# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

Modern organisations increasingly depend on externally managed services to manage internal communications, data processing, and information dissemination. While such interconnected infrastructures improve operational efficiency and capability, they simultaneously expand the attack surface exposed to malicious actors and introduce system dependencies on third-party providers. The 2016 DDoS attack against Dyn brought down major online platforms across Europe and the United States [1]. Furthermore, internal failures within these third-parties can cause large-scale outages, as showcased by the 2025 Cloudflare disruption [2].

Central systems in security-critical groups require safeguarding from untrusted networks, often isolated through network segregation strategies and access control policies. Such measures significantly reduce the likelihood of conventional remote attacks, but they disconnect external resources and services, limiting system functionality [3]. In this scenario, the organisation may have to develop (or integrate) self-hosted solutions to maintain operational capability while meeting security requirements.

Air-gapped systems are physically isolated from external and untrusted networks, having no network interfaces connected to other networks. Air-gapping is generally regarded as an effective hardware security measure for protecting critical systems in national security and financial sectors. Some literature extends the definition of air gapping to include logically isolating a system via firewalls or network routing [3, 4], yet the existence of zero-day exploits challenges the security capabilities of software-based isolation.

Controlled data transfer between systems operating at different trust levels is an eventuality even in highly segregated or air-gapped environments. Although these channels may be intended to be unidirectional, data transfer protocols do not inherently protect against opposing information flows. Covert channels may emerge by leveraging network-layer mechanisms [5]. Unidirectional data transfer solutions that employ hardware-security measures may provide a secure mechanism for air-gapped networks to interact with untrusted external services.

## 1.2 Motivation

The present software ecosystem is predominantly designed for persistent internetworking. Modern applications assume access to cloud-based services and remotely managed tools, making them unsuitable for isolated environments. Moreover, air-gapped organisations often impose organisational-specific requirements that cannot be compensated by commercial solutions.

Such products are designed for a broad customer base and are often distributed as closed-source systems. The lack of lightweight, customisable, and self-contained software solutions represents a practical limitation that motivates the present work.

Air-gapped environments must still exchange information with external networks to support operational requirements. Provisioning such interactions without compromising system isolation presents a major engineering challenge. Conventional bidirectional communication introduces the risk of unintended information flows and network-level exploits. The work discussed is motivated by the need to design and implement a secure data exchange mechanism capable of enabling controlled external data transfer while preserving the security requirements of air-gapped networks.

Present day cloud-based and externally hosted services prioritise the user experience, emphasising seamless integration and accessibility to enable efficient organisational workflows. Secure systems developed for air-gapped environments should achieve comparable measures of usability to ensure effective adoption. Complex configurations and deployment procedures must be relinquished from the end-user to ensure security capabilities are leveraged effectively.

## 1.3 Objective

- *To design and implement a hardware-enforced unidirectional data transfer mechanism for secure communication with air-gapped networks*
- *To develop a lightweight, self-hosted software module capable of operating within air-gapped environments*
- *To integrate a plug-and-play design philosophy for seamless integration and deployment.*

## 1.4 Target Specifications

**Table 1.1:** Target specifications of the Proposed Data Diode

| ID | Specification | Description | Priority |
|---|---|---|---|
| TS-01 | Unidirectional communication | Physical enforcement of one-way transfer. | Primary |
| TS-02 | Air-gapped operation | Operations independent of internet connectivity or external network services. | Primary |
| TS-03 | Lightweight deployment | Self-contained and deployable offline. | Secondary |
| TS-04 | Plug-and-Play Design | Interface shall minimise end-user error and avoid manual workarounds. | Secondary |
| TS-05 | Throughput | $\geq 900\,\mathrm{Mbit\,s^{-1}}$. | Secondary |

## 1.5 Project Work Schedule

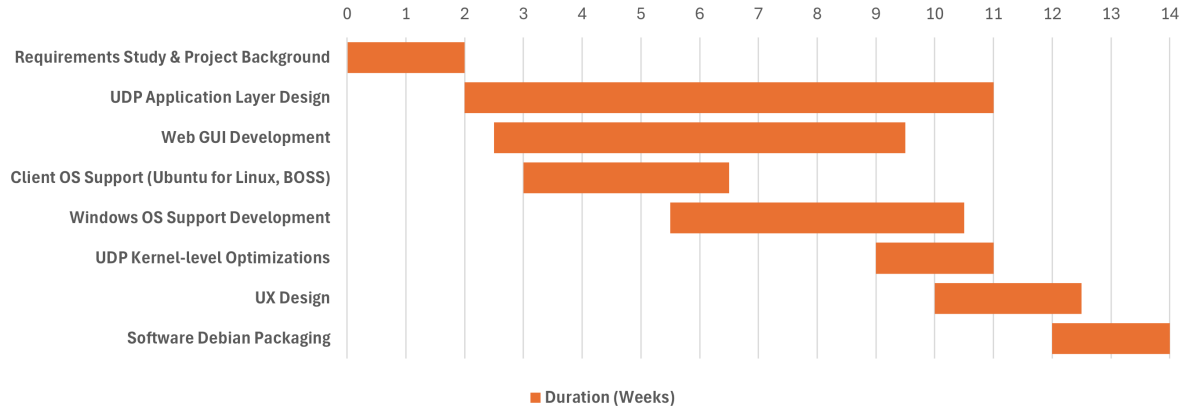



**Figure 1.1:** Project Work Schedule - February to May 2026

## 1.6 Organisation of the Report

This report follows with 4 chapters. Chapter 2 presents the background concepts and reviews related works concerning air-gapped systems, unidirectional data transfer mechanisms, and human-centric software requirements engineering. Chapter 3 specifies the system architecture and implementation of the proposed project. Chapter 4 presents the testing results and system validation. Finally, Chapter 5 summarises the outcomes and discusses potential improvements.

# CHAPTER 2

# BACKGROUND THEORY / LITERATURE REVIEW

Presented is the theoretical background relevant to the development of self-hosted software for air-gapped systems. Fundamental concepts comprising network isolation and simplex data transfer are introduced to establish the foundation of the work. Subsequently, existing research on data diode architectures and hardware-enforced communication mechanisms is examined in a literature review. The chapter aims to highlight the references (and their limitations) that motivate the design of the proposed solution.

## 2.1 Background Theory

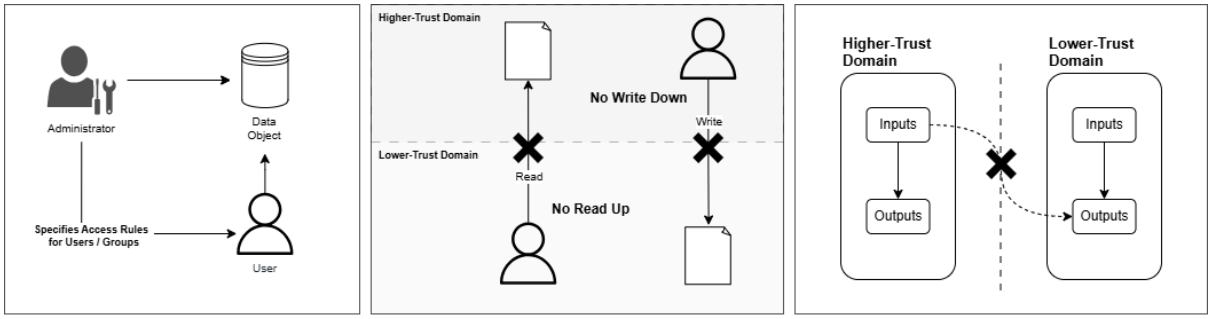
### *2.1.1 Network Isolation and Trust Levels*

Air-gapping has been introduced as an important security mechanism to isolate critical systems. The absence of physical communication channels significantly reduces the risk of network exploitation, scanning, and data [3]. The requirement for network isolation may arise for systems where complications would result in severe operational consequences. This includes defence communications, financial transactions, and industrial control systems. A mission-critical system may operate in a networked environment with distinct trust domains.

