSONAR and NLNOG Ring rely on a globally dispersed network of probes for deployment, whereas RIPE Atlas employs a combination of probes and anchor hosts. SamKnows offers a small hardware device that is installed within the user's residence.

Ultimately, each tool has advantages and disadvantages, and the choice of which tool to employ will depend on the user's particular requirements. Due to its emphasis on E2E network performance measurements, perfSONAR is the probable best option for large-scale research and education networks. RIPE Atlas and NLNOG Ring are two excellent solutions for measuring global Internet connectivity and reachability. And for end-user measurements, SamKnows is the preferred instrument. Table 2.1 briefly compares the solutions presented before. This information was directly obtained from the official perfSONAR documentation.

The work in [32], with collaboration with FCCN, addressed in the past the implementation of a previous version of perfSONAR. This previous attempt never got fully implemented, and since it was made in 2013 has become outdated.

This chapter delved into the current landscape of network monitoring technologies and methodologies. It provided a detailed overview of different types of monitoring metrics, distinguishing between active and passive monitoring methods. The chapter also explored key protocols for time synchronization, discussing their operation, hierarchy, accuracy, and security. Additionally, a review of related work in the field provided a comprehensive understanding of existing solutions and setting the stage for the work elaborated ahead.

Table 2.1: perfSonar, RIPE Atlas, SamKnows, NLNOG RING comparison information from perfSONAR Documentation.

	perfSONAR	RIPE Atlas	SamKnows	NLNOG RING
Type of Measurements	Throughput (TCP and UDP), RTT , one-way delay, one-way packet loss, network path	Built-in: RTT to the first and second hops, ping to predetermined destinations, traceroute to predetermined destinations, DNS queries to root DNS servers, SSL queries to predetermined destinations. User-defined: ping, traceroute, DNS, TLS and NTP query to any destination.	Multi-threaded HTTP download speed, multi-threaded HTTP based upload speed, availability of the connection, jitter, latency (Internet Control Message Protocol (ICMP) and UDP), packet loss (ICMP and UDP), DNS query resolution time, DNS query failure rate, web page loading time, web page loading failure rate, Video streaming performance.	RTT, traceroute, SSH and system tools from predetermined destinations to any other host.
User-Defined Scheduled Measurements	Yes	Yes; Limited by a system of credits based on participation	No	No
On-demand Measurements	Yes	Yes; Limited by a system of credits based on participation	No	Yes
Incoming Measurement Control	Yes; site-programmable	No	No	No
Measurement Target Control	Yes; site-programmable	Any non-local destination	Pre-defined hosts	Other RING hosts
Type of Distribution & Software	Hardware device or software package	Software and hardware	Software	
Measurement Data Storage Distribution	Local or central; many archive types	Central	Central	N/A
Measurement Data Storage Architecture	Data stored in user infrastructure	Data stored in service provider infrastructure + Google BigQuery	Central	N/A
Access to Archive Measurements	Local or central web interface	Central web interface	Central web interface	N/A

3

PerfSonar

In today's digital world, organizations cannot function without networks that are both dependable and high-performing. This is especially true for NRENs that serve the academic and research communities by providing connectivity and other services. As the Portuguese NREN, FCCN job is to help support scientific and educational projects all over the country. To make sure that users are getting the best service possible, it is important to maintain a high-quality network that provides fast, reliable, and safe connections. This is where perfSONAR comes into play. perfSONAR is a collection of open-source software for performing and sharing end-to-end network measurements that has gained widespread popularity among network engineers and administrators. This tool development is being supported through a collaboration between different organizations like GÉANT, Energy Sciences Network (ESnet), Indiana University, Internet2, University of Michigan, and *Rede Nacional de Ensino e Pesquisa (RNP)*. Profiting from its nature as an open-source software, perfSONAR is continuously being developed and improved as a result of a vibrant community of developers and users who work together to ensure that this system remains effective and up-to-date.

With the many configuration options in perfSONAR, network administrators can fine-tune the tool to meet the specific needs of their network. It is adaptable to different network settings and can be used in a range of contexts, be it a aclan or a acwan. For a more comprehensive view of network health, perfSONAR can be combined with other network monitoring tools like Nagios, Icinga, and Zabbix.

perfSONAR uses a variety of protocols and techniques to perform measurements, but

one of the key aspects of this system is its ability to create an abstraction layer where it permits the user not to work directly with his tools but instead with a more user-friendly interface called the perfSonar Scheduler (pScheduler).

In this chapter, we will discuss how perfSONAR can be used in network monitoring as a tool that helps administrators detect and diagnose possible issues. We will also implement the system on the FCCN network, providing a practical example of how perfSONAR can be used to improve network performance and reliability in real-world scenarios.

3.1 General Deployment Architecture

The totality of the perfSONAR environment is composed by the core perfSONAR software in conjunction with the satellite probes that can be divided into five specific perfSONAR layers: visualization, archiving, scheduling, tools, configuration, and discovery. However, the core system components can be placed under a more generic three-layer deployment architecture, consisting of a visualization web server, an application server, and a database server.

The web server for visualization is the initial layer of the perfSONAR architecture. It is accountable for displaying network performance metrics to end users via a web-based Graphical User Interface. The visualization web server processes data received from the application server to generate graphs, charts, and other visual representations of network performance data. The GUI is designed to be user-friendly and straightforward, enabling network managers to access and analyze network performance information with ease.

The application server constitutes the perfSONAR architecture's second tier. Its job is to gather, process, and store information about how well a network is functioning. Data is collected from various locations in the network that have a working probe and sent to the application server, which then evaluates the information and produces actionable performance metrics. Latency, packet loss, and throughput are just a few of the KPIs that are computed with the help of algorithms. Data processing and visualization are carried out by the application server, with the results shown on the visualization web server in the form of charts and graphs.

The database server is the perfSONAR architecture's third layer. It is in charge of archiving and managing statistics regarding the operation of the network. Data created by the application server can be stored in a scalable and effective manner on the database server.

This separation makes it possible to allocate each layer in a different server cluster, which augments the total flexibility, resilience, and scalability of the application, especially when used in conjunction with a mechanism like Docker swarm, which is currently being used as a basis in FCCN.

Docker Swarm is an orchestration platform for containers that facilitates large-scale application deployment and management. Docker Swarm is a flexible and efficient tool for deploying and managing perfSONAR components across several network domains, and it may be used in conjunction with perfSONAR. Using Docker Swarm, perfSONAR's many parts may be deployed as containers, making them highly scalable to meet fluctuating demands in network monitoring.

Because of Docker Swarm's modular nature, each component of perfSONAR's architecture may be deployed independently. For instance, the visualization layer can be set up as a standalone service with numerous copies.

Furthermore, in order to gain redundancy FCCN operates with server clusters instead of isolated servers, this means that each service will be replicated through different servers and the technology docker will choose automatically in what host the service should be running taking in account the available resources in the host machines.

3.2 perfSonar Application Layers Macro-View

The architecture of perfSONAR is composed of diverse layers of services. The Figure 3.1 illustrates the layers present in the old version four of perfSONAR, while the Figure 3.2 represents the architecture of the newest version five. It can be observed that the major modifications implemented were around the archiving layer that includes the database modules. Each one of these layers is then made up as a conjunction of various modules working together to offer a vast number of options and services. It is possible to generally describe these layers as follows:

- **Visualization:** perfSONAR's visualization layer offers a Graphical User Interface for inspecting network performance statistics. OpenSearch Dashboards, Grafana, Graphs, MaDDash, perfSonar Config (pSConfig) WebAdmin, and the Toolkit UI are just some of the tools that are included to help administrators build their own individualized views of network performance.
- **Archiving:** The archiving layer of perfSONAR provides a mechanism for storing and retrieving network performance data. It includes tools such as the perfSONAR Measurement Archive (MA), which allow administrators to store and

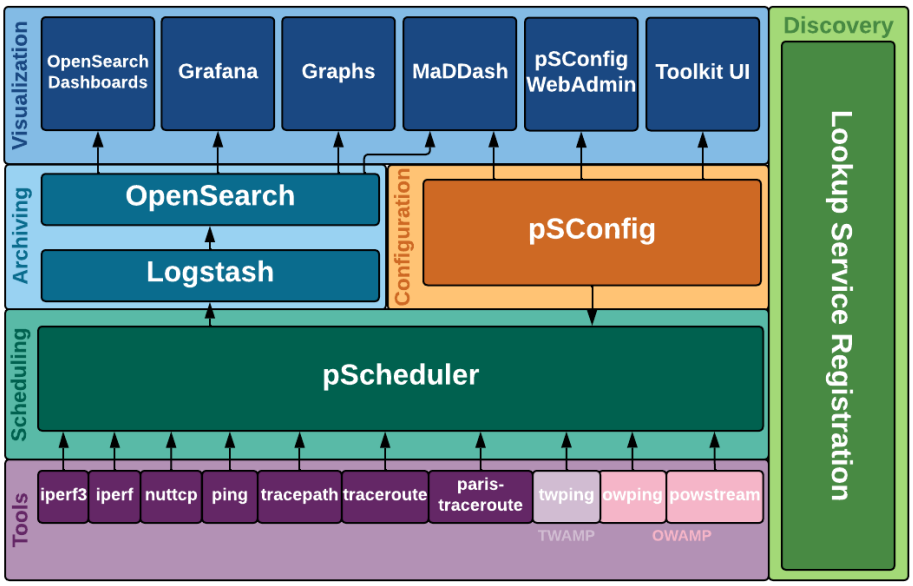

Figure 3.1: Diagram of perfSONAR version 4 (From the official website [33])

analyze historical network performance data.

- **Configuration:** perfSONAR's configuration layer facilitates constructing performance evaluations and guidelines for a network. The perfSONAR configuration manager (pSConfig) is one of the included tools that helps administrators standardize network testing and policy configurations across various perfSONAR installations.

- **Scheduling:** To schedule network performance testing, perfSONAR has a dedicated scheduling layer. Included in this suite is the perfSONAR Test Scheduler, which helps network administrators plan out periodic and ad hoc network testing.

- **Tools:** This layer is present on the MS and includes a variety of network measurement and monitoring tools that allow administrators to conduct a variety of tests to diagnose and troubleshoot network issues.

- **Discovery:** Each perfSONAR node has the option of running the Lookup Service Registration (LSR) daemon, which notifies both public and private lookup services of the node's presence. The registration daemon compiles data on the host machine and each perfSONAR layer. This data is then put to use in a variety of contexts, including troubleshooting issues and locating test hosts for use in creating new setups.

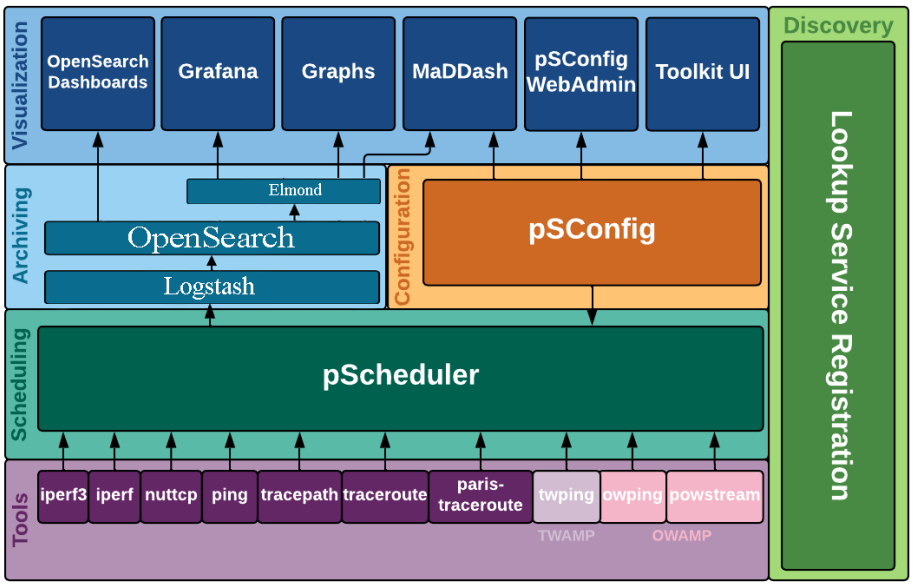

Figure 3.2: Diagram of perfSONAR version 5 (Adapted from the official website [33])

3.2.1 Visualization Layer

The visualization layer is a large module of perfSONAR composed of six different subcomponents that make it possible to display the collected test data to the users:

- **OpenSearch Dashboards**: OpenSearch Dashboards (formerly known as Kibana): OpenSearch Dashboards is a data visualization dashboard. It is designed for real-time, web-friendly access to data stored in OpenSearch (formerly Elasticsearch). It allows network administrators to create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

- **Grafana**: This component is another common open-source data visualization tool. Grafana is a multi-platform open-source analytics and interactive visualization platform. It provides charts, graphs, and alerts for the web when connected to supported data sources, one of which is OpenSearch. It is expandable through a plug-in system and is a popular tool for visualizing time-series data.Grafana is also flexible and may be used in conjunction with other technologies like Prometheus and InfluxDB.

- **Graphs**: This refers to the general ability of the visualization layer to generate various types of graphs based on the network performance data collected. These could include time-series graphs, bar charts, pie charts, scatter plots, and more.

- **MaDDash**: MaDDash stands for Monitoring and Debugging Dashboard. It's a graphical interface that provides a grid view of network paths and corresponding performance data. Each cell in the grid represents a network path, and its color reflects the status of the path (e.g., green for normal, red for problems).

- **pSConfig WebAdmin**: pSConfig WebAdmin is a web-based user interface that allows users to configure perfSONAR tests using pSConfig templates. This involves defining what tests to run, where to run them, and how often they should be performed.

- **Toolkit UI**: The Toolkit UI provides a user-friendly interface for managing a perfSONAR node. This includes configuring tests, viewing test results, and performing basic node management tasks.

3.2.2 Archiving Layer

The archiving layer is the module responsible for storing the network performance data retrieved from the different probes for both immediate and future reference. Network administrators rely heavily on this information because it is the foundation for a wide range of analysis. It is a layer composed by only a small but complex number of components:

- **OpenSearch**: OpenSearch is a powerful, distributed, RESTful search and analytics engine designed to handle large volumes of data quickly and responsively. It can take in unstructured data from various sources, index and store this data, and provide nearly instantaneous search and analysis capabilities. Its feature-rich functionality includes full-text search, distributed search, multi-tenancy, and more. In the context of perfSONAR, OpenSearch efficiently stores network performance data, ensuring it is readily accessible for real-time analytics and long-term trend analysis.

- **Logstash**: Logstash is a server-side data processing pipeline that receives data from a variety of sources, transforms that data, and then sends the transformed data to a number of different destinations. It provides a wide variety of inputs, filters, and output plugins, allowing it to unify data coming from a variety of sources such as logs, metrics, and events. It is not restricted to merely log files; rather, it can also consume from message queues, communicate with APIs, and manage a wide variety of organized and unstructured data types. Logstash

will gather and process network measurement data using perfSONAR, and then transmit it to OpenSearch for storage after the data has been processed.

- **Elmond**: Elmond is a new module that permits the retro compatibility with the users that have not fully transferred to perfSONAR version 5 and still have their data stored in the fourth version.

3.2.3 Configuration Layer

In the configuration layer, intended measurements and storage location instructions are specified. pSConfig is the principal component at this layer. pSConfig is a framework for documenting and configuring a tasks' topology. Some configuration tasks can become challenging if you manage multiple perfSONAR hosts or participate in a distributed community of perfSONAR measurement hosts, such as scheduling tasks to run at each location and maintaining visualization components to display results from multiple hosts. It is composed by the following modules:

- **pSConfig**: pSConfig is a perfSONAR tool that serves as the central point for managing a suite of performance measurements. It uses JSON-based configuration files, called "pSConfig templates", to define measurements. These templates describe the types of tests, their schedule, the parameters to be used, and the targets against which they will be run. Network administrators can use pSConfig to manage testing policies effectively across multiple perfSONAR nodes, ensuring a consistent approach to performance measurements.

The perfSONAR framework also includes a vital component known as the pspublisher package wich can be atribbuted to the pSConfig layer of perfSONAR, previousy described, this module is incredibly important however its module existence is not well described on the official perfSONAR documentation. This package serves a crucial role in facilitating seamless connections between the various layers of perfSONAR, streamlining automation tasks, and enabling efficient data management. Its purpose is to simplify and optimize the process of coordinating and automating tasks within the perfSONAR infrastructure.

At its core, the pspublisher package acts as a bridge that enables efficient communication and data exchange between different components of perfSONAR. This bridging capability enhances the interoperability of the system, making it easier for individual components to work together cohesively.

Moreover, the pspublisher package empowers administrators to enumerate and manage the list of test-points across the entire perfSONAR infrastructure. Administrators can conveniently specify the desired tests to be executed on each test-point, tailoring the network monitoring process to suit specific needs and requirements.

3.2.4 Scheduling Layer

The scheduling layer is comprised of a single tool named pScheduler and is responsible for finding time-slots to run the tools while avoiding scheduling conflicts that would negatively impact results, executing the tools, gathering results, and sending the results to the archiving layer (if necessary).

- **pScheduler**: pScheduler is a daemon that schedules and runs performance tests, keeping track of their results. It considers network conditions, test durations, and system availability when scheduling tests, helping to prevent overloading of the network or hosts due to concurrent tests. pScheduler also interacts with pSConfig, which provides the test specifications. This interaction ensures a comprehensive, well-organized, and efficient approach to network performance monitoring and troubleshooting.

3.2.5 Measurement Tools

Like previously stated, one of the key components of the perfSONAR architecture is pScheduler, a task scheduling system that provides a unified framework for running measurements across different tools and hosts. The list of tools that can be found in each probe includes:

- **iPerf2**, also known as just iPerf, is a common tool used to measure network throughput that has been around for many years. It supports TCP and UDP protocols and offers a variety of options for customizing tests, including setting the test duration and the size of the test packets.

- **iPerf3** is a rewrite of the classic iPerf tool used to measure network throughput and associated metrics. It offers increased functionality over its predecessor, including the ability to measure UDP bandwidth and packet loss, as well as improved reporting capabilities.

