Computer Science Tripos

Part II Project Proposal Coversheet

Please fill in Part 1 of this form and attach it to the front of your Project Proposal.

Part 1

Name:	Jake Hillion	CRSID:	jsh77
College:	Queens'	Overseers: (Initials)	AWM & AV
Title of Project:	A Multi-Path Bidirectional Layer 3 Proxy		
Date of submission:		Will Human Participants be used?	No
Project Originator:	Jake Hillion	Signature:	--
Project Supervisor:	Mike Dodson	Signature:	--
Directors of Studies:	Neil Lawrence	Signature:	--
Special Resource Sponsor:		Signature:	--
Special Resource Sponsor:		Signature:	--

Above signatures to be obtained by the Student

Part 2

Overseer Signature 1: --

Overseer Signature 2: --

Overseers signatures to be obtained by Student Administration.

Overseers Notes:

Part 3

SA Date Received:		SA Signature Approved:	

Introduction and Description of the Work

This project attempts to combine multiple heterogeneous network connections into a single virtual connection, which has both the combined speed and the maximum resilience of the original connections. This will be achieved by inserting a Local Portal and a Remote Portal into the network path, as shown in Figure 1. While there are existing solutions that combine multiple connections, they prioritise one of resilience or speed over the other; this project will attempt to show that this trade-off can be avoided.

The speed focus of this software is achieved by providing a single virtual connection which aggregates the speed of the individual connections. As this single connection is all that's made visible to the client, all applications and protocols can benefit from the speed benefits, as they require no knowledge of how their packets are being split. As an example, a live video stream that only uses one flow will be able to use the full capacity of the virtual connection.

The resilience focus provides similar benefits, in that the virtual connection conceals the failing of any individual network connections from the client and applications. This again means that applications and protocols not built to handle a network failover can benefit from the resilience provided by this solution. An example is a SIP call continuing without a redial.

This system is useful in areas where multiple low bandwidth connections are available, but not a single higher bandwidth connection. This is often the case in rural areas in the UK. It will also be useful in areas with diverse connections of varying reliability, such as a home with both DSL and wireless connections, which may become more common with the advent of 5G and LEO systems such as Starlink. The lack of requirement for vendor support allows for this mixture of connections to be supported.

Some existing attempts to solve these problems, and the shortfalls of each solution, are summarized below:

- Failover: All existing flows must be restarted when failover occurs. There is no speed benefit over having a single connection.

- Session Based Load Balancing: All flows on a failed connection must be restarted. Speed benefit varies between applications, but is excellent in ideal circumstances. This solution is less effective when parameters of the connections vary with time, as with wireless connections. Further, advanced policies can be required on an application level to achieve the best speed.

- Application Support: Many modern protocols that are designed with mobile devices in mind can already handle IP changes (e.g. switching from WiFi to 4G). This allows these applications to handle situations such as Failover (above), as they treat it like any other network change. The downside of requiring application support is older protocols, such as SIP, for which resilience needs to be gained at a higher level.

- MultiPath TCP: MPTCP works best with multiple interfaces on each device that is using it, e.g. a 4G and WiFi connection on a mobile device. This is due to a device on a NAT with access to two WAN connections having no direct knowledge of this. It also requires support on both ends, which isn't common yet (MPTCP is not yet mainlined in the Linux kernel). Further, many modern applications are moving away from TCP in favour of lighter UDP protocols, which wouldn't benefit from MPTCP support.

- OpenVPN over MultiPath TCP: This allows both non-TCP based protocols, and clients that don't support MPTCP to benefit (if it's implemented network wide). Head of line blocking becomes more of an issue when passing multiple entirely different applications over a VPN, as any application can block any other. OpenVPN also adds a lot of unnecessary overhead if a network wide VPN would not otherwise be used.

