UCLouvain

epl

École polytechnique de Louvain

Enabling MPTCP on Windows through WSL

Author: **Bilal BRAROU**
Supervisor: **Tom BARBETTE**
Reader: **Olivier BONAVENTURE, Clement DELZOTTI**
Academic year 2023–2024
Master [60] in Computer Science

Contents

List of Figures

Acknowledgements

This work would not have been possible without the help of multiple people, for that, I would like to express my gratitude to:

Tom Barbette, my supervisor, for his valuable advice and the weekly follow-up he provided.

Clement Delzotti, PhD student supervised by Tom Barbette, for his technical help and review.

Olivier Bonaventure, Professor at UCLouvain, for agreeing to read my document and be part as a member of the jury.

My *parents* for their continuous support.

Tools used

To write this thesis, the tools that I used are :

- the *Grammarly* grammar checker
- the *Quillbot* grammar checker
- *Google translate* to translate part of text written in French to English

Chapter 1

Introduction

The multipath TCP protocol is an extension of the TCP protocol, allowing the use of multiple network interfaces. Multipath TCP was officially integrated into the Linux kernel v5.6 in March 2020 [18] and on iOS/MacOS since 2013 [18]. However, Windows does not implement the MPTCP protocol, making it unusable on this platform. This presents a missed opportunity as more users have multiple internet connection sources, such as 5G and fiber optics. Additionally, given that over 70 percent of Internet users use Windows [21], the absence of MPTCP is noticeable.

Windows users could benefit from better bandwidth by using MPTCP in aggregation mode, where multiple network interfaces are used, and better resilience to network instabilities using the backup mode, where one of the network interface serve as a backup in case the other network interface fails.

The main objective of this thesis is to overcome this problem by first enabling MPTCP on Windows-Subsystem for Linux (WSL), a virtual machine running a Linux kernel specifically integrated into Windows. Next, we will use WSL to enable MPTCP on Windows.

To achieve that, we run an MPTCP proxy on WSL, which will act as an intermediary between Windows applications and remote servers. Instead of the Windows application establishing a TCP connection with the remote server, it is the proxy that makes the connection, and since it is an MPTCP proxy, it will use Multipath TCP to establish the connection.

The state of the art will outline existing methods for enabling MPTCP on operating systems that lack the MPTCP feature by default. These methods include the use of a separate virtual machine with MPTCP enabled, or the use of an aggregator box, a router with MPTCP enabled that converts TCP traffic to MPTCP traffic.

We believe it's crucial to first explain the fundamentals of multi-path TCP, including how to establish an MPTCP connection and use multiple network interfaces. We will briefly describe how WSL works, such as what the different versions of WSL are, what the advantages of each of them are, etc.

In the chapter 4, we'll explain how our application enables MPTCP on WSL. To do this, we'll describe the application's general architecture, then go into more detail about each component and justify the implementation choices we have made.

Then, before we discuss how to enable MPTCP on Windows, we explain in Chapter 5 how proxies works, how they allow us to enable MPTCP on Windows, and finally, which type proxy we used for enabling multipath TCP on Windows.

Next, in Chapter 6, we discuss the general architecture of our application to enable multipath TCP on Windows. Then, similarly to Chapter 4, we describes in detail the necessary components used to enable MPTCP on Windows, the implementation choices made, and the difficulties encountered.

Finally, in Chapter 7, we test and evaluate the performance of our application in a variety of situations, such as using MPTCP for downloading, streaming, and so on.

Chapter 2

State-of-the-art

In this chapter, we will first demonstrate the technique of using an MPTCP-enabled virtual machine to run an MPTCP proxy. Then, we will demonstrate the use of aggregation boxes offered by OpenMPRouter and Tessares. Finally, we will explain how WSL can enable multipath TCP on Windows.

2.1 VM-based Techniques for multipath TCP Activation

Utilizing proxies within virtual machines offers versatile solutions for enabling Multipath on operating systems lacking native support. These techniques leverage the flexibility of virtualized environments to introduce MPTCP capabilities without relying on external hardware such as routers.

2.1.1 Single proxy running inside a VM

One of the techniques for enabling multipath TCP on OSes that do not have native support is to run an MPTCP proxy in a virtual machine that have multipath TCP enabled. Any Windows application could then use the proxy to leverage the usage of MPTCP.

Finally, users must provide the proxy address in the chosen program to take advantage of MPTCP. This technique was proposed by maintainers of Multipath TCP in the Linux kernel to measure the ability of MPTCP to interact with existing middleboxes without doing any kernel change. [18]

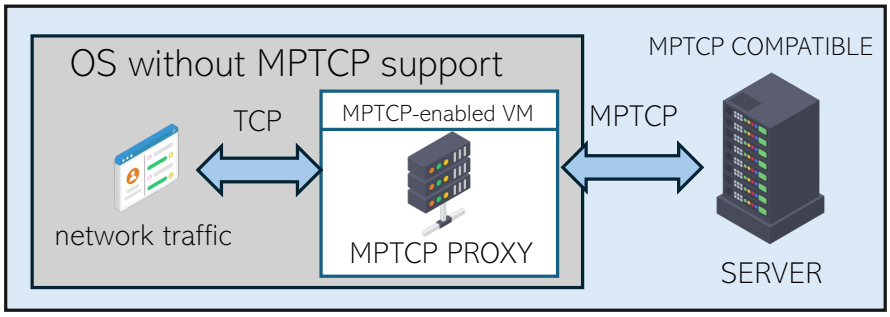

Figure 2.1: Single proxy technique using a VM

The advantage of this technique is that it does not require any external hardware, such as a custom router, because the virtual machine runs locally. This allows MPTCP to be used in any location. The disadvantage of this approach is that we can only use MPTCP on servers that have enabled it.

2.1.2 Double proxy technique

The second approach for enabling MPTCP on operating systems that do not support it natively is like the approach above. However, in this case, the proxy itself will forward the traffic to another MPTCP proxy. This allows multipath TCP to be used even if the distant server doesn't have it implemented or enabled. [13]

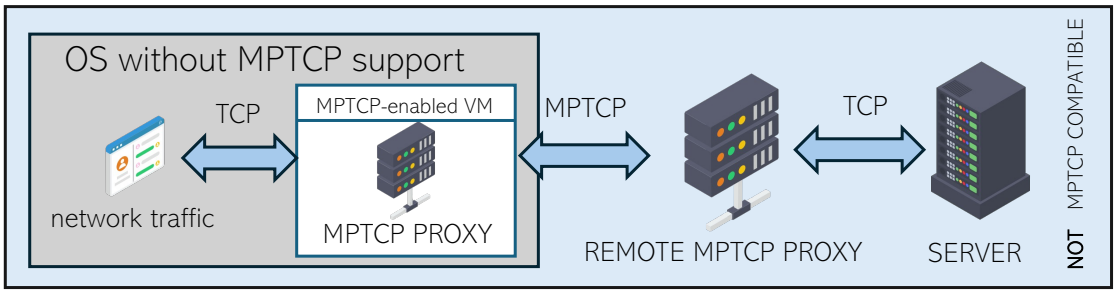

Figure 2.2: Double proxy technique using a VM

2.2 Aggregation box

The second technique is to use an aggregation box, which is simply a router with a MPTCP-enabled kernel connected to multiple network links, permitting the aggregate of multiple bandwidths. The box will convert the received TCP traffic

into multipath TCP traffic and forward it to an MPTCP proxy.

The problem with this approach is that we are dependent on hardware devices and, therefore, on a specific location and network. For example, In a train with a Windows laptop, We could not use MPTCP since we are not connected to the aggregator box.

2.2.1 OpenMPRouter

OpenMPRouter is an open-source software that is based on OpenWrt, a Linux operating system targeting embedded devices, except that the developers of OpenMPRouter modified the kernel to enable MPTCP on it [3]. The advantage of using OpenMPRouter is that it provides a user interface to setup each network link correctly; for example, deciding which network interface will act as a backup. The disadvantage of this method is that not all people have the expertise nor the necessary hardware to do such an installation.

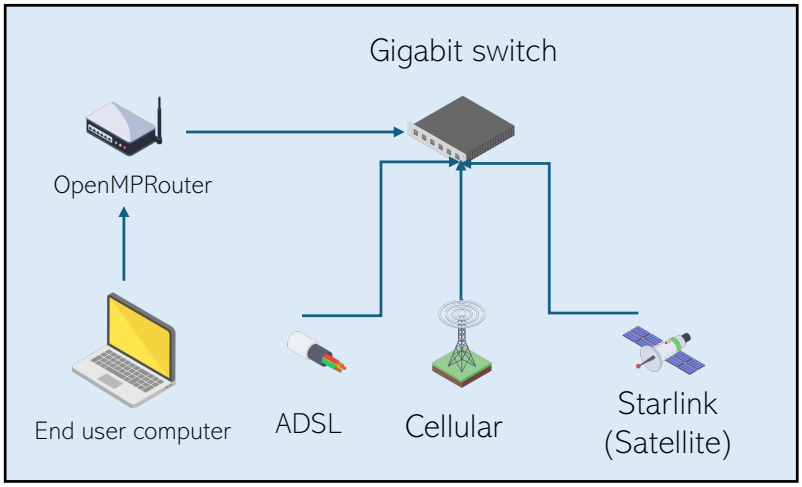

Figure 2.3: Bandwidth aggregation using OpenMPRouter

2.2.2 Tessares

Tessares is an industry leader in seamless multipath connectivity [4], providing software solutions for hybrid access and Wi-Fi Cellular Convergence. Tessares is used by telecommunications companies such as Proximus and KPN to increase

bandwidth and reliability for clients.

To allow their customers to profit from an aggregated bandwidth, Tessares provides software for Customers Premise Equipment (CPE) [5]to enable aggregation based on multipath TCP. They use a Linux kernel with MPTCP enabled and run upon it an MPTCP proxy to convert the TCP traffic into multipath TCP traffic.

Then they use a second MPTCP proxy that runs on a remote server that they call a Hybrid Access Gateway (HAG) [2] and is used to establish a MPTCP connection between the HAG himself and the client. We remind that this is done to force the use of MPTCP even if the remote server does not support it.

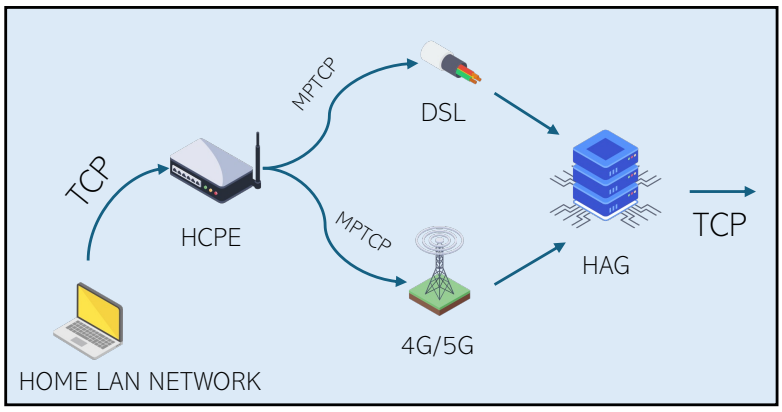

Figure 2.4: bandwidth aggregation using Tessares technology

This approach has the advantage of allowing customers to benefit from a better bandwidth without having to do any configuration beforehand. Also, customers do not have to buy any additional hardware, unlike OpenMPRouter, to benefit from improved bandwidth. Only customers of Internet service providers partnered with Tessares can access this solution, which is a disadvantage. Also, Internet Service Provider that uses Tessares technology forces their clients to use only two Internet connection points, ASDL/VSDL, and 4G/5G, even though the software proposed by Tessares can support more than 2 connection points. In contrast, OpenMPRouter allows users to add up to eight internet connection points.