- **Nuttcp** is another throughput tool with some useful options not found in other tools. It supports both TCP and UDP protocols and allows for the measurement of both bandwidth and latency. It is also designed to be highly scalable and can support large-scale testing across multiple hosts.
- **OWAMP** is a tool primarily used for measuring packet loss and one-way delay. It consists of a set of tools including owping, which performs short-lived tests, and powstream, which runs long-running background tests. OWAMP relies on synchronized clocks between the Measurement Points (MPs) to ensure accurate measurements.
- **Paris-traceroute** is a packet trace tool that attempts to identify paths in the presence of load-balancers. It works by sending packets with different Time to Live (TTL) values and recording the IP addresses of the routers that respond. By analyzing the response patterns, it can identify which routers are part of the same load balancing group.
- **Ping** is the classic utility for determining reachability, RTT, and basic packet loss. It works by sending ICMP echo requests to a destination and measuring the time it takes for a response to be received. Ping is a simple and widely used tool for basic network troubleshooting.
- **Tracepath** is another path trace tool that also measures path Maximum Transmission Unit (MTU) . It works similarly to traceroute but uses UDP packets instead of ICMP packets. This can provide more accurate results in certain network configurations.
- **Traceroute** is a classic packet trace tool used in identifying network paths. It works by sending packets with increasing TTL values and then recording the IP addresses of the routers that respond with ICMP time-exceeded messages. This provides a picture of the path that packets take through the network.
- **Two Wire Active Measurement Protocol (TWAMP)** is a tool primarily used for measuring packet loss and two-way delay. It offers increased accuracy over tools like ping, without the same clock synchronization requirements as OWAMP.

3.2.6 Discovery Layer

This layer is responsible for discovering other perfSONAR nodes within the network and is made up of one single component:

- **Lookup Service Registration**: Increasing the discoverability of nodes and the scalability of perfSONAR deployments rely heavily on the LSR. To facilitate the discovery of additional perfSONAR nodes and the coordination of the scheduling and running of network performance tests, it provides a global register of perfSONAR nodes. To better understand how their networks are performing at different points, network administrators can use the Lookup Service to locate other nodes with which to run testing.

The perfSONAR chapter provided an in-depth look at the tool's architecture and functionalities. It covered the general deployment architecture of perfSONAR within the RCTS network and examined its various application layers, including visualization, archiving, configuration, scheduling, measurement tools, and discovery. This chapter aims to offer a thorough understanding of how perfSONAR operates and its potential benefits for network monitoring and performance validation.

4

Perfsonar Implementation

The deployment of perfSONAR, a widely adopted network performance measurement framework, demands for a carefully planned and executed implementation strategy. To ensure seamless integration into the existing infrastructure and validate its effectiveness, the perfSONAR implementation is typically divided into distinct phases. This chapter explores the rationale behind the phased implementation approach as well as the practical deployments made in each one, focusing on the three main stages: initial POC, intermediate POC, and final deployment. Each phase serves a specific purpose in gradually scaling up the perfSONAR infrastructure, optimizing its performance, and facilitating nationwide deployment.

The phased implementation of perfSONAR, with its distinct stages of initial POC, Intermediate phase with distributed test-points, and final deployment, serves as a comprehensive strategy to seamlessly integrate the framework into the existing network infrastructure. This systematic approach empowers network administrators to measure, monitor, and optimize network performance at various scales, ensuring enhanced end-user experiences and efficient network operations.

Key benefits offered by this strategy:

1. **Risk Mitigation:** By implementing perfSONAR in phases, any potential risks or issues can be identified and rectified early in the process. This mitigates the risk of unforeseen complications arising during the final deployment, ensuring a smoother and more efficient implementation.

2. **Scalability and Performance Optimization:** Each phase allows for performance fine-tuning and optimization, taking into account specific network characteristics and requirements. This iterative approach ensures that the perfSONAR infrastructure scales effectively, capturing accurate performance measurements across the entire network landscape.

3. **Enhanced User Experience:** The phased implementation approach enables network administrators to gradually introduce perfSONAR capabilities, minimizing disruptions to ongoing network operations. This ensures a seamless transition and maximizes user experience throughout the implementation process.

In the implementation of perfSONAR there is a notable absence of a functional Docker installation available. However, a concerted effort will be made to leverage Docker Swarm containers whenever possible. This approach serves multiple purposes, including aligning with the infrastructure of the FCCN and optimizing the system's potential. Additionally, it aims to contribute to the broader community and the advancement of knowledge, ensuring that the benefits of the system can be extended to a wider audience in the future. This approach offers several advantages within the context of the FCCN infrastructure. By leveraging Docker Swarm, the components of perfSONAR can be encapsulated within containers, enabling easier management, deployment, and scalability across multiple nodes within the swarm.

The decision to utilize Docker Swarm aligns with the FCCN infrastructure, which already incorporates containerization technologies. By integrating perfSONAR into a Docker Swarm environment, it becomes compatible with existing infrastructure practices and makes use of the benefits of containerization, such as isolation, portability, and resource efficiency.

Adopting Docker Swarm for the implementation of perfSONAR not only benefits the FCCN infrastructure but also facilitates the development of the system to its fullest potential. The use of containers streamlines the deployment process, allowing for faster iterations, easier updates, and simplified testing in different environments. This contributes to the ongoing development and improvement of the perfSONAR framework, enabling it to evolve and adapt to emerging networking challenges.

Throughout the implementation phases, the servers used for perfSONAR will initially operate as standalone entities. However, as the final deployment phase is reached, a migration will be performed to migrate the front-end and database servers into their own dedicated clusters. This migration aims to improve redundancy and ensure higher availability of services.

By establishing a dedicated cluster for the front-end and databases, redundancy and fault tolerance can be achieved. This architecture helps mitigate the impact of potential server failures, ensuring the continuous operation of perfSONAR services. Redundancy in the form of a cluster also improves scalability, allowing for seamless scaling of resources as the demands of the network grow.

4.1 Network Architecture

In the final design of our network architecture, we have strategically planned the integration of 40 test-points. These test-points are dispersed across various higher education institutions located in different regions of Portugal. The selection of these 40 locations was a meticulous process, driven by a series of key metrics that are of significant importance to the FCCN. Figure 4.1 acts as a brief illustration of this test-points distribution in a general view of Portugal.

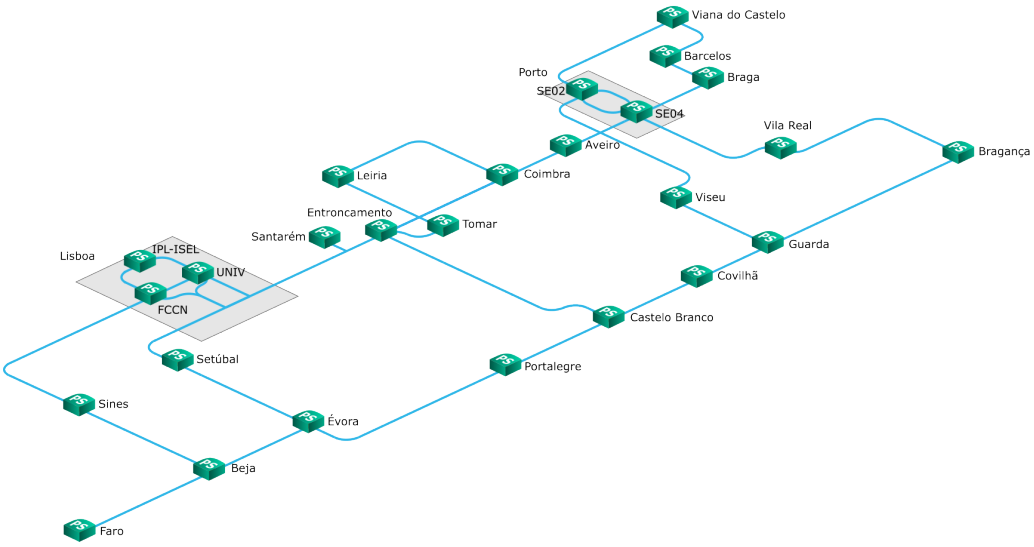

Figure 4.1: Diagram with the test-points distribution

The rationale behind choosing these specific sites lies in their geographic and network diversity, which aligns with the overarching objectives of FCCN. By spreading these test-points across a wide range of locations, we aim to garner a comprehensive and accurate picture of the network's performance and health. This spatial distribution is crucial for diagnosing and resolving potential network issues, as well as for planning future enhancements.

Moreover, the number of locations is not arbitrary. Figure 4.1 was determined after thorough analysis, ensuring that it is sufficiently large to provide a robust dataset for

network analysis yet manageable in terms of maintenance and monitoring. Each test-point will be equipped with perfSONAR tools, enabling detailed performance measurements and facilitating the detection of any anomalies in data transmission, latency, or packet loss.

The implementation of this network architecture, with its strategically placed test-points, represents a significant step forward in our mission to enhance and maintain a high-quality, reliable network for Portugal's higher education sector. This initiative not only reflects our commitment to technological advancement but also underscores our dedication to supporting the vital research and educational endeavors within these institutions.

4.2 Hardware

The hardware selection for our network architecture plays a pivotal role in ensuring optimal performance and integration within the host institutions. We have chosen Supermicro Nexus SuperChassis 506TQC-R301 [34] Servers for this purpose, illustrated in Figure 4.2. These servers are notable for their relatively small dimensions, being just 1 Rack Unit in height with a length of 35 cm. This compact size is crucial as it allows for the hardware to be installed in various sites without disrupting the existing infrastructure or usability of each host institution.

Figure 4.2: Supermicro Nexus SuperChassis 506TQC-R301 Servers (From the experimental laboratory)

Each server is meticulously configured to meet the demands of our network. A key feature of these servers is the inclusion of dual power supplies, providing essential redundancy. This redundancy is vital in maintaining network uptime, as it ensures that the server remains operational even if one power supply fails, thereby enhancing the overall reliability of the network.

Connectivity options in these servers are diverse and tailored to the specific needs of each test-point. The servers are equipped with 1 GB ports and a network adapter that varies depending on the test-point. Specifically, there are two types of network

adapters employed: 7 test-points, deemed more critical by FCCN, are outfitted with 100 GB Mellanox ports, while the remaining 33 test-points are equipped with 10 GB ports. These high-capacity ports are designated for handling the perfSONAR tests, with the 100 GB ports catering to sites requiring higher bandwidth and the 10 GB ports serving the rest. This differentiation in port capacities ensures that each test-point is optimally equipped to handle the network traffic and performance testing relevant to its role in the FCCN network.

A crucial component of these servers is the Integrated Lights-Out (ILO) port, which is active at every site. The ILO port is a dedicated management port that provides out-of-band management capabilities. This means it allows for remote management of the server, independent of the status of the main operating system or the server hardware. The significance of the ILO port cannot be overstated. It enables network administrators to remotely perform a variety of management tasks, such as monitoring server health, updating firmware, and even rebooting servers. This capability is especially important in a distributed network setup like ours, as it allows for efficient management and troubleshooting of servers at various locations without the need for physical presence, thereby greatly enhancing the maintainability and responsiveness of the network infrastructure.

In summary, the hardware configuration with its Supermicro Nexus SuperChassis 506TQC-R301 servers, compact design, dual power supplies, varied network adapters, and the critical ILO port, forms the backbone of our robust and efficient network architecture. This setup not only meets the current needs of the FCCN network but also provides the flexibility and scalability to adapt to future requirements.

4.2.1 Hardware Connectivity

In our network architecture, the connectivity methods for the ILO ports and the network adapters at each test-point are meticulously selected to ensure optimal performance and reliability. The configuration of these connections is tailored to the specific requirements of each type of port.

For the 1GB ILO connections, we have opted to use copper cabling in conjunction with 1 GB TX Small Form-factor Pluggable (SFP) transceivers. This combination is chosen for its reliability and cost-effectiveness. Copper cabling is highly suitable for the ILO management traffic, which does not require high bandwidth but demands stability and minimal latency. The 1 GB TX SFP transceivers provide a secure and stable means of connecting the ILO ports to the network, ensuring robust out-of-band management capabilities across all test-points.

Moving to the 10 GB connections, these are established using Short Reach (SR) SFP+ transceivers paired with multimode LC-LC fibers. The SR SFP+ transceivers are designed for short-range, high-bandwidth connections, making them ideal for the 10 GB network requirements at 33 of our test-points. The multimode LC-LC fiber cables offer a perfect medium for these connections, providing high data transmission rates and reduced signal loss over the required distances. This setup ensures that each of these test-points has a high-performance, reliable connection suitable for conducting the necessary perfSONAR tests.

For the highest bandwidth requirements at the 7 critical test-points, the 100 GB connections utilize SR4 SFP+ transceivers coupled with Multifiber Push On (MPO)-MPO fibers. The SR4 SFP+ transceivers are capable of handling the significantly higher data throughput required by these test-points. The use of MPO-MPO fibers is particularly important for these connections. MPO fibers are designed for high-density connections and are capable of supporting the high data rates and bandwidth required by 100 GB connections. This type of fiber provides a reliable and efficient means of handling the extensive data traffic at these critical nodes.

The selection of copper cabling and 1 GB TX SFP transceivers for the ILO connections, multimode LC-LC fibers with SR SFP+ transceivers for the 10 GB connections, and MPO-MPO fibers with SR4 SFP+ transceivers for the 100 GB connections, is a strategic decision. This configuration ensures that each test-point in the FCCN network is equipped with the most suitable and efficient means of connectivity. This tailored approach to connectivity not only optimizes the performance of each test-point but also enhances the overall reliability and effectiveness of the network infrastructure.

4.3 Phase 1: Initial Proof of Concept (POC)

As represented on the Figure 4.3 the initial phase involves setting up a Staging POC environment where the perfSONAR servers are deployed within the same LAN. This approach allows for controlled testing and evaluation of the framework's functionalities in a controlled environment. By simulating a real-world scenario within a confined network, any initial configuration issues or interoperability challenges can be addressed effectively. This phase also offers an opportunity to fine-tune the perfSONAR configuration and ensure compatibility with the existing infrastructure. Key components of the POC as seen on the Figure 4.4 include:

- **Front-end Server:** This server acts as the central hub and houses modules such

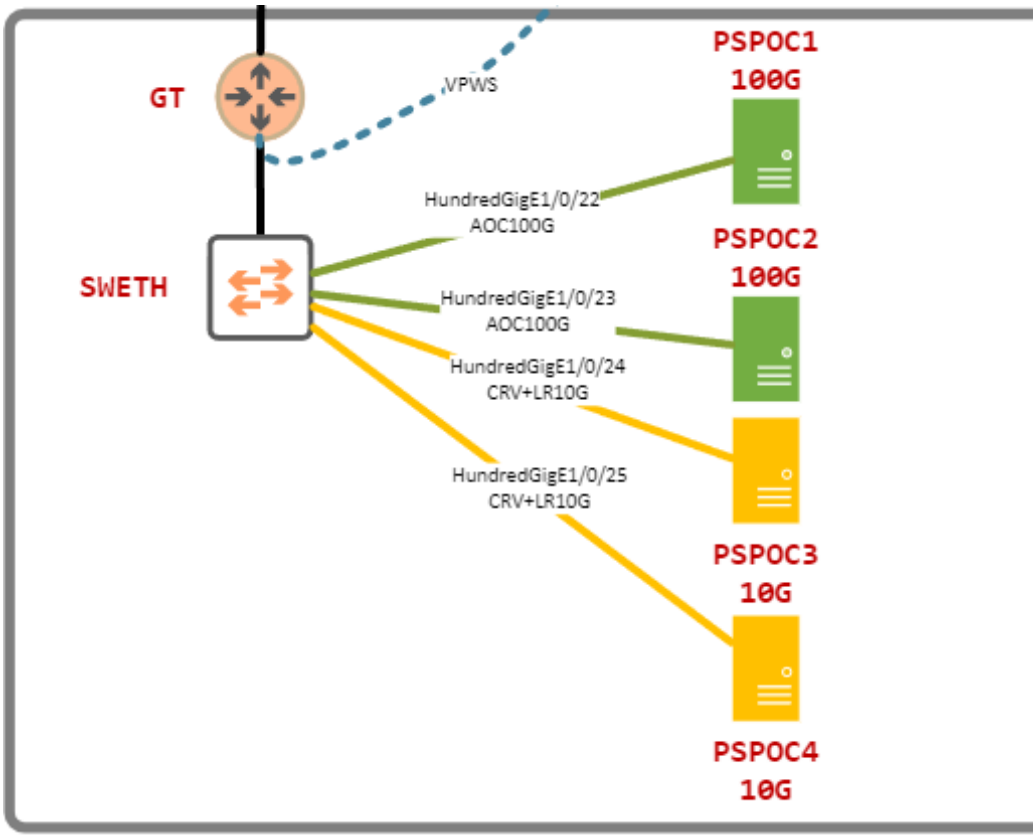

Figure 4.3: perfSONAR Initial POC Diagram

as MadDash, Grafana and Graphs. It is responsible for retrieving data from the database server and presenting it in a user-friendly format.

- **Database Server:** The database modules like OpenSearch, LogStash and Elmond, reside on this server, providing efficient data storage and retrieval capabilities.
- **Test-point Servers:** Four initial test point servers are deployed in the LAN, enabling the collection of performance measurements for local analysis and testing.

During the initial phase of the perfSONAR implementation, the focus was on conducting a step-by-step installation of the different modules directly on the Host systems. This approach allowed us to gain valuable insights into the specific requirements and dependencies of each module, ensuring a thorough understanding of the underlying components before proceeding to containerization with Docker.

