TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Glossary

ADR Asymptotic Dispersion Rate. 27, 183

APNIC Asia Pacific Network Information Center. 7

Ark Archipelago measurement infrastructure. 23

ARPANET Advanced Research Projects Agency Network. 15

AS Autonomous System. 24

ASN autonomous system number. 128, 183

BGP Border Gateway Protocol. 23, 127

BSD Berkeley Software Distribution. 42

CSV comma separated values. 18, 85, 183

CTCP coded TCP. 119

DDOS distributed denial of service. 43

DHCP Dynamic Host Configuration Protocol. 42

DIMES distributed Internet measurement. 23

DMZ demilitarised zone. 42

DNS Domain Name System. 43

DNSSEC Domain Name System Security Extensions. 43

DOF degree of freedom. 118, 183

ECN explicit congestion notification. 115, 183

FEC forward error correction. 59, 136, 183

FIFO first in, first out. 56

GEO geostationary. 48, 94, 183

HSTCP High Speed TCP. 145

ICANN International Corporation for Assigned Names and Numbers. 7

ICMP Internet Control Message Protocol. 23, 183

IETF Internet Engineering Task Force. 22

IP Internet Protocol. 23

IPPM IP Performance Metrics. 22, 53, 183

ISIF Information Society Innovation Fund. 7

ISP Internet Service Providers. 6, 15

ITU-T International Telecommunication Union. 18, 58, 61

LEO low Earth orbiting. 48

MEO medium Earth orbiting. 48, 94, 183

MIB management information base. 21

MIT Massachusetts Institute of Technology. 6

MOS Mean Opinion Score. 5, 16, 61, 70, 87, 183

MTU maximum transmission unit. 23, 135, 183

NAT network address translation. 124, 183

NC network coding. 18, 117

NCE network capacity expansion. 112, 184

NRSC Network Resources Startup Center. 6

NTP Network Time Protocol. 43, 77

OSI Open System Interconnection. 17, 47, 183

OWAMP One-way Active Measurement Protocol. 22, 28

OWD one-way delays. 28

PDV packet delay variation. 52

PEP performance enhancing proxies. 112, 113, 183

PER packet error rates. 121

PESQ perceptual evaluation of speech quality. 58

PICISOC Pacific Internet Society. 7

PIP Pacific Internet Partners. 7

QoS Quality of Service. 24, 49

RIR Regional Internet Registry. 23

RTP Real-time Transport Protocol. 27

RTT round trip time. 25, 26, 86, 113, 183

SCTCP Stream Control Transport Protocol. 113, 183

SLOPS Self Loading Periodic Stream. 28, 183

SNMP Simple Network Measurement Protocol. 21

SPAN switched port analyser. 21

SPC South Pacific Secretariat for the Pacific Community. 7

TCI Telecom Cook Islands. 6

TCP Transmission Control Protocol. 3, 15, 20, 183

TCP/NC TCP over Network Coding. 3, 5, 12, 17, 18, 29, 59, 109, 111, 116, 117, 122, 124, 133, 141, 183

TCPeP TCP Performance-Enhancing proxy. 122

TTC Tuvalu Telecom Corporation. 6

TTL time to live. 3, 25, 183

TTMS Test Traffic Measurement Service. 23

TWAMP Two-way Active Measurement Protocol. 22

UDP User Datagram Protocol. 3

UoA University of Auckland. 126

UPS Uninterruptible Power Supply. 44

VoIP Voice over IP. 3, 15

VoTCP Voice over TCP. 12, 39, 107, 109, 183

WAN wide area network. 135, 183

CHAPTER 1

Introduction

The first inter-connected networks consisted of three different networks: Advanced Research Projects Agency Network (ARPANET), the Packet Radio Network, and the Atlantic Packet Satellite [2]. On November 22nd, 1977, tests were carried out using the TCP protocol to transmit data from a van located in San Francisco to nodes located at the University of Southern California. The success of this experiment was the beginning of the Internet.

From the three inter-connected networks, the Internet has evolved to become the world's largest and most complex network. It is estimated that there are about 800 million hosts connected to the Internet, providing connectivity to more than 2 billion users and the number of hosts grows significantly every year [3, 4].

However, the growth of the Internet forces ISP to regularly upgrade their network infrastructure to meet the high demand for bandwidth. This is costly and represents an on-going cost because the number of hosts increases continuously [3]. Moreover, the Internet is a best effort network and therefore routers process packets in the order they are received. When there is congestion, packets spend more time in router queues and this delayed the arrival time of packets. For *VoIP* and other real-time applications, timely arrival of information is essential to achieve smooth replay at the destination. However this is not always guaranteed, especially when links are congested.