2.3 Using Windows Subsystem for Linux

The approach that we used to be able to use MPTCP was to use the first technique with the virtual machine and proxy. Windows proposes to run a full Linux Kernel in a virtual machine called WSL. We then run a proxy inside the virtual machine,

and users can decide to use the proxy on the applications of their choice.

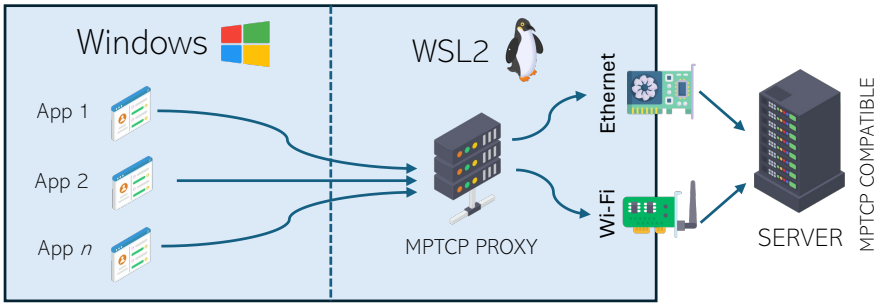

Figure 2.5: Proxy technique using WSL2

Users can also redirect all network traffic to the proxy they created. Fortunately, Windows also allows us to use a custom-built kernel, allowing us to use a compatible MPTCP kernel. Unfortunately, Windows doesn't expose every network interface in the WSL virtual machine, meaning that even if we activated MPTCP on the kernel, we wouldn't be able to use multiple paths with all of our available Windows network interfaces. The section on enabling Multipath TCP on WSL describes the way we solved this problem. The advantage of this solution is that we don't require any aggregator boxes, meaning that we could profit from an aggregated bandwidth anywhere. The only thing that we need is a server with a kernel that has MPTCP enabled on it (to use the double proxy technique).

	Maximum connections Points	Hardware dependent	MPTCP on non-supported servers	User friendly
Single Proxy using VM	8	No	No	No
Double Proxy using VM	8	No	Yes	No
OpenMPRouter	8	Yes	Yes	No
Tessares	depend on client requirement (max. 8)	Yes	Yes	Yes
Double Proxy using WSL	8	No	Yes	Yes

Table 2.1: Comparison of each method for enabling MPTCP on unsupported OSes

In this chapter, we have shown the different methods for enabling multipath TCP on unsupported OSes. The first method is to use an MPTCP-enabled VM with an MPTCP proxy running on it. The second method is to use an aggregation box where all the network traffic is sent to the box, which will convert the TCP traffic into MPTCP.

We concluded that the first method was not user-friendly because the user had to manually install and configure a separate VM. The second method, using OpenMPRouter, is not user-friendly because it requires external hardware and the installation of a special kernel with MPTCP enabled, which is a painful process for users.

We considered the Tessares solution user-friendly because it is an out-of-the-box solution for users, eliminating the need for users to perform any additional installation or configuration. Finally, we demonstrated that enabling MPTCP on Windows using WSL offers the advantages of both the first and second methods. This implies that users only need to install our application to benefit from MPTCP, which is user-friendly. Additionally, this method does not rely on hardware components which allow the possibility of using multipath TCP in any location.

Chapter 3

Background

3.1 Multipath TCP

This section presents the desired goals of using MPTCP, how the protocol manages to use several paths, and what are the potential uses of this protocol.

3.1.1 Goals

The principal goal of multipath TCP is to enhance network throughput while also providing resilience against network outages. However, the protocol's performance in terms of throughput and resilience must not be lower than the standard TCP protocol. Then, multipath TCP must provide the same services as the standard TCP protocol, namely, connection establishment and reliable data exchange.

One of the first constraint that is imposed by the IETF is that the protocol must be backward compatible, meaning that if one of the hosts doesn't support the protocol, the connection must fall back seamlessly to a regular TCP connection.

Also, another aspect of protocol backward compatibility that must be respected is that the protocol must work with the majority of middleboxes, such as NATs, firewalls, and proxies. Also, the protocol must not assume that data segments sent over the cable arrive unmodified, as TCP options can be modified, removed, or duplicated.

Finally, from a security point of view, the level of security must be at or above that of a standard TCP connection, to ensure data confidentiality, integrity and authenticity. [11]

In summary, MPTCP aims to :

- Improve network bandwidth and resilience against network failures
- Maintain the same performance as TCP or better
- Provide backward compatibility with standard TCP protocol
- Ensure reliable transport of data in sequence like TCP
- Ensure that the protocol can work with the majority of middleboxes
- Uphold or exceed security standards set by regular TCP connections.

By fulfilling these goals, MPTCP aims to offer a robust and efficient solution for multipath communication while addressing the requirements of modern networking environments.

3.1.2 MPTCP Mechanisms: An overview

This section demonstrates how the protocol works and what the potential use cases are. First, we will discuss how an MPTCP is established and how it will fall back into a standard TCP connection for backward compatibility. Then we discuss how the protocol is able to establish multiple subflows and how it is able to exchange over these subflows. [12]

MPTCP handshake

A standard TCP connection begins with a three-way handshake [10]. The client initiates the connection by sending a SYN packet to the server to synchronise with it. To acknowledge receipt of the SYN request, the server sends a SYN-ACK packet. Finally, the client sends an ACK packet to acknowledge the reception of the SYN-ACK packet. At this time, the transfer of data can start.

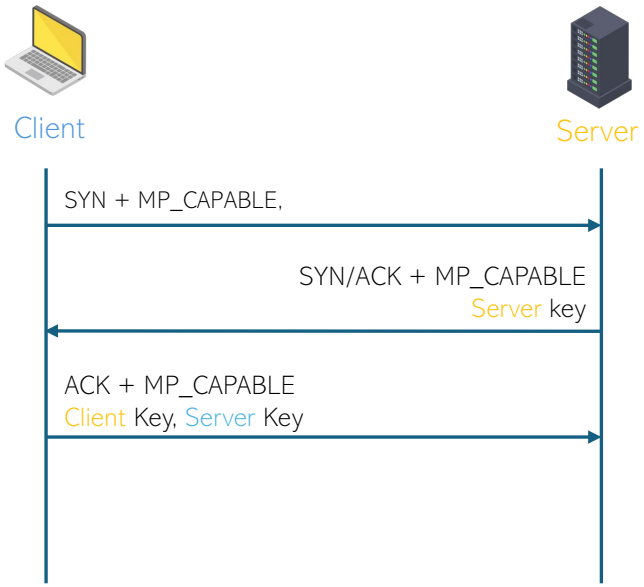

Figure 3.1: MPTCP handshake

On the other hand, a multipath TCP connection is the same as a regular TCP connection, except for the MP_CAPABLE option that is included in the option section of the TCP header. This option indicates that the host supports multi-path TCP. In the case where the receiving end replies back with the MP_CAPABLE option, it means that the receiving end also supports MPTCP and that a MPTCP connection can be started.

However, if the receiving end does not accept MPTCP support by omitting the MP_CAPABLE option in its response or if some middle-box (e.g., firewall) removes the MP_CAPABLE option because it does not accept it or does not recognise it, then the connection immediately falls back to a standard TCP connection using a single path.

In a standard TCP connection, we are able to identify a TCP connection by the 4-tuple (source IP,source port,destination IP,destination port) because only one path is used between both end.

In a multipath TCP connection, using the 4-tuple is no longer sufficient to identify a connection because of the presence of NAT. Therefore, we need to find a mechanism to determine which MPTCP connection the subflow wants to join even through a middlebox. To achieve that, during the MPTCP handshake, an exchange of 64-bit keys generated from both hosts is done. These keys are exchanged only on the handshake, and their purpose is to derive a 32-bit token that allows the

identification of an existing connection on the two hosts and to identify to which multipath TCP connection it wants to join.

Subflow addition

After the MPTCP connection is established, additional subflows can be added. The client or server can add additional subflow by doing a TCP handshake with the MP_JOIN option which includes a token to know to which MPTCP connection it want to join.

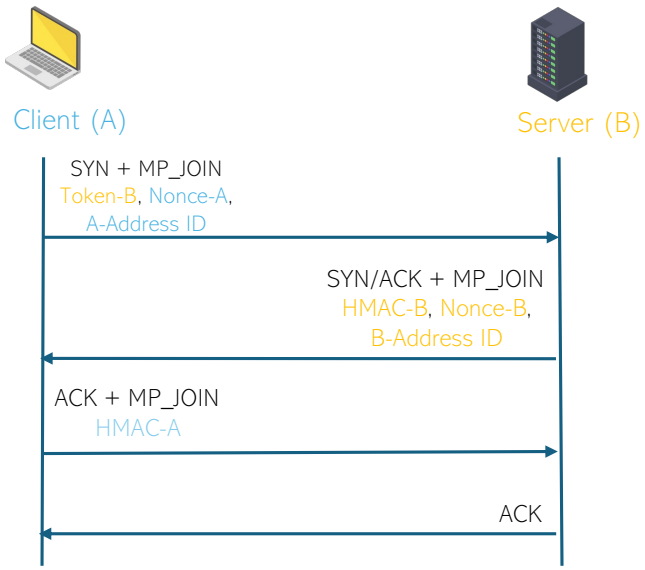

Figure 3.2: Subflow addition

If the token corresponds to an existing MPTCP connection, the server will acknowledge the reception of the SYN packets with the MP_JOIN option. To authenticate the addition of a new subflow, the server computes a hash-based message authentication code (HMAC) based on the nonce received by the client and both of the keys exchanged in the initial handshake.

The client receives the HMAC and a nonce provided by the server. If the client sees that the HMAC is conform, it also computes an HMAC based on the two keys, the nonce, and sends it back to the server. The server also verifies the HMAC's authenticity and replies back with an ACK. Both participants verify the authenticity of each other, which prevents attackers from spoofing or injecting data. [12]

Address addition and deletion

In order to remove a subflow, we need to be able to identify it; therefore, each subflow has a unique identifier named Address ID. The ADD_ADDR or MP_JOIN operations provide this identifier. This identifier is put in place to remove an address even if there is a NAT.

The ADD_ADDR option is sent within the MPTCP connection to inform the other end about new addresses on which a subflow can be established. This operation can be done at any moment of the connection in the case of the addition of a new path.

The REMOVE_ADDR option allows to remove paths and is sent also within the MPTCP connection. This option is mainly useful to inform the other end that an address is no longer available. This operation makes use of the address ID provided in the ADD_ADDR or MP_JOIN option. [12]

Data exchange

Once the connection is established, the transfer of data can begin. To achieve a reliable transfer, a 64-bit Data Sequence Number is shared across all subflows permitting receivers to reorder or re-transmit data even if multiple subflows are used. It is the Data Sequence Signal (DSS) option that contain the Data Sequence Number, the subflow Sequence Number and the length for which this mapping is valid. [12]

Prioritizing subflow

In a typical usage where the goal of using Multipath is to maximize the throughput, all the available paths will be used for data transfer. But there can be other use cases where the user wants to use a subflow as the main one and the second as a backup. For that purpose, there is a "B" flag that, if set to 1, determines if a subflow is used as a backup or not and can be included directly in the MP_JOIN operation to immediately inform the other end that he wants to use this subflow as a backup.

