ELE3001 Interim Report

Michael Allen
40266651

Applicability of Digital Security by Design to Performance-Sensitive Networking Applications

16 January 2023

1 Specification

Electrical & Electronic Engineering, Software & Electronic Systems Engineering Final Year Projects 2022-2023

Applicability of DSbD to Performance-Sensitive Networking Applications

Supervisor: Sandra Scott-Hayward

	Control	x	Embedded Systems		High Frequency Electronics		Microelectronics
	Electric Power	x	Software		Connected Health		MEMS
x	Cyber-Security	x	Wireless Communications		Signal/Image Processing		Intelligent Systems
	Digital Design		Sensor Networks	x	Data Analytics		Electronics

Table 1: The topics covered by this project.

The area of investigation is the application of DSbD technologies to performance sensitive distribution of data from a broker to untrusted consumer plugins. A concrete example of this is in a networking application or appliance (such as a firewall or network monitor) which receives packets from an incoming network device, classifies the packets and distributes them to one or more '3rd-party' untrusted plugins for consumption.

Such a program structure entails an undesirable trade-off between performance (co-located plugins within the main address space rather than separate plugin processes with consequent IPC overhead) and security (isolated address spaces for untrusted code vs their inclusion in the address space of the main process, which may allow them unauthorised access to additional data).

We would like to demonstrate that DSbD can permit elimination of this trade-off and enable an architecture which is both secure and performant.

1.1 Objectives

1. Familiarisation with Morello, CheriBSD, and CHERI concepts.
2. Successfully boot, install, and configure CheriBSD on the supplied Morello board.
3. Port a packet processing library to run with CHERI capabilities enabled on the Morello board.
4. Development of a packet processing application for CheriBSD and the Morello board, which can use both CHERI-enabled in-process plugins ("DSbD design") and a traditional multi-process structure.
5. Add protection to DSbD design (bounds checking and appropriate permissions) and demonstrate that it works (secure).
6. Define simple packet streams (including a range of packet sizes) and transmit those streams to both versions of the packet processing application.
7. Measure and analyse key performance characteristics including packet processing latency and CPU utilisation for both versions and write these up in a final report.

1.2 Learning Outcomes

Upon completion of the project you will expect to have:

1. Hands-on knowledge of CHERI including CheriBSD.
2. Enhanced C and systems/embedded programming capability.
3. Ability to identify, measure, record and interpret key performance metrics.

2 Declaration of Originality

By submitting this report electronically or physically to the School of EEECS for assessment I declare that:

- I have read the University regulations on plagiarism, and that the attached submission is my own original work except where clearly identified.
- I have acknowledged all written and electronic sources used.
- I agree that the School may scan the work with plagiarism detection software.

I declare that this report is my original work except where stated.

Signed: Michael Allen

Date: 16/01/2023

Contents

3 Introduction

3.1 Background

Digital Security by Design (DSbD) is a UK government-backed initiative, which reimagines digital computing in a more secure way. It allows digital computing to have built-in limitations to what data can be accessed and minimises the effect of vulnerabilities which are found [1].

The fundamental technology driving the DSbD initiative is called CHERI (Capability Hardware Enhanced RISC Instructions), developed by the University of Cambridge and SRI International. This technology replaces pointers with capabilities, which have address boundaries and permissions. This means that programming languages which lack intrinsic memory safety features, such as C and C++, can be made more secure by design, and eliminate the need for performance heavy software runtime checks [2]. CHERI also allows for compartments to be created within a process, which have limited access compared to what is available in the full process. This means that if a security breach is found within an application, then only the compartment where the breach is found can be exploited and all other subsystems will remain unaffected [3].

Morello is a research programme directed by ARM, which aims to redesign the way that computer processors are built to achieve higher security. This has resulted in the creation of a prototype development board that uses CHERI concepts, called the Morello board [2] [4].

CheriBSD is an operating system (OS) which extends FreeBSD. It is Unix-like and it enables CHERI capabilities, including memory protection and software compartmentalisation [5].

The CHERI LLVM-project is a fork of the LLVM toolchain, extended to include compatibility with CHERI capabilities. This toolchain includes the CHERI Clang compiler, C/C++ libraries, and many other features [6]. The CHERI Clang compiler allows for C/C++ applications to be compiled in either hybrid or pure capability (PureCap) mode. Hybrid applications will use machine-word pointers by default with an option to upgrade them to CHERI capabilities. However, PureCap applications exclusively use CHERI capabilities [7].

The CHERI GDB debugger is a fork of the GNU GDB debugger, extended to allow for CHERI capabilities [8]. This can be used to debug compiled CHERI C/C++ applications.

Data Plane Development Kit (DPDK) is an open source packet processing framework, originally written by Intel in 2010, with the goal of allowing high speed packet processing. It is written in C and is compatible with a wide variety of platforms [9]. However, it does not currently support the ARM Morello architecture.

Pytilia is a Belfast-based software solutions company, established in 2020 [10]. They have previously completed work to port DPDK to the Morello fixed virtual platform (FVP) [11], which resulted in a fork of `DPDK 20.11.7` [12]. This project is being conducted with Pytilia, in collaboration with Queen's University Belfast.