While external networks are considered untrusted, internal domains may also have differing privileges. Network segregation through physical isolation establishes a rigid access policy. Yet, operational requirements necessitate periodic interactions; a mission-critical network or system still needs external process updates. Considerations must be made to augment network channels into controlled information streams.

### *2.1.2 Access Control and Information Flow*

Data confidentiality in multi-user systems was introduced as discretionary, user-based permission mechanisms, which later evolved into policy-driven security actions. Early approaches regulated *who* could access a resource, with frameworks such as the Bell-LaPadula model and Biba model formalising restricted interactions between differing privilege levels. Recent studies have extended this concept beyond access policies toward regulating information propagation. The approach of Noninterference dictates system behaviour and response: actions performed at a higher security level shall not influence behaviour observed at lower levels. Hardware states in operating systems may expose high-level interactions to low-level processes.




**Discretionary Access Control (DAC)**
Data Object owners / administrators explicitly define access permissions for respective users or groups.

**Bell-LaPadula Model**
Lower-trust users shall not read higher-trust data and Higher-trust user interactions should not leak over to a lower-trust domain.

**Noninterference (Information Flow Control)**
Lower-level inputs should provide the same output regardless of higher-trust level interactions. Higher-level inputs should not influence lower-level output.

**Figure 2.1:** Conceptual Evolution of Confidentiality Enforcement Models, introduced as explicit permission policies, later improving security in multi-user systems through policy-driven access management and information flow control.

Noninterference identifies the scenario of interactions between lower-trust and higher-trust domains but is generally considered too strict to apply in actual systems [6]. A hardware-enforced communication mechanism that constrains information flow between domains of differing trust levels provides a practical realisation of this concept.

### *2.1.3 Unidirectional Data Transfers*

Asymmetric connectivity serves as the foundation for developing a secure communication channel between systems operating at different trust levels. A receiving system cannot transmit any information back to the transmitting system. Standard network protocols assume receiver feedback is available and depend on acknowledgement mechanisms, session establishment, and retransmission requests. Even when pathways are physically unidirectional, communication protocols incorporate feedback mechanisms. For example, UART simplex channels are coupled with hardware flow control mechanisms to establish a bidirectional link.

Logically isolated systems (via firewalls or routing controls) still allow potential reverse signalling through their underlying network-layer mechanisms. When communication is strictly unidirectional, these assumptions no longer hold. Consequently, implementing unidirectional transfer as a data diode introduces several engineering challenges. Reliability must be ensured without acknowledgements. These constraints distinguish a hardware-enforced data diode from logical restrictions.

The User Datagram Protocol (UDP) is found to be suitable for implementing communication systems. Unlike TCP/IP, UDP does not require session establishment, message acknowledgement, or retransmission mechanisms. While UDP does not guarantee message delivery, ordering, or reliability, these responsibilities may be delegated to the application-layer specialised to control data transfer across a unidirectional channel.

## 2.2 Literature Survey

**Table 2.1:** Summary of Literature Survey

| Ref. | Methodology | Findings |
|---|---|---|
| [4] | Penetrates air-gapped network via removable USB attack device. | Data exfiltration success rate of 99.706%. Highlights the MITRE ATT&CK framework as a threshold for attack techniques. |
| [7] | Encodes sensitive data as electromagnetic transmissions emanated over Ethernet cables. | Error-free data exfiltration up to a distance of $4.5\,\mathrm{m}$. Operates successfully under specific deployment assumptions. |
| [8] | Combines a data diode, encryption, and a genuine random number generator. Modifies Serial port to allow data diode insertion. | Distinguishes the proposed low-cost implementation of the data diode as opposed to speed-optimised commercial solutions, prioritising system security by coupling robust encryption. |
| [9] | Implements a data diode through hardware-level optical isolation, transmitting data over UDP. | Reports 97.5% successful data transmissions with an average duration of $2.07\,\mathrm{s}$ at an assumed bitrate of $1\,\mathrm{Gbit\,s^{-1}}$. |
| [10] | Couples hardware-enforced optical isolation with file hashes and rate-limiting to improve reliability. | Identifies potential optimisations in hardware configurations to improve throughput to $\approx 750\,\mathrm{Mbit\,s^{-1}}$. |

This section reviews studies relevant to air-gapped system security and unidirectional data transfer methods. The proposed solution references recent advancements in information flow control and open-source data diode prototypes.

### *2.2.1 Data Exfiltration in Air-gapped Networks*

Offensive security research has shown that air-gapped systems may be vulnerable under specific threat assumptions. N. Mohamed *et al.* presented a method for data exfiltration from air-gapped networks without the need for privilege escalation, utilising a removable USB drive as the attack vector [4]. Their methodology assumes having initial access to the target system, typically achieved through social engineering attacks such as phishing. In operational environments where hardware interfaces are strictly managed, this scheme may not suffice. As a result, the threat model pertaining to the proposed work shifts its focus from external removable media to securing the managed data channel itself.

Electromagnetic leakage from communication interfaces and data storage devices has been explored as a potential attack vector; physical media may unintentionally function as a covert transmission channel. M. Guri demonstrated a data exfiltration technique in which sensitive

data was encoded into electromagnetic emissions radiated from Ethernet cables [7]. The study reported error-free reception at distances up to $4.5\,\mathrm{m}$. However, the authors also noted that the attack can be effectively prevented by physical isolation and the use of shielded Ethernet cables. While side-channel exfiltration remains feasible, such attacks rely on specific physical conditions and can be avoided through mindful hardware deployment.

### *2.2.2 Data Diode Architectures*

L. Gaina *et al.* conducted a systematic survey of unidirectional communication solutions to identify paradigms aligning with requirements in IoT security [11]. The survey provides a structured framework for analysing data diode implementations with respect to design philosophy and deployment context. Krause and Essig [8] proposed a solution utilising a serial communication interface with the receive-pin physically disabled, permitting the insertion of a digital buffer to hardware-enforce one-way communication. The reliance on serial communication significantly limits throughput, making the architecture unsuitable for real-time large volume transfers.

Peter Story's implementation of a data diode was designed to enable secure interaction for journalists operating within air-gapped environments [9]. The work establishes several fundamental technical requirements that align with the objectives of the presented research. However, the proposed work extends this foundation by systematically approaching the design considerations from an end-user perspective, aiming to improve deployment flexibility and scalability. Story highlights a cost-effective method for hardware isolation by enforcing physical isolation, referenced from the public OSDD project: Unidirectional communication is achieved by bridging Ethernet interfaces using media converters [12]. The study observes that cost-effective diode implementations do exist but fail to support diverse data-flow scenarios. For example, the godiode project [10] integrates a reliable UDP transfer mechanism with rate limiting but supports only batch directory transfers.

Evaluation within Story's study is performed using metrics based on the percentage of successful data transfers and the average transfer duration across configured link speeds of $100\,\mathrm{Gbit\,s^{-1}}$ to $1000\,\mathrm{Gbit\,s^{-1}}$. While these measurements provide insight into reliability at different transfer rates, the evaluation fails to reflect goodput and instead focuses on the assumed hardware link capability. File transfers and data streams are application-layer operations and should not be evaluated solely at the physical/data-link level.

Finally, the research reviews existing solutions based on the required technical expertise, aiming to develop a comparably easy-to-deploy system. Despite using relatively simple hardware, the implementation does not detail design specifications to support usability or deployment flexibility. It is important to note that while using simple hardware may streamline assembly processes, it does not necessarily improve deployment or practical use.