In a congested link, VoIP applications encounter delays, but as long as the delay is relatively constant and small, VoIP applications can buffer to achieve smooth replay. However, when there are variable-length delays in the arrival time of packets (jitter), VoIP applications may discard overly delayed packets, and this may cause short breaks in voice replay [5].

The problem with jitter is more common in high latency and low bandwidth satellite connections because of the limited capacity to transmit large amounts of voice, video and data packets simultaneously. This is the case in particular in remote countries in the Pacific where satellite is the common medium of Internet connectivity.

This thesis makes the following contributions: Firstly, we develop an active measurement software, our `beacon` software, and install it in hosts in the Pacific, North America, Africa, Europe and Asia. These hosts regularly exchange synthesised UDP and TCP packets to perform two types of experiment: simulated VoIP and simulated file download. The beacon software is also capable of simulating other real-time and streaming applications. However because of the limited bandwidth in some of our research partners' networks and limited number of experiment time slots across our beacons network we restricted ourselves to the aforementioned experiments.

The VoIP experiment uses the UDP protocol to perform a simulated VoIP call between two nodes. In addition, we also use TCP to perform VoIP experiments between two nodes. Our file download experiment mimics a situation where a client uses the TCP protocol to repeatedly download a file from a server over an extended period of time. At the time of writing, we have archived our experiment data into a network of file servers and have accumulated an estimated 491 gigabytes of experimental data. This is four years worth of data collection.

Secondly, we use our data to determine path characteristics such as jitter, out-of-order arrivals, estimated *MOS*, latency, changes in the path of packets (with TTL values), and packet loss.

As a third contribution, we developed two methods to detect different causes of jitter. One method uses T-entropy [6] together with jitter, while the other method

uses the quantile function of the Gaussian distribution [7]. We use these methods to interpret our results. Lastly, we developed a web portal that we use to showcase our results and to publish our data for interested researchers.

In our paper on the measurement of packet train arrivals for high latency networks [8], we mentioned that links in countries in the Pacific are mostly impaired by high latency, jitter, and packet loss. In our studies in [9] we reflect on queue oscillation and point out its contribution to high packet loss simply because of the slow start mechanism of TCP.

As a fourth contribution, we attempt to improve TCP throughput over high latency links by investigating the tunneling of TCP connections over network coded UDP links. One outcome of this investigation is that we contributed design information and ideas on how to further improve the TCP/NC kernel module based on an existing TCP/NC library by Steinwurf ApS. Note that we did not contribute to the development of the TCP/NC algorithm.

The results of this thesis should be of interest to both ISPs and over the top providers such as the VoIP telephony industry or streaming services. For example, an offshore call centre provider might want to know about long term jitter and latency trends before committing to an investment in a particular location.

1.1 Overview

This thesis is structured as follows. In Chapter 2, we present a literature review on existing network measurement studies.

In Chapter 3, we unveil the motivation in the development of our active measurement tool (`beacon` software). In addition, we discuss the difference between active and passive network measurement methods.

In Chapter 4, we describe the functional, non-functional, and hardware requirements of our software. The functional requirements include the input parameters

expected from the user, the way our software processes packets, and the expected performance of our software. The non-functional requirements are utilities and operating system configurations that support the core functionality of our software. The hardware requirements are the minimum hardware components that we require from a physical host to support the operation of our software.

In Chapter 5, we provide background information on jitter. We use the Open System Interconnection (OSI) model to discuss delays in the travel time of packets and possible network events that contribute to jitter. We follow with a review of various methods for estimating jitter and compare these methods to our jitter estimator. The rest of this chapter is dedicated to available solutions for reducing the impact of jitter.

In Chapter 6, we review the use of jitter to estimate the quality of a network, and present our method of estimating voice quality. We begin with an introduction into complexity measures and their relation to entropy measures. Then we use previous work on entropy measure to introduce T-entropy. Next we describe the mapping of inter-arrival time of packets into strings, and how we compute T-entropy from strings. We then combine entropy with jitter to describe the quality of a channel between two nodes. The last section of this chapter discuss the use of jitter as one of metrics for the estimation of MOS with the International Telecommunication Union (ITU-T) E-model.

In Chapter 7, we outline the limitations of our software and the challenges we encountered in the implementation/deployment of our global network of active measurement hosts.

In Chapter 8, we give a brief history of the tools and methods we use to archive our data into comma separated values (CSV) files.