3.2 Aim

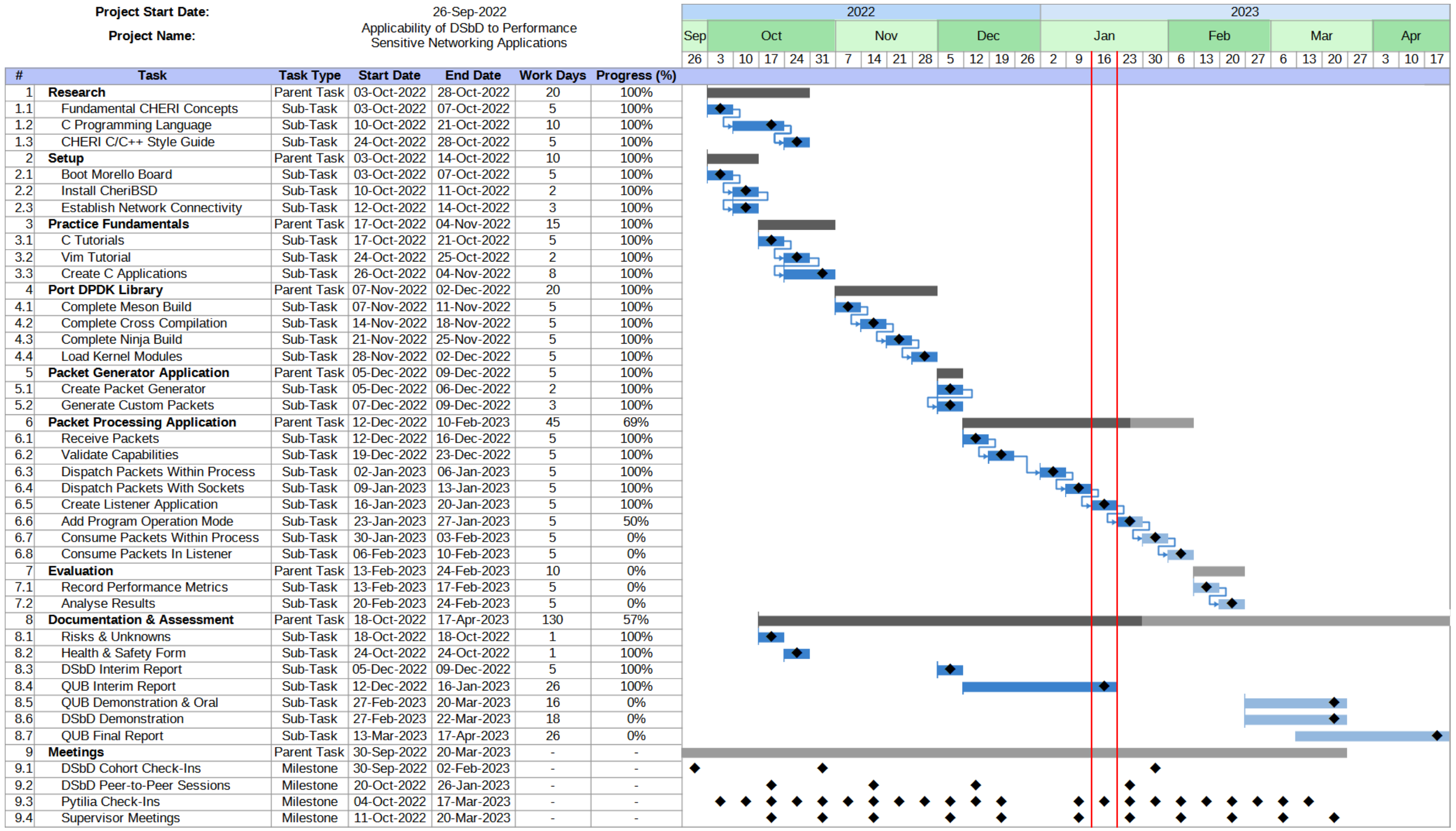
This project aims to investigate the application of DSbD initiatives for networking applications.

To enable this investigation, CheriBSD must be set up on a Morello board. A C/C++ packet processing library, such as DPDK, must be ported to the Morello board with CHERI capabilities enabled. This build could be done natively or cross-compiled for use on the Morello board.

Once setup is complete, a packet processing application should be created using the ported library. This application should be able to receive and classify incoming packets before dispatching them to consumers. The application should allow for CHERI-enabled plugins to be run in the same process as the application. This should be demonstrably secure, with capability bounds and permission enforcement checks. This application should also allow for a traditional inter-process communications (IPC) method to be used with the help of sockets to achieve the same goal.

Packet streams with a range of packet sizes should be created and used with both versions of the application. This will enable performance metrics to be measured and recorded, including packet processing latency and CPU utilisation. These metrics should be analysed and compared to evaluate the effectiveness of DSbD initiatives in simple networking applications.

4 Project Plan



Project Start Date:	26-Sep-2022
Project Name:	Applicability of DSbD to Performance Sensitive Networking Applications