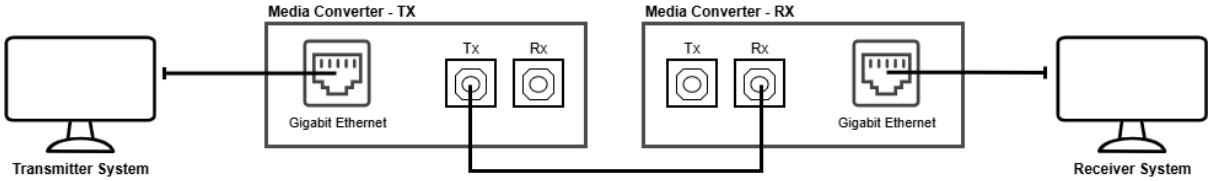



**Figure 2.2:** Hardware-enforced optical isolation of an Ethernet link achieved using media converters. The optical *Tx* interface of the transmitting converter is connected to the *Rx* interface of the receiving converter. Such converters are customised to bypass auto-negotiation requirements, by dead-ending or PCB modification.

## 2.3 Summary of Literature Review

Security in air-gapped systems depends less on the absence of network connectivity and more on definitive threat assumptions and controlled information flow. The reviewed works concerning data exfiltration in air-gapped networks demonstrate that successful attacks rely on external attack vectors rather than weaknesses inherent to the isolated environment itself. Notably, these studies indicate that enforcing strict control over communication interfaces is more critical than trying to eliminate potential covert channels.

Surveyed data diode implementations reveal a trade-off between streamlined deployment (and mutually, simplicity of design) and technical capability. Low-cost diode designs successfully enforce one-way communication but suffer from slow transfer rates. Optical isolation of a communication channel has been proven to be reliable; however, existing research provides limited insight into system performance under diverse data-flow scenarios.

## 2.4 Conclusions

The theoretical foundations and conclusions drawn from the literature survey collectively indicate that secure operations in air-gapped environments require the hardware-level enforcement of data confidentiality. The proposed work would translate this requirement into a deployable system.

Existing works reveal a disconnect in security between theoretical models and operational environments. While many implementations successfully prevent bidirectional communication, they do not adequately compensate for the limitations imposed by unidirectional channels. As such, the engineering challenge is no longer reinforcing one-way data transfer, but designing a software environment capable of operating effectively within that constraint. The following chapter showcases the design methodology for implementing the proposed solution.

# CHAPTER 3

# METHODOLOGY

This chapter presents the methodology adopted for the design and implementation of the proposed hardware-enforced data diode. The design approach is guided by the system requirements inferred from the theoretical foundations and literature review detailed in Chapter 2. Notably, open-source data diode implementations provide a practical reference for hardware configurations that enforce one-way data transfer through optical isolation [12]. Accordingly, this description focuses on the development of software modules dedicated to controlled information flow and the design of a flexible data transfer mechanism.

## 3.1 Design Methodology

The proposed system was supported by a design methodology aiming to preserve network segregation while supporting robust inter-domain communication. Rather than treating unidirectional transfer solely as a hardware-level problem, the design considers software operations a critical component of a deployable and usable tool for air-gapped environments.

Existing implementations compensate for the absence of feedback mechanisms by introducing data redundancy and forward error correction (FEC) into the transfer process [9]. Such approaches add protocol overhead that is inherently unsuitable for real-time transfer systems hoping to mirror conventional file-transfer behaviour. The proposed work seeks to simplify the communication design by practically evaluating the reliability of the underlying data link and introducing kernel-level optimisations to improve the efficiency of standard UDP data transfer.

### *3.1.1 System Assumptions*

The proposed architecture is intended to operate within or between air-gapped environments where network segregation is already established within the organisation's infrastructure. While this tool may be integrated across any communication channel connecting trusted and untrusted domains, the design assumes that the protected client would be air-gapped. A system compromise of either client endpoint does not breach the data diode's communication boundaries.

The work assumes that communication between the trust domains occurs exclusively across the provided managed data channel and that no alternative (covert) channels exist between the connected environments. The data diode is intended to enforce unidirectional communication independent of the security conditions of either endpoint. Consequently, the presented work evaluates the security and usability of the proposed transfer mechanism rather than that of the host network architecture. Table 3.1 specifies the technical assumptions supporting the design process.

**Table 3.1:** System Assumptions made in the Proposed Design

| Category | Assumption |
|---|---|
| Network Environment | Client endpoints are air-gapped and have no internet connectivity. |
| Trust Model | Channel connects a lower-trust domain to a higher-trust domain. |
| Threat Scope | Physical tampering and compromise of the system hardware is considered outside the scope of work. |
| Communication Path | Client endpoints interact exclusively through the data diode. |
| Transfer Requirement | Clients require transfer capability mirroring conventional Ethernet transfers. |
| Hardware Limitation | Application-layer UDP software is designed for a hardware-imposed upper limit of $1\,\mathrm{Gbit\,s^{-1}}$. |

## 3.2 System Architecture

The proposed system design comprises two principal components: a software module dedicated to managing the internal UDP-based unidirectional communication, and a user-facing client management service intended to simplify end-user interactions and deployment processes. This separation allowed the communication logic and user interaction to be developed independently while preserving a modular structure. The web-based GUI provides a visual interface for the client endpoint to review and interact with the lower-level UDP communication channel. Figure 3.1 encapsulates this abstraction.

The presented architecture implements a one-to-one data transfer model, where information originates from a single defined source and is delivered to a single corresponding sink in the receiving domain. This design choice is not a restriction imposed by the operational environment; it serves as an initial model to showcase the design philosophy and simplify performance evaluation. The software may be specialised to support alternative network topologies.

### *3.2.1 Overview of Data Flow*

The unidirectional data transfer tool (hereafter referred to as data diode for convenience) exposes a dedicated Ethernet interface to both client endpoints. Through this interface, the data diode hosts a web-based GUI that enables initial configurations for real-time data access with each client. The transmitter network sends files over this interface. The data diode streams this data across an optically isolated unidirectional communication channel. Upon reception, the data diode delivers the data to the receiver network through its corresponding client-facing Ethernet interface. Figure 3.2 showcases the data flow.

The data diode comprises two identical controller nodes that act as the transmitting and receiving internal endpoints of the optically isolated unidirectional channel. System operations and data management are entirely handled by these controller nodes. Chapter 3.2.2 describes the UDP-based communication service using a bottom-up abstraction approach.

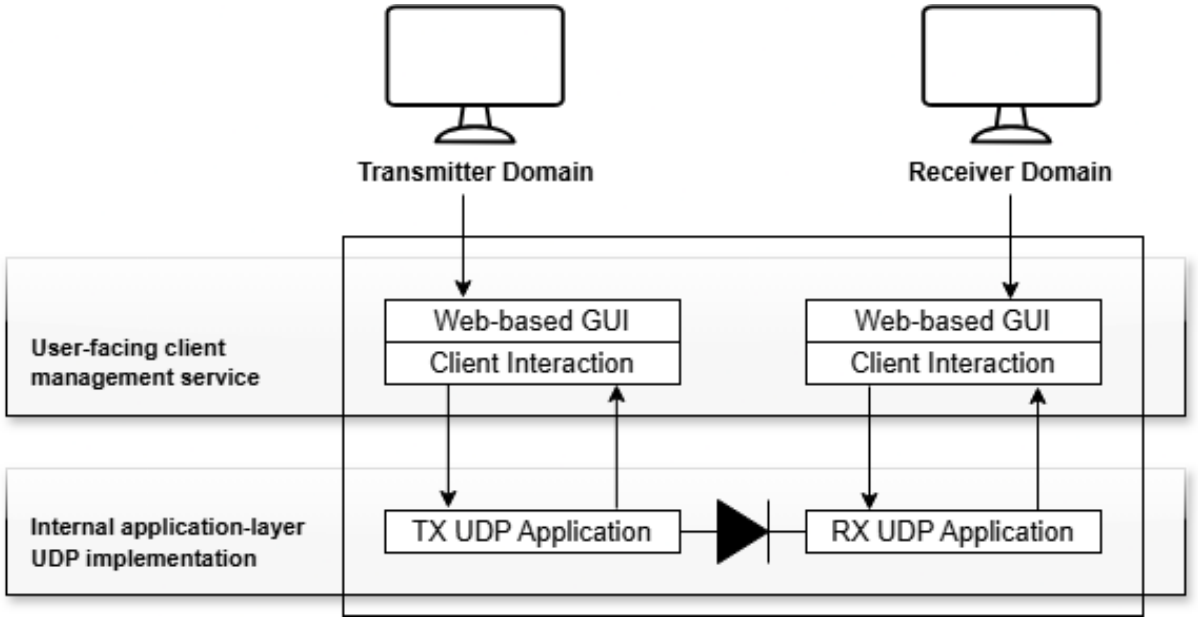



**Figure 3.1:** Principal components of the data diode. The graphical interface allows the client to control and review UDP behaviour.

