High-availability Internet Gateway for Uptime-Sensitive Medical Telemetry Applications

Isaac True*, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 151231A
Swinburne University of Technology
Melbourne, Australia
isaac@is.having.coffee, garmitage@swin.edu.au

***Abstract*—For some devices and services, a consistently reliable connection to the internet is crucial; a failure to report to an internet service could result in significant financial or property damage or loss of life. In this document we propose a solution through the development and testing of a "High-availability Internet Gateway Device" (HaIG) which can be installed into a network and utilise multiple redundant internet connections in order to guarantee uptime for a secure tunnel for medical devices. Three potential solutions are identified and prototyped: Multipath TCP (MPTCP), Layer 2 Bonding (L2B), and SCTP. MPTCP and L2B were found to be less suitable than SCTP at providing a reliable, high-availability fail-over solution. We were able to successfully integrate the SCTP method with a consumer networking device to provide a service which can be used in remote cardiac monitoring to provide reliability and performance guarantees using redundant internet connections. The effectiveness of this method was proven using network analysis tools and a controlled testbed.**

I. Introduction

THE aim of this project was to research possible software and protocol solutions for an in-home High-availability Internet Gateway (HaIG) device based on existing consumer hardware. The purpose of this gateway was to provide networked clients with an always-on, highly reliable connection to the internet using several redundant internet uplinks.

Typically, home gateways will only provide an internet connection via one uplink, for example an ADSL2+ modem, with a minority of devices providing a fall-back to 3G when the main connection fails. For these devices, it can typically take up to several minutes for it to determine that its main connection has completely failed and that it should switch over to its fall-back.

The goal of this research report was to ascertain the best method of redundancy fail-over using existing technologies that best fit the criteria laid down by the use case. Specifically, we focused on the testing and implementation of the protocol in a medical telemetry setting, where high response times are an absolute requirement. Emergency alarms generated by medical devices require immediate attention and response, and the failure of the internet connection through which these devices report their situation could result in catastrophic damage or loss of life.

II. Literature Review

We factored down the research aims for this project into three key questions:

1) What are the network requirements of a real-time remote cardiac monitoring device?
2) What existing technologies, protocols, and research are available, which are free to use and free of charge, that can provide or help to implement the desired service and can be integrated into a single functional unit?
3) What performance metrics can be used to accurately determine the reliability, performance, and failure response time of such a gateway in different testing scenarios?

Question 3 was critical to the development and final analysis of the device, as it must be determined whether this internet gateway device can actually be beneficial and improve reliability, and whether it is worth the cost and performance overhead of maintaining and polling the state and quality of multiple internet connections.

A. Medical Telemetry

Reliability is key for networked medical devices: "Packet losses during the transmission of medical information may have disastrous impacts on a patient's

*This work was originally developed as part of the author's BEng Final Year Research Project, 2015

diagnosis". [1] It was therefore vital that ensuring the reliability of the platform and service, especially when faced with a packet loss scenario, was a paramount focus of this project.

The bandwidth required for transmitting medical telemetry over an internet connection needed to be taken into account, especially when operating over potentially costly mobile internet connections. Table I shows the approximate bandwidth requirements for various types of raw medical telemetry data for a patient in bits per second.

Table I: Bandwidth requirements for monitoring services [2]

Biomedical Measurement	Rate (bps)
ECG	15000
Heart sound	120000
Heart rate	600
EEG	4200
EMG	600000
Respiratory Rate	800
Temperature of body	80

B. Network Redundancy

1) Link Aggregation: This is the process of linking physical network interfaces together at the software level, also called teaming. There are many different approaches to this, mostly proprietary and manufacturer-dependant, however most focus around splitting network traffic at Layer 2 of the OSI Model across the multiple interfaces in order to utilise the aggregate bandwidth of all the links.