Host may want also after the subflow creation, change a specific subflow to a backup or vice-versa, Then, the MP_PRIO option can be used to change the "B" flag of the subflow where it is sent. When a subflow is used as a backup, only other non-backup subflows will be used to transfer the data. If all normal subflows lose the connection, then the backup subflow will take over to finish the transfer of the data. [12]

Connection closing

In a standard TCP connection, a FIN packet is sent to announce that there is no more data to send. In multipath TCP, a FIN packet only affects the subflow in which it was sent, and like in TCP, both participants have to acknowledge each other's FIN packet until the subflow is successfully closed.

When the sender decides that there is no more data to send, the DATA_FIN is signaled with the "F" flag set to 1 and is contained in the Data Sequence Signal option. Also, the DATA_FIN can be used by the receiver to verify if all the data has been successfully received from the sender. Finally, once DATA_FIN's of both ends have been acknowledged, all remaining subflows are closed with classic FIN exchanges. [12]

3.2 Windows-Subsystem for Linux

WSL is a virtual machine closely integrated with Windows that allows users to use a complete functional Linux VM without having to install third-party virtualization software separately.

3.2.1 WSL Architecture

The latest version of WSL uses Hyper-V, the official Windows virtualization tool, to create a lightweight virtual machine that can any Linux kernel provided from Microsoft. The advantage of using Hyper-V as a hypervisor is that the virtual machine runs directly on the computer hardware without requiring an underlying host operating system, unlike type 2 hypervisors, which run on top of an existing operating system. [20]

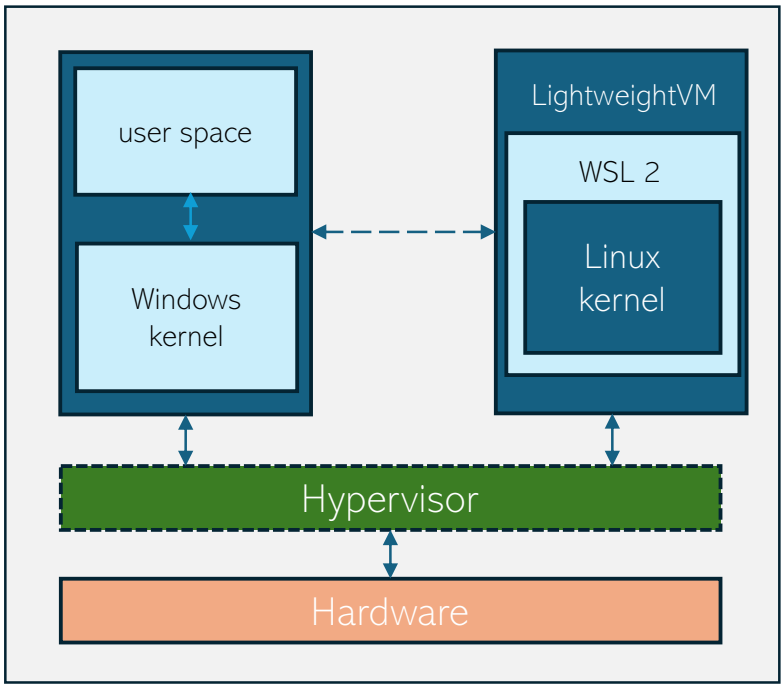

Figure 3.3: WSL2 architecture

3.2.2 WSL1 and WSL2

WSL1 is the first version proposed by Microsoft and allows you to run most Linux distribution by using a compatibility layer that translates Linux system calls into Windows system calls. This imposes limitations such as not being able to run Graphical User Interface, In contrast to WSL1, WSL2 uses a virtualization platform based on Hyper-V, allowing you to run a full Linux kernel.[20]

3.2.3 WSL1 advantages

WSL 1 has a lower system overhead than WSL 2 since it does not involve running a full Linux kernel inside a virtual machine; therefore, it may be a better choice for low-resource machines. Also, WSL1 has a simplified network architecture compared to WSL2. WSL1 mirrors the network configuration of the Windows host machine. So if there are multiple interfaces on Windows, all of them will appear on WSL1. The main disadvantage of WSL1 is that there are some system-call incompatibilities. Finally, WSL1 does not allow to run a custom kernel, and because of that limitation, enabling MPTCP on Windows would be impossible. [20]

3.2.4 WSL2 advantages

The advantage of choosing WSL2 is that it offers the possibility of completely launching a Linux kernel inside a virtual machine, where all system calls are compatible without translation layers. Finally, WSL2 allows us to run a custom-built Linux kernel provided by Microsoft, which will allow us to activate MPTCP on WSL2 by enabling MPTCP on the kernel configuration file. The disadvantage of using WSL2 for our application is that it does not expose all network interfaces and uses NAT; NAT stands for Network Address Translation; each WSL2 distribution has its IP address that is only accessible from the Windows device on which it is running. To let WSL2 communicate with the external world, Windows translates outgoing WSL connections to look like they are coming from the Windows host. [20]

3.2.5 Hyper-V

As said before, WSL2 use a lightweight virtualization platform that is built on top of Hyper-V. Hyper-V is the official virtualization tools that is built on the Windows Kernel, Hyper-V allows their users to run virtual machine. The difference between other hypervisors is that they are type 2 hypervisors which load kernels drivers in the user-space, while Hyper-V is a type 1 hypervisor, which means that it runs at the Windows kernel level. [20, 7]

Hyper-V virtual switch

An Hyper-V virtual switch is an Hyper-V functionality allowing to manage the network of VM's. it allows virtual machine running on Hyper-v to communicate between VM's and with the external network. [7] Multiple type of Hyper-V switches can be created :

- the *External* virtual switch connect the virtual machine directly to the physical network by doing a binding with the network adapter. This type of switch will be used for our application to expose all network interfaces to WSL2.
- the *Private* virtual switch allows communications between virtual machines on the same Hyper-V host. So there is no connectivity with the Windows host machine itself or with the external network.
- the *Internal* virtual switch allows the communications between virtual machines on the same Hyper-V host and with the host machine itself but not with the external network.

3.2.6 Network limitation on WSL2

WSL2 uses a single internal switch to communicate with the external world, so it is not exposed to every Windows network interface. This means that even if we used an MPTCP-enabled Linux kernel, we would not be able to profit from multiple network interfaces. Also, WSL2 only supports IPV4. We will see in the next chapter how we were able to expose all network interfaces.

Chapter 4

Enabling Multipath TCP on WSL

This section presents the general architecture of the application permitting the activation and the correct usage of MPTCP on WSL2. We will then define in detail each important step allowing its use and we will discuss the implementation choices that have been for our application.

4.1 General architecture

Our implementation is straightforward to understand, it is a service in Windows that is running on the background, once the service sees that WSL is running, this service will detect network interface in Windows and making them usable inside WSL by creating a virtual switch for each Windows network interface.

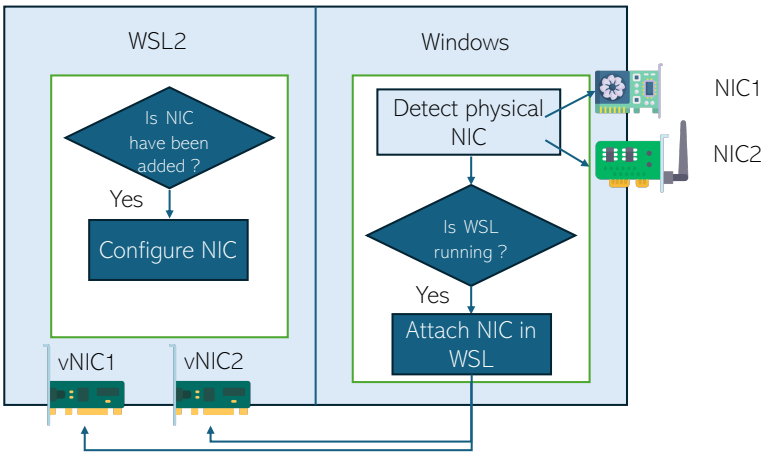

Figure 4.1: General mechanism for enabling MPTCP on WSL

The service also change the WSL configuration file to specify the path of custom built MPTCP-enabled kernel . Without that even if all interfaces where exposed we wouldn't be able to use MPTCP since it is not available on the kernel.

When installing a WSL distribution, there is no network manager pre-installed, meaning that there is no automatic network configuration for each network interface. Another service will be running inside WSL2 to detect the presence of each network interface and configure them properly. In our case, we used Network Manager to configure each interface; we will discuss later on section 4.3.6, why we used Network Manager to configure WSL network interfaces.

The script running by Network Manager inside WSL will configure each network interface according to a configuration file located in Windows, allowing users to properly configure each interface according to their preference. After the script configures each interface, users will be able to use MPTCP inside WSL.

4.2 Steps for activating and using MPTCP

4.2.1 Compiling the kernel

The kernel serves as the core component of an operating system, acting as an interfaces between the hardware and software and is the foundation upon which the entire operating systems operates playing a critical role in determining system performance, stability, and functionality.

The default Linux kernel provided by Microsoft doesn't have MPTCP enabled by default, we need then to enable it by our self by changing the Linux build configuration file.

Once the kernel with MPTCP enabled we can the specify the path of it into the WSL configuration to tell the hypervisor to boot the linux virtual machine with the MPTCP-enabled kernel. We recall that it is not sufficient to enable MPTCP on the kernel, since not all Windows network interface are exposed to WSL2.

4.2.2 Exposing all network interfaces

We remind that WSL2 have only one interface which is an internal virtual switch. When packets are sent from WSL instance to external networks, then routed into

an internal virtual switch created by the Windows host.

The virtual switch performs Network Address Translation (NAT), which involves translating the IP address of outgoing packet coming from WSL2 to the IP address of the Windows interface before sending them out to the external networks.

Using TUN interfaces

The first approach that we thought of is to expose every Windows network interface to WSL is to create TUN interface for each physical NIC. A TUN interface is simply a virtual network interface that operate at the Layer 3.

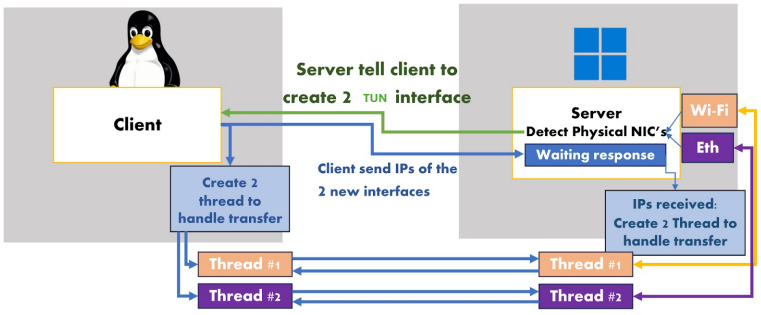

Figure 4.2: Exposing Windows interface using TUN interface

We could then create a thread in Windows for each physical network interface, we would then NAT the receiving packet of WSL to the corresponding IP address of the Windows interface. Each thread will be responsible for receiving/transmitting data to the corresponding interface.