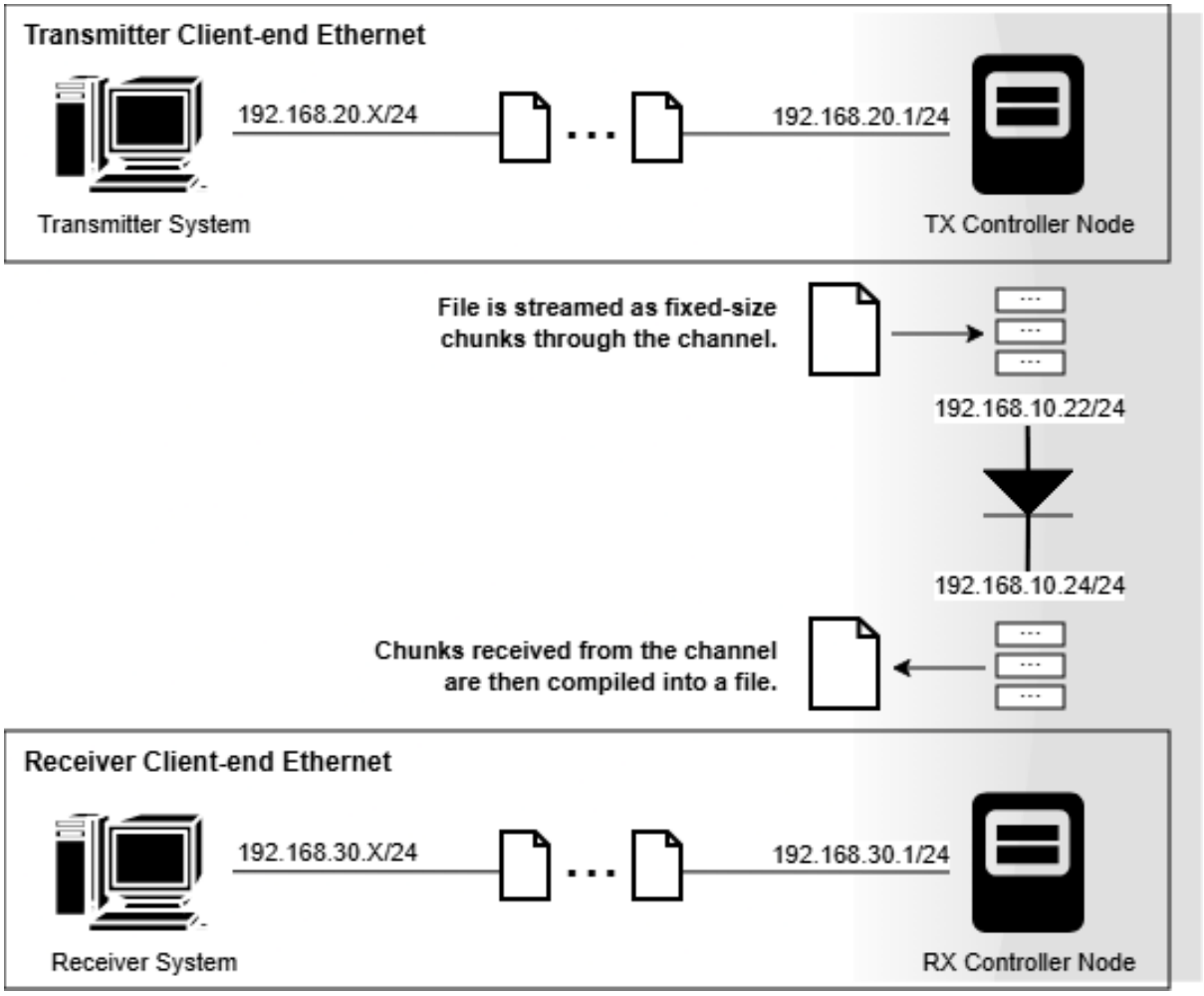



**Figure 3.2:** Overview of file transfer from a transmitter to receiver system. Files are shared in real-time with the clients and streamed as chunks across the unidirectional channel.

### *3.2.2 Reliable UDP Transfer Design*

The application layer of the UDP communication module is designed to guarantee successful file transfer over an inherently unreliable protocol. In this context, reliability is defined as the successful reconstruction of the transmitted file at the receiving endpoint, without missing data segments and with correct ordering. In practice, this may be validated by transmitting executable files whose behaviour remains identical after transmission through the diode. Rather than developing error correction or redundant transmission features, the design prioritises deterministic data delivery under the known link specifications. This allows the UDP channel to preserve high throughput while maintaining the intended security guarantees.

The transmitter controller node's (hereafter called the *sender*) transmission rate is regulated by an inter-packet gap which defines the delay between consecutive chunks. Unit testing showed that setting this inter-packet gap to $40\,\mu\mathrm{s}$ produced a throughput confined to approximately $957\,\mathrm{Mbit\,s^{-1}}$ which showcases a realistic upper bound (refer Table 3.1). Appendix A.1 details the inter-packet gap evaluation scheme.

Under these conditions, the receiver controller node (hereafter called the *receiver*) becomes the performance bottleneck due to protocol overhead and application-layer operations. Specifically, throughput worsens when the rate at which the receiver processes packets becomes slower than the sender transmission rate, or when the application-level disk write operations aggregate I/O latency during file reconstruction. To compensate, the receiver socket buffer size is increased to $2048\,\mathrm{MiB}$, allowing temporary absorption of network traffic at the cost of increased RAM usage. The receiver application incorporates a producer-consumer multithreading model where the producer thread continuously drains the receiver socket buffer into an application-layer queue, while multiple consumer threads process the queued chunks and write them to a temporary file. Appendix A.3 explains the receiver-end operations.

When transferring a file, the sender initially transmits a META packet containing the file's UUID, filename, and total file size. Subsequent data packets contain a fixed $40\,\mathrm{B}$ header, storing the corresponding file UUID, packet sequence number, expected total chunk count, and an MD5 message digest to verify file integrity. upon successful transmission, the sender transmits a DONE packet indicating that all file chunks have been sent over. This final signal prevents the receiver from indefinitely checking for incoming packets in case packet losses have occurred. At the receiving end, data chunks are written directly to their corresponding offsets within the temporary reconstruction file according to their attached sequence numbers. Thus, packet reordering does not interrupt the reconstruction process. Figure 3.3 depicts the application-level file transfer mechanism.