Some implementations of link aggregation, such as the Bonding driver in the Linux kernel, offer high configurability and flexibility, and offer multiple functional modes depending on the requirements. In the case of the Bonding driver, it offers active and passive forms of load balancing across the interfaces, mirroring of data, backup links, and 802.3ad compatibility modes. [3]

2) MPTCP: Multipath TCP (MPTCP) is a new experimental standard ratified by the IETF in 2013, providing a higher level implementation of network redundancy, in that is based around OSI Layer 4 network communication. "MPTCP operates at the transport layer and aims to be transparent to both higher and lower layers. It is a set of additional features on top of standard TCP ... MPTCP is designed to be usable by legacy applications with no change" [4] Additionally, there is a project aimed at bringing MPTCP support to Linux-based systems through a patch-set for the Linux kernel and user-space tools, which will greatly aid in development of the internet gateway device. [5]

3) SCTP: The Stream Control Transmission Protocol (SCTP) is a session-oriented alternative transport protocol, operating at the same level as TCP and UDP. It was created with the intent of adding new capabilities to IP-based transport protocols, such as multi-homing (utilising multiple end-points) and multi-streaming (partitioning data into multiple independent streams in order to increase fault tolerance). The multi-homing capabilities are of particular interest to the project. This protocol is included in the Linux kernel. [7]

C. Network Performance Metrics

There are many network analysis tools designed for various scenarios and testing criteria. TCP Experiment Automation Controlled Using Python (TEACUP) is a set of software for the management of a suite of such tools for use in automated testbeds. "Based on a configuration file TEACUP can perform a series of experiments with different traffic mixes, different bottleneck configurations (such as bandwidths, queue mechanisms), different emulated network delays and/or loss rates, and different host settings (e.g. used TCP congestion control algorithm). For each experiment TEACUP automatically collects relevant information that allows analysing TCP behaviour, such as tcpdump files and TCP stack information." [8]

III. Methodology

The focus of the evaluation of the redundancy solutions was testing their ability to maintain an OpenVPN connection between the gateway device (simulating the user's 'home router') and an endpoint (simulating the 'hospital') using OpenVPN v2.3.8. The OpenVPN connection maintained a layer 3 routed IP network between the end-point and router, through which the traffic generated by `iperf` under TEACUP v1.0 was routed. This involved setting static routes on the gateway and endpoint for routing the traffic, as well as configuring OpenVPN for bidirectional external routing. For L2B and SCTP-based setups, UDP was used for the OpenVPN transport layer, while TCP (in order to facilitate the automatic use of MPTCP) was used for the MPTCP setup.

Traffic was generated on a 'client device' (hostname: `testhost1`), sitting on the home user network, and sent to the 'server' (`testhost2`), sitting on the 'hospital' network behind the hospital endpoint (`endpoint-router`), via the OpenWRT device (`openwrt`). Traffic from `testhost1` was routed to `openwrt` via a third host,

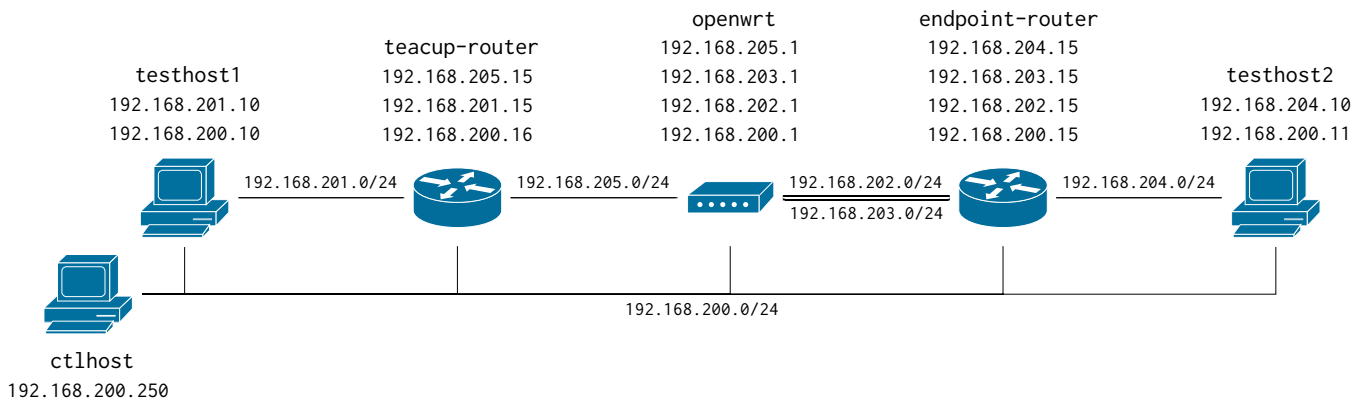


Figure 1: Test-bed network layout [6]

`teacup-router`. `ctlhost` was used as the TEACUP 'control host', which orchestrates the TEACUP tests over the management network `192.168.200.0/24`.