When incoming packet arrive on Windows, we just check if the incoming packet are destinated to WSL2 and forward it to the corresponding TUN interfaces. With that WSL would be able to use all available interfaces.

The disadvantages of this method are that the forwarding of packets is done by ourselves, and we also have to detect changes in the network. If a Wi-Fi or Ethernet connection suddenly changes or the IP address of the interface changes, we would have to find a mechanism to be informed about these events.

Also, Windows doesn't provide any API to be able to parse packets and modify them to be able to send the packet to the right WSL interface. We will see that there is a better option for exposing every network interface to WSL.

Using Hyper-V Switch

Another approach to expose every network interface to WSL is to create a virtual external switch using Hyper-V utilities; these switches allow building a bridge between network interfaces on Windows and any Hyper-V virtual machine. We create a Hyper-V external switch for each interface and attach them to WSL2. This way, we avoid data transfer management, and performance will undoubtedly be improved since the management of data transfer is entirely done by Windows. This is the method we chose for exposing all network interfaces into WSL for the reasons cited above.

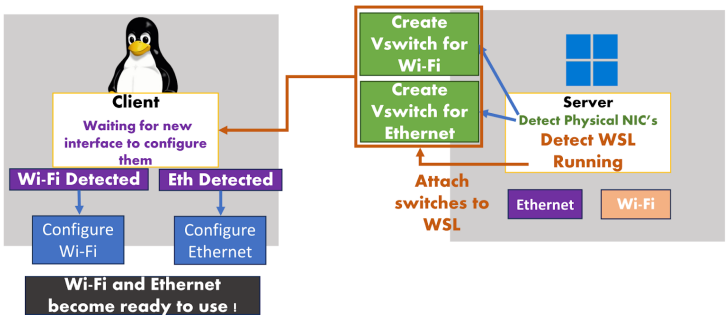

Figure 4.3: Exposing Windows interface using Hyper-V switch

Unfortunately, Windows does not allow us to attach a virtual switch to WSL2 natively. To counter that, we found an open-source solution named WSLAttachSwitch [6] that allowed us to attach a virtual switch into WSL. It simply retrieves the GUID of a virtual switch and updates the configuration of the WSL2 virtual machine to include the virtual switch that we want to add.

Once all network interfaces are attached, we need to configure them correctly. We mean by that to give an IP Address, create a route, and decide which interfaces will be a normal subflow or a backup one. We will see later in this chapter how we managed to do that.

4.2.3 Encountered issues

The switches attached are disconnected when WSL goes down, which means that each time we have to use all available network interfaces on WSL2, we have to reattach the switches. Also, since WSL2 does not have any network manager, each time that WSL2 restarts, we must reconfigure each interface. This is quite painful for users who want to use MPTCP.

Also, Windows does not allow the identification of each WSL instance, meaning if multiple users are running the WSL instance, we would not be able to tell to which users the instance belongs. Hence, we could not attach interfaces properly.

4.3 Implementation details

This section explains in detail how our implementation works to effectively use MPTCP on WSL2. First, we will define how the implementation can detect only physical network interfaces. Then, we will look at how the implementation can attach each network interface. Finally, we will see how we configure each network interface after the network interface has been attached.

4.3.1 Network Monitor

The core part of our implementation is to detect each physical network interface; this is the job of the Network Monitor; when he detects the addition or deletion of any physical network interface, it triggers an event that another component of the application can use. Unfortunately, no mechanism offered by the Windows API triggers an event when there is an addition or deletion of network interfaces. Therefore, the Network Monitor will poll periodically on another thread and fire events when any change occurs.

Encountered issues

The API provided by .NET Core and C# (the language we used to develop the application) to get information about network give all network interfaces including the virtual switches, also, the API doesn't make the difference between a virtual switch and a physical network interfaces. The solution for this problem is to use Windows Management Instrumentation (WMI), an infrastructure for management data and operations, which allows to gather information about hardware component and the operating system. We could ask a query to WMI to gather all physical network interfaces names that are either PCI or USB network interfaces.

4.3.2 Hyper-V Manager

HyperVManager is responsible for creating an external hyper-v switch when a network interface is detected; HyperVManager will then attach a function to the NetworkMonitor event; if it is an addition of an interface and that interface doesn't have a Hyper-V switch created, it will make one with the name prefix: "MPTCP -"

and an event will be triggered to inform another component of the application that a new Hyper-V switch has been created.

Encountered issues

Our application is multi-threaded and Windows does not permit the simultaneous creation of virtual switches, meaning that we must use a synchronization mechanism called Mutex that prevents the creation of two switches simultaneously.

4.3.3 WSLAttacher

When a network interface has an external virtual switch, WSLAttacher will be responsible for attaching the interface to WSL. The issue arises because attaching an interface necessitates the launch of WSL; therefore, attaching an interface solely upon the creation of a switch is insufficient.

WSLAttacher checks if WSL is running, and polling (checking each time interval) is not necessary because the Windows API (Windows Management Instrumentation) provides a way to execute code when a specific process is launched without the need of polling; this way, we can efficiently attach each network interface to WSL.

4.3.4 NetworkConfig

A configuration file is essential to enable users to configure each endpoint properly. NetworkConfig contains multiple parameters to configure the multipath TCP properly, such as:

- *subflow* : This parameter is an integer value that define the maximum number of subflow that can a MPTCP connection can create.
- *add_addr_accepted* : This parameter limits the number of addresses that are learned over each Multipath TCP connection. This parameter is used to protect the Multipath TCP implementation against attacks where two many addresses are advertised
- *ManageKernelLocation* : Let users decide if the program manage automatically the path of kernel location.
- *network information* : List of information about network interfaces like the interface name, the Windows MacAddress, the Linux MacAddress of the corresponding Windows MacAddress, and for each network interface, a list of endpoints that define how the network interface should behave in the match context, meaning if the interface will serve as a backup or a normal sub flow.

- *Proxy* : Users can set a proxy in order to use the double Proxy Technique to use MPTCP even if the distant server is not MPTCP compatible. We will talk in more detail about proxying in the next chapter.

These details are all saved into a configuration file located in Windows, and then the service in charge of configuring network interfaces will read this file to configure them in accordance with this file.

4.3.5 FileManager

The FileManager allow to the program to get path from various important file used for enabling MPTCP like the configuration file as described above, the logging file and the kernel image.

Encountered Issues

A first attempt was to store the configuration file in the ProgramData folder. This folder is generally used for storing application configuration files; unfortunately, a Windows service runs by default under a LocalService Account, which does not have enough permission to write in that folder. An alternative was found and was to store the configuration file under the AppData folder.

4.3.6 NIC's configuration in WSL

No network configuration is done by default when network interfaces are added to WSL since there is no network manager. This section shows all the possibilities that have been tried to configure correctly each network interface.

udev

udev is software that monitors specific kernel changes, like the addition or deletion of a new device. It then allows system administrators and users to tell how to react to these events by describing them as rules. The Linux kernel primarily generates the events that Udev's daemon receives in response to physical events related to peripheral devices. [14]

We could use udev to configure network interfaces. To accomplish this, we create a rule that executes a script whenever it detects a network interface addition or deletion. The script would then receive the interface's name in arguments, and we could assign an IP address, create a route, configure endpoints, and so on.

The problem is that the script is run only when a network interface is added or removed; imagine if the IP address of the interface has changed; we would not be able to adjust to route configuration accordingly because it only triggers the script when an interface is added, moreover, when the interface goes down to up all route configuration are deleted, that means we have to find a way to trigger the script when the network interfaces card is added or IP address changed.

dhclient

dhclient provides a way to assign IP address using the Dynamic Host Configuration Protocol (DHCP). The DHCP protocol allows a client to contact the DHCP server, which maintains a list of available IP addresses that may be temporarily assigned for network communication. The good thing about dhclient is that it allows users to run a script when an IP address changes; this allows them to adapt the route accordingly. [1]

The inconvenience of using dhclient is that it does not provide any mechanism to run a script when the interface goes down or up, or even when an interface is attached. Therefore, we could not configure each interface as soon as network interfaces appear. We could use a combination of dhclient and udev to configure it, but we will find out later that there are better options for configuring network interfaces.

ifupdown

Ifupdown is a set of utilities for configuring and managing network interfaces [16] , allowing you to properly configure each interface according to a configuration file. Ifupdown allows you to run a script whenever a network interface goes up or down, even if the interface's IP address changes. The only issue we encountered with ifupdown was its deprecation.

NetworkManager

NetworkManager, like ifupdown, is software that allows administrators or users to configure network interfaces, whether using a configuration file or by using the command line. [17] Network managers provide mechanisms for running scripts when interfaces go from down to up, when the interfaces change IP addresses, and even when the interfaces appear.

We chose network manager because it provides all the necessary components to properly configure each network interface in an efficient way. More over, network manager is not deprecated.

Also, when an user decide to change network connection in the Windows side, we need to be able to detect those change and act upon it. If we we're using a combination of dhclient and udev, it would be a painful process to re-configure interface when for example users decide to change Wi-Fi connection for example. Fortunately, NetworkManager is able to detect those changes and trigger the configuration script accordingly.

4.3.7 Case : retrieving user settings

When attaching an interface to WSL, there is no way to tell to which Windows interface it belongs, so we need to find a mechanism to identify the mapping between the Windows and WSL network interfaces.

Unfortunately, WSLAttachSwitch doesn't allow you to specify the interface name that it will create on WSL. If that were possible, we would be able to create a mapping between the Windows and WSL interfaces by matching interface names.

We found a workaround with WSLAttachSwitch because it allows us to specify the MAC address of the newly created interface, and then we just save into a configuration file the corresponding Windows MAC address and the WSL MAC address. The script responsible for configuring the NIC will be able to retrieve for each network interface the corresponding configuration.

Chapter 5

Proxies

This section presents some theoretical background about proxying, which types of proxies have been used and, more specifically, transparent proxying, and lastly, why proxies will be used to enable MPTCP on Windows.

5.1 Definition

A proxy server acts as an intermediary between a client and another server from which the user requests information. The proxy receives network requests from the client and executes them on their behalf. Once the response is received from the server, the proxy server sends it back to the client. Users with a proxy can hide their identity because the request is not made on behalf of their own machine and IP address.

5.2 Types of Proxy Servers

This section presents the different types of proxy that exist, their own properties and use cases.