At the highest-level of abstraction, the UDP software incorporates mechanisms relevant to end-user interaction. To confirm whether the UDP channel is active, client endpoints must manually verify operational success at their respective ends. To simplify this process, the sender periodically generates observable signals. At startup, the sender transmits a three-packet heartbeat

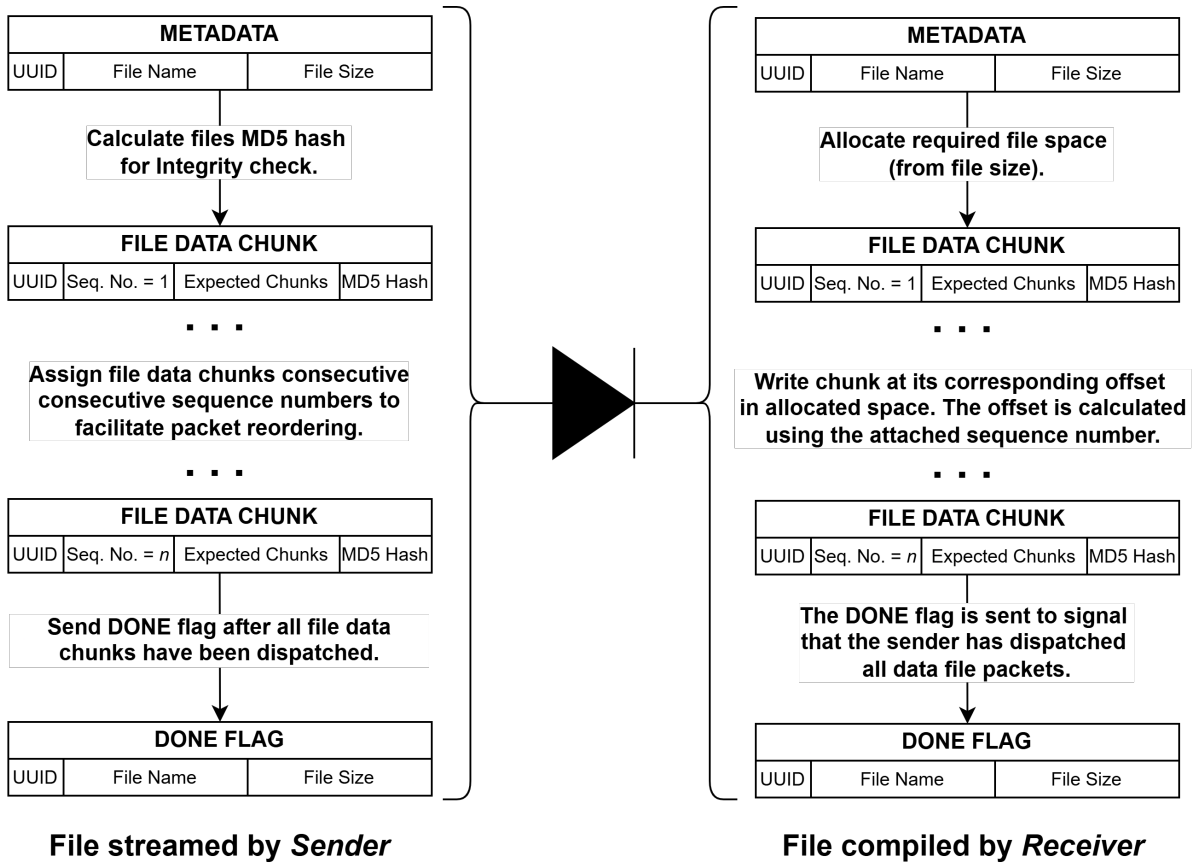



**Figure 3.3:** UDP file transfer at the application-level requires embedded metadata in each packet to enable chunk identification and deterministic file reconstruction through packet reordering.

sequence, followed by a scheduled link-test packet transmitted every 20 minutes. Additionally, the communication interfaces of the controller nodes are configured with an MTU of $9000\,\mathrm{B}$ to support Ethernet jumbo frames. Therefore, the combined size of the application-layer metadata and file data payload remains within this limit to avoid fragmentation.

### *3.2.3 Client-end Interactions*

The data diode utilises the Network File System (NFS) protocol to provide remote filesystem access to the client, aiming to minimise latency at the interface connecting the diode and client domain. Each controller node shares a local directory; this is also the working directory for the UDP application operations. The files supplied by the transmitting domain are automatically detected by the sender node and streamed across the unidirectional channel. After the receiver node reconstructs the file, it exposes the compiled data through its corresponding shared directory, allowing the receiving domain to access the data in real time. Figure 3.4 illustrates this workflow. This design choice allows the data diode to integrate into existing client-side data pipelines without requiring modifications to the client codebase or any underlying system configurations. Appendix B details the employed filesystem sharing protocols.

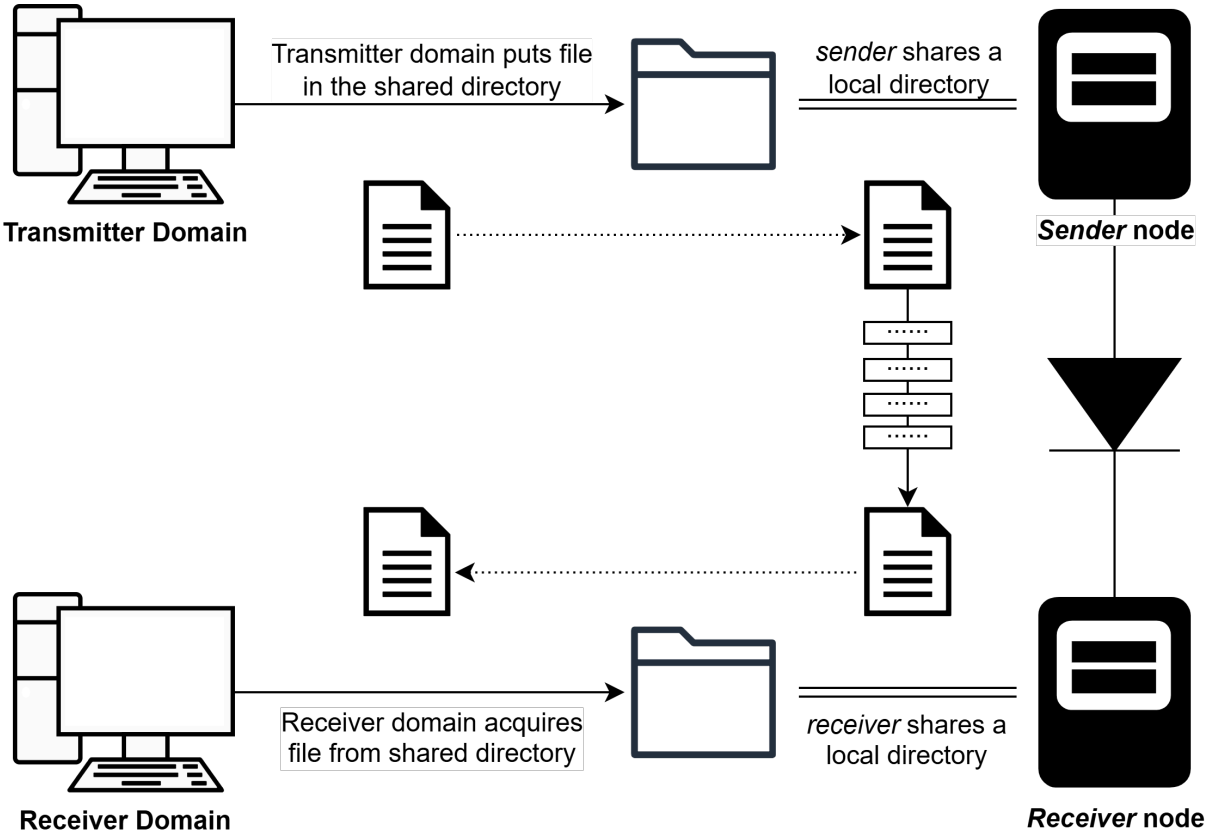



**Figure 3.4:** Network file system sharing with the client endpoints is implemented via OS-specific protocols. The transmitter and receiver domains are able to share data with the data diode in real-time.