TEACUP provided a solid platform and testing template for use with the evaluation of the link redundancy solutions. The focal testing tool used by TEACUP during testing was `iperf`.

TEACUP is however (at the time of writing) incompatible with the `busybox` utilities used by OpenWRT in place of the standard GNU `coreutils` found on a typical Linux distribution. Due to this, the OpenWRT-powered router was not managed by TEACUP; instead, `teacup-router` was used as a 'shim' router between `testhost1` and the OpenWRT device. This simply routed traffic between the `testhost1` network and a dedicated interstitial network between it and the OpenWRT device.

A diagram showing the network layout and each host's assigned IP addresses is shown in Figure 1. The `192.168.202.0/24` network emulated the 'main' (ADSL, Cable, etc.) internet connection between `openwrt` and `endpoint-router`, while the `192.168.203.0/24` network emulated a 'fall-back' 3G connection. In order to emulate the 3G connection, the `tc` tool was used to introduce a 200ms delay and 3.5Mbps bandwidth limit to the fall-back network on `openwrt`. This also served the purpose of identifying when traffic was being routed through the fall-back network instead of the main when analysing the results, as traffic flowing through the fall-back was at a much lower throughput and higher latency. As identified in the literature review, 1000kbps is enough bandwidth to support any combination of the likely biomedical measurement data, including OpenVPN and other overheads. [9]

The two hosts and routers, `testhost1`, `testhost2`, `endpoint-router`, and `teacup-router`, along with `ctlhost`, were all virtual machines running on a desktop computer, virtualised using QEMU v2.4.1. The host operating system on the desktop computer was an up-to-date at time of writing Arch Linux installation. The desktop was physically connected to the `openwrt` host device using a single Ethernet cable. The six separate network segments were simulated using VLAN tagging – all traffic was trunked over the single connection. The VM hosts were connected to their respective networks using the bridging functionality of QEMU.

All five virtual machines were running Debian 8.2.0 and were configured to use the `virtio-net` drivers for network interfaces for maximal performance. The OpenWRT device firmware was built using code checked out from the official OpenWRT `git` repository, at commit `1f66f11`. [10]

As TEACUP uses the Web10g Linux kernel extensions for probing the network stack and gathering statistics when running under Linux, and as Debian does not include these extensions by default, a custom kernel was required for these virtual machines. This was achieved by cloning the Web10g fork of the Linux kernel, and compiling it with the standard kernel configuration from Debian 8.2.0 with Web10g additionally enabled. This kernel was then used together with the "Direct kernel boot" feature of QEMU, booting the VM with the kernel directly, bypassing the OS bootloader. This simplified the setup by allowing a single customised kernel image file for all VM's to be saved on the host machine and loaded from there. The source code for the Web10g was downloaded from the project's GitHub page, at commit `444b451`, corresponding to Linux 3.18.0. [11] [12]

A. L2B

The design of the L2B approach was to create a layer 2 tunnel interface using OpenVPN on each end of each internet connection, and to bond them together using the Linux bonding driver. A standard OpenVPN tunnel was to then be created using this new bonded interface, and traffic routed through that. Figure 2 shows the testbed setup.

During preliminary testing of the L2B method, the setup was found to be unstable and prone to failures. Functionally, it worked and was able to facilitate the transmission of data from `testhost1` to `testhost2`, and provided fail-over when the main connection went down. However, various components would sporadically fail, halting the flow of data across the connection and requiring the entire stack to be restarted. The exact cause of this is unknown; the Linux bonding driver was constructed with physical link interfaces in mind, not virtual interfaces such as these layer 2 tunnels, and as such, could have been put into an invalid or undefined state due to some lacking characteristic or functionality of these tunnel interfaces.