5.2.1 HTTP proxy

A Hyper Transfer Protocol proxy handles HTTP requests on behalf of another client as if they originated from the proxy's IP address. To achieve that, the client will send the HTTP request to the proxy server. Since the data are unencrypted, the proxy server can look at the HTTP header to find the final destination with the "host" parameter. The proxy server will then make a TCP connection to the final destination, get the result back, and send it back to the client. [22]

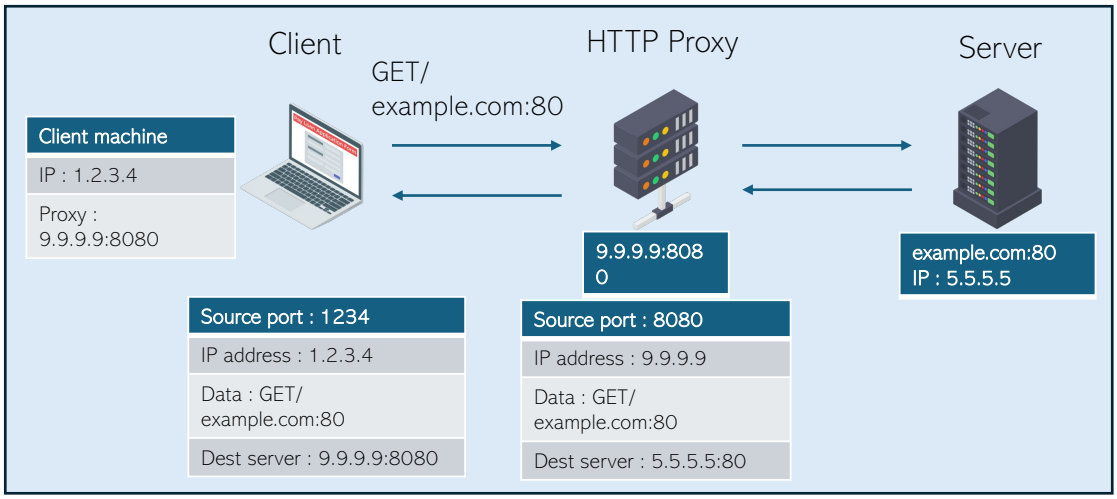

Figure 5.1: How a HTTP proxy works

5.2.2 HTTPS proxy

the HTTPS proxy works a bit the same like the HTTP proxy, but in this case, the data is encrypted. the client establish a TLS connection with the proxy server, and since the proxy server is in the part of the connection he can decrypt the content of the client request, then, the proxy make a establish a secure connection with the final destination, get the result back, and send it back to the client. an HTTPS proxy can be source of the vulnerabilities since the proxy servers sees every data in clear.

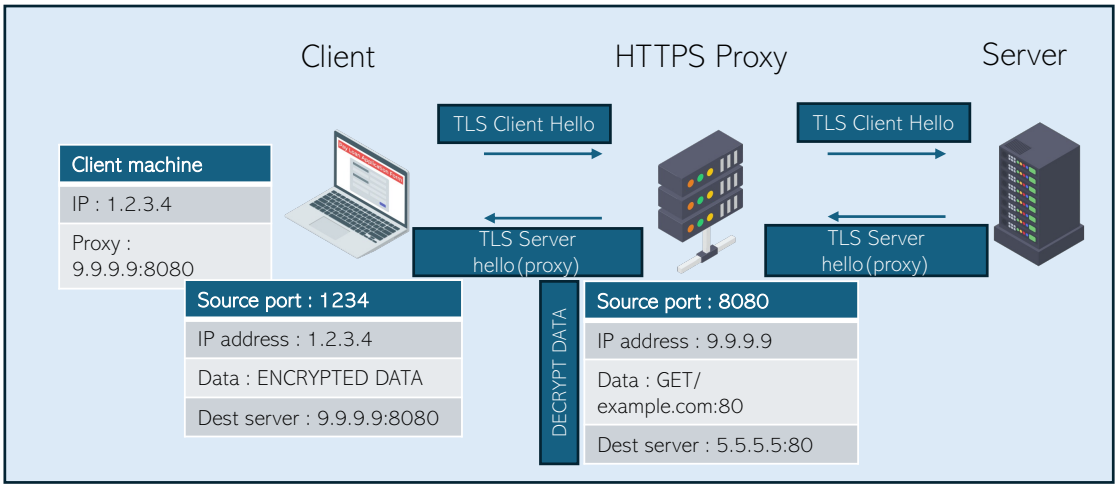

Figure 5.2: How a HTTPS proxy works

5.3 SOCKS proxy

SOCKS stands for Socket Secure, a protocol developed by David Kobals and improved by Ying-Da Lee. It acts on the transport layer and can be used as a proxy for any application that uses TCP, unlike the HTTP proxy, which can only be used with HTTP content. Different versions of the protocol have been implemented, and we will show how they differ between them. [15]

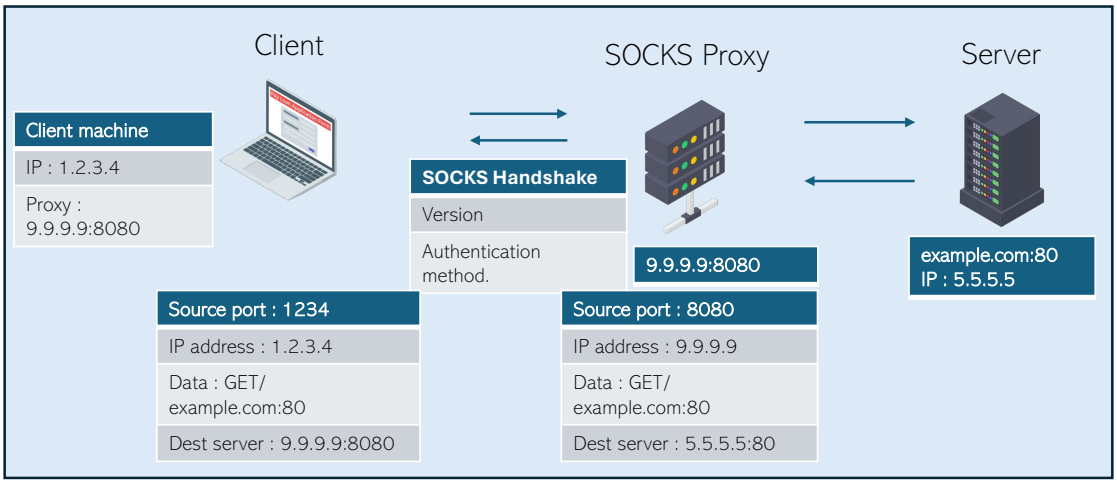

Figure 5.3: How a proxy works

SOCKS v4

Socks4 is an older version of the SOCKS protocol and is less secure compared to SOCKS5. Socks4 works by establishing a TCP connection between the client and the proxy server. The client sends a connection request to the proxy server, which then forwards the request to the server. The advantage of Socks4 proxies is that they are faster than SOCKS5 and don't require authentication, but they are less secure compared to SOCKS, don't support UDP, and don't resolve DNS.

SOCKS v4a

Socks4a is a modification of the Socks4 version that adds support for domain name resolution. The mechanism for sending the data from the client to the server works the same.

SOCKS v5

Socks5 is the last version of the SOCKS protocol and is more secure compared to SOCKS4. It has additional features, such as authentication and support for UDP

and IPV6. However, SOCKS5 proxies are generally slower than SOCKS5.

5.4 Transparent proxying

The method of transparent proxying intercepts network traffic and forwards it to a proxy server. Unlike a traditional proxy, where the user must specify a proxy for a specific application, a transparent proxy forces the user to have a proxy for all applications. This is a clear advantage when an application does not allow the use of a proxy in its settings.

5.4.1 TProxy

The Linux kernel provides TPROXY, a functionality for transparent proxying. Unlike other methods of transparent interception, TPROXY can maintain the original source and destination IP addresses of the traffic. The Linux kernel facilitates this by marking packets that require forwarding to the proxy server. Because the packet's IP sender is preserved, the proxy server must use the (SOL_IP, IP_TRANSPARENT) socket option due to IPV4 limitations. That way, we will be able to send datagrams from non-local IP addresses. [19]

5.4.2 Redsocks

Redsocks is transparent proxy software that is open-source and allows to redirect network traffic into HTTP,HTTPS, SOCKS4, and SOCKS5 proxies.[8] unlike TPROXY, which is primarily designed for transparent proxying with protocols such as HTTP and HTTPS.

5.4.3 Usage in our case

To redirect the traffic to WSL, we're going to use transparent proxying, which will transform the TCP traffic into MPTCP. The transformation of the TCP traffic into MPTCP traffic takes place on the proxy server, not the transparent proxy, since the transparent proxy only acts as a link between the Windows traffic and the proxy server running on WSL2. We used Redsocks because it was easier to setup and also support out-of-the-box the usage of an SOCKS5 proxy.

5.5 Proxy chaining

Multiple proxy chaining involves using several proxy servers before contacting the remote server. This technique adds an extra layer of security by hiding the connection's originator, making it more difficult to trace or intercept traffic. [9] We use chaining for none of the reasons listed above, and we describe in section 5.5.2 why we use proxy chaining for our application.

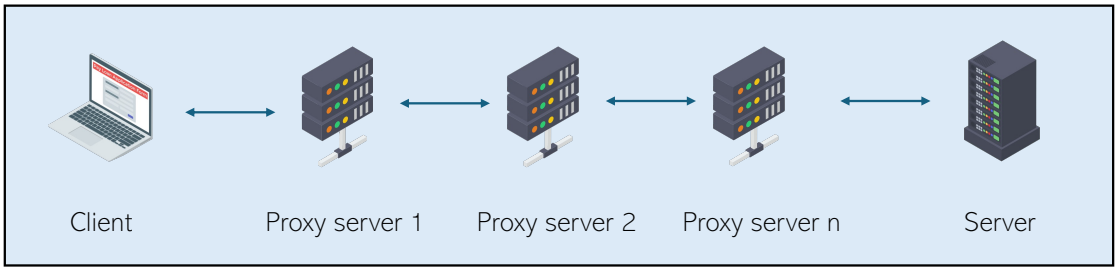

Figure 5.4: proxy chaining illustration

5.5.1 Proxychains

Proxychains is Unix software that allows for proxy chaining and can force any TCP connection made by any given application to be redirected into an HTTP or SOCKS proxy. Proxychains requires a configuration file that contains information about the chaining structure, including the IP address of each proxy and its type (socks4,http,...). This is the software we will use to chain two MPTCP proxy.

5.5.2 Usage in our case

To be able to use several interfaces during a connection, both the client and server must support and activate MPTCP. This is a problem when remote servers don't support MPTCP. A solution to this problem is to use the double proxy technique, as we mentioned in section 2.1.2. The client sends its traffic to the MPTCP proxy on WSL, which then establishes an MPTCP connection with a remote proxy. This remote proxy then forwards the traffic to the final remote server. This makes it possible to use MPTCP, even with a server that doesn't implement it.

Chapter 6

Enabling MPTCP on Windows

This section will present how we managed to enable MPTCP on Windows. We will first present the general architecture for how this works and take a deeper look at our implementations.

6.1 General architecture

The architecture for enabling MPTCP on Windows is very similar to enabling MPTCP on WSL. We will use the application we built to enable and configure MPTCP properly on WSL. Then, we redirect all traffic to the WSL internal switch. WSL will have a firewall rule to forward the TCP traffic to the Redsocks transparent proxy. Finally, Redsocks will forward it to the MPTCP proxy to convert the TCP traffic into MPTCP traffic.