To establish the filesystem share, configuration steps are required for both the data diode and the client endpoint. While the data diode's operations can be automated, the client domain configurations would require manual user intervention. This poses a challenge to the specified system requirement of following a plug-and-play design philosophy (refer TS-04 in Table 1.1). To resolve this, the controller node acquires remote access to the client through Secure Shell (SSH) and automatically executes the required configuration procedures. This enables the client system to connect to the data diode's exposed shared directory without requiring direct user interaction with low-level system parameters.

To establish SSH access, the controller node requires authentication details from the client system, namely the system username and password. These credentials are specified through the web-based GUI. Both controller nodes operate as DHCP servers for their respective client-end interfaces and dynamically assign an IPv4 address to the connected client systems. Thus, the assigned IPv4 address acts as a unique identifier for a user session.

With these system credentials, the controller node starts a user session. During an active session, the controller prevents other devices on the client network from accessing the shared directory. This restriction ensures that only a single data source transmits data through the data diode at a given time, simplifying data ownership logic.

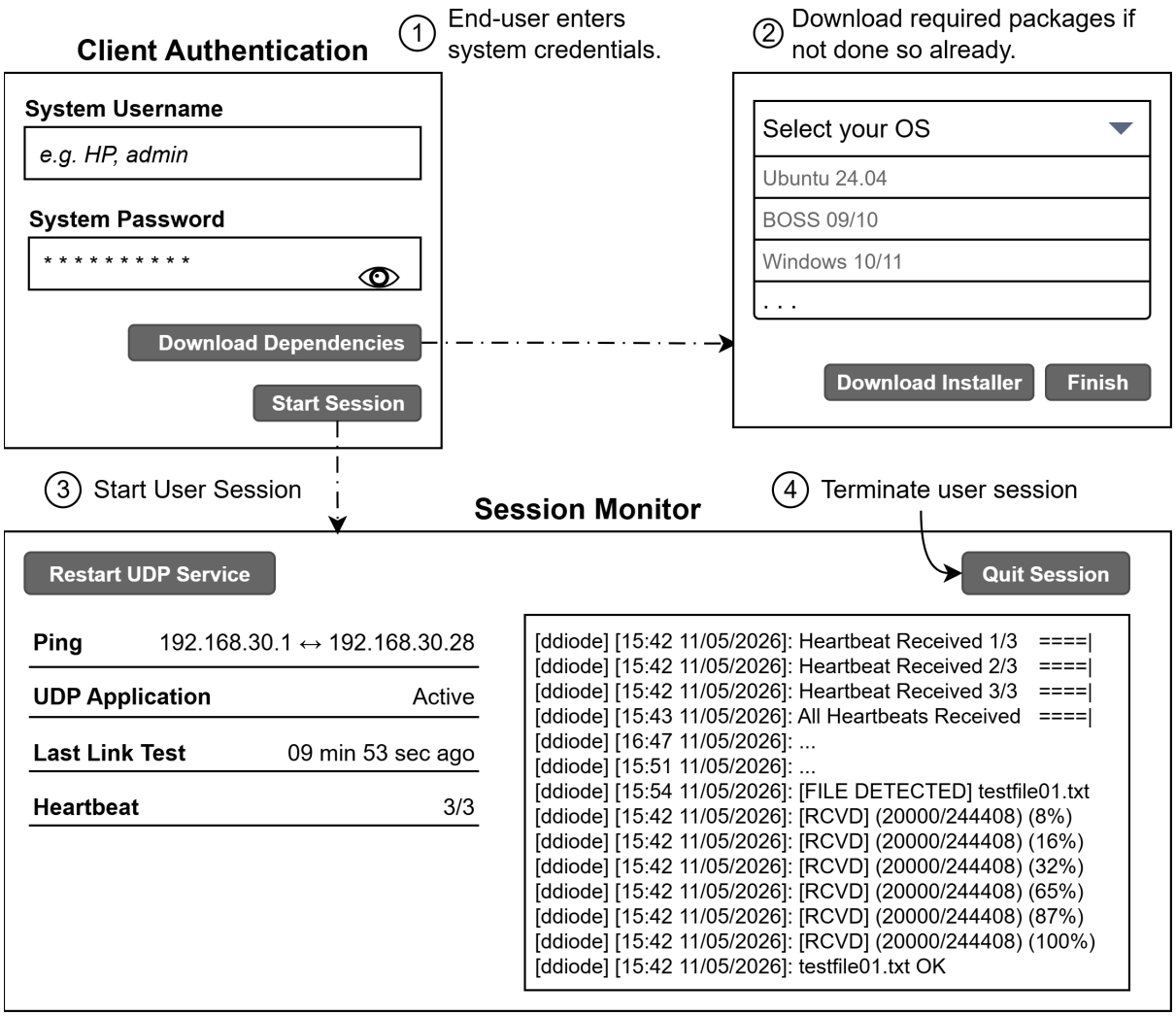



**Figure 3.5:** Client-end interaction workflow comprising system authentication, the offline dependency provisioning mechanism, and real-time data transfer monitoring.

To support services such as NFS and SSH, the client endpoint must have the required software dependencies installed. For example, Ubuntu systems would require the `nfs-common` and openSSH utilities. Considering that the proposed work assumes an air-gapped environment, the data diode provisions these packages through the web-based GUI. During initial setup, the client system specifies its operating system, following which the controller node provides a corresponding offline dependency installer for download. The client may execute this installer locally without requiring internet access. Figure 3.5 depicts the overall client user-interaction.

The data diode maintains persistent user sessions to reduce operational disruption during system reboots. Relevant session information is retained by the controller node, allowing the authenticated client system to automatically re-establish its session after reboot. Additionally, the filesystem access configuration is preserved for the duration of the active session and remains restricted exclusively to the authenticated client endpoint.

## 3.3 Module Specifications

The proposed solution is segmented into distinct functional modules to simplify the development, testing, and maintenance processes of the system. Table 3.2 encapsulates the principal modules implemented in the proposed system and their corresponding operational responsibilities.

**Table 3.2:** Module Specifications of the proposed Data Diode

| Module | Function | Key Operations |
|---|---|---|
| Hardware Isolation Layer | Unidirectional Communication | Strictly unidirectional data flow achieved via specialised hardware |
| UDP Transfer Application | Unidirectional File Transfer | Data streaming, packet encapsulation and reordering, throughput optimisations |
| Web-based GUI Application | Client-end Interaction | Shared directory access, session management, DHCP assignment |

## 3.4 Tools Used

**Table 3.3:** Detailed Specifications of Data Diode Components

| Tool | Purpose |
|---|---|
| 1-Gbit Media Converter | Hardware-enforced unidirectional stream via optical isolation |
| Raspberry Pi 5 | Controller Node for Data Diode |
| Raspberry Pi OS (64-bit) | Operation environment for present implementation |
| Python 3 | Scripting language for Service Development |
| Flask (python) | Framework for web-based GUI |
| OpenSSH | Remote Shell Access for client-side configuration |
| NFS Protocol | Local filesystem share protocol for Linux-based clients |
| Samba (SMB) | Local filesystem share protocol for Windows OS clients |
| Debian (arm64) | Dependency packaging logic for Raspberry Pi 5 deployment |

The tools specified in Table 3.3 support a streamlined operational workflow. Linux-based systems provide robust abstractions for low-level kernel and networking operations without introducing significant implementation complexity. Additionally, Python is a flexible and accessible language suitable for both web-based GUI development and UDP socket programming, simplifying the development and maintenance of the project.