The primary objective of this phase was twofold: to learn the specific requirements of the perfSONAR framework and to prepare for the development of Dockerfiles and Docker Compose files. The in-depth knowledge gained during the step-by-step installation served as a solid foundation for creating Docker containers that could accurately

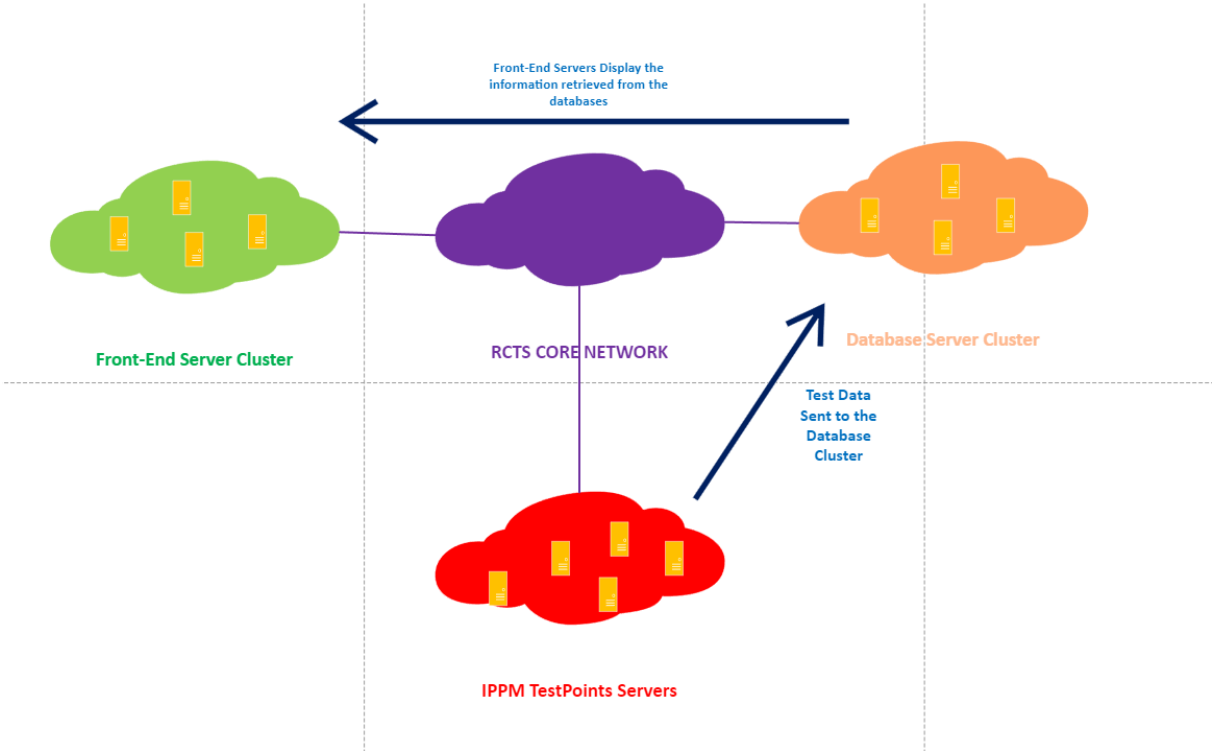

Figure 4.4: perfSONAR Modules Diagram

reflect the real-world setup of the perfSONAR system. These files enable the automation of the installation and configuration process, simplifying the deployment of perfSONAR components in the future. By encapsulating each module within a Docker container, we can ensure consistency and reproducibility across different environments, eliminating potential inconsistencies that might arise from manual installations.

The decision to employ Docker containers offered numerous advantages, including enhanced portability, isolation, and version control. Docker provided a convenient way to package each module and its dependencies, allowing for seamless migration between different Host systems and environments. The containerization approach also promoted better resource utilization and scalability, facilitating the deployment of perfSONAR on various infrastructure setups.

Moreover, by using Docker Compose files, we are able to define the orchestration and interconnection of multiple containers. This streamlined the management of the perfSONAR deployment, automating the process of bringing up the entire perfSONAR ecosystem with its interconnected modules. Docker Compose provided a clear and concise way to manage complex setups, ensuring consistent and reliable deployments in various scenarios.

The Figure 4.5 shows one result of a connectivity test made between two test-points present in this initial POC, we are able to observe a connection of almost 10 Gbps

between them.

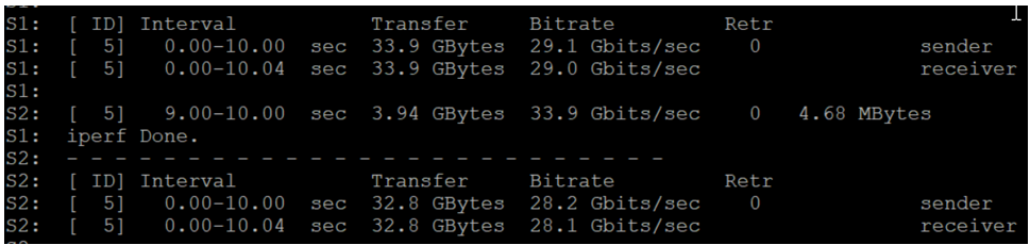

Figure 4.5: perfSONAR Connectivity test between two initial POC testpoints

4.3.1 Architecture Installation Step-By-Step

4.3.1.1 Front-end Server

The front-end server as the name suggests is the server responsible for displaying the data collected by the test-points and stored by the archive server. Since the data is stored in the OpenSearch database, it is possible to display it by using a large number of different graphical systems. In this implementation we chose to use the MadDash service that was for a long time the primary display used and developed by GÉANT, the installation of wich can be seen bellow, and also Grafana since FCCN already works and prefers this service.

1. **Access a server with the Debian operating system installed**

2. **Install Required Packages** Inside the running Debian container, update the package list and install the necessary packages using the following commands:

```
1
2     \item \textbf{Add \ac{perfsonar} Repository and Install Packages}
      Navigate to the /etc/apt/sources.list.d/ directory and download the
      \ac{perfsonar} repository file. Add the \ac{perfsonar} GPG key to
      the system and update the package list. Then, install the \ac{
      perfsonar} frontend-related packages using the following commands:
3     \begin{lstlisting}
4     cd /etc/apt/sources.list.d/
5
6     curl -o perfsonar-release.list
7     http://downloads.perfsonar.net/debian/
8     perfsonar-release.list
9
10    curl
```

```
    http://downloads.perfsonar.net/debian/
    perfsonar-official.gpg.key | apt-key add -

    apt update

    apt install maddash --no-install-recommends

    apt install apache2 --no-install-recommends

    apt install perfsonar-psconfig-maddash
    --no-install-recommends

```

a

3. **Start the different needed services** After installing the required packages, start the maddash-server and Apache web server using the following commands:

```
    /etc/init.d/maddash-server start
    /etc/init.d/apache2 start
    /etc/init.d/psconfig-maddash-agent start

```

4. **Create Admin User** If you need to create an admin user to access the maddash-webui, use the htpasswd tool to create the user. Execute the following command and replace myadmin with the desired username:

```
    htpasswd /etc/maddash/maddash-webui/admin-users myadmin

```

The installation and configuration of the perfSONAR frontend server involve pulling the Debian Docker image, installing required packages, adding the perfSONAR repository, and starting essential services. Once these steps are completed, the perfSONAR frontend server will be up and running, ready to provide web-based access to the network performance monitoring data through the maddash-webui.

4.3.1.2 Database Server

Now we will proceed with the implementation of the achieve module on the data base server, in order to do it the following steps were needed:

1. Update Package List and Install Required Packages

```
apt update

apt install curl gnupg2 dialog procps

```

2. Add perfSONAR Repository

```
cd /etc/apt/sources.list.d/

curl -o perfsonar-release.list
http://downloads.perfsonar.net/debian/
perfsonar-release.list

```

3. Add GPG Key

```
curl
http://downloads.perfsonar.net/debian/
perfsonar-official.gpg.key | apt-key add -

```

4. Update Package List (Again) and Install perfSONAR Archive. The –no-install-recommends flag is used to prevent the installation of recommended packages, which can help keep the system lean and avoid unnecessary dependencies.

```
apt update

apt install perfsonar-archive --no-install-recommends

```

5. Go to directory of the apache-logstash.conf file, and edit it in order to have the database accept loggins originated from the test-point, you can accept a dns, a domain, an IP or a subnet like for example:

```
vi /etc/apache2/conf-enabled/apache-logstash.conf

"Example: Place the following lines in the file"
```

```
    Require host tespoint1.fccn.pt
    Require host test-point2.fccn.pt
    Require host test-point3.fccn.pt

```

6. Generate the information needed for the pSConfig.json file, this will be explained in the phase 2 of the implementation.

```
    /usr/lib/perfsonar/archive/
    perfsonar-scripts/psconfig_archive.sh


```

7. The output of the command above will look similar to the following:

```
    {
        "archiver": "http",
        "data": {
            "schema": 2,
            "_url": "https://{% scheduled_by_address %}/logstash",
            "op": "put",
            "_headers": {
                "x-ps-observer": "{% scheduled_by_address %}",
                "content-type": "application/json",
                "Authorization":"Basic eXamPleTokEn"
        }
    }
}

```

4.3.1.3 Test-points

perfSONAR test-point is actually the only module of perfSONAR that already has a working Docker image available on DockerHub. In order to to install it we will have to follow the steps listed below:

1. **Pull the perfSONAR/test-point Docker:** The first step is to pull the perfSONAR/test-point Docker image from the Docker Hub repository. This can be accomplished by executing the following command:

```
docker pull perfsonar/test-point
```

2. **Run the test-point Containe:** After pulling the Docker image, you can run the perfSONAR/test-point container in detached mode while connecting it to the host's network stack. This ensures that the perfSONAR tools can access and monitor network performance directly from the host system. Execute the following command:

```
docker run -d --net=host perfsonar/test-point

```

3. **Access the test-point Container's Shell:** To interact with the running perfSONAR test-point container, you need to access its shell. For this purpose, you can use the docker exec command, followed by the container ID obtained from the previous step. Use the following command:

```
docker exec -it <container ID from above> bash

```

4. **Troubleshoot with pscheduler:** Once inside the test-point container's shell, you can use pScheduler, a performance scheduling and measurement toolkit for perfSONAR. To troubleshoot the network, execute the command shown bellow, which helps identify potential network performance issues and diagnose any problems affecting network throughput or latency.

```
pscheduler troubleshoot

```

5. **Perform a Throughput Measurement Test:** To perform a specific network measurement test, such as measuring network throughput, you can use pScheduler tasks. In this example, we are conducting a throughput test to a specified destination hostname. Use the following command:

```
pscheduler task throughput --dest hostname

```

Replace 'hostname' with the actual hostname or IP address of the destination you wish to test the network throughput to.

In conclusion, installing the simple version of perfSONAR test-point involves pulling the Docker image, running the container in detached mode, accessing its shell, using pScheduler to troubleshoot the network, and performing specific network measurement tests. This method makes it easy to monitor and analyze network performance by deploying perfSONAR test-point in a containerized fashion.

4.3.1.4 PsPublisher Package

One of the key features of the pspublisher package is its ability to publish relevant information inside a json file format in a web-page published using the apache2 package. This json file can also be called pSConfig file and acts as a configuration file that can be accessed and modified with ease. Administrators have the flexibility to configure the location of the database server to which the test-points should send their collected data. This feature streamlines the process of data aggregation, ensuring that the collected measurements are efficiently and accurately stored for further analysis.

By automating jobs and centralizing the management of perfSONAR components, the pspublisher package significantly simplifies the network performance monitoring workflow. The package contributes to the overall efficiency of perfSONAR, promoting a streamlined and cohesive environment that optimizes the use of resources and minimizes potential bottlenecks.

To install and configure this module the following steps should be followed:

1. Install the pspublish package:

```
1    apt install perfsonar-psconfig-publisher
2
```

2. Place the pSConfig file in a server of our choosing, in this case we picked the database server since every test point already had to be able to reach it.

3. Access the file directory and give the following command:

```
1    psconfig publish 'name of the json psconfig file'
2
```

4. Copy the outputed URL and use it in the command that will be applied in every test-point and in the front-end server:

```
1    psconfig remote add 'url'
2
```

5. Check if the file was correctly added with the command:

```
1    psconfig remote list
2
```

4.3.2 Architecture Installation Docker Deployment

After carefully describing the step-by-step manual installation of the modules required for the perfSONAR implementation, the next phase involved creating the necessary files and images to transition these modules into the Docker environment. This approach aimed to automate the deployment process and leverage the benefits that Docker provides, such as portability, scalability, and resource efficiency.

To begin this phase, we identified the specific components and dependencies needed to run each module within Docker containers. For each module, we crafted Dockerfiles that contained the necessary instructions to build the corresponding Docker image. These Dockerfiles served as blueprints for creating the containerized environments with the exact configuration and packages as the manually installed modules.

To facilitate the deployment and management of the entire system, we utilized Docker Compose. This tool allowed us to define a multi-container application in a single file, specifying the relationships and dependencies between the various modules. Docker Compose streamlined the process of launching and managing multiple interconnected containers, ensuring that the system components functioned harmoniously.

4.3.2.1 Front-end Server MadDash Service Deployment

MaDDash as shown in the Figure 4.6, constitutes a pivotal component in network monitoring, particularly within the realm of research and educational networks. It plays a vital role by providing a web-based platform for overseeing the performance and status of numerous network endpoints, often referred to as 'nodes.' MadDash's primary

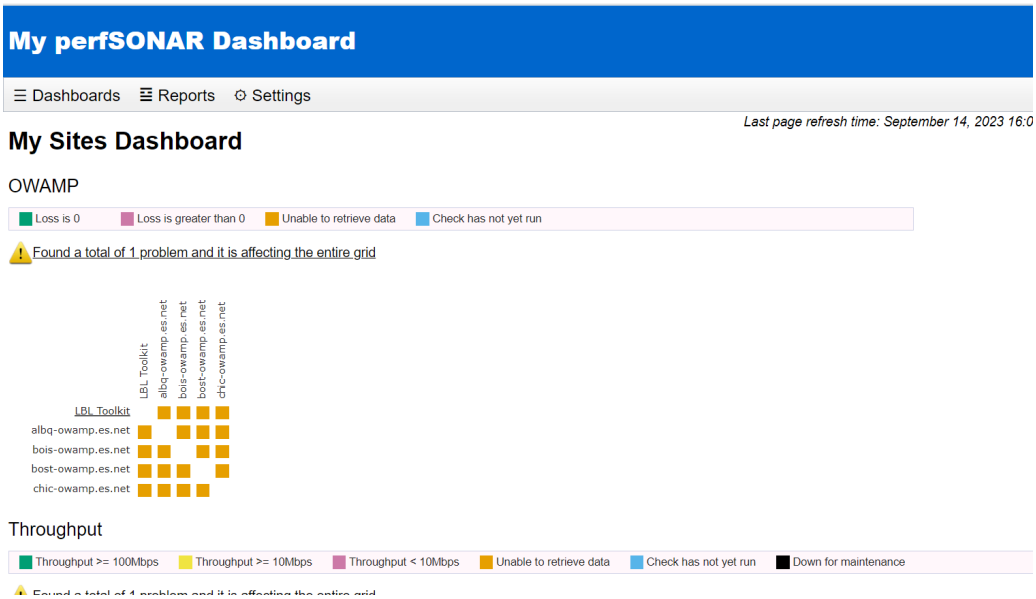

Figure 4.6: MaDDash Home Page

function is to centralize data and offer a dynamic dashboard that displays real-time and historical network information from multiple perfSONAR nodes. This tool empowers network administrators and operators to gain valuable insights, swiftly detect anomalies, troubleshoot network issues, and ensure the overall robustness of the network infrastructure.

Nonetheless, similar to any software, MadDash is not immune to occasional issues and bugs. In our case, we encountered a perplexing situation where, despite meticulously configuring the software on our end, MadDash persistently failed to update the list of nodes it should display. Instead, it stubbornly presented the default example nodes that come pre-packaged with the initial installation. This predicament not only hindered our network monitoring efforts but also posed a significant challenge to the core functionality of the MadDash tool.

In response to this issue, we took the proactive step of reporting it to the GÉANT development team. This team is responsible for the ongoing development and maintenance of perfSONAR and MadDash. Collaborating closely with the GÉANT developers, we embarked on a systematic troubleshooting process. Our approach involved testing multiple releases of the software in an attempt to resolve the problem. Regrettably, despite our concerted efforts, the bug persisted, leaving us with a vexing and elusive challenge.

Instances such as these are not uncommon in the realm of software development and deployment. They can stem from a myriad of factors, including software bugs, compatibility issues, or specific configurations unique to our network environment. In

such situations, effective collaboration with software developers and active engagement with the user community become paramount. By working collectively with these stakeholders, we aimed to pinpoint the root cause and rectify the issue. Additionally, the act of reporting persistent issues to the developers serves a broader purpose — it contributes to the enhancement and refinement of the software for the benefit of all users. This ensures that critical bugs are addressed and resolved in future software releases, thereby advancing the utility and reliability of the MadDash tool.

perfSONAR MaDDash DockerFile:

The Dockerfile presented below provides instructions for building a Docker image that, when run as a container, will configure and execute the MadDash network monitoring tool. The base image used for this Docker image is Debian Buster.

```
FROM debian:buster AS maddash

# Instala dependencias
RUN export DEBIAN_FRONTEND=noninteractive \
     && apt-get update \
     && apt-get install -yq --no-install-recommends \
          curl \
          gnupg2 \
          supervisor \
     && apt-get clean \
     && rm -rf /var/lib/apt/lists/*

RUN curl -o /etc/apt/sources.list.d/perfsonar-release.list
http://downloads.perfsonar.net/debian/perfsonar-release.list \
    &&  curl http://downloads.perfsonar.net
    /debian/perfsonar-official.gpg.key
    | apt-key add -

RUN export DEBIAN_FRONTEND=noninteractive \
     && apt-get update \
     && apt-get install -yq --no-install-recommends \
          maddash \
     && apt-get clean \
     && rm -rf /var/lib/apt/lists/*

COPY files/supervisord.conf /etc/supervisor
/conf.d/supervisord.conf

COPY files/maddash.sh /maddash.sh

RUN mkdir -p /var/log/supervisor \
```

```
32      && chmod +x /maddash.sh
33
34  EXPOSE 80 443
35
36  CMD ["/usr/bin/supervisord"]
```

Dockerfile Explanation:

The following sections break down the contents of the Dockerfile step by step:

```
1  FROM debian:buster AS maddash
```

This line specifies the base image for the Docker image, using the official Debian Buster image as a starting point. It also assigns the alias "maddash" to this stage.

```
1  RUN export DEBIAN_FRONTEND=noninteractive && apt-get update && apt-get
       install -yq --no-install-recommends curl gnupg2 supervisor
```

This command executes a series of `apt-get` commands within the Docker image. It sets the environment variable `DEBIAN_FRONTEND` to "noninteractive" to prevent certain prompts during package installation. It then updates the package list with `apt-get update`, followed by the installation of several packages:

- `curl`: A command-line tool for transferring data with URLs.
- `gnupg2`: GNU Privacy Guard - cryptographic software.
- `supervisor`: A process control system.