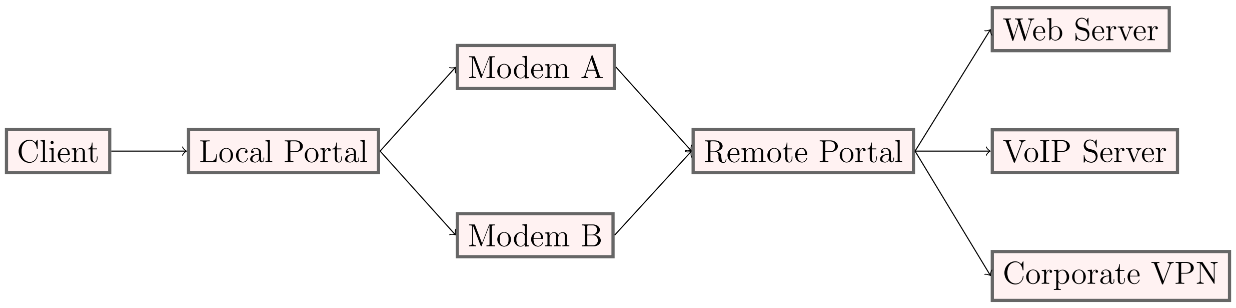

Figure 1: A network applying this proxy

By providing congestion control over each interface and therefore being able to share packets without bias between connections, this project should provide a superior solution for load balancing across heterogeneous and volatile network connections. An example of a client using this is shown in Figure 1. This solution is highly flexible, allowing the client to be a NAT Router with more devices behind it, or the flows from the Local Portal to the Remote Portal being tunnelled over a VPN.

Starting Point

I have spent some time looking into the shortfalls and benefits of the available methods for combining multiple Internet connections. The Part IB course *Computer Networking* has provided the background information for this project. I have significant experience with Go, though none with lower level networking. I have no experience with Rust, and my C++ experience is limited to the Part IB course *Programming in C and C++*.

While I am not aware of any existing software that accomplishes the task that I propose, Wireguard performs a similar task of tunnelling between a local and remote node, has a well regarded interface, and is a well structured project, providing both inspiration and an initial model for the structure of my project.

Substance and Structure of the Project

The system will involve load balancing multiple congestion controlled flows between the Local Portal and the Remote Portal. The Local Portal will receive packets from the client, and use load balancing and congestion control algorithms to send individual packets along one of the multiple available connections to the Remote Portal, which will extract the original packets and forward them along a high bandwidth connection to the wider network.

To achieve this congestion control, I will initially use TCP flows, which include congestion control. However, TCP also provides other guarantees, which will not benefit this task. For this reason, the application should be structured in such a way that it can support alternative protocols to TCP. An improved alternative is using UDP datagrams with a custom congestion control protocol, that only guarantees congestion control as opposed to packet delivery. Another alternative solution would be a custom IP packet with modified source and destination addresses and a custom preamble. Having a variety of techniques available would be very useful, as each of these has less overhead than the last, while also being less likely to work with more complicated network setups.

When the Local Portal has a packet it wishes to send outbound, it will place the packet and some additional security data in a queue. The multiple congestion controlled links will each be consuming from this queue when they are not congested. This will cause greedy load balancing, where each connection takes all that it can get from the packet queue. As congestion control algorithms adapt

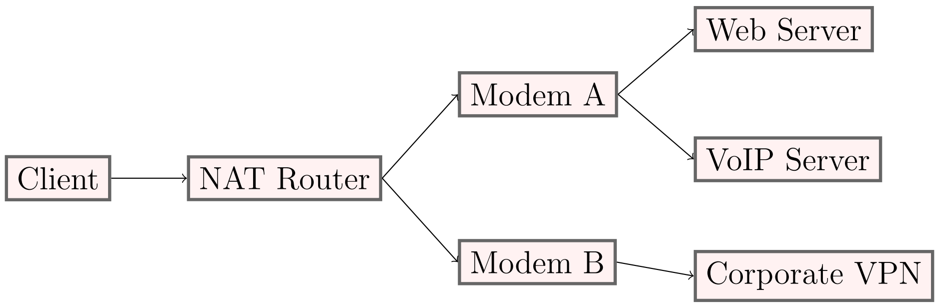

Figure 2: A network with a NAT Router and two modems

to the present network conditions, this load balancing will alter the balance between links as the capacity of each link changes.