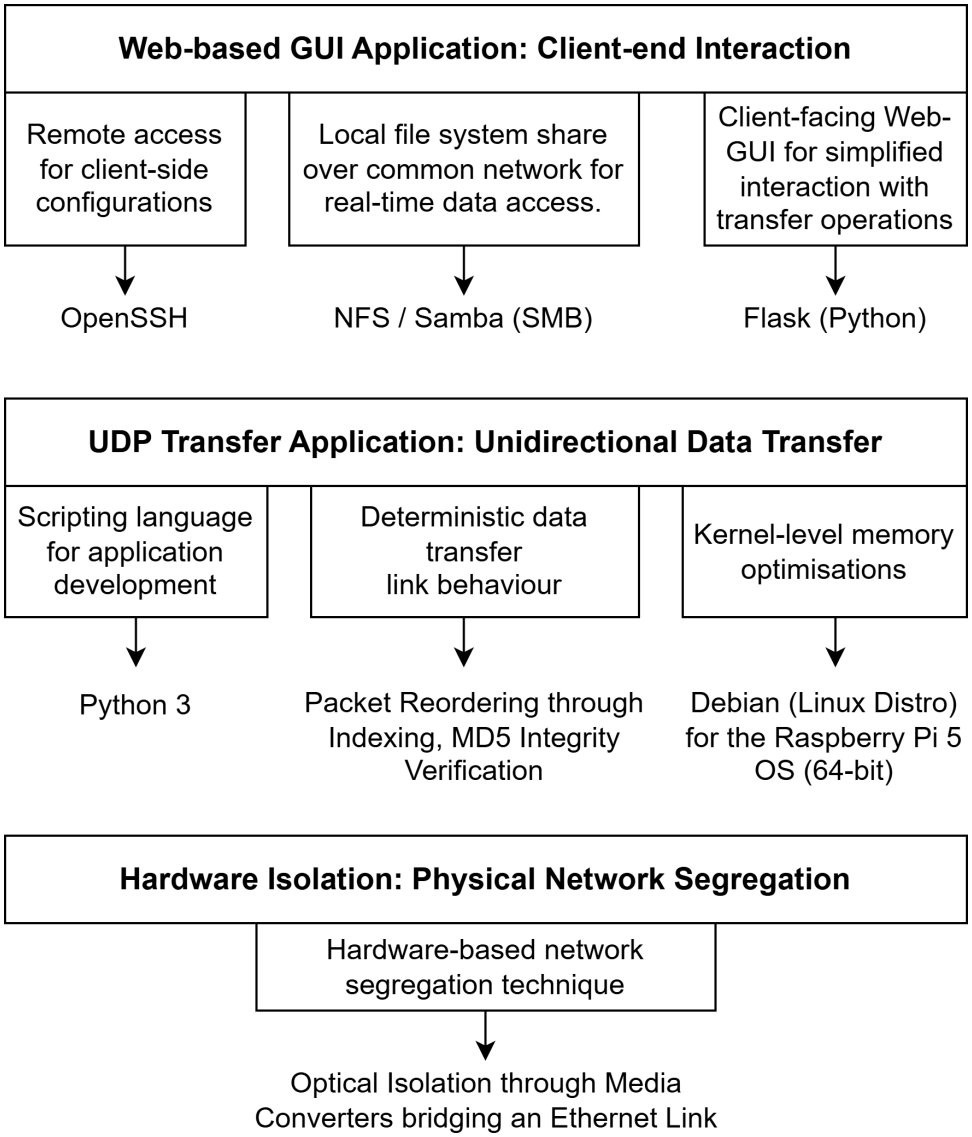



**Figure 3.6:** Top-down Abstraction of Module Specifications and Relevant Tools Used. The depicted modules represent decoupled components of the data diode which may be implemented through any capable software / hardware stack.

## 3.5 Conclusions

The presented methodology showcases a practical architecture for securing unidirectional communication within air-gapped networks. The work combines hardware-enforced optical isolation with a software-defined UDP transfer mechanism capable of optimising throughput. Rather than relying solely on logical isolation guarantees, the design methodology approaches this challenge as a combination of hardware-enforced restrictions and the system-level customisations necessary for practical deployment within operational environments.

# CHAPTER 4

# RESULT ANALYSIS

This chapter evaluates the operational capabilities of the proposed system under practical deployment conditions and testing procedures. The discussion analyses transfer throughput, unidirectional channel reliability, and client-end usability of the presented data diode architecture. These evaluations validate the extent to which the proposed work satisfies the target specifications defined in Table 1.1. It is important to note that the presented results are interpreted within the design assumptions and system limitations summarised in Table 3.1.

## 4.1 Throughput Analysis

Throughput was evaluated to determine the practicality of the proposed unidirectional transfer mechanism for sustained file-transfer operations. Since communication is physically restricted to a link rate of $1\,\mathrm{Gbit\,s^{-1}}$, the presented analysis compares the achieved application-layer throughput (goodput) against the hardware-imposed upper bound. While Appendix A.1 derives a theoretical lower bound for the inter-packet gap, the effective goodput was measured experimentally through repeated transfer tests. For consistency, system goodput was calculated on a per-file basis as the ratio of file size (in bits) to the total transfer duration (in seconds). The transfer duration was measured at the receiver controller node as the time difference between receiving the META packet and the last DATA packet.

$$\text{Goodput (bit/s)} = \frac{\text{Size of File Transferred (in bits)}}{\text{Total Transfer Duration (in seconds)}} \tag{4.1}$$

A set of test files ranging in size from $100\,\mathrm{MiB}$ to $8\,\mathrm{GiB}$ was maintained for repeated throughput evaluation. Each file was transferred multiple times to verify the consistency of the achieved transfer rate.

All tests were conducted with the inter-packet gap configured to $40\,\mu\mathrm{s}$. Prior to implementing the receiver-side memory optimisations described in Section 3.2.2, reducing the inter-packet gap to this value resulted in frequent packet drops. This behaviour was later attributed to memory limitations and receiver-side buffering constraints of the Raspberry Pi 5. Following the revised configuration, the measured application-layer goodput stabilised at approximately $957\,\mathrm{Mbit\,s^{-1}}$ during sustained transfer operations. Notably, the UDP transfer application depends almost entirely on the available system RAM and onboard storage. If either of these resources gets exhausted mid-operation, throughput can degrade significantly and, in severe cases, the transfer process can terminate unexpectedly. These observations are detailed in Appendix A.4.

## 4.2 Link Reliability

The presented evaluation focuses on packet loss and the operational stability of the UDP unidirectional communication channel. In the initial prototype, packet losses frequently occurred when the inter-packet gap was reduced. Link reliability was improved by increasing the kernel-level socket buffer size at the receiver controller node. Following this, throughput optimisation no longer posed any proportional risk of increasing packet losses. However, packet drops were still observed when testing large-file transfers (exceeding $6\,\mathrm{GiB}$). The root cause of this was identified as Linux's default feature of aggregating disc writes. Appendix A.4 describes the memory-management approach adopted to mitigate this problem. After all the detailed adjustments, no packet drops were observed throughout repeated file-transfer testing.

**Table 4.1:** Iterations of System Configuration to Improve Link Reliability

| System Configuration | Outcome |
|---|---|
| Initial Prototype Build | Packet drops occur when reducing the inter-packet gap below $100\,\mu\mathrm{s}$. |
| Increased Reciever Socket Buffer | Packet drops occur on sending large-sized files, or batch transfers exceeding $6\,\mathrm{GiB}$. |
| Improved Memory Management | No packet drops observed during repeated testing. |

## 4.3 Project Usability and Deployment

To evaluate the plug-and-play capability of the proposed system, deployment testing was conducted progressively throughout development. During initial testing, the data diode was deployed between two virtual machines that had default OS configurations. Test users interacted with each endpoint as independent clients; all deployment errors and issues with the end-user interaction were addressed in subsequent builds. This iterative process helped organise the requirements relevant to usability and deployment simplicity. As discussed in Section 3.2.3, offline client-end installers were developed to provide any missing dependencies for air-gapped client systems. These installers were also tested on default virtual machines for the supported operating systems.