After installation, the `apt-get clean` command is used to clean up the package cache, reducing the image size. The `rm -rf /var/lib/apt/lists/*` command removes cached package lists.

```
1  RUN curl -o /etc/apt/sources.list.d/perfsonar-release.list http://downloads
       .perfsonar.net/debian/perfsonar-release.list && curl http://downloads.
       perfsonar.net/debian/perfsonar-official.gpg.key | apt-key add -
```

This command adds a perfSONAR repository to the list of package sources for APT (Advanced Package Tool) and adds the perfSONAR GPG key for package verification.

```
RUN export DEBIAN_FRONTEND=noninteractive && apt-get update && apt-get
    install -yq --no-install-recommends maddash
```

Similar to previous commands, this set of commands installs the `maddash` package from the perfSONAR repository after updating the package list.

```
COPY files/supervisord.conf /etc/supervisor/conf.d/supervisord.conf
```

This line copies the `supervisord.conf` file from the host's `files` directory to the container's `/etc/supervisor/conf.d/` directory. This configuration file is used by Supervisor to manage processes.

```
COPY files/maddash.sh /maddash.sh
```

This line copies the `maddash.sh` script from the host's `files` directory to the root of the container. The script will be used later for launching MadDash.

```
RUN mkdir -p /var/log/supervisor && chmod +x /maddash.sh
```

This command creates a directory at `/var/log/supervisor` and makes the `maddash.sh` script executable by changing its permissions.

```
EXPOSE 80 443
```

This instruction specifies that the container will listen on ports 80 and 443. However, it does not actually publish these ports to the host. Port exposure is typically managed when running the container.

```
CMD ["/usr/bin/supervisord"]
```

This is the default command that will be executed when the container starts. It runs `supervisord`, which is used to manage multiple processes within the container. In this case, it will be used to start and manage the MadDash service.

This Dockerfile sets up a Debian-based environment, installs required packages, including MadDash, copies configuration files, and prepares the container to run the MadDash service using Supervisor. This container image can be used to deploy MadDash in a containerized environment.

perfSONAR MaDDash Gitlab CI/CD:

The following Dockerfile creates a Docker image for running a perfSONAR MaDDash.

```
variables:
  PACKAGE_PATH: /
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  CONTAINER_MADDASH_IMAGE:
    $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME-maddash
  CONTAINER_test-point_IMAGE:
    $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME-test-point

stages:
  - maddash

maddash:
  stage: maddash
  image: docker:stable
  services:
    - docker:dind
  before_script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login --username
        "$CI_REGISTRY_USER" "$CI_REGISTRY" --password-stdin
  script:
    - docker build -f Dockerfile-maddash
        --target=maddash --pull -t
        $CONTAINER_MADDASH_IMAGE .
    - docker push $CONTAINER_MADDASH_IMAGE
  only:
    - tags
```

GitLab CI/CD Configuration for MadDash Explanation

The following GitLab CI/CD configuration file is designed to automate the process of building and pushing Docker images for the MadDash software. It defines a pipeline with a single stage called "maddash." Let's examine each section of this configuration file in detail:

Variables Section

```
variables:
  PACKAGE_PATH: /
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  CONTAINER_MADDASH_IMAGE:
    $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME-maddash
  CONTAINER_test-point_IMAGE:
    $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME-test-point
```

In this section, various environment variables are defined for use throughout the CI/CD pipeline:

- `PACKAGE_PATH`: Specifies the path within the Docker image where packages will be located, set to the root directory (`/`).
- `DOCKER_DRIVER`: Indicates the Docker storage driver being used (overlay2).
- `DOCKER_TLS_CERTDIR`: Specifies the directory for Docker TLS certificates (`/certs`).
- `CONTAINER_MADDASH_IMAGE`: Defines the image name for MadDash based on the current Git commit reference name.
- `CONTAINER_test-point_IMAGE`: Specifies the image name for the test-point image based on the current Git commit reference name.

Stages Section

```
stages:
  - maddash
```

The `stages` section defines the stages in the CI/CD pipeline. In this case, there is only one stage, "maddash," which is used for building and pushing MadDash Docker images.

MadDash Stage

```
maddash:
  stage: maddash
  image: docker:stable
  services:
    - docker:dind
```

```
  before_script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login --username
        "$CI_REGISTRY_USER" "$CI_REGISTRY" --password-stdin
  script:
    - docker build -f Dockerfile-maddash
        --target=maddash --pull -t $CONTAINER_MADDASH_IMAGE .
    - docker push $CONTAINER_MADDASH_IMAGE
  only:
    - tags
```

The "maddash" stage is defined here with the following details:

- `stage`: Indicates that this stage belongs to the "maddash" stage defined in the `stages` section.
- `image`: Specifies the base Docker image used for this stage, which is "docker:stable."
- `services`: Lists additional services to be run alongside the main container, including "docker:dind" for Docker in Docker functionality.
- `before_script`: Contains commands executed before the main script. It logs into the GitLab Container Registry using provided credentials.
- `script`: Contains the main script, performing two Docker-related commands:
  - `docker build`: Builds a Docker image based on a specified Dockerfile (`Dockerfile-maddash`) and target (`maddash`), ensuring the base image is up-to-date.
  - `docker push`: Pushes the built Docker image to the GitLab Container Registry using the image name defined in `$CONTAINER_MADDASH_IMAGE`.
- `only`: Specifies that this pipeline stage runs only when Git tags are created.

This GitLab CI/CD configuration automates the building and pushing of Docker images for the MadDash software. It sets environment variables, defines the pipeline stages, and executes Docker commands to create and publish the MadDash image when specific Git tags are generated.

4.3.2.2 Database Server Deployment

The deployment of the database component within the Docker environment for the perfSONAR Archive faced a series of challenges that eventually necessitated a specialized solution. Unfortunately, these challenges were substantial enough to prevent the successful conclusion of the deployment process. As a result, a distinctive approach had to be taken to address the issues encountered.

Numerous factors contributed to the complexity of deploying the database module. These included intricate dependencies, compatibility concerns, and the specific requirements of the perfSONAR Archive module. Despite concerted efforts, the prevailing issues could not be overcome through conventional methods. It became evident that a more targeted solution was required, one that could address the intricacies unique to the perfSONAR Archive component.

To address this situation, a decision was made to create a specialized Docker image of OpenSearch, tailored precisely to the demands of the perfSONAR Archive module. OpenSearch is a distributed search and analytics engine that is a core component of the perfSONAR infrastructure. However, the creation of this customized Docker image required an in-depth understanding of the perfSONAR architecture and the specific requirements of the Archive module.

Regrettably, the complexity and nuances of the perfSONAR system were beyond the scope of our current knowledge and expertise. The resolution of these challenges demanded insights that were accessible only to the developers deeply familiar with the perfSONAR framework. This inherent complexity made it practically unfeasible for us to proceed independently.

Consequently, the deployment of the database component within the Docker environment for the perfSONAR Archive module remained unresolved due to the need for a specialized Docker image created by the perfSONAR developers. The intricacies of the perfSONAR architecture, coupled with the specific requirements of the Archive module, necessitated an in-depth familiarity with the system that was beyond our current capabilities.

4.3.2.3 Test-Point Servers Deployment

The Dockerfile presented here creates a Docker image for running a perfSONAR testpoint, configuring it with a remote configuration source, exposing necessary ports for network measurement tools, and using Supervisord to manage the processes within the container.

perfSONAR test-point DockerFile:

```
# perfSONAR test-point

FROM perfsonar/test-point:v5.0.2 AS test-point

RUN psconfig remote add "https://'HostIP'/psconfig/pscfg-fccn.json"



The following ports are used:
# pScheduler: 443
# owamp:861, 8760-9960
# twamp: 862, 18760-19960
# simplestream: 5890-5900
# nuttcp: 5000, 5101
# iperf2: 5001
# iperf3: 5201
EXPOSE 443 861 862 5000-5001 5101 5201 8760-9960 18760-19960

# add pid directory, logging, and postgres directory
VOLUME ["/var/run", "/var/lib/pgsql", "/var/log", "/etc/rsyslog.d" ]

CMD /usr/bin/supervisord -c /etc/supervisord.conf
```

DockerFile description:

- ```
  FROM perfsonar/test-point:v5.0.2 AS test-point
  ```

  This line sets the base image for the Docker image, which is based on the `perfsonar/test-point:v5.0.2` image. This base image already contains the necessary components and configurations to run a perfSONAR test point.

- ```
  RUN psconfig remote add "https://'HostIP'/psconfig
  /pscfg-fccn.json"
  ```

  This line runs the `psconfig` command inside the Docker image. `psconfig` is a utility used to configure perfSONAR components. In this case, the command adds a remote configuration source for the test point using the JSON file

located at "https://'HostIP'/psconfig/pscfg-fccn.json". This JSON file likely contains configuration settings for the test point to function correctly within the specific network environment.

- ```
  EXPOSE 443 861 862 5000-5001 5101 5201 8760-9960 18760-19960
  ```

The following line exposes various ports to allow incoming connections from the host or other containers. These ports are used for different network measurement tools provided by Perfsonar. The exposed ports are as follows: - pScheduler: 443 - OWAMP: 861, 8760-9960 - TWAMP: 862, 18760-19960 simplestream: 5890-5900 - nuttcp: 5000, 5101 - iPerf2: 5001 - iPerf3: 5201

- ```
  VOLUME ["/var/run", "/var/lib/pgsql", "/var/log", "/etc/rsyslog.d"]
  ```

This line creates volumes in the Docker image. Volumes are used to persist data outside the container, so any data written to these directories will be stored on the host system or a specified external storage. The directories defined as volumes are: - "/var/run": This volume is used for storing process IDs (PIDs). - "/var/lib/pgsql": This volume likely contains data for the PostgreSQL database. - "/var/log": This volume contains log files. - "/etc/rsyslog.d": This volume is likely used for rsyslog configuration.

- ```
  CMD /usr/bin/supervisord -c /etc/supervisord.conf
  ```

This line specifies the default command to run when a container is started based on this image. In this case, the `supervisord` command is used with the `-c` flag to specify the configuration file `/etc/supervisord.conf`. `supervisord` is a process control system used to manage multiple processes within a single container. It ensures that all the necessary services and processes required for the perfSONAR test-point are running.

test-point GitLab CI/CD File

The following GitLab CI/CD configuration file outlines a continuous integration and continuous deployment (CI/CD) process designed for deploying a perfSONAR test-point using Docker images. This configuration file serves as a tool to automate both the build and deployment processes, thereby ensuring the creation of a consistent and streamlined deployment environment.
  ```

```
variables:
  PACKAGE_PATH: /
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  CONTAINER_TEST_IMAGE: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
  CONTAINER_RELEASE_IMAGE: $CI_REGISTRY_IMAGE:latest

  CONTAINER_test-point_IMAGE:
  $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME-test-point

stages:
- test-point

test-point:
  stage: test-point
  image: docker:stable
  services:
    - docker:dind

  before_script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login --username
        "$CI_REGISTRY_USER" "$CI_REGISTRY" --password-stdin


  script:
    - docker build -f Dockerfile-test-point --target=test-point --pull -t
       $CONTAINER_test-point_IMAGE .

    - docker push $CONTAINER_test-point_IMAGE
  only:
    - tags
```

In order to comprehend the purpose and workflow of this configuration file, let's delve into its various sections:

Defining Environment Variables

At the beginning of the configuration file, a series of environment variables is defined. These variables play a critical role in enabling the configuration and deployment of Docker images. The specific variables defined include:

- `PACKAGE_PATH`: Specifies the path within the Docker image where packages will be located.

- `DOCKER_DRIVER`: Indicates the Docker storage driver being utilized, with the current setting as `overlay2`.
- `DOCKER_TLS_CERTDIR`: Sets the directory for Docker TLS certificates.
- `CONTAINER_TEST_IMAGE`: Defines the image name for testing, based on the current Git commit reference name.
- `CONTAINER_RELEASE_IMAGE`: Establishes the image name for the latest release version.
- `CONTAINER_test-point_IMAGE`: Specifies the image name for the perfSONAR test-point image.

Defining CI/CD Stages

The subsequent section of the configuration file delineates the stages of the CI/CD pipeline. In this particular instance, there is only one stage defined, which is aptly named `test-point`. This stage has a distinct purpose - to facilitate the building and deployment of the perfSONAR test-point image.

Details of the test-point Stage

The `test-point` section provides comprehensive details regarding the `test-point` stage's execution. Within this stage:

- The base image for the CI/CD pipeline is set to a stable version of Docker.
- The `docker:dind` service is specified, enabling Docker-in-Docker functionality within the pipeline.
- The `before_script` section handles the authentication process by logging into the GitLab Container Registry using the supplied credentials.
- The `script` section carries out the key deployment tasks. This involves building the Docker image for the perfSONAR test-point, leveraging the provided `Dockerfile-test-point`. The target `test-point` is specified, and the image is tagged using the previously established environment variable.
- The resultant image is pushed to the GitLab Container Registry.
- The `only` directive ensures that this pipeline exclusively runs when Git tags are created.

In a broader context, this GitLab CI/CD configuration file efficiently automates the process of deploying a perfSONAR test-point through the utilization of Docker images. The file employs environment variables, defines pipeline stages, and orchestrates the necessary steps required for building, tagging, and pushing the test-point image into the GitLab Container Registry. The automated deployment process streamlines the entire deployment trajectory, augmenting uniformity within the deployment environment.

4.3.2.4 Docker Compose

The following Docker Compose file defines a multi-service application for deploying perfSONAR components, specifically the "maddash" service and the "test-point" service. Docker Compose is a tool used to define and manage multi-container Docker applications. This file plays a crucial role in orchestrating the deployment and configuration of the perfSONAR components within a Docker environment.

perfSONAR Docker Compose File

```
version: "3.7"
services:
  maddash:
    image: registry-gitlab/image:0.0.1
    ports:
      - target: 80
        published: 80
        protocol: tcp
        mode: host
      - target: 443
        published: 443
        protocol: tcp
        mode: host
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
            - node.hostname == perfsonar
  test-point:
    image: registry-gitlab/image:0.0.13-test-point
    networks:
      - outside
    deploy:
      mode: global
```

```
      placement:
        constraints:
            - node.labels.type == ippm
networks:
  outside:
    name: "host"
    external: true
```

File Description:

Now let's break down the Docker Compose file and understand each section and its purpose:

```
version: "3.7"
```

This line specifies the version of the Docker Compose file format being used. In this case, it is version 3.7.

```
services:
```

The "services" section of the Docker Compose file defines the different services (containers) that compose the application.

```
maddash:
  image: registry-gitlab/image:0.0.1
  ports:
    - target: 80
      published: 80
      protocol: tcp
      mode: host
    - target: 443
      published: 443
      protocol: tcp
      mode: host
  deploy:
    mode: replicated
    replicas: 1
    placement:
      constraints:
          - node.hostname == perfsonar
```

The "maddash" service definition starts here. This service uses the Docker image named "registry-gitlab/image:0.0.1" as its base image. This image contains the perfSONAR MadDash image built by the MadDash dockerfile.

The "ports" section maps the exposed container ports to the corresponding host ports. The "maddash" service is published on host ports 80 and 443, and the mode is set to "host," which means the container's network stack is not isolated from the host.

The "deploy" section specifies the deployment options for the "maddash" service. It is set to run in "replicated" mode with one replica, meaning there will be one instance of the "maddash" service running. The "placement" constraints ensure that the "maddash" service is deployed on a node with the hostname "perfSONAR."

```
test-point:
  image: registry-gitlab/image:0.0.13-test-point
  networks:
    - outside
  deploy:
    mode: global
    placement:
      constraints:
          - node.labels.type == ippm
```

The "test-point" service definition starts here. This service uses the Docker image named "registry-gitlab/image:0.0.13-test-point," which contains the perfSONAR test-point Image built from the dockerfile created before.

The "networks" section defines the networks that the "test-point" service will be connected to. In this case, it connects to the "outside" network.

The "deploy" section specifies the deployment options for the "test-point" service. It is set to run in "global" mode, which means there will be one instance of the "test-point" service on each node in the swarm. The "placement" constraints ensure that the "test-point" service is deployed on nodes labeled with the type "ippm".