As a result of this instability and in the interest of time, the L2B approach was deemed unsatisfactory, and no further testing was carried out.

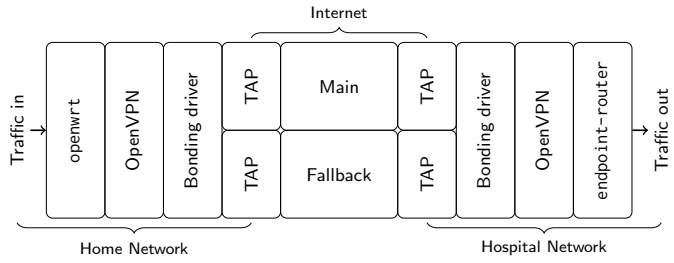


Figure 2: L2B testbed setup

B. MPTCP

As MPTCP is a transparent replacement for TCP, to be used it needed to be enabled in the kernel configuration, and OpenVPN configured to use TCP as its transport layer protocol, in order to function. The testbed setup that we proposed for MPTCP is shown in Figure 3.

However, as MPTCP is not currently in the mainline Linux kernel, a fork including the necessary changes needed to be used. An already established MPTCP-enabled fork of OpenWRT exists, which was intended to be used to carry out the testing. However, due to the MPTCP fork lagging behind the mainline OpenWRT distribution version, several incompatibilities with required software versions were discovered. This introduced errors in the OpenWRT compilation process, which proved to be too pervasive and difficult to fix in the time available.

Furthermore, subsequent analysis of the protocol found that MPTCP implementations are likely to split traffic across all paths to maximise throughput, which does not coincide with the goal of the project. This could add considerable expense for the user, as traffic could be constantly sent using the more costly 3G path.

Thus, in light of the aforementioned observations, the decision was made to abort the MPTCP testing and focus on other methods.

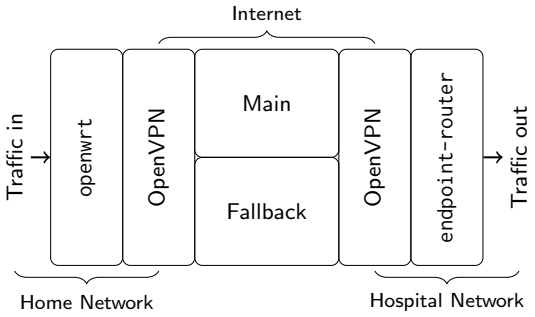


Figure 3: MPTCP testbed setup

C. SCTP

As OpenVPN does not natively support using SCTP as its transport-layer protocol, OpenVPN UDP traffic needed to be tunnelled using a socket relay application – `socat`. One instance of `socat` was configured to listen for UDP connections from the OpenVPN client on `openwrt` and forward them to another instance of `socat` on `endpoint-router`. This second instance was configured to listen for SCTP connections and forward the tunnelled UDP packets to the OpenWRT server on the same device. SCTP automatically makes use of alternative available paths. Figure 4 shows the testbed setup used in the experiment. Listings 1 and 2 show the commands used to initialise both `socat` instances.

The Linux SCTP implementation makes several tunable parameters available to the user, and can be modified using operations on the files under the `/proc/sys/net/sctp` directory. Of particular interest to the project were the `rto_initial`, `rto_min`, and `rto_max` parameters, which govern the initial, minimum, and maximum receive time-outs (RTO). Of these, `rto_min` had the most dramatic effect on the performance of the fail-over during preliminary testing. Thus, in order to further explore the efficacy of SCTP as a redundancy solution, further

tests were carried out observing the effects of setting the `rto_min` parameter. The values used (representing milliseconds) were: 50, 100, 1000 (default), and 3000.