Later software builds of the data diode were deployed between operational data streams within the organisation to evaluate the system under practical conditions. This stage of testing exposed issues pertaining to packet loss and deployment integration. The final firmware release was observed to provide a simplified deployment process with minimal end-user interaction. The system was successfully integrated into existing data workflows. Additionally, the set-up procedure was also simplified by creating Debian installer packages for the Raspberry Pi 5 controller nodes. These packages bundle the software source code and automatically set up the required system and network configurations for the controller nodes' operations.

## 4.4 Significance of Result Analysis

The system evaluation was conducted primarily to evaluate the capability of the proposed data diode design to support practical data transfers in air-gapped environments. The target specifications defined in Table 1.1 encapsulate the operational behaviour necessary for sustained practical use of the data diode. Table 4.2 relates the outcomes of the result analysis to the target specifications.

**Table 4.2:** Validation of Target Specifications (TS) through Result Analysis

| **Specification** | **Observed Outcome** |
|---|---|
| Unidirectional Communication | Hardware-enforced one-way transfer successfully achieved through optical isolation. |
| Air-gapped Operation | System operates independent of internet connectivity and external services. |
| Lightweight Deployment | $25\,\mathrm{MiB}$ Debian packages for initial configuration of low-cost Raspberry Pi 5 controller nodes. |
| Plug-and-Play Design | Client-interactions minimised and configuration procedures are automated. |
| Throughput | Application-layer throughput stabilised at approximately $957\,\mathrm{Mbit\,s^{-1}}$. |

It is important to note that the stabilised throughput and deterministic link behaviour were achieved under specific system limitations. The proposed architecture is primarily restricted by the installed RAM and onboard data storage. Moreover, the unidirectional channel itself is limited to a link rate of $1\,\mathrm{Gbit\,s^{-1}}$. While these constraints may be alleviated by implementing better hardware, the deployment testing process described in Section 4.3 verified that the present architecture can reliably support real-world data flows.

## 4.5 Conclusions

The presented evaluation demonstrated that the proposed data diode architecture can uphold practical high-throughput data transfers while preserving hardware-enforced unidirectional communication. Through memory optimisations and a distinct application-level UDP transfer implementation, the system achieved a stabilised goodput approaching $1\,\mathrm{Gbit\,s^{-1}}$ without requiring data redundancy or error correction mechanisms. The deployment testing process showcased that the data diode can integrate into existing client workflows with minimal end-user interaction. Though the system is constrained by the limitations of unidirectional communication, the presented work confirms that a production-ready data diode system can be implemented using cost-effective hardware and open-source software tools.

# CHAPTER 5

# CONCLUSION AND FUTURE SCOPE

## 5.1 Summary

This report presented the design and implementation of a hardware-enforced data diode intended to support practical unidirectional data transfer within air-gapped environments. The design methodology enforces network segregation through hardware isolation and incorporates controlled information flow, keeping in mind link reliability, deployment capability, and overall system usability. Existing works regarding air-gapped systems were reviewed to highlight a critical research gap in production-ready data diode implementations.

The implemented architecture utilised optical isolation to physically enforce one-way communication over an Ethernet link. A custom UDP socket transfer application was developed to support packet reordering and integrity verification, enabling deterministic file transfer link behaviour without depending on bidirectional communication mechanisms. The system incorporated a client-facing web interface coupled with an automated configuration service and offline dependency provisioning to simplify integration steps and minimise client-side efforts.

Evaluation through deployment testing verified that the proposed architecture could sustain practical application-layer throughput upwards of $900\,\mathrm{Mbit\,s^{-1}}$ while maintaining file integrity over transmissions. Further testing showed that the system could integrate into existing workflows with minimal end-user interaction.

## 5.2 Conclusions

In practical operational scenarios, a reliable high-throughput communication mechanism for a hardware-enforced unidirectional channel can be achieved through application-layer communication design, memory management, and a simplified deployment architecture. Existing works attribute limited data transfer capability to the inherent restrictions posed by a unidirectional channel. However, the presented work showcases that transfer instability originates from memory and storage limitations. The file transfer process can be made deterministic without relying on forward error correction methods or redundant retransmissions, by reducing receiver-side bottlenecks. Deployment testing highlighted that usability and integration processes are critical to improving the adoption of such a project in practical operational environments. Existing data diode implementations prioritise hardware enforcement while overlooking the inherently complex deployment procedures. The presented system addresses this limitation by incorporating a system architecture that simplifies deployment and minimises end-user configuration efforts, enabling the system to operate as a plug-and-play solution.

## 5.3 Future Scope

The presented framework's operational capability is bounded by the implemented hardware. The achievable throughput is limited by the physical Ethernet link rate of $1\,\mathrm{Gbit\,s^{-1}}$. Additionally, the file transfer operations are constrained by the installed RAM and onboard data storage size of the controller nodes. Consequently, the UDP data transfer is not showcased as globally deterministic but is rather validated to be deterministic under this specific deployment scenario through rigorous deployment testing.

The presented work implements a one-to-one transfer model between singular data sources from trusted and untrusted domains. Future implementations should extend this design to support a variety of network topologies. A many-to-one model is essential for handling data workflows in larger organisations.

Future research may focus on developing an OS-independent solution for client-end interactions. Similarly, platform-agnostic firmware may be developed for the data diode's controller nodes. Though the presented architecture is modularised and can be implemented through any capable toolset, the design methodology incorporates functionalities distinct to Linux-based operating environments.

# REFERENCES


[1] A. Hern. (2016, October) Major websites including twitter and reddit hit by huge internet outage. The Guardian. Accessed: 2026-05-16. [Online]. Available: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet

[2] C. E. Team. (2025, November) November 18, 2025 outage. Cloudflare Blog. Accessed: 2026-05-16. [Online]. Available: https://blog.cloudflare.com/18-november-2025-outage/

[3] M. R. Na and K. Sundharakumar, "A study on air-gap networks," in *2024 5th International Conference on Innovative Trends in Information Technology (ICITIIT)*. IEEE, 2024, pp. 1–6.

[4] N. Mohamed, S. K. Almazrouei, A. Oubelaid, M. Elsisi, B. M. ElHalawany, and S. S. Ghoneim, "Air-gapped networks: Exfiltration without privilege escalation for military and police units," *Wireless Communications and Mobile Computing*, vol. 2022, no. 1, p. 4697494, 2022.

[5] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[6] E. Cecchetti, A. C. Myers, and O. Arden, "Nonmalleable information flow control," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1875–1891.

[7] M. Guri, "Lantenna: Exfiltrating data from air-gapped networks via ethernet cables emission," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2021, pp. 745–754.

[8] A. F. Krause and K. Essig, "Protecting privacy using low-cost data diodes and strong cryptography," in *Science and Information Conference*. Springer, 2022, pp. 776–788.

[9] P. Story, "Building an affordable data diode to protect journalists," in *Proc. Workshop Privacy Eng. Pract.(PEP), USENIX Symp. Usable Privacy Secur.(SOUPS)*, 2023, pp. 1–8.

[10] Klockcykel, "godiode," https://github.com/klockcykel/godiode, accessed 2026-02-07.

[11] L. Gaina, C. S. Stangaciu, D. Stanescu, B. Gusita, and M. V. Micea, "Unidirectional communications in secure iot systems—a survey," *Sensors*, vol. 24, no. 23, p. 7528, 2024.

[12] R. Vrolijk, "Osdd," https://github.com/Vrolijk/OSDD/, accessed 2026-02-07.