#	Task	Task Type	Start Date	End Date	Work Days	Progress (%)
1	**Research**	Parent Task	03-Oct-2022	28-Oct-2022	20	100%
1.1	Fundamental CHERI Concepts	Sub-Task	03-Oct-2022	07-Oct-2022	5	100%
1.2	C Programming Language	Sub-Task	10-Oct-2022	21-Oct-2022	10	100%
1.3	CHERI C/C++ Style Guide	Sub-Task	24-Oct-2022	28-Oct-2022	5	100%
2	**Setup**	Parent Task	03-Oct-2022	14-Oct-2022	10	100%
2.1	Boot Morello Board	Sub-Task	03-Oct-2022	07-Oct-2022	5	100%
2.2	Install CheriBSD	Sub-Task	10-Oct-2022	11-Oct-2022	2	100%
2.3	Establish Network Connectivity	Sub-Task	12-Oct-2022	14-Oct-2022	3	100%
3	**Practice Fundamentals**	Parent Task	17-Oct-2022	04-Nov-2022	15	100%
3.1	C Tutorials	Sub-Task	17-Oct-2022	21-Oct-2022	5	100%
3.2	Vim Tutorial	Sub-Task	24-Oct-2022	25-Oct-2022	2	100%
3.3	Create C Applications	Sub-Task	26-Oct-2022	04-Nov-2022	8	100%
4	**Port DPDK Library**	Parent Task	07-Nov-2022	02-Dec-2022	20	100%
4.1	Complete Meson Build	Sub-Task	07-Nov-2022	11-Nov-2022	5	100%
4.2	Complete Cross Compilation	Sub-Task	14-Nov-2022	18-Nov-2022	5	100%
4.3	Complete Ninja Build	Sub-Task	21-Nov-2022	25-Nov-2022	5	100%
4.4	Load Kernel Modules	Sub-Task	28-Nov-2022	02-Dec-2022	5	100%
5	**Packet Generator Application**	Parent Task	05-Dec-2022	09-Dec-2022	5	100%
5.1	Create Packet Generator	Sub-Task	05-Dec-2022	06-Dec-2022	2	100%
5.2	Generate Custom Packets	Sub-Task	07-Dec-2022	09-Dec-2022	3	100%
6	**Packet Processing Application**	Parent Task	12-Dec-2022	10-Feb-2023	45	69%
6.1	Receive Packets	Sub-Task	12-Dec-2022	16-Dec-2022	5	100%
6.2	Validate Capabilities	Sub-Task	19-Dec-2022	23-Dec-2022	5	100%
6.3	Dispatch Packets Within Process	Sub-Task	02-Jan-2023	06-Jan-2023	5	100%
6.4	Dispatch Packets With Sockets	Sub-Task	09-Jan-2023	13-Jan-2023	5	100%
6.5	Create Listener Application	Sub-Task	16-Jan-2023	20-Jan-2023	5	100%
6.6	Add Program Operation Mode	Sub-Task	23-Jan-2023	27-Jan-2023	5	50%
6.7	Consume Packets Within Process	Sub-Task	30-Jan-2023	03-Feb-2023	5	0%
6.8	Consume Packets In Listener	Sub-Task	06-Feb-2023	10-Feb-2023	5	0%
7	**Evaluation**	Parent Task	13-Feb-2023	24-Feb-2023	10	0%
7.1	Record Performance Metrics	Sub-Task	13-Feb-2023	17-Feb-2023	5	0%
7.2	Analyse Results	Sub-Task	20-Feb-2023	24-Feb-2023	5	0%
8	**Documentation & Assessment**	Parent Task	18-Oct-2022	17-Apr-2023	130	57%
8.1	Risks & Unknowns	Sub-Task	18-Oct-2022	18-Oct-2022	1	100%
8.2	Health & Safety Form	Sub-Task	24-Oct-2022	24-Oct-2022	1	100%
8.3	DSbD Interim Report	Sub-Task	05-Dec-2022	09-Dec-2022	5	100%
8.4	QUB Interim Report	Sub-Task	12-Dec-2022	16-Jan-2023	26	100%
8.5	QUB Demonstration & Oral	Sub-Task	27-Feb-2023	20-Mar-2023	16	0%
8.6	DSbD Demonstration	Sub-Task	27-Feb-2023	22-Mar-2023	18	0%
8.7	QUB Final Report	Sub-Task	13-Mar-2023	17-Apr-2023	26	0%
9	**Meetings**	Parent Task	30-Sep-2022	20-Mar-2023	-	-
9.1	DSbD Cohort Check-Ins	Milestone	30-Sep-2022	02-Feb-2023	-	-
9.2	DSbD Peer-to-Peer Sessions	Milestone	20-Oct-2022	26-Jan-2023	-	-
9.3	Pytilia Check-Ins	Milestone	04-Oct-2022	17-Mar-2023	-	-
9.4	Supervisor Meetings	Milestone	11-Oct-2022	20-Mar-2023	-	-

Figure 1: The Gantt chart for the project, as of Monday 16th January 2023.

Figure 1 shows the Gantt chart for the project. The dark blue colour shows that a task has been completed, whereas the light blue colour shows work still remains to be completed. Milestones are marked with a diamond. The current week is marked on the Gantt chart by the red vertical lines. This timeline shows the progress thus far is ahead of schedule by approximately one week. Furthermore, there are approximately 4 weeks of contingency time before the project demonstrations during the week commencing Monday 20th March 2023.

5 Risks and Unknowns

This section covers the risks and unknowns that could have a negative impact on this project, and how they have been or will be mitigated.

The Morello board could take a long time to be delivered. It could also be faulty or damaged upon arrival.

- The Morello board was ordered by Pytilia before the project start date. This allowed the board to arrive before any work began.
- The board that arrived at Pytilia was not faulty or damaged.

There is limited documentation available on CHERI, CheriBSD, and Morello.

- There is a Morello community forum hosted on the ARM website [13]. This forum contains many answered questions about using the Morello board and allows for new questions to be posted.
- There is a CHERI Slack workspace [14]. Many of the DSbD team are available for contact in this workspace.
- Pytilia has two employees who have firsthand experience with CHERI and CheriBSD. These employees have made themselves available for contact to help with this project if required.
- Pytilia has contact with various companies and individuals who have undertaken work relating to CHERI.

Presently, the CHERI compartmentalisation feature has incomplete implementation.

- After discussion with representatives from the DSbD team and Pytilia, it was determined that this feature was not required for use in this project.
- The project requirement to validate compartmentalisation within the packet processing application has been removed.

There is a chance for work which has been completed to get corrupted or deleted.

- An external backup has been created for all work that has been completed digitally. This backup will continue to be updated regularly.
- Version control software (VCS) has been used for all of the code which has been created or modified. The git VCS has been chosen for use in this project. GitLab has been chosen as the git hosting service for storing the repositories remotely.

The DPDK library has very limited use with CHERI and the Morello board.

- There is a DPDK port for the Morello FVP that was created by Pytilia [15]. The Morello FVP is similar to the physical Morello board, which reduced the effort required for porting the library.
- A contact was found in the CHERI Slack workspace who had attempted to port the DPDK library to the Morello board for personal use. This contact was able to provide help with the porting process for this project.

The project could run behind schedule due to unexpected difficulties, sickness, or a high university workload for other modules.

- The project timeline described in Section 4 has approximately 4 weeks of contingency time allocated. This allows for delays due to unforeseen circumstances.

6 Setup

This section covers the set up process of the Morello board, including the physical set up procedure and software configuration. This relates to task 2, ‘Setup’, shown on the project timeline in Figure 1. It also fulfils project objective 2 listed in Section 1.1.