```
networks:
  outside:
    name: "host"
    external: true
```

The "networks" section defines the custom network called "outside." The "host" network driver is used, meaning the "test-point" service shares the network namespace

with the host machine. The "external: true" attribute indicates that the "outside" network is an externally managed network and not created by this Docker Compose file. In this project this method was used since Docker Compose does not yet permit to expose ports in a range, meaning that to not use this workaround, the user would have to open each port one by one. This solution is not optimal and must only be used until some fix is made to this Docker Compose feature, since opening every port can possible be a major safety issue.

4.3.3 Troubleshoot PerfSonar

Because implementing a new system is never an easy task, we will address some ways that a system engineer can debug their infrastructure when any problem is detected in the perfSONAR architecture.

We start with making sure that all the points are functioning properly:

- Verify that the test-points received the measurement schedule
- Check if test-points record measurement data in the central archive
- Verify that the measurement data is actually stored in the central archive
- Check that maddash and/or grafana can access the central archive and display the results

The system administrators can also check some files and commands that can help with troubleshooting some problems:

1. test-points Receiving Measurement Schedule To ensure that the test-points are correctly receiving the measurement schedule, you can use the following commands:
    - `pscheduler monitor`: This command provides real-time monitoring of test-point activities, showing the current measurement schedule and tasks.
    - `pscheduler schedule -PT1H`: This command displays the scheduled tasks for the next hour, verifying if the measurement schedule is active.
2. Additionally, check the log files on the test-points, specifically:
    - `/var/log/perfsonar/psconfig-pscheduler-agent.log`: This log file records the configuration and activity of the pScheduler agent on the test-points.

- `/var/log/pscheduler/pscheduler.log`: This log file provides comprehensive information about pscheduler's operations and task execution on the test-points.

3. **test-points Recording Measurement Data in the Central Archive:** To confirm that the test-points are successfully recording measurement data in the central archive, check the following log file on the archive server:

   - `/var/log/apache2/access.log`: This log file records HTTP access to the central archive, including incoming measurement data from the test-points.

4. **Measurement Data Stored in the Central Archive:** To verify that the measurement data is indeed being stored in the central archive, review the log file on the archive server:

   - `/var/log/opensearch/opensearch.log`: This log file documents the operations and activities of the central archive, including the data indexing process.

5. **MadDash Accessing the Central Archive and Displaying Results** To ensure that MadDash can access the central archive and display the results correctly, check the log file on the MadDash host:

   - `/var/log/maddash/psconfig-maddash-agent.log`: This log file tracks the configuration and activity of the psconfig-maddash-agent, which is responsible for communicating with the central archive and processing data for MadDash.

6. **MadDash UI** Furthermore, evaluate the MadDash user interface to gather valuable insights:

   - Check the MadDash UI to see if different grids are displayed, representing distinct network measurement tasks and test-points.
   - Observe the color-coded squares on the MadDash grids. If all squares are orange (indicating missing data), there might be issues with data retrieval or communication between MadDash and the central archive.

By using these commands and inspecting the relevant log files, administrators can effectively troubleshoot their perfSONAR implementation. These diagnostic steps provide a comprehensive overview of the system's performance and allow for a quick resolution of any potential issues.

4.3.4 Special Note: IPv6 in Docker Swarm

When it comes to using Docker or Docker Swarm with IPv4, the integration typically works well, and the containers can efficiently communicate with each other and with external networks. However, the situation changes when we consider IPv6, as there are some significant challenges that arise due to differences in network configurations and support.

The main issue arises from the fact that Docker Swarm doesn't automatically handle the necessary network configuration and rule creation for IPv6. With IPv4, the default networking setup often works seamlessly, but in the case of IPv6, manual intervention and configuration are necessary to enable proper communication between containers and external networks.

To ensure successful IPv6 communication within a Docker Swarm, administrators must manually configure the required IPv6 network settings, routing rules, and firewall rules. This manual configuration process can be complex and error-prone, especially for those not well-versed in IPv6 networking.

Additionally, some operating systems may not have full support for IPv6 or may require additional configuration steps to enable it. This lack of uniformity in IPv6 support across different platforms and distributions further complicates the process of getting Docker Swarm to work effectively with IPv6.

Furthermore, applications running within Docker containers must be explicitly designed and configured to handle IPv6 addresses correctly. Some applications may not have full IPv6 support, potentially leading to compatibility issues and disruptions in communication.

Because it is a real and hard to solve problem that was encountered during this implementation the proper solution will be offered in order for anyone who reaches the same obstacle. To enable Docker Swarm to work with IPv6, follow these steps:

- **Remove the Node from the Swarm**: Before making changes to the networking setup, remove the node from the Docker Swarm.
- **Recreate the docker_gwbridge Network with IPv6 Enabled**:

  Create a new docker_gwbridge network with IPv6 support using the following command:

```
1    docker network create --driver bridge
2    --ipv6 --subnet 10.20.1.0/24 --gateway 10.20.1.1
```

```
3
4       --subnet fd01::/64 --gateway fd01::1 --opt
5       com.docker.network.bridge.name=docker_gwbridge
6
7       --opt com.docker.network.bridge.enable_icc=false --opt
8       com.docker.network.bridge.enable_ip_masquerade=true
9       docker_gwbridge
```

This command creates the docker_gwbridge network with both IPv4 and IPv6 settings. The IPv4 subnet is specified as 10.20.1.0/24, and the IPv6 subnet is specified as fd01::/64.

- **Add Firewall Rules**: Execute the following ip6tables commands to add necessary firewall rules:

```
1       sudo ip6tables -I FORWARD -i docker_gwbridge
2       ! -o docker_gwbridge -p ipv6-icmp -j ACCEPT
3
4       sudo ip6tables -I FORWARD ! -i docker_gwbridge
5       -o docker_gwbridge -p ipv6-icmp -j ACCEPT
6
7       sudo ip6tables -t nat -I POSTROUTING -s fd01::/64
8       -j MASQUERADE
```

These rules allow IPv6 ICMP traffic to pass through the docker_gwbridge network, enabling communication between containers. The last rule enables IPv6 masquerading for outgoing traffic from the fd01::/64 subnet.

- **Join the Node to the Docker Swarm**: On a manager node, obtain the command to join a node to the Docker Swarm using:

```
1       docker swarm join-token worker
```

This command generates the token required for a worker node to join the Docker Swarm. To add the node to the Swarm, run the generated command on the worker node:

```
1       docker swarm join --token <token> <ip>:2377
```

Replace <token> with the generated token and <ip> with the IP address of the manager node.

- **Persistently Save the IPTables Rules**: Save the iptables rules in the following file to ensure their persistence across reboots.

```
/etc/iptables/rules.v6
```

- **Troubleshooting Network Removal (if needed)**: In case there are problems with removing the docker_gwbridge network, execute the following commands:

```
sudo apt install bridge-utils
sudo systemctl stop docker
sudo brctl show docker_gwbridge
sudo ifconfig docker_gwbridge down
sudo brctl delbr docker_gwbridge
sudo systemctl start docker
```

These steps help enable Docker Swarm to work with IPv6 by creating the appropriate network, adding the required firewall rules, and allowing proper communication between nodes in the Swarm. The process ensures that IPv6 traffic is correctly handled, allowing for a seamless and robust Docker Swarm deployment with IPv6 support.

4.4 Phase 2: Intermediate Phase with Distributed Test-points

After the successful validation of the POC, the implementation proceeds to the intermediate phase, which involves the distribution of three test-point servers to their final destinations while still maintaining the same database and front-end servers. The objective is to assess performance across different geographic locations and network routing as well as experimenting with "cleaner" servers that have not been subject to massive amounts of changes and experimental configurations like the initial test-points. As shown in the Figure 4.7 the chosen test-points for this phase are:

- **IPPM-ISEL:** This test point server is situated at the ISEL University, contributing to the regional coverage and enabling performance monitoring within the university's network infrastructure.

- **IPPM-FCCN:** Located at the FCCN datacenter in Lisbon, this test point server extends the coverage to the datacenter environment, offering insights into network performance within the datacenter's context.
- **IPPM-Porto:** This test point server, situated at the FCCN datacenter in Porto, expands the coverage to another major city, allowing performance monitoring within Porto's network infrastructure.

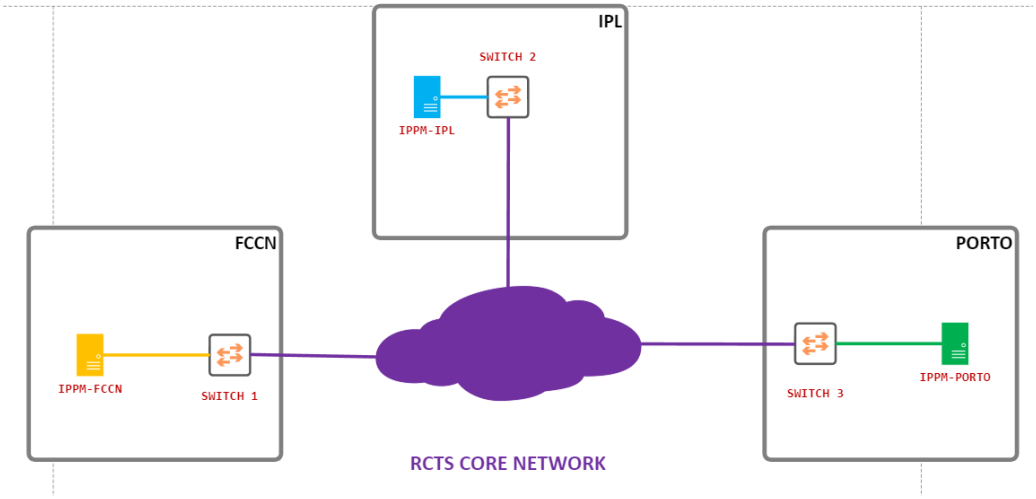

Figure 4.7: perfSONAR Intermediate Phase Diagram

This step aims to validate the framework's performance and reliability in a more realistic setting, closely resembling the target deployment environment. By introducing the distributed setup, network administrators can assess the impact of varying network conditions and address any potential scalability or interconnection challenges that may arise. Rigorous testing during this phase ensures that the perfSONAR infrastructure can effectively capture performance metrics across diverse network segments and provide accurate measurements.

4.4.1 Initial Deployment and Manual Testing

This stage is integral to our implementation strategy, as it involves conducting a series of manual tests to ensure that the servers are functioning as intended and are effectively communicating with each other.

The primary objective of this manual testing phase is to validate the operational integrity of the deployed servers. Given that these servers are freshly installed and have not been subjected to extensive changes or experimental configurations, it is imperative to ascertain their baseline performance and connectivity. This step is crucial before

advancing the project further, as it allows us to identify and rectify any initial issues that could impact the project's overall success.

The manual testing process involves executing specific commands on each server to check for various essential aspects, such as:

Network Connectivity: We will manually test the network connectivity of each server to ensure they are correctly connected to the network and can communicate with other servers. This involves pinging between the servers and conducting traceroute tests to verify the network paths.

Firewall Configuration: An essential part of these tests includes verifying that the firewall settings on each server and within the network do not impede the necessary communications. This step is critical to ensure that the servers can freely exchange perfSONAR-related data without any hindrance.

Server Configuration Verification: Manual inspection of each server's configuration settings is conducted to ensure that all parameters are correctly set as per the project requirements. This includes checking network adapter settings, ILO port configurations, and ensuring that the correct SFP transceivers and cabling are in use as planned.

Baseline Performance Metrics: Initial performance metrics are gathered to establish a baseline for future comparison. This step is crucial for understanding the default operational state of each server in its respective environment.

By meticulously executing these manual tests, we aim to establish a solid foundation for the subsequent phases of the project. Identifying and addressing any connectivity, configuration, or performance issues at this early stage is crucial for the smooth progression of the project. This careful validation ensures that we have a reliable and accurately functioning network of test-points, setting the stage for the effective implementation of the perfSONAR framework in a real-world environment.

4.4.2 Automation using PSConfig File

While manual access to individual test-points allows us to initiate tests on demand, there is a more streamlined and automated approach for certain tests, such as latency measurements. This automation is facilitated through the utilization of the pSConfig file. This file plays a crucial role in the automation and scheduling of desired tests within the perfSONAR infrastructure.

The pSConfig file is dynamically distributed by the pspublisher module, which operates within the perfSONAR ecosystem. This module serves as a crucial mediator, communicating between different layers of perfSONAR and enabling the orchestration of

various network tests. The pSConfig file essentially acts as a configuration blueprint that specifies the types of tests to be conducted, the test parameters, scheduling details, and the involved test-point nodes.

By utilizing the pSConfig file, we can define a set of network tests, such as latency measurements, that need to be performed automatically and periodically. These configurations can be tailored to match the specific requirements of our network monitoring objectives. Once the pSConfig file is broadcasted across the network, it is consumed by the relevant test-points, which then execute the designated tests based on the defined schedule.

This automated approach significantly reduces the need for manual intervention in initiating routine tests. It enhances the efficiency of network monitoring by enabling timely and consistent measurements. Furthermore, it allows network administrators to focus on analyzing the generated data rather than spending time initiating tests manually.

A similar file to the one used on this project is available at the A

The provided `psconfig.json` file is a configuration file utilized within the perfSONAR infrastructure for the automation and scheduling of network tests. This JSON file encompasses several distinct sections, each of which specifies various aspects of the network testing setup. Let's delve into the key components and elucidate their roles:

- **Administrators and Display Name:** The `_meta` section defines administrators' contact information and a display name for the configuration. This information aids in identifying the responsible organization for the configuration.
- **Addresses:** This section defines test-point addresses along with their corresponding hosts. These addresses serve as identifiers for different test-points within the configuration.
- **Archives:** The `archives` section specifies configurations for data archiving of test results. It includes details about the archiver, data transmission, and metadata. In this instance, the configuration employs the "logstash-docker" archive.
- **Groups:** The `groups` section delineates various groups of test-points. Each group is assigned a type (mesh or disjoint) and a list of addresses. These groups facilitate the organization of tests based on their target test-points.
- **Hosts:** The `hosts` section furnishes information regarding individual test-point hosts. It encompasses display names and the archives associated with them.

- **Schedules:** The `schedules` section defines diverse schedules for tests. Each schedule specifies the repetition interval, slip interval, and slip randomness. These schedules determine the frequency of different test executions.
- **Tasks:** The `tasks` section outlines various network tests to be executed. Each task includes a display name, group affiliation, schedule, test type, and tools employed for the test. Tasks indicate which tests to perform, their scheduling, and their specific parameters.
- **Tests:** The `tests` section provides detailed specifications for each test type. It encompasses parameters such as test type, specifications for various test types (latency, throughput, trace, DNS, HTTP), and assorted test parameters.

4.4.3 Front-end Server Grafana Integration and Configuration for perfSONAR Visualization

Grafana is a widely used, open source visualization platform, offers powerful capabilities for creating interactive dashboards. With support for various data sources, including the default OpenSearch database bundled with the perfSONAR Toolkit and Archive packages, Grafana becomes an invaluable tool for visualizing perfSONAR performance metrics. This section outlines the steps to:

1. Install and Set Up Grafana: While this guide demonstrates installing Grafana through Docker, alternative installation methods are available for various operating systems.
2. Configure OpenSearch Data Source: Setting up the OpenSearch data source within Grafana allows seamless communication with perfSONAR. This connection enables data retrieval and visualization from the perfSONAR installation.
3. Install Matrix Visualization Plugin: This step introduces the installation of the Matrix visualization plugin, which enhances dashboard capabilities by providing summarization for tests.
4. Import Pre-built Dashboards: Utilize pre-existing dashboards to start visualizing perfSONAR data quickly. These dashboards, developed by the perfSONAR team, offer insightful insights and serve as examples for creating custom visualizations.

The following prerequisites are necessary for utilizing this guide:

- A perfSONAR host with version 5.0 or later, having either the Toolkit or Archive package installed.
- A separate host capable of running Grafana, which can access port 443 on the perfSONAR host.

Step 1: Install and Set Up Grafana

This step covers the setup of Grafana using a Docker container. It's important to note that Docker setup outlined here is suitable for non-production environments and is not designed for persisting changes.

1. Log in to the host where Grafana will be installed, and ensure Docker is present (Docker installation instructions can be found on Docker's website).
2. Start the Grafana container using the command below (Linux users might need to use `sudo`):

   ```
   docker run -d -p 3000:3000 -name grafana grafana/grafana-oss
   ```

3. Check if Grafana is running by opening a web browser and navigating to `http://localhost:3000` (or replace "localhost" with your host's name if Grafana is installed remotely). You should see the Grafana login page.
4. Log in using the default credentials: admin/admin. You'll be prompted to change the default password.

Step 2: Configure OpenSearch Data Source

To connect Grafana with perfSONAR, the OpenSearch data source plugin is required:

1. Access Grafana's settings by clicking the cogwheel icon on the left and selecting "Plugins."
2. Search for "OpenSearch" in the plugin search bar.
3. Install the "OpenSearch" plugin from the results.
4. Create an OpenSearch data source by clicking "Create a OpenSearch data source" at the top of the page.

On the configuration page:

- Name: Provide a name for the data source, such as "perfSONAR Measurements".
- URL: Enter the URL in the format:

  `https://PERFSONAR_DataBase_HOST/opensearch`, replacing "PERFSONAR_DataBase_HOST" with the address of your perfSONAR archive host.
- Skip TLS Verify: Enable this toggle due to self-signed certificates typically used in perfSONAR installations. Disable if a trusted certificate is in place.
- Index name: Set as `pscheduler*`.
- Time field name: Enter `pscheduler.start_time`.

Typically, configuring Grafana involves a streamlined process where users can set up their desired data sources and dashboards without requiring authentication. However, in a specific case related to the opensearch packages, a software bug has surfaced. This bug mandates the usage of valid login credentials in the authentication fields during configuration.

Due to this issue, users who intend to configure Grafana with the opensearch data source need to provide appropriate login details for authentication. This is a temporary workaround necessitated by the bug that affects the opensearch packages.

It's worth noting that this bug has been identified and reported to the GÉANT team, responsible for the development of perfSONAR and its related components. The team is actively working on resolving this issue, and a fix is anticipated in the near future. This demonstrates the proactive approach taken to address software discrepancies and ensure the smooth functioning of the perfSONAR ecosystem.

Once the bug is rectified, users will likely be able to resume the streamlined configuration of Grafana without requiring login credentials. This scenario highlights the collaborative nature of software development and the commitment to providing users with a robust and user-friendly experience.

Save and test the configuration. A successful message indicates Grafana's successful connection to the perfSONAR Archive.

Step 3: Install the Matrix Plugin

To enable the Matrix panel for summarizing tests:

1. Access Grafana's settings and go to "Plugins."
2. Search for "Matrix" in the plugin search bar.

3. Install the "ESnet Matrix Panel" plugin.

Step 4: Import Pre-built Dashboards

By using the pre-built dashboads we can simplify data visualization:

1. In the Grafana menu, click the "four square" icon and select "+ Import."
2. Copy the content of the JSON file present in the perfSONAR Overview Grafana Dashboard JSON website [35] into the "Import via panel json" text box.
3. Click "Import."
4. Repeat the above steps for the perfSONAR Endpoint Pair Grafana Dashboard JSON website [36]

With imported dashboards, explore visualizations and understand data displayed. Importantly, these dashboards serve as templates for your custom creations.

Step 5: Build Your Own Line Graph

This final step guides you in creating a line graph to display maximum packet loss over time:

1. In the Grafana menu, select "+ New dashboard."
2. Click "Add a new panel" and select the "perfSONAR Measurements" data source and "Time series" visualization.
3. Configure the graph title to "Maximum Packet Loss Over Time."
4. Edit the query to calculate the "Max" value of the "results.packets.loss" field.
5. Add units to the Y axis to display the maximum packet loss as a percentage.
6. Save your dashboard and explore customizations.

This completes the setup of Grafana for perfSONAR visualization, enabling you to create insightful and dynamic dashboards that enhance your network monitoring capabilities. In the Figure 4.8 it is possible to observe a first draft of the graphs, this were populated with data from the Phase 2 (4.4) implementation.

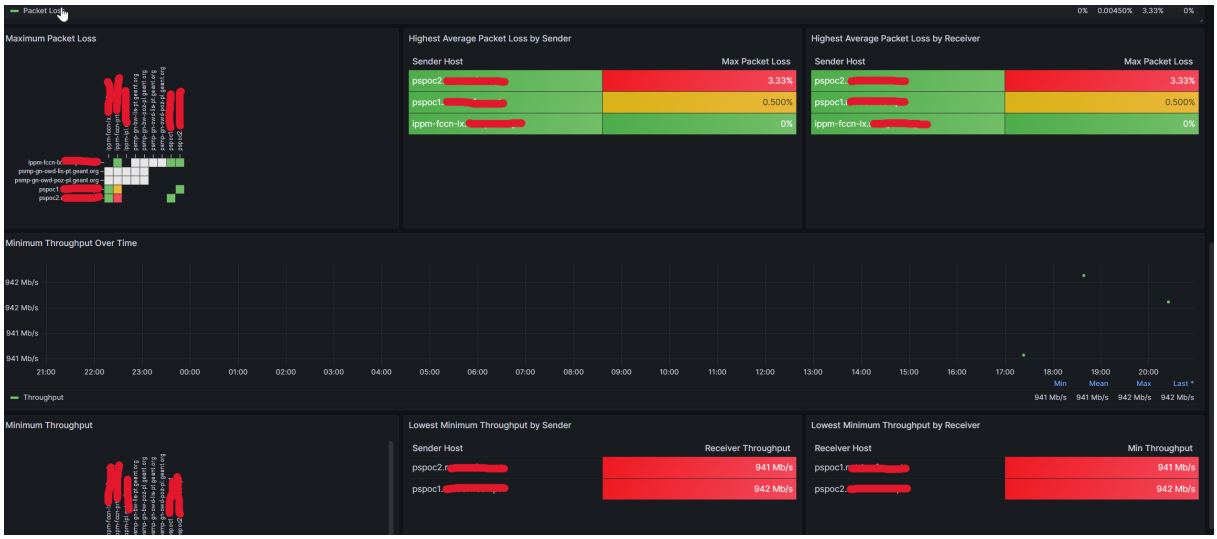

Figure 4.8: Initial Grafana Draft with Phase 2 Data

4.5 Phase 3: Nationwide perfSONAR Test-point Server Deployment

Phase 3 represents a significant expansion in the scope of our perfSONAR implementation, as we extend the network to cover the entire nation. This phase is both ambitious and complex, involving the deployment of additional perfSONAR test-point servers across various geographical regions and network domains. The primary goal of this phase is to achieve comprehensive coverage and enable extensive monitoring and optimization of network performance on a national scale.

The nationwide deployment of perfSONAR test-point servers in this phase necessitated a carefully coordinated logistical plan. Due to the diverse locations of the test-points, each deployment required a unique approach, factoring in geographical challenges, local network infrastructure, and institutional requirements. Consequently, this phase involved a varied number of trips to set up each test-point. Some locations were easily accessible and required minimal setup time, while others presented logistical challenges, necessitating multiple visits to ensure proper installation and integration with the local network infrastructure.

The setup of each test-point was meticulously executed. This process included installing the Supermicro Nexus SuperChassis 506TQC-R301 servers, configuring network connections (including 1 GB ILO connections, 10 GB, and 100 GB ports as required), and ensuring that all hardware and software components were functioning correctly. The setup also involved testing each server's connectivity and performance, both individually and in relation to the broader network, to confirm that there were no

underlying network or firewall issues.

Upon the successful setup of all test-points, a crucial step was the update of the pSConfig file. This file, which is central to the perfSONAR architecture, was meticulously revised to include every newly deployed test-point. The updated pSConfig file ensured that each test-point was correctly registered and integrated into the network, allowing for seamless communication and data exchange among all nodes. This step was essential for enabling the comprehensive monitoring and analysis capabilities of the perfSONAR system across the entire network.

A significant aspect of this phase was the migration of front-end and database servers from a staging to a production environment. This transition marked a major upgrade in terms of system availability and reliability. By moving to a production environment, the front-end and database services were no longer hosted on a single server but were instead distributed across a cluster with significant geographic redundancy. This setup provided enhanced fault tolerance, better load balancing, and improved overall system resilience, which are critical for supporting the expanded network infrastructure.

As we progressed with the deployment of our nationwide perfSONAR network, the integration of Grafana for visualization became increasingly crucial. Grafana, with its powerful graphing capabilities, was tailored to provide insightful and real-time visualizations of the network's performance data. However, we faced significant troubleshooting challenges due to issues with our database backend, specifically the OpenSearch database.

The primary problem we encountered was related to the way our OpenSearch database was handling the large number of test-points in our expanded network. OpenSearch databases use a sharding mechanism to distribute data across multiple nodes, enhancing performance and scalability. However, each shard has resource limitations, and with our growing network, we found that the database was consistently reaching the maximum number of available shards.

This issue was exacerbated by the default database configuration, which created a new index each day. Since every new index consumed multiple shards, the rapid accumulation of daily indices led to an excessive consumption of shards, severely impacting the database's performance and scalability.

To address this challenge, we shifted our strategy from the daily creation of indices to the implementation of datastreams. Datastreams are a feature in OpenSearch that allow for the continuous, time-series data management more efficiently than traditional index-based approaches. By adopting datastreams, we were able to streamline how data was stored and managed in our database. The use of datastreams brought several

key advantages:

1. **Improved Shard Management:** Datastreams significantly reduced the number of shards being consumed. This was because datastreams allow for better control over the lifecycle of the data, meaning that we could more efficiently manage how data was stored and aged out of the system.

2. **Scalability and Performance:** With the reduced shard usage, the overall scalability and performance of our OpenSearch database improved. This was crucial in managing the increased volume of data from the expanded number of test-points.

3. **Enhanced Data Organization:** Datastreams provided a more organized way of handling time-series data, which was essential for our network performance data. This organization made it easier to query and visualize data in Grafana, leading to more efficient and insightful monitoring.

The transition to datastreams was a pivotal move in enhancing our network monitoring capabilities. By resolving the database shard limitation issue, we were able to better tailor the Grafana graphs to the user experience, ensuring that network administrators could access clear, timely, and relevant performance data. This change significantly improved our ability to monitor, analyze, and optimize the network performance, contributing to the overall success of the project. The graphs can be observed in the following Figures 4.9 and 4.10.

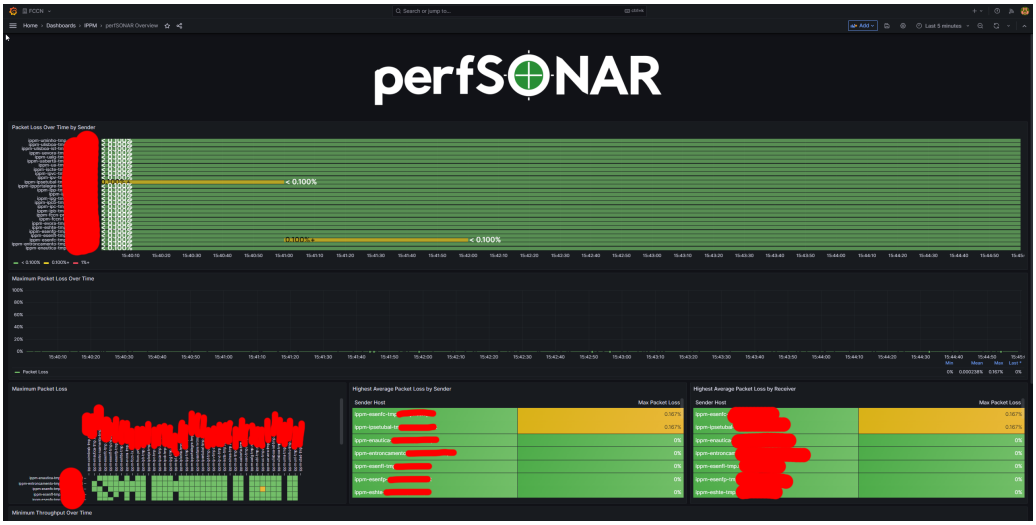

Figure 4.9: Grafana Phase 3 NationWide Graphs 1/2

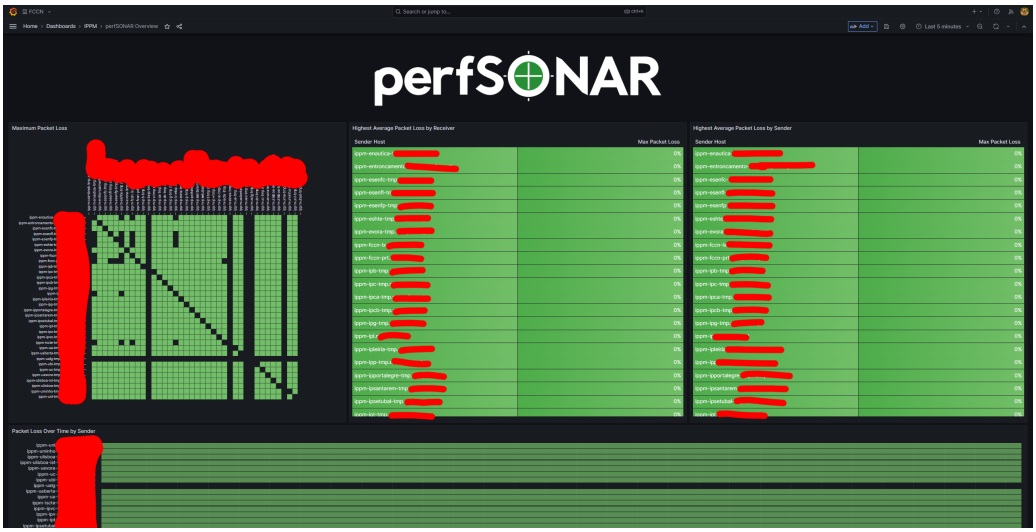

Figure 4.10: Grafana Phase 3 NationWide Graphs 2/2

4.6 PerfSonar Testing

perfSONAR is distinct from other monitoring tools like Icinga, Telemetry, or Zabbix due to its focus on end-to-end network performance testing and its ability to measure network paths across multiple network domains. This makes perfSONAR particularly suited for research and educational networks where understanding the performance over long-distance and multi-domain paths is crucial. The importance of the perfSONAR tests, as opposed to other systems, lies in their methodology of testing directly between two points rather than through a central server. While tools like Icinga and Zabbix are excellent for monitoring the status and health of network devices and services, perfSONAR specializes in measuring metrics such as latency, packet loss, and throughput over large-scale networks. This approach offers a more accurate representation of the network's performance along the specific paths data travels. It provides a realistic measure of network conditions experienced by end-users, which is particularly valuable in optimizing and troubleshooting large, distributed networks like those in educational and research institutions. Additionally, perfSONAR's wide adoption in the research and education community allows for collaborative troubleshooting and network performance optimization.

In the following figures, we will see sample outputs from latency (Figures 4.11, 4.12, 4.13), trace (Figure 4.14), clock (Figure 4.15), and RTT (Figure 4.16) tests performed between two perfSONAR test points. These outputs provide insight into the network performance between these points, highlighting key aspects such as data transmission delays, clock synchronization, and the total RTT of packets in the network. These

results are crucial for assessing and optimizing network performance in educational and research environments. From this single tests we manage to get the information that at the time of testing we were experiencing a time offset of 5,507 miliseconds, or that the connection between both testpoints did not have any packet loss and had a mean latency of 0,35 miliseconds. However the real value of this tests is manly not in the single use scenario, but in the mass repetition throughout the day, the values are thereafter stored in the database servers and later displayed in Grafana. When displayed in Grafana we are able to more easily detect if for example the values of latency are changing in a meaningful way at any time of the day.