Security is an important consideration in this project. Creating a multipath connection and proxies in general can create additional attack vectors, so I will perform a review of some existing security literature for each of these. However, as the tunnel created here transports entire IP packets, any security added by the application or transport layer will be maintained by my solution.

Examples are provided showing the path of a packet with standard session based load balancing, and with this solution applied:

Session Based Load Balancing

A sample network is provided in Figure 2.

1. NAT Router receives the packet from the client.
2. NAT Router uses packet details and Layer 4 knowledge in an attempt to find an established connection. If there is an established connection, the NAT Router allocates this packet to that WAN interface. Else, it selects one using a defined load balancing algorithm.
3. NAT Router masquerades the source IP of the packet as that of the selected WAN interface.
4. NAT Router dispatches the packet via the chosen WAN interface.
5. Destination server receives the packet.

This Solution

A sample network is provided in Figure 1.

1. Local Portal receives the packet from the client.
2. Local Portal wraps the packet with additional information.
3. Local Portal sends the wrapped packet along whichever connection has available capacity.
4. Wrapped packet travels across the Internet to the Remote Portal.
5. Remote Portal receives the packet.
6. Remote Portal dispatches the unwrapped packet via its high speed WAN interface.
7. Destination receives the packet.

Success Criteria

1. Demonstrate that a flow can be maintained over two connections of equal bandwidth with this solution if one of the connections becomes unavailable.

2. Any and all performance gains stated below should function bidirectionally (inbound/outbound to/from the client).

3. Allow the network client behind the main client to treat its IP address on the link to the Local Portal as the IP of the Remote Portal.

4. Provide security that is no worse than not using this solution at all.

5. Demonstrate that more bandwidth is available over two connections of equal bandwidth with this solution than is available over one connection without.

Extended Goals

1. Demonstrate that more bandwidth is available over two connections of unequal bandwidth than is available over two connections of equal bandwidth, where this bandwidth is the minimum of the unequal connections.

2. Demonstrate that more bandwidth is available over four connections of equal bandwidth than is available over three connections of equal bandwidth.

3. Demonstrate that if the bandwidth of one of two connections increases/decreases, the bandwidth available adapts accordingly.

4. Demonstrate that if one of two connections is lost and then regained, the bandwidth available reaches the levels of before the connection was lost.

5. My initial design requires the Remote Portal to have two interfaces: one for communicating with the Local Portal, and one for communicating with the wider network. This criteria is achieved by supporting both of these actions over one interface.

6. Support a metric value for connections, such that connections with higher metrics are only used for load balancing if no connection with a lower metric is available.

Stretch Goals

1. Provide full support for both IPv4 and IPv6. This includes reaching the Remote Portal over IPv6 but proxying IPv4 packets, and vice versa.

2. Provide a UDP based solution of tunnelling the IP packets which exceeds the performance of the TCP solution in the above bandwidth tests.

3. Provide an IP based solution of forwarding the IP packets which exceeds the performance of the UDP solution in the above bandwidth tests.

Although these tests will be performed predominantly on virtual hardware, I will endeavour to replicate some of them in a non-virtual environment, though this will not be a part of the success criteria.

Timetable and Milestones

12/10/2020 - 1/11/2020 (Weeks 1-3)

Study Go, Rust and C++'s abilities to read all packets from an interface and place them into some form of concurrent queue. Research the positives and negatives of each language's SPMC and MPSC queues.

Milestone: Example programs in each language that read all packets from a specific interface and place them into a queue, or a reason why this isn't feasible. A decision of which language to use for the rest of the project, based on these code segments and the status of SPMC queues in the language.

02/11/2020 - 15/11/2020 (Weeks 4-5)

Set up the infrastructure to effectively test any produced work from this point onwards.

Milestone: A virtual router acting as a virtual Internet for these tests. 3 standard VMs below this level for each: the Local Portal, the Remote Portal and a speed test server to host iPerf3. Behind the Local Portal should be another virtual machine, acting as the client to test the speed from. Backups of this setup should also have been made.