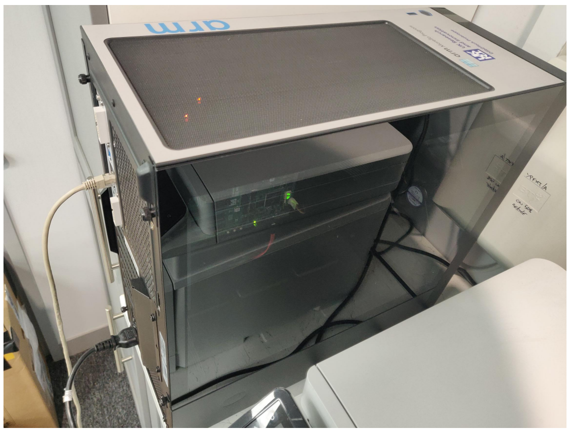


Figure 2: Side view of the Morello board.

Figure 3: Back of the Morello board.

The Morello board arrived prebuilt, so it did not need to be constructed from the individual parts. The powered on board with the cables connected can be seen in Figures 2 and 3. However, there was no external monitor, keyboard, or mouse, so these parts needed to be sourced and connected separately to enable interaction with the board.

The Morello board did not come preinstalled with any software, including the OS. Therefore, a prebuilt image of `CheriBSD 22.05p1` was downloaded from the official CheriBSD website [5]. This image was flashed onto a universal serial bus (USB) storage device for the installation process.

For the installation of the OS, a MacBook Pro was connected to the Morello board via a USB connection. The `screen` terminal emulator was used to interact with the board from the MacBook. This connection was used to configure the date and time, as well as copy the required firmware binaries onto the board. The bootable USB drive was connected to install CheriBSD.

Other tasks which were completed during this setup process include:

- Creation of a local user.
- Establishment of a network connection through the use of Ethernet.
- Creation of a secure shell (SSH) key to allow an external computer to remotely connect to the Morello board.
- Assignment of a static IP address to the Morello board.
- Installation of packages such as `Vim`, `Python`, `Meson`, and `Ninja`.

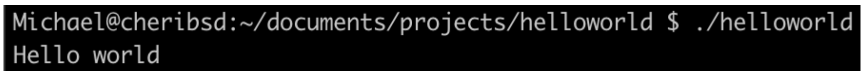
6.1 Issues

When establishing network connectivity, an Ethernet connection was set up between the Morello board and the MacBook Pro. This setup was impractical to use, so a Wi-Fi connection was configured to replace this. However, this proved to be unsuccessful. An alternative solution was found in which the Morello board was connected to a router through an Ethernet cable. An SSH key was generated to allow for remote access to the Morello board with this setup.

When setting a static IP address on the Morello board, the network connection was lost even though an SSH connection could still be established. This was fixed by removing the static IP address from the board and adding dynamic host configuration protocol (DHCP) reservation on the router. This achieved the same goal of having a consistent IP address for the Morello board after a reboot.

7 C Applications

This section covers the creation of example C applications on the Morello board. All of the applications in this section were compiled in PureCap using the CHERI Clang compiler. This covers task 3, shown on the project timeline in Figure 1.

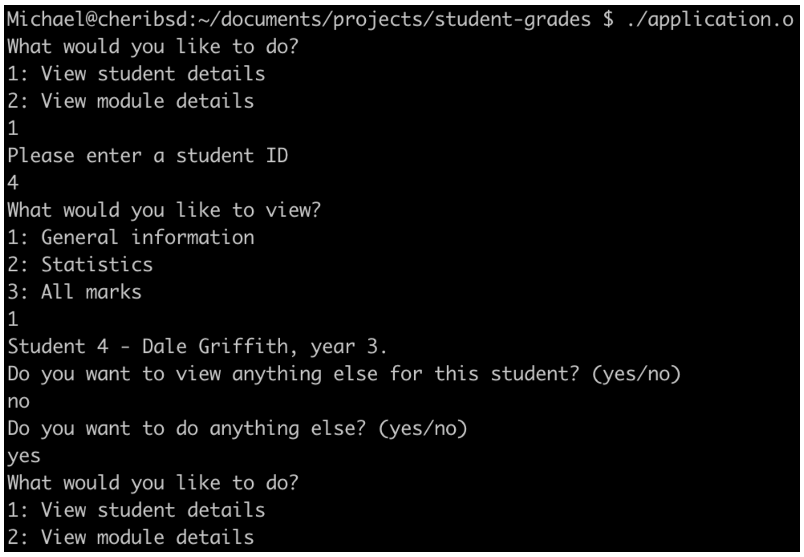


```
Michael@cheribsd:~/documents/projects/helloworld $ ./helloworld
Hello world
```

Figure 4: The C `helloworld` application.

The first program to be created was a `helloworld` program. This program output the phrase 'Hello World!' into the terminal when run, as shown in Figure 4. Completion of this program showed that the compiler was installed and configured correctly.

The next program to be created was a student information application. This program received inputs for a number of students, as well as the name, age, and grades of each student. The student information would be displayed to the terminal after inputs were completed. The student information was stored in a `struct`. The inputs for the application occurred through the terminal, which validated that input was possible on the Morello board. Furthermore, the code for the `struct` and all relevant functions was written in an external C header and source file, which showed that the linker was working correctly.



```
Michael@cheribsd:~/documents/projects/student-grades $ ./application.o
What would you like to do?
1: View student details
2: View module details
1
Please enter a student ID
4
What would you like to view?
1: General information
2: Statistics
3: All marks
1
Student 4 - Dale Griffith, year 3.
Do you want to view anything else for this student? (yes/no)
no
Do you want to do anything else? (yes/no)
yes
What would you like to do?
1: View student details
2: View module details
```

Figure 5: The final student information application.