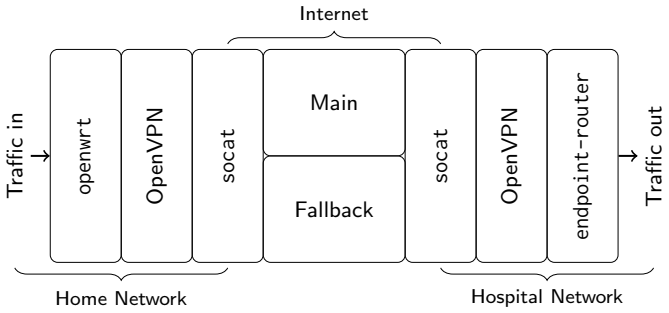


Figure 4: SCTP testbed setup using `socat`

Listing 1: `openwrt socat` initialisation

```
socat 'UDP-LISTEN:1194,fork' \
      'SCTP:192.168.202.15:8888'
```

Listing 2: `endpoint-router socat` initialisation

```
socat 'SCTP-LISTEN:8888,fork' \
      'UDP:localhost:1194'
```

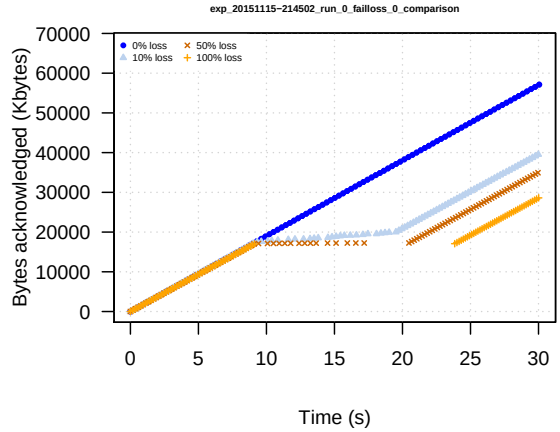
D. Testing

TEACUP was used to carry out 30 second long experiments using `iperf`. This generated TCP traffic from `testhost1` to `testhost2` for the duration of the tests. At approximately 10 seconds into the experiment, an artificial traffic loss rate was added to the 'main' internet connection of `openwrt` using `tc`. After 5 seconds, this was set back to the default of 0% traffic loss, simulating an internet connection failure. Four traffic loss rates were used to demonstrate the showcase the redundancy method's behaviour during different degrees of connection failure: 0% (no failure), 10% (small failure), 50% (large failure), and 100% (total failure). The 0% loss rate was used as a zero-point reading for comparing the results of the higher loss rate scenarios against normal traffic flow.

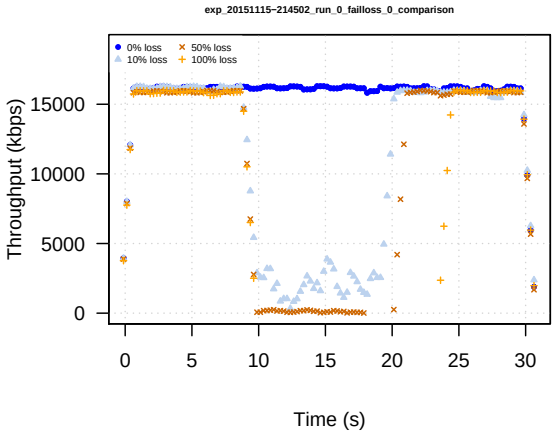
IV. Results

A baseline test of the traffic over the main internet connection was carried out in order to observe the normal behaviour of the test-bed without any redundancy methods. The results for this experiment can be seen in Figure 5. These figures show the TCP performance using the ACK sequence and associated throughput.

Figure 6 shows the TCP performance of the SCTP fail-over redundancy method. Figure 7 shows the ACK sequences during failure comparing the effects of the `rto_min` parameters at 50% and 100% loss rates.



(a) TCP ACK sequence



(b) Throughput

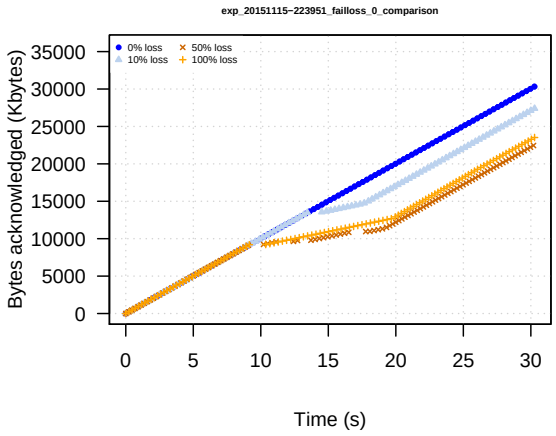
Figure 5: Baseline results over OpenVPN connection

All result plots were generated using TEACUP's built-in R-based plotting functions.