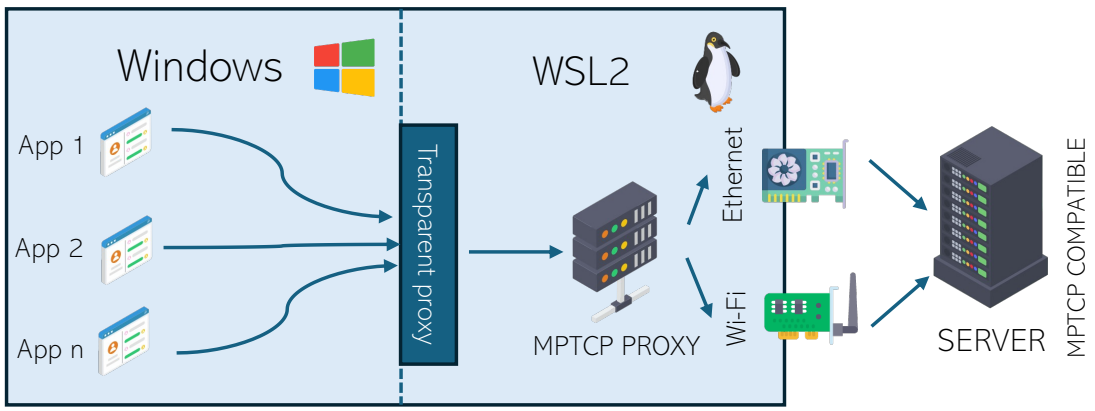

Figure 6.1: Architecture for enabling MPTCP on Windows

Users can decide also if they don't want to redirect all the traffic into WSL, to specify the proxy server address into the application settings, this way users can control where they want to use the MPTCP proxy.

6.2 Routing Windows traffic

To route the traffic into WSL, we change the default Windows route settings with the IP address of the WSL2 virtual machine. It seemed easy at first, but when we first tried, we discovered that the virtual machine interface always had the highest metric value, meaning that the WSL NIC would never be used for handling traffic. The solution we found is to change the metric value of the network interface itself, and not the route itself. This way, the route going to the WSL internal switch will always be preferred.

6.3 Choosing the WSL internal switch as gateway

We need to remind them that WSL2 runs over Hyper-V, which contains the Linux kernel. The internal switch provided by Hyper-V to connect between Windows and WSL is always eth0. Since eth0 no longer handles network traffic, it only functions as a gateway between Windows and WSL2. Initially, we could attempt to route the traffic into interfaces we added through our application, but the challenge lies in the DHCP server's ability to change the IP address of these interfaces at any time, unlike eth0, whose IP address remains constant.

6.4 Redirecting traffic to transparent proxy

The Windows incoming traffic that goes into the eth0 interface passes first by the pre-routing chain. This chain makes any routing-related decisions before sending any packets. We add a rule to this chain to redirect the TCP traffic to the redsocks transparent proxy.

6.5 Converting TCP into Multipath TCP

Once the transparent proxy has received TCP frames, it will forward them to the proxy. The MPTCP proxy will then connect with an MPTCP socket on behalf of external requests. This is how we can convert TCP frames into MPTCP ones.

Chapter 7

Performance evaluation

This chapter presents the tests that were carried out during the use of MPTCP on WSL and Windows. We will first demonstrate the use of MPTCP on a classic file download, then talk about using MPTCP during video streaming. Next, we will talk about using MPTCP for live streaming. Finally, we discuss about using MPTCP for remote desktop access. In each section, we'll discuss the use of MPTCP in aggregation or backup mode, i.e., an interface will take over if the other subflow fails.

7.1 Test setup

All tests were carried out under Windows 11, with a 35 Mb/s wired connection to my ISP (VDSL) and a 40 Mb/s connection to my 4G access point with a Wi-Fi USB stick. The Linux kernel version used in WSL is 6.1 with MPTCP enabled. Also, to use the dual proxy technique, Google Cloud services were used to host a socks5 proxy, hosted on Ubuntu 23.02 with multipath TCP enabled.

7.2 Latency test

Latency is an important parameter for measuring the characteristics of a network. We will therefore show the latency for each possibility, i.e. without proxy, using the proxy technique and finally using the double proxy technique.

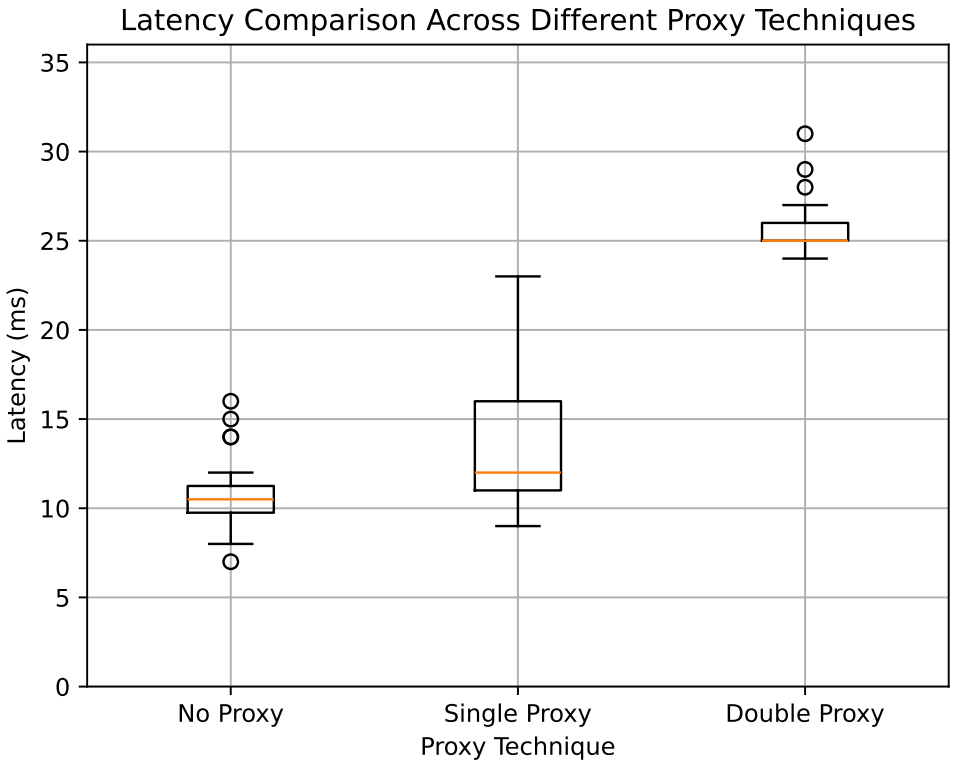

Figure 7.1: Latency for each scenario

As we can see, using a proxy has an impact on the application's latency. However, it has a minor impact, since the difference between not using a proxy and using one is 2 ms. The difference is more noticeable when using the double proxy technique, as there is a difference of approximately 15 ms.

7.3 Bandwidth test

7.3.1 Downloading file

Here's we will presents the use of MPTCP when downloading file, data in the internet become more and more heavy, so we will shows that MPTCP using aggregation mode will help us download data faster than using a regular TCP connection.

Aggregation Mode

As we can see the graph below we are able to use multiple network interfaces when downloading the file. Of course, the throughput depend on your Internet Service Provider, your subscription, or if u are using 4G or 5G.

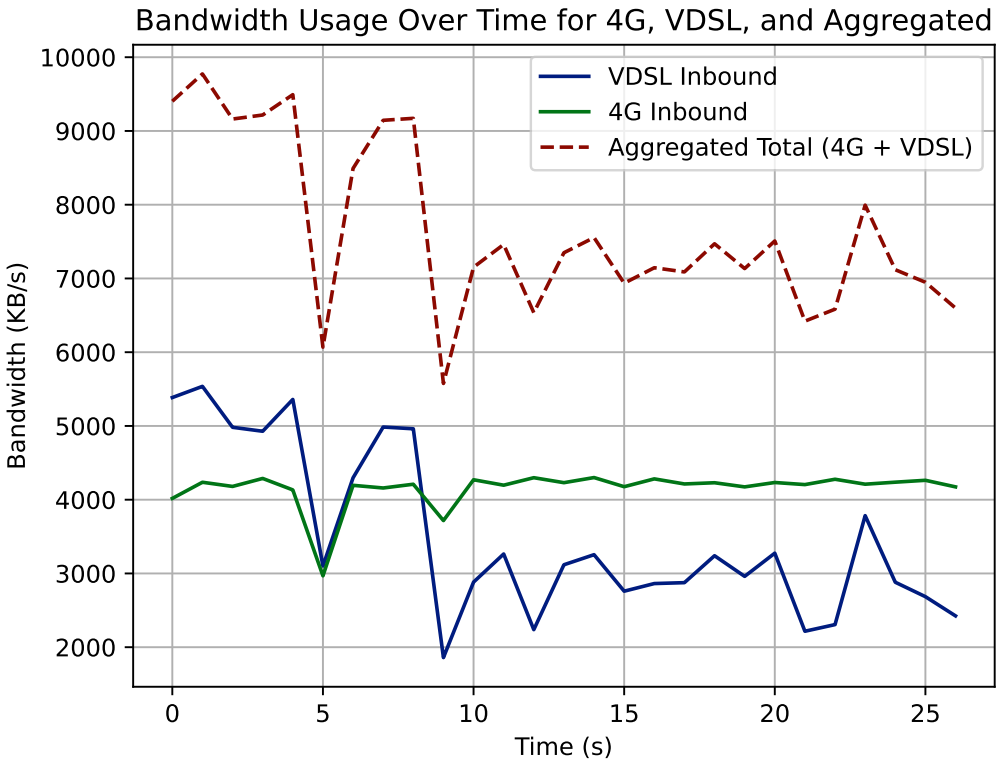

Figure 7.2: Aggregation mode when downloading file

Backup Mode

When using the backup mode, only the non-backup subflow will be used, as we can see in the graph when the primary connection get down, the backup subflow will take the relay to finish the download. Also when the non-backup subflow is available, it will take over to participate on the download.

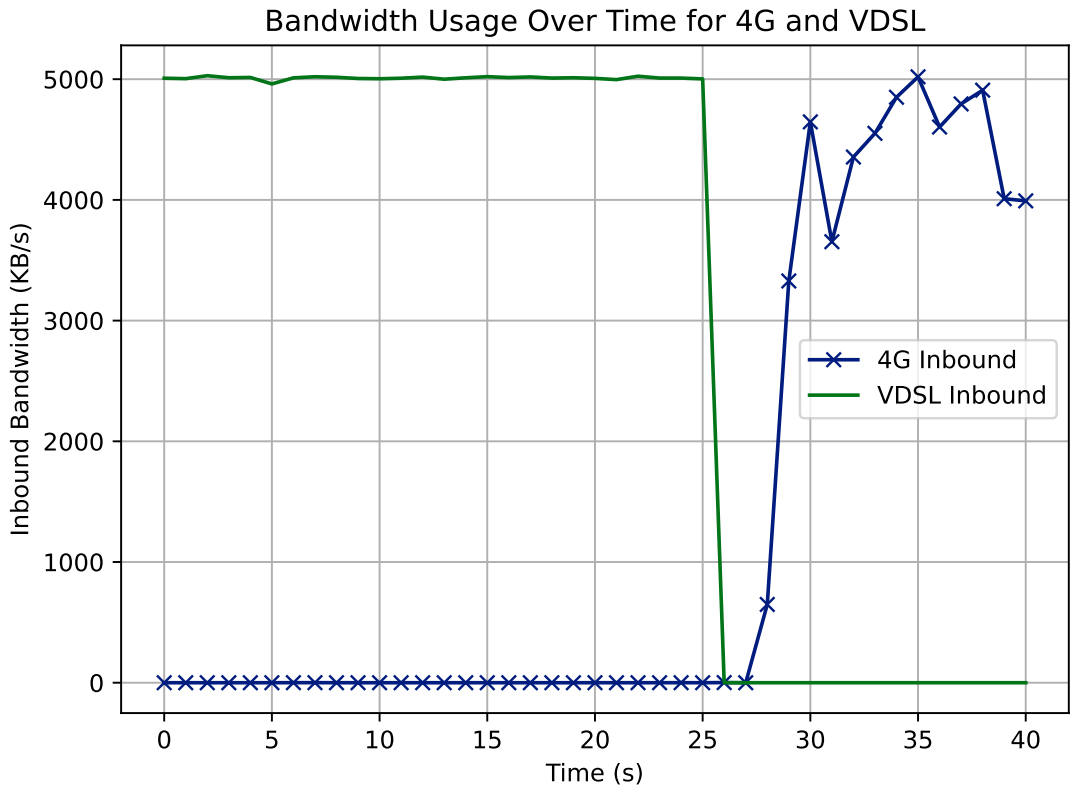

Figure 7.3: Backup mode when downloading

7.3.2 Video streaming

Now, we will present the usage of MPTCP when we are watching any video online, we will shows both aggregation mode where all available subflow will be used and backup mode where one subflow serve as backup.