Another student program was created with more functionality. This program read students, module information, and student grades from text files. These text files used a comma separated variable (CSV) format. The students could have different academic years, which influenced the number of modules that were expected to be input. The average, maximum, minimum, and range of grades for each student or module could be calculated on demand. These statistics, generic student information, or module details could be displayed based on inputs received from the terminal. An example execution of this application can be seen in Figure 5. A `Makefile` was also created for the program. This program was added to the version control software `git`, and pushed upstream to GitLab [16].

This application validated that all required C features worked correctly on the Morello board and would successfully compile in PureCap. These features include pointer functions, double pointers, and linked lists. Moreover, this program showed that text files were a viable source of input on the Morello board.

7.1 Issues

Some C functions were unable to be used in PureCap the same way as in normal C applications. For example, the `memcpy` function would sometimes cause buffer overrun errors and had to be replaced with the `strcpy` function. This meant that these error-prone C functions should not be used for the developmental work of this project.

8 DPDK Port

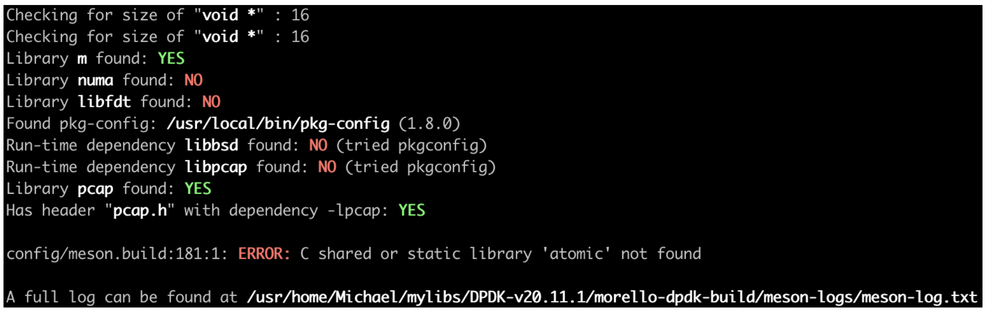
This section covers the porting process of the DPDK library to the Morello board. The Pytilia fork of `DPDK 20.11.7` [12] was used as the starting point for this process. This relates to task 4, “Port DPDK Library”, shown on the project timeline in Figure 1, and fulfils project objective 3 listed in Section 1.1.

The build process for DPDK involves two stages, which are as follows:

- The `Meson` build - The `meson setup build` command generates a build directory, containing the necessary files for the next step.
- The `Ninja` build - The `ninja` command is run from inside the directory generated by the previous command. This compiles all of the C code, including libraries, drivers, and applications.

8.1 Attempt 1 - Native Compilation

This section covers the first attempt of compiling DPDK, which was a native compilation.



```
Checking for size of "void *" : 16
Checking for size of "void *" : 16
Library m found: YES
Library numa found: NO
Library libfdt found: NO
Found pkg-config: /usr/local/bin/pkg-config (1.8.0)
Run-time dependency libbsd found: NO (tried pkgconfig)
Run-time dependency libpcap found: NO (tried pkgconfig)
Library pcap found: YES
Has header "pcap.h" with dependency -lpcap: YES

config/meson.build:181:1: ERROR: C shared or static library 'atomic' not found

A full log can be found at /usr/home/Michael/mylibs/DPDK-v20.11.1/morello-dpdk-build/meson-logs/meson-log.txt
```

Figure 6: The `atomic` dependency error during the `Meson` build.

One error occurred during the `Meson` build, which was due to a missing dependency of the `atomic` library, as shown in Figure 6. This was fixed by editing one of the `meson.build` configuration files, which checked if the size of the `void *` data type was 4 or 8 bytes long. This data type has a size of 16 bytes on the Morello board. This is because it is a CHERI capability rather than a pointer, which it would be on all other machine architectures. Capabilities require 16 bytes for 64-bit machines in order to store the associated permissions and bounds alongside the memory address. The configuration files revealed that only 32-bit machines require the `atomic` library for DPDK to build, so the 64-bit Morello board did not need this dependency. Editing the configuration file such that DPDK recognised the correct system architecture allowed the `Meson` build to complete.

The `Ninja` build for DPDK raised multiple errors, including one which related to emulated thread local storage (TLS). This error was resolved by removing the compiler flag for using emulated TLS, `-femulated-tls`, so that the system would use regular TLS instead. This flag was required for the Morello FVP because it is virtual. However, the Morello board does not support emulated TLS and can only use regular TLS.

The next error raised in the `Ninja` build process related to atomic operations. This error occurred because some atomic operations, such as compare and swap, were undefined. These references should have been unnecessary based on the knowledge gained from the `Meson` build step, so they were removed from the code. The compiler flag `-mno-outline-atomics` was also added to ensure that no atomic operations would have been used, in addition to the flags `-march=morello+c64` and `-mabi=purecap`, which specify the use of a Morello board with PureCap compilation. However, this proved to be unsuccessful because many network drivers that were included in the compilation required atomic operations. Removing these operations resulted in more errors which were too difficult to resolve.

8.1.1 Issues

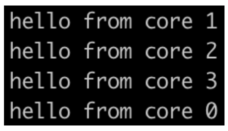
An attempt to resolve the atomic operations error in the `Ninja` build step involved adding a `native_file.txt` text file which was used with the `Meson` build flag `--native-file=native_file.txt`. This file took precedence over system defaults which were detected, and included compiler flags, machine architecture, and paths to binary files. However, this did not fix the build error and was abandoned.

8.2 Attempt 2 - Cross-Compilation

This section covers the second attempt of compiling DPDK, which was a cross-compilation. A MacBook Pro was used for the cross-compilation process.