```
^C[root@ippm-fccn-lx /]# pscheduler task latency --dest ippm-ipl.[redacted]
Submitting task...
Task URL:
https://ippm-fccn-lx/pscheduler/tasks/97fea029-0753-4d0d-8645-2d0c8bbbf50d
Running with tool 'owping'
Fetching first run...

Next scheduled run:
https://ippm-fccn-lx/pscheduler/tasks/97fea029-0753-4d0d-8645-2d0c8bbbf50d/runs/5eab6505-ee7e-43ff-b394-e
0001818a87c
Starts 2024-01-20T19:40:27-05:00 (~2 seconds)
Ends   2024-01-20T19:40:49-05:00 (~21 seconds)
Waiting for result...

Packet Statistics
-----------------
Packets Sent ......... 100 packets
Packets Received ..... 100 packets
Packets Lost ......... 0 packets
Packets Duplicated ... 0 packets
Packets Reordered .... 0 packets

One-way Latency Statistics
--------------------------
Delay Median ......... -0.36 ms
Delay Minimum ........ -0.51 ms
Delay Maximum ........ -0.25 ms
Delay Mean ........... -0.35 ms
Delay Mode ........... -0.30 ms
Delay 25th Percentile ... -0.39 ms
```

Figure 4.11: perfSONAR Latency test output 1/3

4.6.1 Phase 3: Testing Results

To analyze and visualize the network's performance, we employed a structured approach leveraging the capabilities of both the perfSONAR monitoring system and advanced data analysis tools. Initially, we configured perfSONAR to conduct automatic, scheduled tests across the network. These tests were meticulously designed to measure various aspects of network performance, including latency, packet loss, and throughput, among others. Upon completion, the results of these perfSONAR tests were systematically transmitted to an OpenSearch database. OpenSearch, a scalable search and analytics engine, served as a centralized repository for storing the test data, facilitating efficient data management and retrieval processes.

Subsequently, we utilized Grafana, a powerful open-source platform for monitoring