16/11/2020 - 29/11/2020 (Weeks 6-7)

This section should focus on the security of the application. This would include the ability for someone to maliciously use a Remote Portal to perform a DoS attack. Draft the introduction chapter.

Milestone: An analysis of how the security of this solution compares, both with other multipath solutions and a network without any multipath solution applied. A drafted introduction chapter.

30/11/2020 - 20/12/2020 (Weeks 8-10)

Implementation of the transport aspect of the Local Portal and Remote Portal. The first data structure for transport should also be created. This does not include the load sharing between connections - it is for a single connection. To enable testing, this will also require the setup of configuration options for each side. At this stage, it would be reasonable for the Remote Portal to require two different IPs - one for server communication, and one as the public IP of the Local Router. The initial implementation should use TCP, but if time is available, UDP with a custom datagram should be explored for reduced overhead.

Milestone: A piece of software that can act either as the Local Portal or Remote Portal based on configuration. Any IP packets sent to the Local Portal should emerge from the Remote Portal.

21/12/2020 - 10/01/2021 (Weeks 11-13)

Create mock connections for tests that support variable speeds, a list of packet numbers to lose and a number of packets to stop handling packets after. Finalise the introduction chapter. Produce the first draft of the preparation chapter.

Milestone: Mock connections and tests for the existing single transport. A finalised introduction chapter. A draft of the preparation chapter.

11/01/2021 - 07/02/2021 (Weeks 14-17)

Implement the load balancing between multiple connections for both servers. At this point, connection losses should be tested too. The progress report is due soon after this work segment, so that should be completed in here.

Milestone: The Local Portal and Remote Portal are capable of balancing load between multiple connections. They can also suffer a network failure of all but one connection with minimal packet loss. The progress report should be prepared.

08/02/2021 - 21/02/2021 (Weeks 18-19)

Finalise the drafted preparation chapter. Draft the implementation chapter. Produce a non-exhaustive list of graphs and tests that should be included in the evaluation section.

Milestone: Completed preparation chapter. Drafted implementation chapter. A plan of data to gather to back up the evaluation section.

22/02/2021 - 21/03/2021 (Weeks 20-23)

Finalise the implementation chapter. Gather the data required for graphs. Draft the evaluation chapter. Draft the conclusions chapter.

Milestone: Finalised implementation chapter. Benchmarks and graphs for non-extended success criteria complete and added. First complete dissertation draft handed to DoS and supervisor for feedback.

22/03/2021 - 25/04/2021 (Weeks 24-28)

Flexible time: divide between re-drafting dissertation and adding additional extended success criteria features, with priority given to re-drafting the dissertation.

Milestone: A finished dissertation and any extended success criteria that have been completed.

26/04/2021 - 09/05/2021 (Weeks 29-30)

New additions freeze. Nothing new should be added to either the dissertation or code at this point.

Milestone: Bug fixes and polishing.

10/05/2021 - 14/05/2021 (Week 31)

The project should already be submitted a week clear of the deadline, so this week has no planned activity.

Resources Required

- Personal Computer (AMD R9 3950X, 32GB RAM)
- Personal Laptop (AMD i7-8550U, 16GB RAM)

Used for development without requiring the lab. Testing this application will require extended capabilities, which would not be readily available on shared systems.

- Virtualisation Server (2x Intel Xeon X5667, 12GB RAM)
- Backup Virtualisation Server (2x Intel Xeon X5570, 48GB RAM)

A virtualisation server allows controlled testing of the application, without any packets leaving the physical interfaces of the server.

I accept full responsibility for the above 4 machines and I have made contingency plans to protect myself against hardware and/or software failure. All resources will be backed up according to the 3-2-1 rule. This would allow me to migrate development and/or testing to the cloud if needed.

Go(Lang) code written will use a version later than that available on the MCS, as the version currently on the MCS (1.10) does not support Go Modules. Rust is not available on the MCS at the time of writing. This can be managed by using personal machines or cloud machines accessed via the MCS.