Due to the unsuccessful compilation attempt in Section 8.1, an attempt was made to cross-compile DPDK for the Morello board. This was the method used by Pytilia when porting DPDK to the Morello FVP. This cross-compilation involved extra steps compared to the native compilation process, including:

- Building the Morello FVP using the `cheribuild` tool.
- Cloning and running a PureCap Docker container to be used as the build environment.
- Replacing relative paths with absolute paths in `meson.build` configuration files.
- Creating a `cross_file.txt` text file to store configuration for the compilation, including compiler flags, machine architecture, and paths to binary files. This was accompanied by the use of a `Meson` build flag `--cross-file=cross_file.txt`.
- Editing file paths in the `build.ninja` configuration file.



```
hello from core 1
hello from core 2
hello from core 3
hello from core 0
```

Figure 7: The DPDK `helloworld` program.

Having completed these steps, the cross-compilation completing using the MacBook was successful. This completed build produced the DPDK `helloworld` application, which is provided with the DPDK library. This application ran successfully on the Morello FVP. The application uses four cores, numbered 0 through 3. Every core should output the phrase "hello from core `X`", where `X` is the core number. The successful output of this application is shown in Figure 7.

However, when the cross-compiled library was copied to the Morello board, it raised an error relating to emulated TLS. This was the same error from the native `Ninja` build step described in Section 8.1, which was resolved by removing the `-femulated-tls` compiler flag. However, removing this compiler flag for cross-compilation raised an error during the `Ninja` build, stating that TLS was unsupported. Therefore, cross-compilation was not a viable option for porting DPDK to the Morello hardware.

8.2.1 Issues

The cross-compilation was initially attempted on a MacBook Pro without using the required Docker container. This resulted in missing dependencies when attempting the build. However, a Docker container with a PureCap build environment was provided in the documentation of the DPDK port for the Morello FVP, which Pytilia had previously created [15] [17]. This container used Linux, which contained all of the required dependencies. This allowed the cross-compilation to complete.

8.3 Attempt 3 - Native Compilation

This section covers the third attempt of compiling DPDK, which was a native compilation.

The successful cross-compilation from Section 8.2 provided build logs which could be compared with the failing native build. This highlighted that the native compilation was attempting to compile and link approximately 1,600 files during the `Ninja` build, whereas the cross-compilation only used approximately 900 files. The corresponding `Meson` build logs showed that fewer drivers were built during the cross-compilation. This meant that there were unnecessary and incompatible drivers being built during the native compilation process. This knowledge allowed for successful debugging of the native compilation from Section 8.1.

The issue with disabling drivers on the Morello board occurred in the `Meson` build step. However, this did not result in an error during this step, but instead during the `Ninja` build. This was caused by a bug when parsing which drivers to disable from the `meson_options.txt` configuration file. These drivers were able to be disabled by removing them from their corresponding `meson.build` configuration files, which allowed the native compilation to complete.

Using this compilation, the DPDK `helloworld` application could be executed on the Morello board in the same way as the cross-compiled version described in Section 8.2. The completed port was added to git and uploaded to GitLab [18].

8.4 Multiple Processes

This section covers the attempt to allow multiple processes to be used by the DPDK port created in Section 8.3.

The DPDK `helloworld` application requires two command line arguments to be specified, `--no-shconf` and `--no-huge`, which only allow for one process to be used by the application. The DPDK documentation states that this is because two kernel modules called `contigmem` and `nic_uio` are required in order to use contiguous memory on the FreeBSD OS [19]. The prebuilt image of CheriBSD does not include these modules.

The `cheribuild` tool was used to create a disk image of CheriBSD with the `contigmem` and `nic_uio` kernel modules installed. The current release of `cheribuild` did not work on a MacBook Pro. However, a fork of `cheribuild` created by Pytilia did work, so it was used instead. Using this version of the tool, a new CheriBSD image was created and loaded onto the Morello board. This fixed the error relating to missing kernel modules. However, this resulted in multiple CHERI capability related errors which could not be fixed.

The final version of the DPDK port was limited to only one process. This was sufficient for the packet processing application to be developed. Therefore, none of the project objectives had to be modified.

8.4.1 Issues

To build the kernel modules on the Morello board natively, a capability permissions error was fixed in the source code. However, the Morello board could not load the kernel modules because they were compiled as PureCap ABIs. Changing the `BSDmakefile` to compile the modules in hybrid instead of PureCap had no effect.

A MacBook Pro was also used to build the required kernel modules. However, they also could not be loaded on the Morello board. This was because the kernel versions did not match.

9 Packet Generation

This section covers the generation of network packets. This covers task 5, 'Packet Generator Application', shown on the project timeline in Figure 1.

The `Python` library `python-libpcap` was used to create packet streams and store them in a packet capture (`pcap`) file. These `pcap` files provide the input traffic for the packet processing application. All of the packet streams contained raw Ethernet packets, and only the packet payloads were defined. This application was added to git and pushed upstream to GitLab [20].

Stream	Packet count	Packet length (Bytes)
1	6	75 - 131
2	10,000	75 - 131
3	20,000	75

Table 2: Details of the packet streams.

This application has produced 3 distinct packet streams, as per project objective 6, which are detailed in Table 2. Stream 1 is a very small stream with a range of packet sizes, which enables fast testing of the packet processing application. Stream 2 contains 10,000 packets of variable lengths, which is the maximum that the packet processing application can handle concurrently. Stream 3 contains 20,000 packets, which is above than the limit of 10,000 which can be handled at any one time. However, because these packets have a constant length, the full stream can be read in by the application.

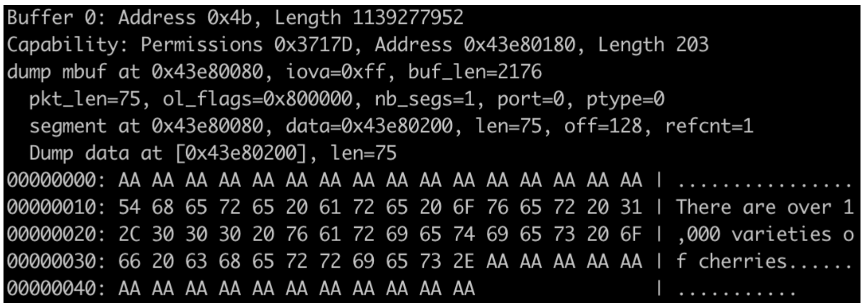
10 Current Situation

This section covers the progress relating to the current task being completed for the project, which is the development of a packet processing application which uses the DPDK framework. This relates to task 6, 'Packet Processing Application', shown on the project timeline in Figure 1. This application also fulfils project objectives 4 and 5 listed in Section 1.1.