```
Delay 25th Percentile ... -0.39 ms
Delay 75th Percentile ... -0.30 ms
Delay 95th Percentile ... -0.26 ms
Max Clock Error ..... 0.32 ms
Common Jitter Measurements:
    P95 - P50 ........ 0.10 ms
    P75 - P25 ........ 0.09 ms
    Variance ......... 0.00 ms
    Std Deviation .... 0.06 ms
Histogram:
    -0.51 ms: 1 packets
    -0.47 ms: 1 packets
    -0.45 ms: 2 packets
    -0.44 ms: 3 packets
    -0.43 ms: 3 packets
    -0.42 ms: 2 packets
    -0.41 ms: 8 packets
    -0.40 ms: 3 packets
    -0.39 ms: 7 packets
    -0.38 ms: 7 packets
    -0.37 ms: 9 packets
    -0.36 ms: 5 packets
    -0.35 ms: 7 packets
    -0.34 ms: 1 packets
    -0.33 ms: 1 packets
    -0.32 ms: 4 packets
    -0.31 ms: 5 packets
    -0.30 ms: 10 packets
    -0.29 ms: 8 packets
    -0.28 ms: 6 packets
```

Figure 4.12: perfSONAR Latency test output 2/3

and visualization, to fetch and interpret the data stored in the OpenSearch database. Grafana's sophisticated visualization capabilities allowed us to create a series of dashboards and graphs that provide comprehensive insights into the network's operational status and performance metrics. This integration not only streamlined the process of data analysis but also enabled us to present the results in an easily digestible and visually appealing format. By harnessing the combined power of perfSONAR, OpenSearch, and Grafana, we established an effective workflow for continuous network monitoring, data storage, and result visualization, thus enhancing our ability to maintain optimal network performance and reliability.

In the subsequent figures, we are presented with a comprehensive visual representation of the network's performance testing outcomes, effectively illustrating the intricacies of our monitoring approach. Initially, we encounter a general matrix of testing on Figure 4.19, a pivotal component of our analysis. This matrix showcases the results derived from each test point within the network conducting tests to every other test point. It serves as a holistic overview, enabling us to swiftly identify patterns, potential bottlenecks, or areas requiring attention across the entire network infrastructure.

Following the general matrix, the figures transition to detailed graphs that delve into the specific metrics of network performance between two designated test points. These graphs meticulously chart the results of testing focused on critical parameters such as jitter Figure 4.20, latency Figure 4.17, and packet loss Figure 4.18. Jitter graphs depict the variability in time delay between packets arriving, which is crucial for understanding the stability of the network's time-sensitive data transmission. Latency graphs

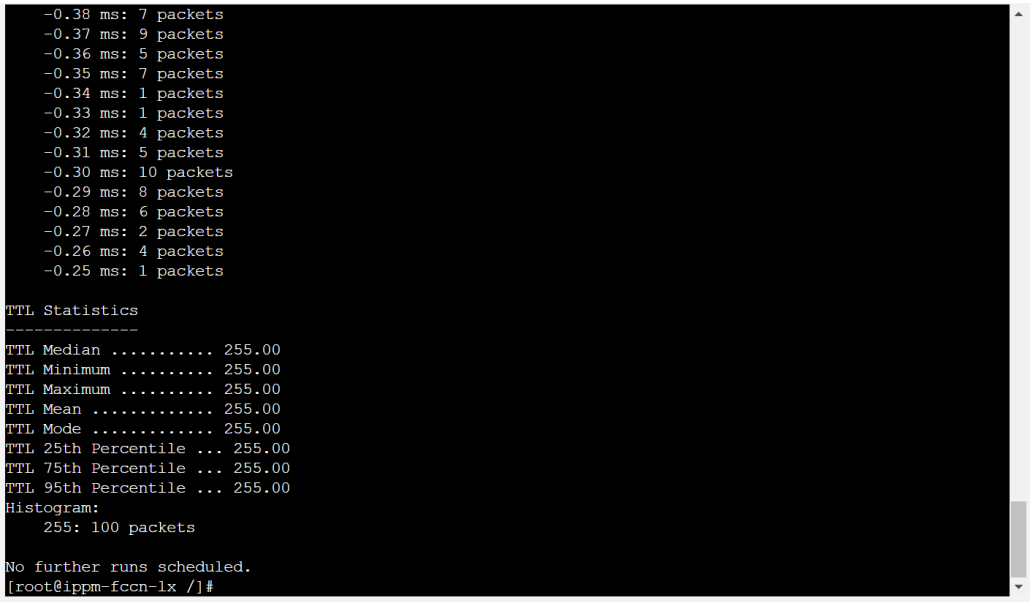

Figure 4.13: perfSONAR Latency test output 3/3

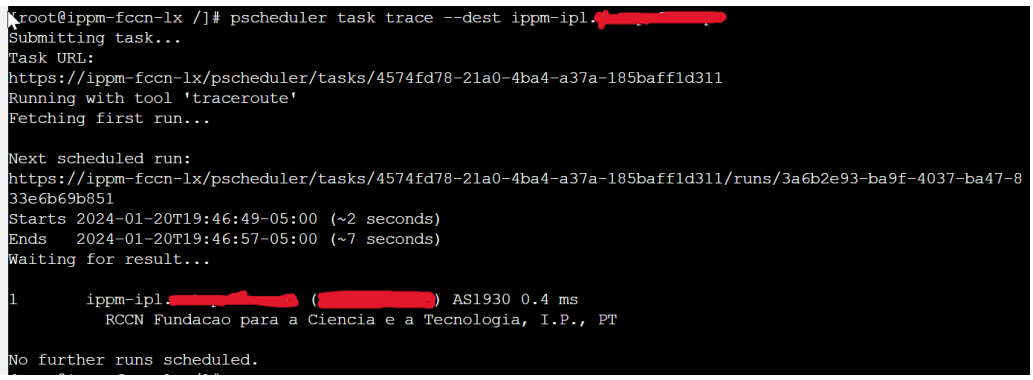

Figure 4.14: perfSONAR Trace test output

illustrate the time taken for a packet to travel from one test point to another and back, offering insights into the network's efficiency and responsiveness. Lastly, packet loss graphs reveal the percentage of packets that fail to reach their destination, highlighting potential reliability issues within the network. In the Figures 4.13 and 4.14 shown in this case we may infer that the packet loss value stayed constant at zero, and that has expected latency and jitter varied but under a reasonable window of variance indicating that there were no problems with the connection.

Together, these Figures provide a dual-layered perspective on network performance, combining a macroscopic overview with targeted, microscopic analyses. This comprehensive approach allows network administrators to not only grasp the network's overall health but also to pinpoint specific segments or connections that may be under performing or experiencing issues, thereby facilitating targeted troubleshooting and

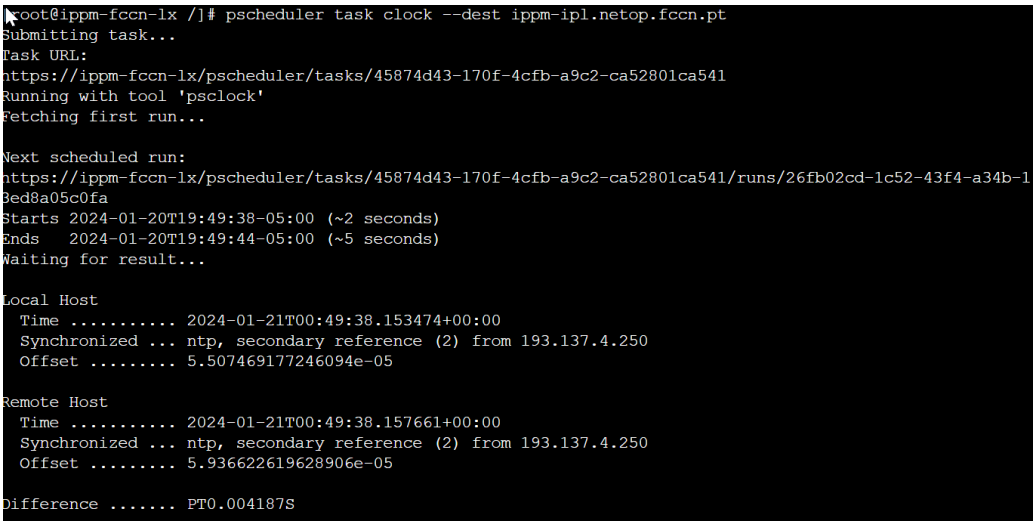

Figure 4.15: perfSONAR Clock test output

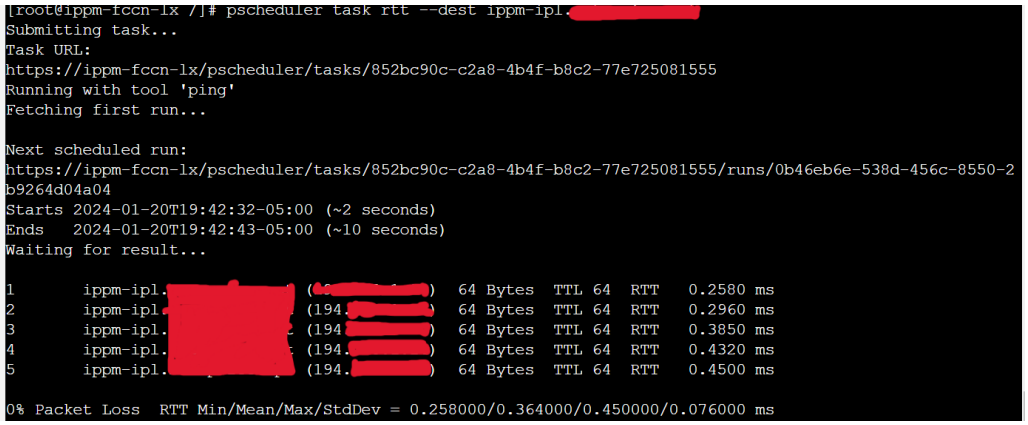

Figure 4.16: perfSONAR RTT test output

optimization efforts.

4.6.2 PerfSonar vs Other Monitoring Systems Already Implemented in FCCN

Currently, FCCN Monitoring Service encompasses multiple applications divided by the following branches: Alarmistic, Visualization and Support Systems. Icinga, primarily an alarm tool, effectively runs templated predefined checks, which can be greatly enhanced with the use of scripts or a simple ICMP or SNMP-get, with the sole purpose to build a robust alert dashboard. In that sense, it provides real-time reporting capabilities that allow for proactive network management. Furthermore, its scalable architecture and easy-to-use web interface make it ideal for networks of different sizes. Figures 4.21 and 4.22 show a series of tests conducted using the Icinga monitoring tool.

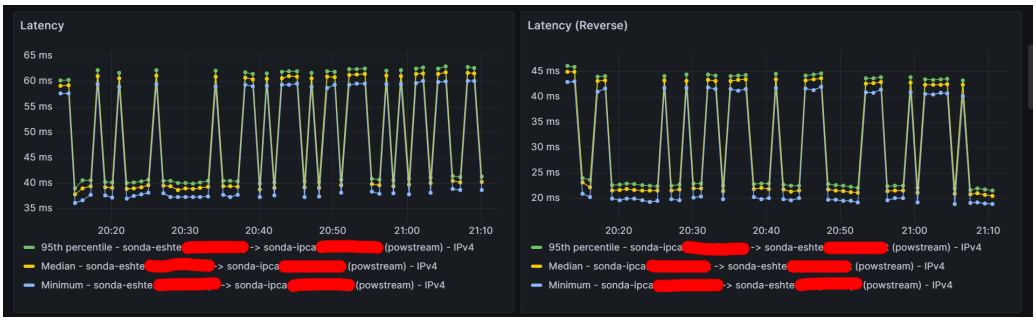

Figure 4.17: perfSONAR Latency Grafana

Figure 4.18: perfSONAR PacketLoss Grafana

We can observe that although some tests seem similar the truth of the manner is that they are inherently different, the values of packet loss and latency showed on Icinga are referring to tests made between the central Icinga server and a remote endpoint, however when using perfSONAR we are not dependent of a central server meaning that if a location A becomes unreachable, from a location B and C, but not from a location D, we may quickly pinpoint were exactly the network problem is occurring, this being something that is not possible using only Icinga. Even more has every Icinga test originate on the central server we are unable to reproduce a key test of perfSONAR, that being one-way-delay.

In the Visualization department, a great effort was made the last two years to migrate the primary visualization tool, from Cacti to Grafana, providing state-of-the-art network and systems graphs and analytics. Support this applications, there are a backbone of databases, data transformation/treatment, inventory and data collection applications. Although the systems are fairly new and stable, they still rely heavily on SNMP, to fetch crucial data to feed Icinga and Grafana for an instance.

This causes constrains in terms of visibility and performance, since SNMP collection interval is slow (1-5m) for today's standards and since there is a ramp of applications wanting to use SNMP and since it's an old and ineffective protocol, it can cause performance problems of the all solution and even of the network equipment's. That's why, a

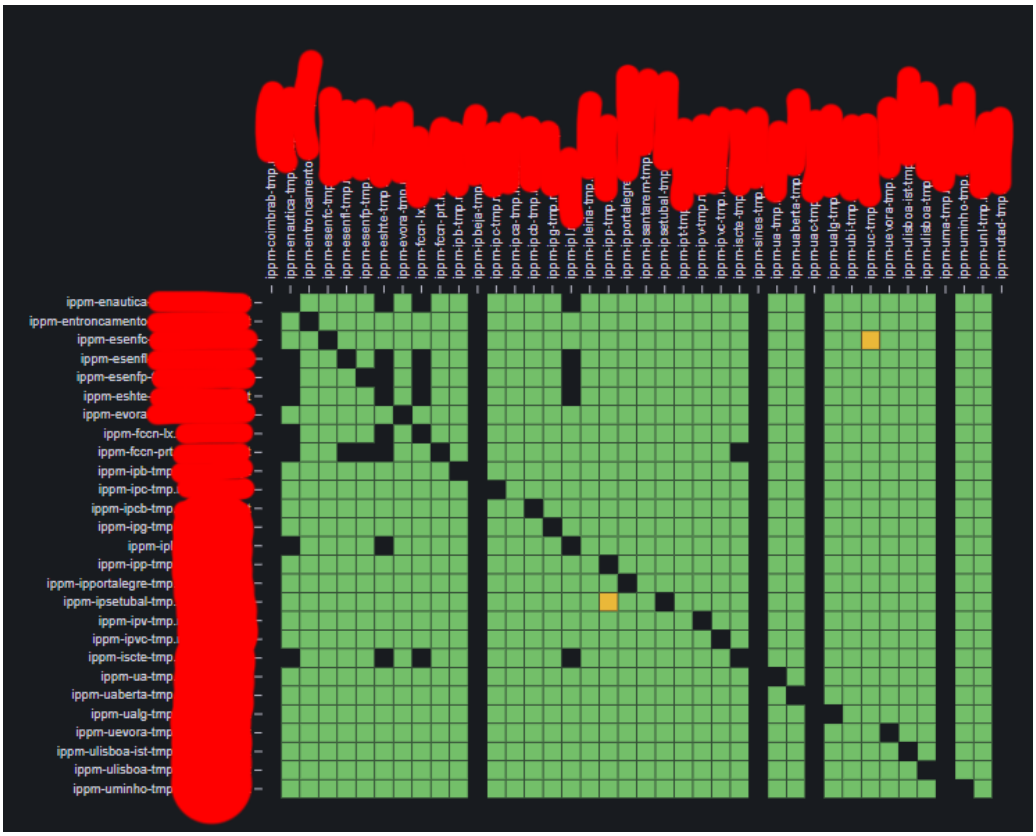

Figure 4.19: perfSONAR Matrix Grafana

model is being tested and evaluated this year to use Streaming Telemetry alongside the Telegraph, InfluxDB & Grafana (TIG) stack to improve visibility to help troubleshoot and that equals to faster problem resolution. The drive to change was motivated by the inability to fetch important data in equipment's, ie: Ethernet Virtual Private Network (EVPN)/perf-measurments and short spaced glitches where is very hard to see.

Despite the strengths of the current monitoring system and Streaming Telemetry, there is an opportunity for further enhancement through the introduction of perfSONAR. perfSONAR is a network measurement toolkit designed to provide detailed active measurements of network performance metrics such as latency, packet loss, and throughput. This tool can automate and schedule network performance tests, providing a clear advantage over manually configuring network equipment to generate active traffic. Moreover, instead of running tests between individual network equipment, perfSONAR allows for end-to-end testing due to the availability of multiple test point servers. This not only saves time but also provides a more comprehensive picture of network performance. perfSONAR's synthetic traffic generation also contributes to its excellent fault detection capabilities. It allows for the identification and mitigation of network issues before they impact end users, enhancing the overall network fault

Figure 4.20: perfSONAR Jitter Grafana

detection and mitigation framework provided by the other tools. The combination of Streaming Telemetry and perfSONAR will provide FCCN with a holistic and multi-dimensional view of network performance. While Icinga offers system and network device monitoring, Streaming Telemetry provides passive network performance insights, and perfSONAR delivers detailed active network performance measurements. This comprehensive network monitoring framework, complete with automated and end-to-end testing, will significantly enhance the overall management and reliability of FCCN's network operations.

In this fourth chapter we find documented, the practical aspects of implementing perfSONAR in the RCTS network. It described the network architecture, hardware components, and connectivity solutions used in the deployment. The implementation process is detailed through various phases, from initial proof of concept to nationwide deployment. Each phase includes step-by-step instructions, troubleshooting tips, and special considerations, such as IPv6 in Docker Swarm. The chapter also presents testing results and comparisons with other monitoring systems, highlighting the effectiveness of perfSONAR.

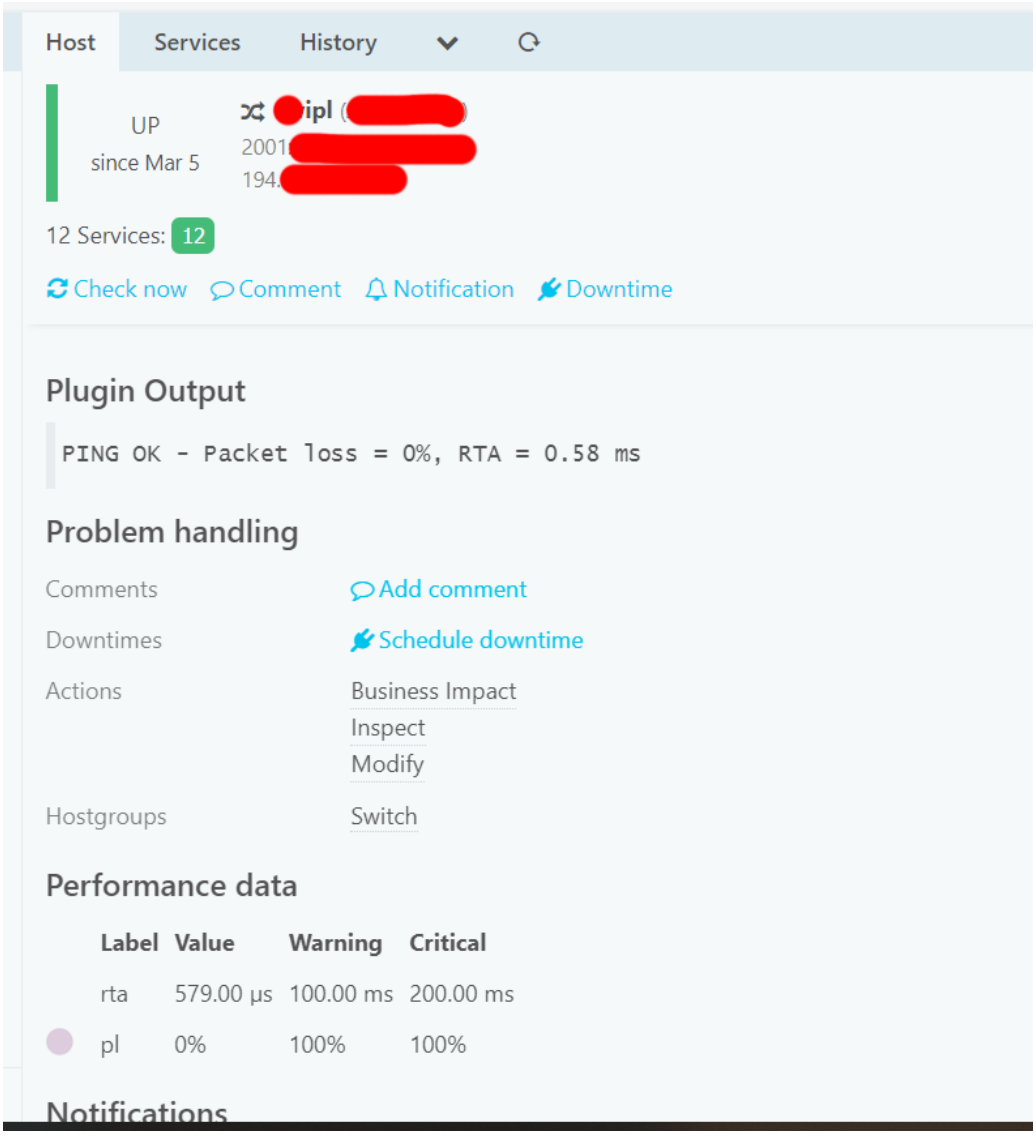

Figure 4.21: Icinga Tests 1/2

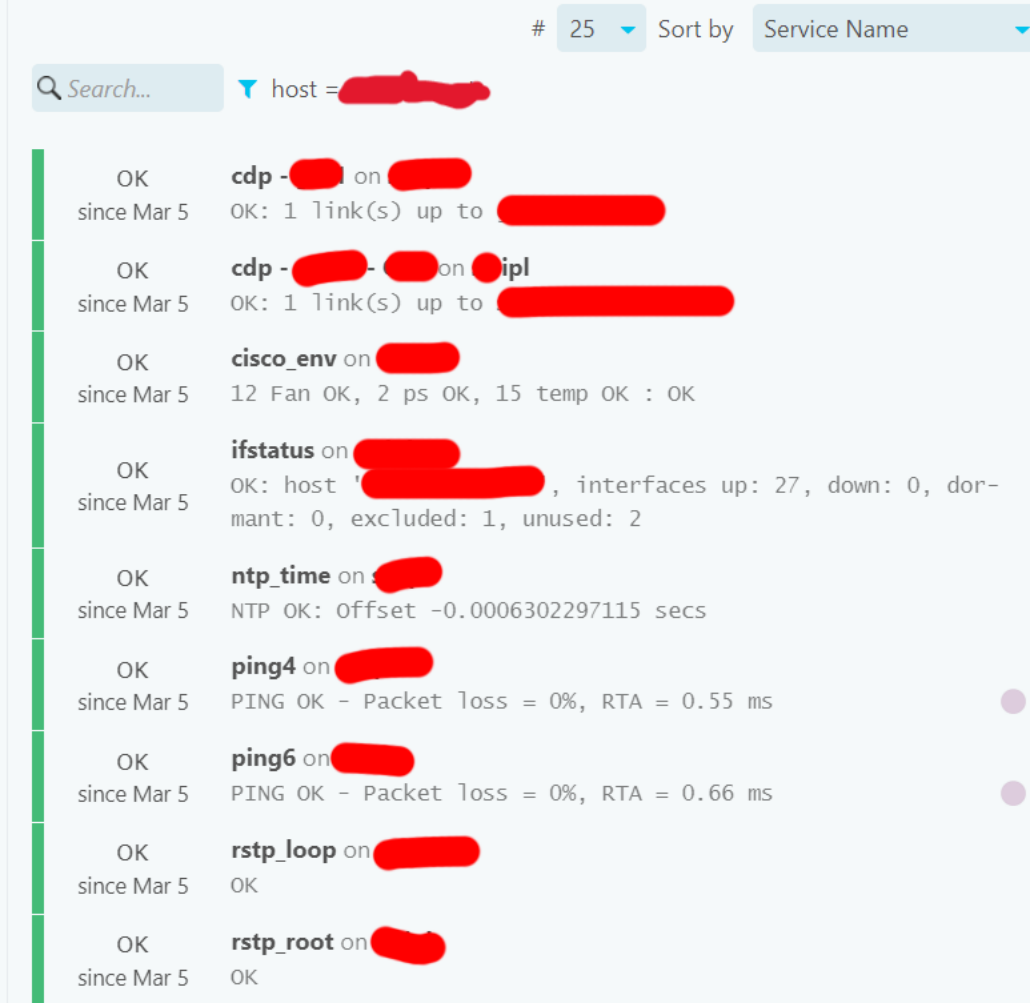

Figure 4.22: Icinga Tests 2/2

5

Conclusions and Future Work

This dissertation represents a significant journey in the deployment and effectiveness analysis of the perfSONAR network across Portugal's higher education institutions, a pivotal step in advancing the digital infrastructure within the academic and research domains. The initiative aimed at enhancing network performance and monitoring, addressing the evolving needs of a complex and dynamic digital landscape.

The project commenced with an initial POC, a critical phase that laid the foundation for what was to become a comprehensive network performance monitoring system. The success of the POC was instrumental in demonstrating the feasibility and potential impact of the perfSONAR network. It served as a litmus test, validating the conceptual underpinnings of the project and setting the stage for more extensive implementation phases. This phase was characterized by meticulous planning, rigorous testing, and an unwavering commitment to achieving a high standard of network performance analysis.

Following the POC, the project transitioned into an intermediate phase, marked by the strategic distribution of test-point servers across various geographic locations. This phase was instrumental in broadening the scope of the project, enabling a more comprehensive assessment of network performance across diverse environments. It provided a unique opportunity to gauge the adaptability and resilience of the perfSONAR network in different institutional settings, offering invaluable insights into network behaviors and performance under varying conditions.

The apex of the project was the full-scale, nationwide deployment of the perfSONAR

network. This phase was a monumental undertaking, extending the network's reach to cover the entire educational landscape of Portugal. Achieving comprehensive network coverage, this phase allowed for the collection of extensive data and insights from across the educational network, fostering a deeper understanding of the network's overall health and efficiency. The nationwide deployment was not just a technical achievement but also a testament to the project's scalability and its ability to cater to a wide array of network environments.

Throughout the project, several challenges were encountered, particularly in the realm of database management. The OpenSearch database system, initially struggling with shard capacity limitations, posed a significant obstacle. The innovative solution to this problem involved transitioning from traditional index creation to the more efficient datastreams methodology. This change significantly enhanced the database's performance and scalability, addressing one of the critical bottlenecks in the network's operation. Furthermore, the integration of Grafana for data visualization emerged as a pivotal element in the project. It transformed the way complex network data was interpreted, facilitating more effective troubleshooting and providing a user-friendly interface for network administrators.

Looking to the future, the potential for expanding the range of perfSONAR tests is considerable. Collaborations with entities like GÉANT are set to propel the development of new and innovative tests, expanding the project's scope and capabilities. One particularly promising area of development is the creation of tests designed to monitor and validate Eduroam authentication processes. Given the widespread use of Eduroam in educational institutions, robust mechanisms for seamless and secure authentication are paramount. The proposed tests aim to focus on real-time monitoring of the authentication process, identifying any issues or delays that could impact user experience and system reliability.

The integration of these new tests with advanced tools like Grafana is anticipated to enhance our ability to create detailed and customized visualizations. Such graphical representations are crucial in simplifying the interpretation of complex data, enabling quick identification of trends and anomalies. Customizing graphs to reflect specific metrics and performance indicators related to Eduroam authentication will equip network administrators with insightful and actionable information, aiding in decision-making and problem resolution.

The continuous development and integration of these tests into the existing perfSONAR framework is imperative. It ensures that the network monitoring capabilities remain robust, scalable, and responsive to the evolving demands of the network infrastructure.

This adaptability is particularly important in the diverse and expanding landscape of network infrastructure, where new challenges and requirements emerge continually.

In conclusion, the successful deployment of the perfSONAR network represents a significant stride forward for Portugal's educational and research network infrastructure. Enhancing current capabilities and laying a solid foundation for future technological advancements, this project underscores the critical importance of strategic planning, innovative problem-solving, and adaptability in the rapidly evolving domain of network performance and optimization. Reflecting on this journey, the dissertation highlights the transformative power of technology in enhancing the capabilities of the academic and research communities. The deployment of the perfSONAR network, from its inception to its nationwide implementation, serves as a testament to the collaborative spirit, technical acumen, and forward-thinking approach essential for advancing digital infrastructure in the 21st century.

References

[1] T. P. Team, *Active Vs. Passive Monitoring: Which is Best for Your Network? - WhatsUp Gold*, [Online; accessed 2022-12-17], 2020. [Online]. Available: https://www.whatsupgold.com/blog/active-vs.-passive-monitoring-which-is-best-for-your-network.

[2] *Understanding the Differences Between Active vs. Passive Monitoring - Instatus blog*, [Online; accessed 2023-11-02]. [Online]. Available: https://instatus.com/blog/active-vs-passive-monitoring.

[3] A. Morton, *Rfc 7799: Active and Passive Metrics and Methods (with Hybrid Types In-Between)*, [Online; accessed 2023-10-01]. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7799.html#section-3.1.

[4] *Projeto de redes - métricas de qualidade de serviço em redes de computadores*, [Online; accessed 2023-10-28]. [Online]. Available: https://www.projetoderedes.com.br/artigos/artigo_metricas_qos_em_redes.php.

[5] E. Stephan, L. Liang, and A. Morton, *Rfc 5644: Ip Performance Metrics (IPPM): Spatial and Multicast*, [Online; accessed 2023-01-08]. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5644.

[6] V. Paxson, Lawrence Berkeley, G. Almes, J. Mahdavi, and M. Mathis, *Framework for IP Performance Metrics*, [Online; accessed 2023-01-05], May 1998. [Online]. Available: https://www.ietf.org/rfc/rfc2330.txt.

[7] *Rfc ippm Metrics for Measuring Connectivity*, [Online; accessed 2023-11-06], 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2498.

[8] *Ippm (IP Performance Metrics) (Linktionary term)*, [Online; accessed 2023-12-07]. [Online]. Available: https://www.linktionary.com/i/ippm.html.

[9] E. Grossman and Ed., *Deterministic Networking Use Cases*, [Online; accessed 2022-12-17]. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8578.txt.

[10] D. Mills, *Rfc 4330: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, [Online; accessed 2022-12-27]. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4330.

[11] J. Postel and K. Harrenstien, *Rfc 868: Time Protocol*. [Online]. Available: https://www.rfc-editor.org/rfc/rfc868.

[12] J. Postel, "User Datagram Protocol", Internet Engineering Task Force, RFC 768, 1980. [Online]. Available: https://doi.org/10.17487/RFC0768.

[13] J. Postel, "Transmission Control Protocol", Internet Engineering Task Force, RFC 793, 1981. [Online]. Available: https://doi.org/10.17487/RFC0793.

[14] *Difference between NTP and PTP - GeeksforGeeks*, 2021. [Online]. Available: https://www.geeksforgeeks.org/difference-between-ntp-and-ptp/.

[15] "White Paper Introduction to NTP", *Endrun Technologies*, [Online; accessed 2022-11-12]. [Online]. Available: https://endruntechnologies.com/pdf/NTP-Intro.pdf.

[16] *Rfc 8173: Precision Time Protocol Version 2 (PTPv2) Management Information Base*, 2017. [Online]. Available: https://datatracker.ietf.org/doc/rfc8173/.

[17] V. Shankarkumar, L. Montini, T. Frost, and G. Dowd, *Rfc 8173: Precision Time Protocol Version 2 (PTPv2) Management Information Base*. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8173.html.

[18] *Conceitos básicos de PTP e SyncE com a configuração do Cisco IOS XR*, [Online; accessed 2022-11-13]. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-xr-software/217579-configure-ptp-and-synce-basics-with-cisc.html.

[19] *Everything You Have To Know About PTP or Precision Time Protocol? - Moniem-Tech*, [Online; accessed 2023-02-18], 2022. [Online]. Available: https://moniem-tech.com/2022/05/13/everything-you-have-to-know-about-ptp-or-precision-time-protocol/.

[20] *Precision Time Protocol Software Configuration Guide for IE 2000u and Connected Grid Switches*, 2019. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cg-switch-sw-master/software/configuration/guide/ptp/b_ptp_ie2ku.html.

[21] *Configuring Precision Time Protocol (PTP)*, [Online; accessed 2022-11-13]. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3650/software/release/16-12/configuration\textunderscore{}guide/lyr2/b\textunderscore{}1612\textunderscore{}lyr2\textu.

[22] *Profiles | IEEE P1588 Working Group*, [Online; accessed 2022-12-18]. [Online]. Available: https://sagroups.ieee.org/1588/ptp-profiles/.

[23] *Precision time protocol software configuration guide for ie 2000u and connected grid switches*. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cg-switch-sw-master/software/configuration/guide/ptp/b_ptp_ie2ku.html.

[24] Douglas Arnold, *Ntp vs PTP: Network Timing Smackdown!*, [Online; accessed 2022-10-30], 2012. [Online]. Available: https://blog.meinbergglobal.com/2013/11/22/ntp-vs-ptp-network-timing-smackdown/.

[25] Teodor Neagoe, Valentin Cristea, and Logica Banica, "Ntp versus ptp in computer networks clock synchronization", in *2006 IEEE International Symposium on Industrial Electronics*, vol. 1, 2006, pages 317–362. DOI: 10.1109/ISIE.2006.295613. [Online]. Available: https://datatracker.ietf.org.

[26] *Perfsonar*, [Online; accessed 2023-03-10], 2021. [Online]. Available: https://network.geant.org/perfsonar/.

[27] *Rcts Network*, 2021. [Online]. Available: https://www.fccn.pt/en/quem-somos/rede-rcts-rede-ciencia-tecnologia-e-sociedade/.

[28] *GéANT*, [Online; accessed 2023-03-04], 2019. [Online]. Available: https://geant.org/.

[29] *Atlas Console*, [Online; accessed 2023-03-13]. [Online]. Available: https://atlas.ripe.net/landing/probes-and-anchors/.

[30] *Introduction*, [Online; accessed 2023-03-13]. [Online]. Available: https://ring.nlnog.net/introduction/.

[31] *Samknows*, [Online; accessed 2023-03-03]. [Online]. Available: https://www.samknows.com.

[32] Pedro Queirós, Nuno Santos, and Maria João Nicolau, "Monitorização da qualidade de serviço da rede portuguesa de investigação e ensino (rcts)", 2013.

[33] *What is perfsonar? — perfsonar toolkit 5.0.7 documentation*, https://docs.perfsonar.net/intro_about.html, Accessed: INSERT-DATE-OF-ACCESS, n.d.

[34] Supermicro, *Sc506tqc-r301*, Accessed: 2024-06-06, 2024. [Online]. Available: https://www.supermicro.com/en/products/chassis/mini-1u/506/sc506tqc-r301.

[35] GÉANT, *Perfsonar overview grafana dashboard json*, https://raw.githubusercontent.com/perfsonar/grafana/main/examples/overview.json, Retrieved November 20, 2023, n.d.

[36] GÉANT, *Perfsonar endpoint pair grafana dashboard json*, https://raw.githubusercontent.com/perfsonar/grafana/main/examples/endpoints.json, Retrieved November 20, 2023, n.d.

PSConfig template file