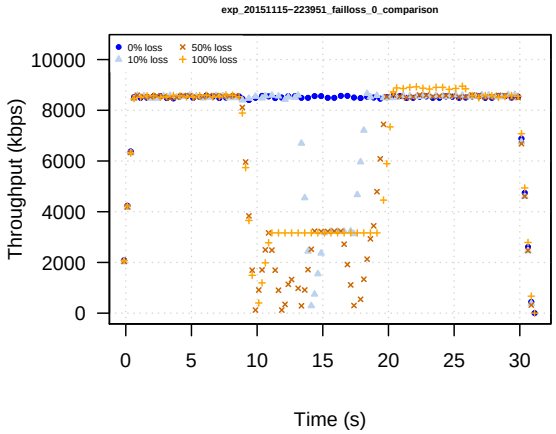
V. Discussion

The ACK sequence graphs are the most important to consider when analysing data derived from this experiment and considering the speed of fail-over and degree of reliability. They show the amount of bytes acknowledged as received by the hosts. This is a cumulative value, which shows the sum total for the test. As the two internet connections have very different bandwidth capabilities, it is easy to see when the device is transmitting over each; the rate of increase in the ACK sequence is far lower for the fall-back internet connection than that of the main.

The throughput graphs shown represent the data rate at which packets are received by `testhost2` over time.

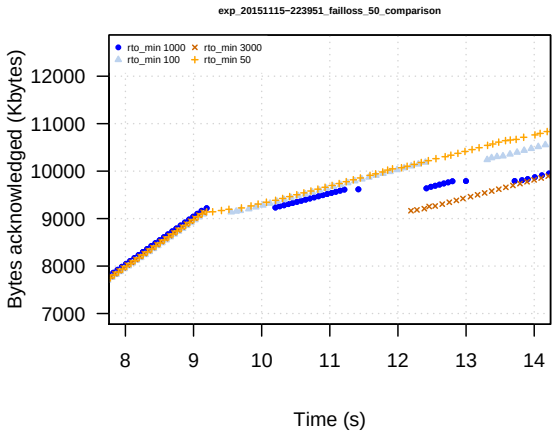


(a) TCP ACK sequence

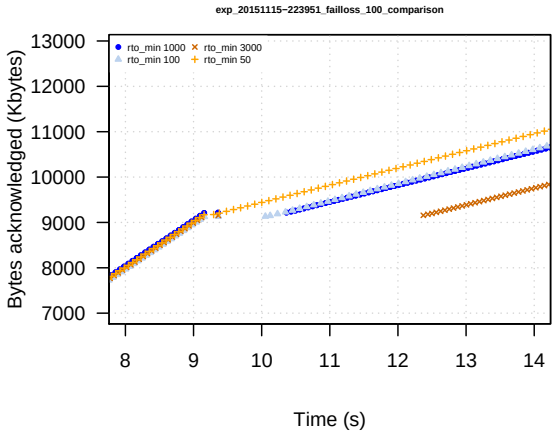


(b) Throughput

Figure 6: SCTP results over OpenVPN connection



(a) 50% Packet loss



(b) 100% Packet loss

Figure 7: TCP ACK sequence at start of failure with SCTP

Higher values represent the traffic being routed through the main internet connection, while zero values represent an interruption in traffic flow, or total loss in packets, and a lower value of approximately 3.5Mbps represents traffic being routed over the fall-back internet connection due to the bandwidth rate limit.

The baseline results are shown in Figure 5, with the TCP ACK sequence numbers, TCP RTT, and `iperf` throughput over time with a range of loss rates as the artificial link failure. As can be seen from the results, the introduction of the link failures just prior to the 10 second mark causes various degrees of response in the transmission continuity – no or limited traffic is received and/or acknowledged by `testhost2`.