This application is called 'CHERI Networking' and is available on GitLab [21]. It is based on the previous application developed by Pytilia, 'Limelight' [22]. The initial findings from this application are detailed in the following subsections.

10.1 Packet Interaction

This section covers the extent to which CHERI Networking can currently interact with packets.



```
Buffer 0: Address 0x4b, Length 1139277952
Capability: Permissions 0x3717D, Address 0x43e80180, Length 203
dump mbuf at 0x43e80080, iova=0xff, buf_len=2176
  pkt_len=75, ol_flags=0x800000, nb_segs=1, port=0, ptype=0
  segment at 0x43e80080, data=0x43e80200, len=75, off=128, refcnt=1
  Dump data at [0x43e80200], len=75
00000000: AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA | ................
00000010: 54 68 65 72 65 20 61 72 65 20 6F 76 65 72 20 31 | There are over 1
00000020: 2C 30 30 30 20 76 61 72 69 65 74 69 65 73 20 6F | ,000 varieties o
00000030: 66 20 63 68 65 72 72 69 65 73 2E AA AA AA AA AA | f cherries......
00000040: AA AA AA AA AA AA AA AA AA AA AA                | ...........
```

Figure 8: Packet information displayed by CHERI Networking.

CHERI Networking takes a `pcap` file for the input of incoming packets. These packets can be received by the application without error and have their details printed to the terminal. An example of the output can be seen in Figure 8.

Up to 10,000 packets can be read concurrently in separate buffers, and each buffer can be cleared when processing of a packet is finished. If all of the packets are the same length, then an infinite number of packets can be read in from the input `pcap` file. However, if the packets are of a variable size, then there is a limit of 10,000 packets which can be read reliably without error. This is because a buffer cannot store a packet which has a larger length than what it stored previously. This is because the capability bounds associated with each buffer are set to match the length of the most recent packet that is read in to that buffer, and these bounds cannot be widened to store a larger packet. Therefore, if a larger packet is received, a capability bounds error will be raised.

The application has two modes of operation, as per project objective 4. The first is single process mode, which can be selected by the command line argument `-s`, or the environment variable `PYTILIA_SINGLE_PROCESS`. The second mode uses IPC instead, which can be selected with the command line argument `-i`, or the environment variable `PYTILIA_INTER_PROCESS`. The ability to select which mode for the application to run in has been added. However, the functionality is currently limited to printing which mode was selected to the terminal.

Both methods of dispatching packets have been implemented. However, due to the missing functionality of the aforementioned setting, both methods of dispatching packets occur when the application is run.

Presently, no consumers have been created to interact with these packets upon receiving them.

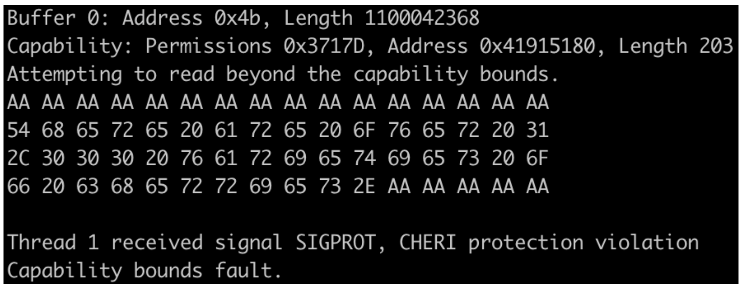
A listener application, called "Packet Receiver" [23], has been created to receive the packets dispatched using the IPC method. However, this application does not yet process the packets that it receives because no consumers have been created. Instead, the application prints the contents of the packet to the terminal to show that the packet has been received correctly.

10.2 Capability Validation

This section covers the capability validation used in the single process mode of the application.

The CHERI capabilities associated with incoming packets must be validated when using the single process setting, as per project objective 5. This ensures that the application is secure. This validation is not required for the IPC mode of operation, because it does not rely on CHERI capabilities for security.

To perform this validation, two checks must be made. The first is to ensure that memory addresses outside of the bounds cannot be accessed, such as would occur during a buffer overrun. The other check is to ensure that packets have the correct permissions enforced, for example attempting to write to a capability with read only permissions should raise an error.

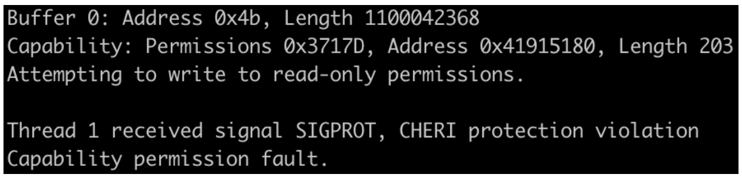


```
Buffer 0: Address 0x4b, Length 1100042368
Capability: Permissions 0x3717D, Address 0x41915180, Length 203
Attempting to read beyond the capability bounds.
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
54 68 65 72 65 20 61 72 65 20 6F 76 65 72 20 31
2C 30 30 30 20 76 61 72 69 65 74 69 65 73 20 6F
66 20 63 68 65 72 72 69 65 73 2E AA AA AA AA AA

Thread 1 received signal SIGPROT, CHERI protection violation
Capability bounds fault.
```

Figure 9: A capability bounds fault successfully being raised.

The bounds associated with the CHERI capabilities for the received packets are enforced correctly by the application. Attempting to read data outside the specified bounds can be performed by the use of a command line argument, `-x`, or an environment variable, `PYTILIA_BOUNDS_ERROR`. This raises an appropriate error, as shown in Figure 9. This check previously existed in the Limelight application and remains fundamentally the same in this application. However, the implementation has been refactored according to the differences in source code of the applications.



```
Buffer 0: Address 0x4b, Length 1100042368
Capability: Permissions 0x3717D, Address 0x41915180, Length 203
Attempting to write to read-only permissions.

Thread 1 received signal SIGPROT, CHERI protection violation
Capability permission fault.
```

Figure 10: A capability permissions fault successfully being raised.