Aggregation Mode

We can observe the utilization, of both network interface for downloading/watching the video, also we can see that there's is no more download and the reasons of that is because there is enough buffering for the video.

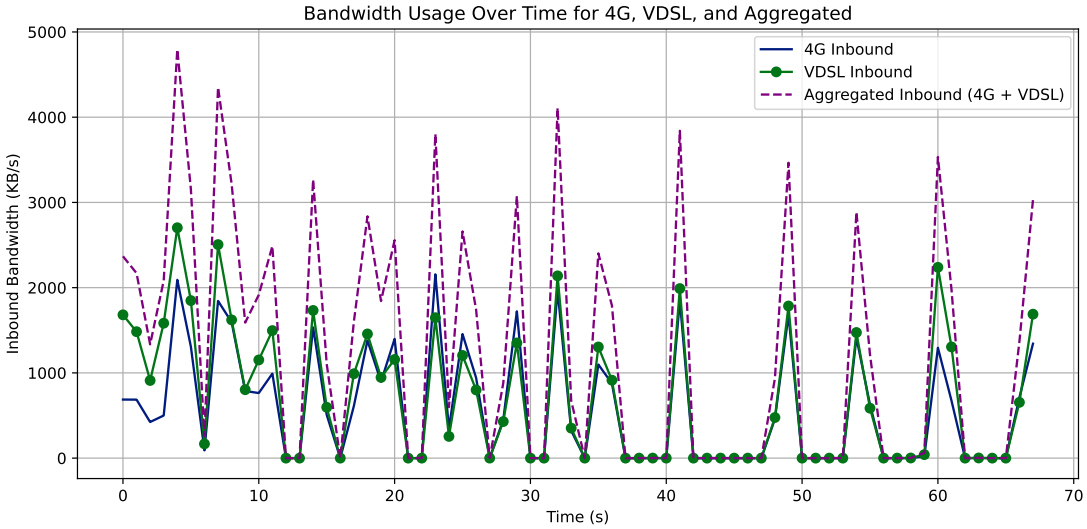

Figure 7.4: Aggregation mode when video streaming

Backup Mode

As the graph shown here, we can see that the non-backup subflow is used to download the video, and when the main subflow goes down or loose connectivity, the backup subflow take over without loosing the MPTCP connection.

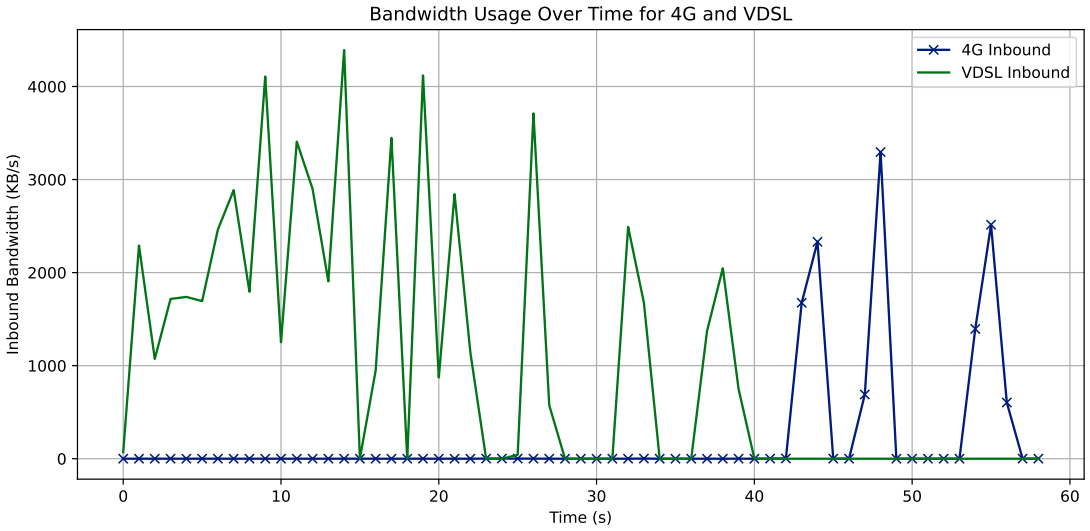

Figure 7.5: Backup mode when video streaming

7.3.3 Live Streaming

Here we will see the usage of MPTCP in the context of Live Streaming, Twitch is known platform that allows users to either stream contents or watch other people stream. We are using Twitch for our experimentation because Twitch use HTTPS Live Streaming (HLS) for watching or uploading content, hence, TCP is used.

Aggregation Mode

As we can see in the graph below, multiple interface are used since HLS is based on TCP. Aggregation mode will be particularly useful in Live Streaming because it is resource consuming especially if someone want to stream in a high resolution like 4K.

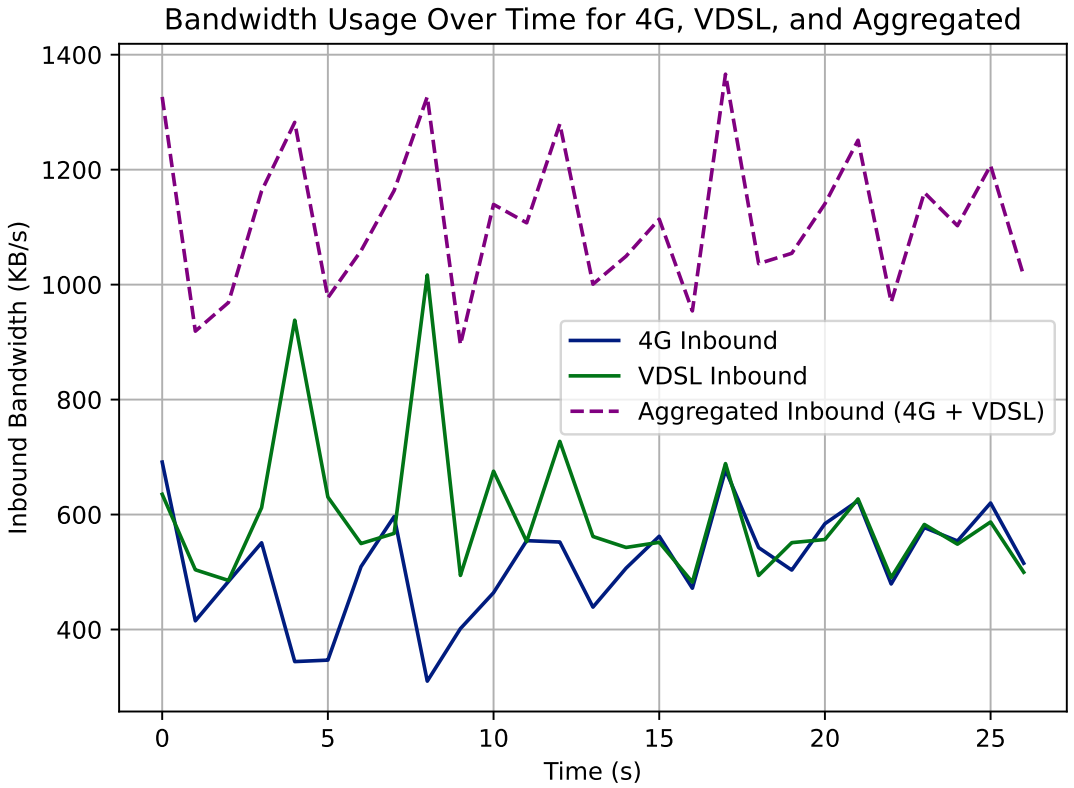

Figure 7.6: Aggregation mode when live streaming

Backup Mode

Streamers nowadays can loose of audience because of a low quality stream or because the stream has crashed due to loose of internet connectivity, Therefore using a network interface has a backup subflow can save you audience and potentially money. The graph below show that the non-backup subflow is used for the stream and as soon as the main subflow loose connection, the backup subflow take the relay.

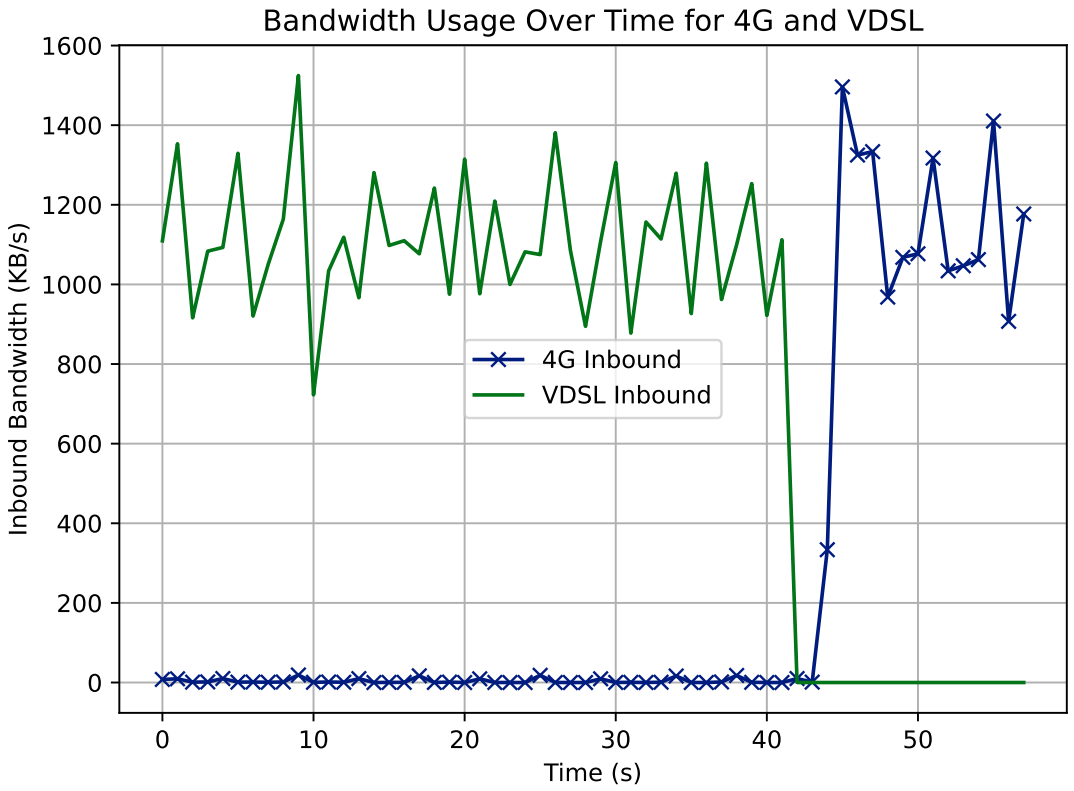

Figure 7.7: Backup mode when live streaming

7.3.4 Remote Desktop Accessing

In this section we will show the usage of MPTCP in video conferencing, We will show that MPTCP can be useful in Remote Desktop Accessing if one single network interface does not provide enough bandwidth then aggregation mode will be perfect for that case or if that there is instabilities, backup mode will be preferred.