As can be seen in the SCTP results presented in Figure 6, the overall bandwidth of the system was lower than that of the baseline system with VPN. This was due to the overhead involved with encapsulating the OpenVPN packets over the SCTP connection. However, the throughput was still high enough to satisfy the requirement of 18kbps for cardiac monitoring devices. This means that assuming the connection is able to provide the reliability guarantees, this is a viable solution.

Analysis of the results for SCTP proved that it is indeed able to function as a fail-over solution, providing a sustained connection over the duration of a failure. This is evident in the ACK sequence graph, where a mostly continuous increase in the ACK sequence numbers was apparent at all packet loss rates. This was corroborated with by the corresponding throughput graph, where data was visibly being sent and received.

From the results shown in Figure 7, it can be seen that setting the `rto_min` parameter to a lower value allowed a quicker fail-over to be achieved, while a higher value limited the response time. This had a direct impact on the

degree of reliability of the solution. The default `rto_min` value of 1000ms was effective when faced with a total connection failure (100% packet loss), however it was unreliable at lower loss rates due to flip-flopping between the main and fall-back connections. 100ms was most effective at both low and total packet loss rates, where it quickly failed-over to the fall-back connection and back again to the main when the failure period was over. 50ms was most effective at medium packet losses, as it had a very short fail-over time, and did not flip-flop like 100 and 1000ms. It was, however, ineffective at high failure rates, do to its apparent tendency to latch on to the fall-back path. Finally, 3000ms was not overly effective at any rate of packet loss – it switched between the main and fall-back paths correctly, although it was very slow to do so.

VI. Conclusion

Three solutions were identified which could potentially satisfy the demanding requirements needed by a remote health monitoring serivce: Multipath TCP (MPTCP), Layer 2 Bonding (L2B), and SCTP. Due to time constraints and development issues, both the MPTCP and L2B approaches were discontinued and testing only carried out on the SCTP method.

Through multiple tests performed on real-world consumer hardware using TEACUP and analysis of the resulting TCP performance metrics, SCTP proved to be an effective method of introducing connection fail-over to an OpenVPN connection. It successfully detected link failures, and re-routed traffic to the fall-back emulated 3G connection.

We have concluded that encapsulating traffic over an SCTP connection provides an effective, reliable, and sufficiently high data-rate solution for the demanding requirements of at-home medical devices which need to communicate over the internet to a remote server.

References

[1] D. D. Vergados, "Simulation and modeling bandwidth control in wireless healthcare information systems", *Simulation*, vol. 83, pp. 347–364, Apr. 2007.

[2] D. J. Vergados, D. D. Vergados, and I. Maglogiannis, "Applying wireless diffserv for qos provisioning in mobile emergency telemedicine", in *Presented as part of the Global Telecommunications Conference, 2006 (GLOBECOM '06)*, San Francisco, CA: IEEE, Nov. 2006, pp. 1–5, ISBN: 1-4244-0356-1.

[3] T. Davis *et al.* (Nov. 2009). Bonding, The Linux Foundation, [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/bonding.

[4] A. Ford *et al.*, "RFC 6824, TCP extensions for multipath operation with multiple addresses", Internet Engineering Task Force, Jan. 2013.

[5] C. Paasch, S. Baare, *et al.* (Apr. 2015). Multipath TCP in the Linux kernel, [Online]. Available: http://www.multipath-tcp.org/.

[6] (Oct. 2015). Icons for printed collateral, visio, video, and multi-media, Cisco, [Online]. Available: http://www.cisco.com/web/about/ac50/ac47/2.html.

[7] L. Ong and J. Yoakum, "RFC 3286, An introduction to the stream control transmission protocol (SCTP)", IETF, May 2002.

[8] S. Zander and G. Armitage, "Teacup v1.0 - a system for automated tcp testbed experiments", Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529A, Apr. 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf.

[9] (Nov. 2009). netem, The Linux Foundation, [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[10] (Nov. 2015). OpenWRT.org git repositories, OpenWRT, [Online]. Available: http://git.openwrt.org/?p=openwrt.git.

[11] (Nov. 2015). The Web10G project, Web10G, [Online]. Available: https://www.web10g.org/.

[12] (Nov. 2015). Development git for Web10g Project, Web10G, [Online]. Available: https://github.com/rapier1/web10g/tree/web10g.