Additionally, the permissions associated with the CHERI capabilities are enforced correctly by the application. Attempting to write when read only permissions are given can be performed by the use of a command line argument, `-y`, or an environment variable, `PYTILIA_PERMISSIONS_ERROR`. This raises an appropriate error, as shown in Figure 10. The Limelight application contains a similar validation check. However, the implementation was incompatible with this application, so it had to be created with a new method for the CHERI Networking application.

11 Future Plans

This section gives details on the remaining tasks still to be completed for the project, including both required tasks and stretch goals.

11.1 Required Tasks

The setting relating to which mode of operation is to be used in the packet processing application is not fully implemented yet. This setting needs functionality added which will allow for only the required packet dispatch method to be used, instead of both.

The application needs to classify packets and dispatch them to one of two or more consumers. The classification of packets has not yet been implemented. This may be done by arbitrary criteria for the purposes of this application, such as packet length.

The packet consumers need to be created. These consumers may be a simple counter variable, a struct, or a function which needs to be executed.

The performance of the packet processing application must be measured and analysed. This will allow for comparison between the single process mode of operation and the IPC mode of the application. Performance metrics which need to be recorded include packet processing latency and CPU utilisation, as per project objective 7. These performance metrics then need to be analysed and evaluated, to show the advantages and disadvantages of each mode of operation.

11.2 Stretch Goals

If time permits, some additional tasks which are surplus to the project specification may be completed, including:

- Allow a buffer to be reused with a larger packet than it previously contained.
- Add error messages whenever no mode or both modes of program operation are specified.
- Create more sophisticated IP/TCP/UDP packets for use in the application.
- Add more sophisticated packet consumers.
- Port `DPDK 22.11.1` to the Morello architecture, as this is the most recent long term stable release of the library.

12 References

[1] Digital Security by Design. Digital security by design, Sep 2022. URL https://www.dsbd.tech/. Accessed: 2023-01-02.

[2] Digital Security by Design. How it works - digital security by design, Sep 2022. URL https://www.dsbd.tech/how-it-works/. Accessed: 2023-01-02.

[3] Robert N. M. Watson. Department of computer science and technology: Capability hardware enhanced risc instructions (cheri), Jan 2022. URL https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/. Accessed: 2023-01-02.

[4] ARM. Morello program - arm, Jan 2023. URL https://www.arm.com/architecture/cpu/morello. Accessed: 2023-01-02.

[5] Digital Security by Design. Cheribsd, Jan 2023. URL https://www.cheribsd.org/. Accessed: 2023-01-02.

[6] University of Cambridge. Github - ctrsd-cheri/llvm-project, Jan 2023. URL https://github.com/CTSRD-CHERI/llvm-project. Accessed: 2023-01-02.

[7] Robert N. M. Watson et al. Cheri c/c++ programming guide. Technical Report 947, University of Cambridge Computer Labratory, Jun 2020. URL https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf.

[8] University of Cambridge. Github - ctrsd-cheri/gdb, Jan 2023. URL https://github.com/CTSRD-CHERI/gdb. Accessed: 2023-01-02.

[9] DPDK Project. About - dpdk, Nov 2022. URL https://www.dpdk.org/about/. Accessed: 2023-01-02.

[10] Pytilia Limited. Pytilia - software development, Sep 2022. URL https://pytilia.io/. Accessed: 2023-01-11.

[11] Digital Security by Design. Pytilia - digital security by design, Feb 2022. URL https://www.dsbd.tech/success-story/pytilia/. Accessed: 2023-01-02.

[12] pytilia/dpdk-v20.11.1, Oct 2021. URL https://github.com/pytilia/DPDK-v20.11.1. Accessed: 2023-01-11.

[13] ARM. Arm community, Jan 2023. URL https://community.arm.com/support-forums/f/morello-forum/. Accessed: 2023-01-13.

[14] University of Cambridge. Cheri, Jan 2023. URL https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-slack.html. Accessed: 2023-01-13.

[15] Eileen O'Reilly. Dpdk - limelight - confluence, Oct 2021. URL https://pytilia.atlassian.net/wiki/spaces/LIM/pages/294617089/DPDK. Accessed: 2023-01-05.

[16] Student grades - gitlab, Oct 2022. URL https://gitlab.eeecs.qub.ac.uk/40266651/student-grades. Accessed: 2023-01-11.

[17] Tim Silversides. tsilversides/morello-purecap-buildenv - docker image, May 2021. URL https://hub.docker.com/repository/docker/tsilversides/morello-purecap-buildenv. Accessed: 2023-01-05.

[18] Dpdk-v20-11-1 - gitlab, Jan 2023. URL https://gitlab.eeecs.qub.ac.uk/40266651/DPDK-v20-11-1. Accessed: 2023-01-11.

[19] DPDK. 3. compiling the dpdk target from source, Nov 2022. URL http://doc.dpdk.org/guides/freebsd_gsg/build_dpdk.html. Accessed: 2022-12-20.

[20] Packet generator - gitlab, Jan 2023. URL https://gitlab.eeecs.qub.ac.uk/40266651/packet-generator. Accessed: 2023-01-11.

[21] cheri-networking - gitlab, Jan 2023. URL https://gitlab.eeecs.qub.ac.uk/40266651/cheri_networking. Accessed: 2023-01-11.

[22] pytilia/limelight_dpdk_build, Oct 2021. URL https://github.com/pytilia/Limelight_DPDK_Build. Accessed: 2023-01-11.

[23] packet-receiver - gitlab, Jan 2023. URL https://gitlab.eeecs.qub.ac.uk/40266651/packet-receiver. Accessed: 2023-01-11.