Aggregation Mode

In the graph below, we can see that both of interfaces are used for screen sharing, video and audio transmission.

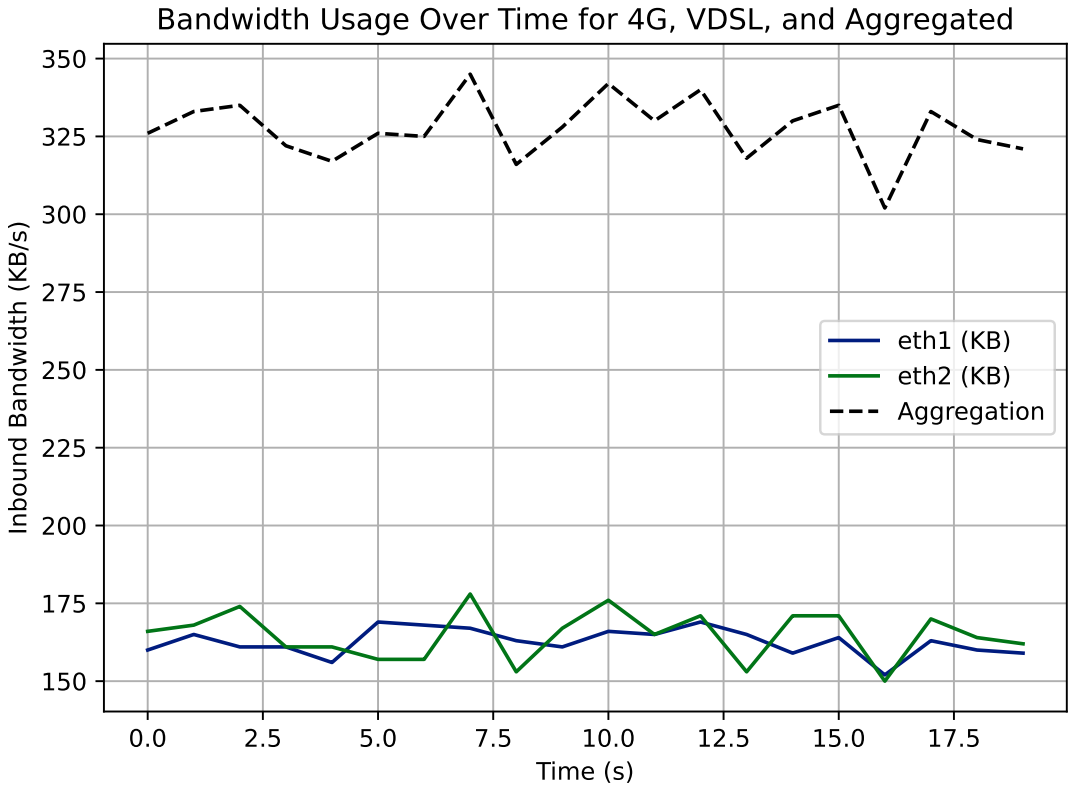

Figure 7.8: Aggregation mode when accessing a Remote Desktop

Backup Mode

In backup mode like other scenario, the main subflow is used for remote desktop accessing, and as soon as the main subflow goes down, the backup subflow take the relay. It is particularely useful, because no one want to be interrupted by a loose of connectivity, Thanks to the use of MPTCP, we are able to solve this problem.

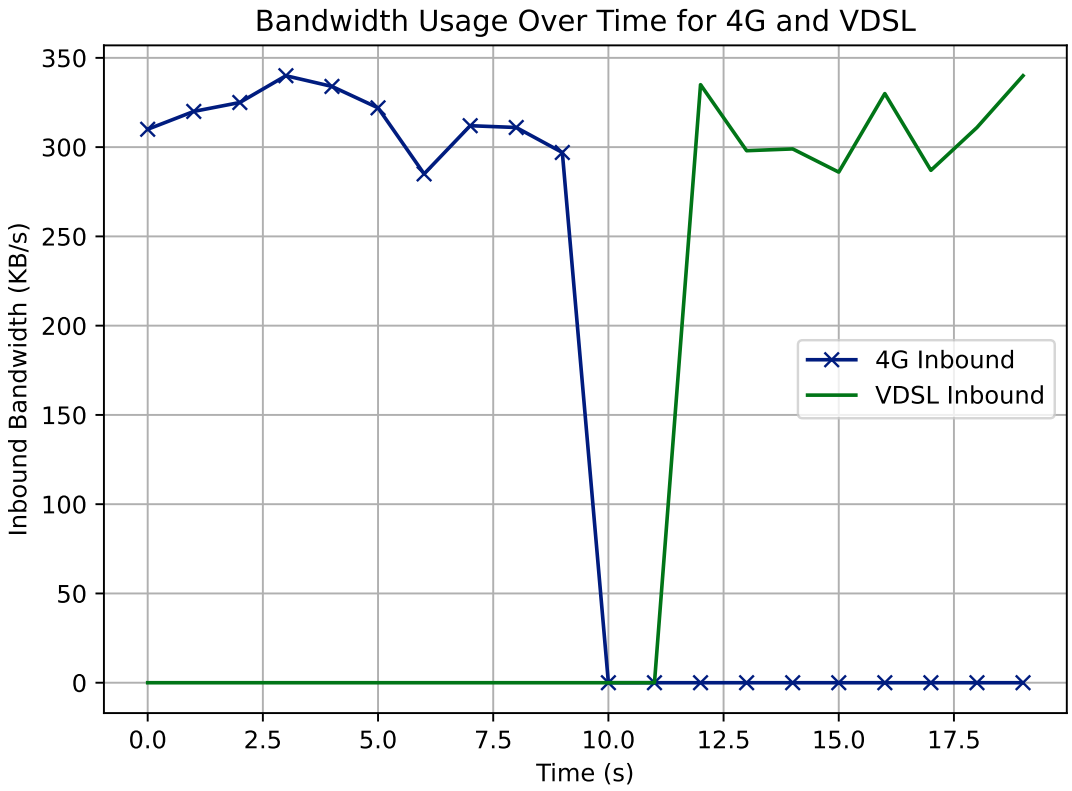

Figure 7.9: Backup mode when accessing a Remote Desktop

Chapter 8

Conclusion

This thesis demonstrates the use of the multipath TCP protocol in a Windows operating system. The measurements obtained have highlighted that using multipath TCP on Windows was possible using multiple network interfaces in aggregation or backup mode and in various contexts such as File Downloading, Live Streaming, Remote Desktop Accessing, and Video Streaming. We proved that activating multi-path TCP was possible without external hardware , which is a clear advantage.

In chapter 2, we presented the existing possibilities for using multi-path TCP in an operating system that doesn't support it by default, such as using a virtual machine where a proxy runs on it and also using an aggregator box that will do the trick of converting the TCP traffic into a MPTCP traffic.

In chapter 3, we presented some theoretical background about the objective of the multi-path TCP protocol, how the protocol works to use multiple bandwidths. We also have presented how WSL works, its architecture, the advantage of WSL1 and WSL2, and why we chose WSL2 over WSL1 to enable MPTCP on Windows.

Then, in the chapter 4, we presented the application that enables the activation and use of the MPTCP protocol in WSL, as well as the implementation choices and difficulties encountered during its implementation.

The 5 chapter discusses the theoretical aspects of proxies and the main difference between each proxy type, such as HTTP or socks. We discussed transparent proxy and why it is crucial in our use case. Finally, we discussed proxy chaining and how we can use it to use MPTCP on unsupported remote servers.

Next, in chapter 6, we discussed how we enabled the use of MPTCP directly on Windows and the implementation choices and difficulties encountered during

the application's implementation.

Finally, in chapter 7 we evaluated the performance of MPTCP, such as the latency and bandwidth in various applications such as downloading files, video streaming, Live streaming, and Remote Desktop Accessing. In each of these scenarios, we tested MPTCP on aggregation mode, where all subflows are used, and a backup mode, where one subflow serves as a backup in case the other fails.

Through the development of this application, we hope that users will be able to benefit from the protocol's advantages and that Windows operating system developers will be encouraged to integrate multipath TCP directly in Windows or on Windows Subsystem for Linux.

Bibliography

[1] dhclient - linux manual page. https://linux.die.net/man/8/dhclient.

[2] Hybrid Access Gateway. https://www.tessares.net/hybrid-access-gateway-software/.

[3] OpenMPTCProuter. https://www.openmptcprouter.com/.

[4] Tessares. https://www.tessares.net.

[5] Tessares Customer Premise Equipment. https://www.tessares.net/cpe-software/.

[6] WSLAttachSwitch. https://github.com/dantmnf/WSLAttachSwitch.

[7] V.R. Apolinário. *Learning Hyper-V*. Professional expertise distilled. Packt Publishing, 2015.

[8] darkk. redsocks. https://github.com/darkk/redsocks.

[9] Raghunath Deshpande. Tcp proxies chaining: Performance implications.

[10] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, August 2022.

[11] Alan Ford, Costin Raiciu, Mark J. Handley, and Olivier Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[12] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684, March 2020.

[13] Jiangping Han, Kaiping Xue, Wenjia Wei, Yitao Xing, Jianqing Liu, and Peilin Hong. Transparent multipath: Using double mptcp proxies to enhance transport performance for traditional tcp. *IEEE Network*, (5), 2021.

[14] Seth Kenlon. An introduction to udev. https://opensource.com/article/18/11/udev.

[15] Marcus D. Leech. SOCKS Protocol Version 5. RFC 1928, March 1996.

[16] Linode. Guides - network configuration using ifupdown. https://www.linode.com/docs/products/compute/compute-instances/guides/ifupdown/.

[17] NetworkManager. Networkmanager documentation. https://networkmanager.dev/docs/. Accessed: 2024-06-02.

[18] Christoph Paasch, Sebastien Barre, et al. Multipath tcp in the linux kernel. https://www.multipath-tcp.org, 2012.

[19] PowerDNS. Powerdns tproxy documentation. https://powerdns.org/tproxydoc/tproxy.md.html.

[20] P. Singh. *Learn Windows Subsystem for Linux: A Practical Guide for Developers and IT Professionals*. Apress, 2020.

[21] StatCounter. Desktop operating system market share worldwide. https://gs.statcounter.com/os-market-share/desktop/worldwide.

[22] Martin Sysel and Ondřej Doležal. An educational http proxy server. *Procedia Engineering*, 2014.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**