```
{
"_meta": {
    "administrators": [
        {
            "email": "admin1@fccn.pt",
            "name": "Admin1"
        },
        {
            "email": "admin2@fccn.pt",
            "name": "Admin2"
        }
    ],
    "display-name": "FCCN"
},
"addresses": {
    "pspoc1": {
        "address": "pspoc1.dns.pt.pt",
        "host": "pspoc1"
    },
    "pspoc2": {
        "address": "pspoc2.dns.pt.pt",
        "host": "pspoc2"
    },
    "ippm-ipl": {
        "address": "ippm-ipl.dns.pt.pt",
        "host": "ippm-ipl"
```

```
        },
        "ippm-fccn-lx": {
            "address": "ippm-fccn-lx.dns.pt.pt",
            "host": "ippm-fccn-lx"
        },
        "ippm-fccn-prt": {
            "address": "ippm-fccn-prt.dns.pt.pt",
            "host": "ippm-fccn-prt"
        }

    },
    "archives": {
        "logstash-docker": {
            "archiver": "http",
            "data": {
                "schema": 2,
                "op": "put",
                "_url": "https://perfsonar-archive.dns.pt.pt
                /logstash",
                "_headers": {
                    "x-ps-observer": "{% scheduled_by_address %}",
                    "Content-Type": "application/json"
                }
            },
            "_meta": {
                "esmond_url": "https://perfsonar-archive-dns
                /esmond/perfsonar/archive/"
            }
        }
    },
    "groups": {
        "grp_latency": {
            "type": "mesh",
            "addresses": [
                { "name": "pspoc1" },
                { "name": "pspoc2" },
                { "name": "ippm-ipl" },
                { "name": "ippm-fccn-lx" },
                { "name": "ippm-fccn-prt" }
            ]
        },
        "grp_throughput": {
            "type": "mesh",
            "addresses": [
                { "name": "pspoc1" },
```

```
                { "name": "pspoc2" },
                { "name": "ippm-ipl" },
                { "name": "ippm-fccn-lx" },
                { "name": "ippm-fccn-prt" }
            ]
        },
        "grp_dns_http": {
            "type": "disjoint",
            "unidirectional": true,
            "a-addresses": [
                { "name": "pspoc1" }
            ],
            "b-addresses": [
                { "name": "pspoc2" }
            ]
        }
    },
    "hosts": {
        "pspoc1": {
            "_meta": {
                "display-name": "perfSONAR One"
            },
            "archives": [ "logstash-docker" ]
        },
        "pspoc2": {
            "_meta": {
                "display-name": "perfSONAR Two"
            },
            "archives": [ "logstash-docker" ]
        },
        "ippm-ipl": {
            "_meta": {
                "display-name": "perfSONAR ippm-ipl"
            },
            "archives": [ "logstash-docker" ]
        },
        "ippm-fccn-lx": {
            "_meta": {
                "display-name": "perfSONAR ippm-fccn-lx"
            },
            "archives": [ "logstash-docker" ]
        },
        "ippm-fccn-prt": {
            "_meta": {
                "display-name": "perfSONAR ippm-fccn-prt"
```

```
            },
            "archives": [ "logstash-docker" ]
        }
    },
    "schedules": {
        "schedule_long": {
            "repeat": "PT21600S",
            "slip": "PT10800S",
            "sliprand": true
        },
        "schedule_medium": {
            "repeat": "PT3600S",
            "slip": "PT1800S",
            "sliprand": true
        },
        "schedule_short": {
            "repeat": "PT600S",
            "slip": "PT300S",
            "sliprand": true
        }
    },
    "tasks": {
        "latency": {
            "_meta": {
                "display-name": "One Way Delay"
            },
            "group": "grp_latency",
            "test": "tst_owd_4"
        },
        "throughput": {
            "_meta": {
                "display-name": "Throughput"
            },
            "group": "grp_throughput",
            "schedule": "schedule_long",
            "test": "tst_bw_4",
            "tools": [ "iperf3" ]
        },
        "traceroute-latency": {
            "_meta": {
                "display-name": "Traceroute Latency group"
            },
            "group": "grp_latency",
            "schedule": "schedule_short",
            "test": "tst_trace_4"
```

```
        },
        "traceroute-throughput": {
            "_meta": {
                "display-name": "Traceroute Throughput group"
            },
            "group": "grp_throughput",
            "schedule": "schedule_short",
            "test": "tst_trace_4"
        },
        "dns": {
            "_meta": {
                "display-name": "DNS resolution"
            },
            "group": "grp_dns_http",
            "schedule": "schedule_medium",
            "test": "tst_dns"
        },
        "http": {
            "_meta": {
                "display-name": "HTTP connexion"
            },
            "group": "grp_dns_http",
            "schedule": "schedule_medium",
            "test": "tst_http"
        }
    },
    "tests": {
        "tst_owd_4": {
            "type": "latencybg",
            "spec": {
                "bucket-width": 0.0001,
                "dest": "{% address[1] %}",
                "flip": "{% flip %}",
                "ip-version": 4,
                "packet-count": 600,
                "packet-interval": 0.1,
                "packet-padding": 0,
                "source": "{% address[0] %}"
            }
        },
        "tst_bw_4": {
            "type": "throughput",
            "spec": {
                "dest": "{% address[1] %}",
                "duration": "PT20S",
```

```
                "ip-version": 4,
                "omit": "PT10S",
                "source": "{% address[0] %}",
                "bandwidth": "1G"
            }
        },
        "tst_trace_4": {
            "type": "trace",
            "spec": {
                "dest": "{% address[1] %}",
                "hops": 64,
                "ip-version": 4,
                "length": 40,
                "probe-type": "udp",
                "source": "{% address[0] %}"
            }
        },
        "tst_dns": {
            "type": "dns",
            "spec": {
                "record": "a",
                "query": "{% address[1] %}"
            }
        },
        "tst_http": {
            "type": "http",
            "spec": {
                "url": "{% address[1] %}"
            }
        }
    }
}
```