In Chapter 9, we give examples of the kind of result that we can extract from the data collected. For example, we plot a graph on packet loss and out-of-order arrivals to determine whether congestion (which causes packet loss) is a contributing factor to changes in the path of packets.

Chapter 10 focuses on TCP/NC. We begin with a review of existing solutions for high latency networks and the limitations of these solutions. Then we introduce network coding (NC) from theory to the kernel module application that was developed by Steinwurf, and outline the reason that TCP/NC is a more suitable solution for improving goodput. The last section of this chapter describes the design and deployment of the experimental version of the kernel module and also proposes a possible design for a commercial version of the software.

In Chapter 11, we discuss the challenges involved in the implementation of TCP/NC and present results from experiments conducted between nodes in New Zealand, Tuvalu, Cook Islands and Niue.

In the final chapter, Chapter 12, we conclude this thesis and proposes areas that may be of interest for future study.

CHAPTER 2

Literature Review

In this chapter, we describe the motivation for the development of our beacon software, and follow this up with discussions on network measurement methods.

One of the problems associated with the growth of the Internet is network congestion. As the number of hosts increases, the amount of traffic emitted by routers increases accordingly. As a result, router queues vary significantly in length and can at times fill up completely. This causes variations in the arrival time of packets (*jitter*). As packets continue to arrive at the queue, routers are forced to perform tail drop on the queue causing packet loss.

In the event of congestion, applications that uses TCP as a transport protocol experience degraded performance because the TCP protocol was not designed for high latency networks (see queue oscillation [10, 11, 12] on Chapter 11). Responding to packet loss and jitter, network measurement tools were developed to monitor networks and to provide appropriate solution for packet loss and jitter. In this chapter, we will discuss some of the active measurement tools and studies using them.

There are two methods to monitor a network; passive and active measurements [13]. The difference between the two methods is that passive measurement

records information from the flow of existing packets.

In contrast, active measurement introduces artificial traffic into the network for the purpose of monitoring the impact of router queue and routing behaviour on the transmission of packets.

2.1 Passive Measurement

Passive measurement is the process of monitoring the status of networks without modifying or introducing any traffic. The monitoring process may involve the use of network interface status, traffic load in routers, length of queues, and routing tables to assess the current state of a network [13].

There exist two models of passive measurement. Firstly, there is the pull model used in the Simple Network Measurement Protocol (SNMP) [14]. In this model, SNMP requires the installation of a hierarchical database known as the *management information base (MIB)* [15]. After the installation of the MIB in a server (SNMP manager), the MIB pulls metrics such as platform resource utilisation, network traffic, and error counts [16] from routers/switches and stores them as records.

In the second approach, passive measurement utilities such as *NTOP* [17] with *Nprobe* [18], or *Cisco netflow* [19] use the push model for monitoring traffic. In this model, a device known as *netflow/IPFIX* collector records information seen on a network interface, such as the IP address, network/application layer protocols, and inter-arrival time of packets.

There are two ways to install a collector. It can be activated on a Cisco switch/router with switched port analyser (SPAN) and related netflow commands [20], or it can be installed in a dedicated machine. A good example of such a method is the installation of *Nprobe* with various configurations of *IPFIX templates* [21] to intercept and classify traffic into characteristics such as the traffic utilisation of hosts, access attempts from external IP addresses etc.

The results from traffic classification are also exportable into analysis applications such as *NTOP* that display information in the form of graphs and tables. Despite the differences between the above models, passive measurement utilities monitor and capture information for reporting on events such as the current network load, network congestion, malicious activities, and packet errors. In this thesis, we use NTOP and Nprobe to gain satellite link utilisation information on Tuvalu (see Chapter 11).

2.2 Active Measurement

Active measurement generally involves the exchange of artificially generated traffic between hosts. During the exchange process, one of the host transmits traffic whereas the other host awaits the arrival of packets. Using the IP address to identify the transmitting host, a port number to identify the service, and a serial number to identify packets, the receiving node records information such as the amount of data, the receive time of packets, information from packet headers, and the number of packets that were received.

By processing this information, one can observe characteristics in networks such as bandwidth, link capacity, delay variation (jitter), packet loss, changes in path, and out-of-order arrivals of packets. Examples of active measurement utilities will be discussed in Section 2.4 of this chapter.

Our research was motivated by a number of standards. Firstly the standards on One-way Active Measurement Protocol (OWAMP) [22] and Two-way Active Measurement Protocol (TWAMP) [23] provided a step by step guide on unidirectional and bidirectional active measurement. The two standards were also adopted in the active measurement tool OWAMP, and we discuss the details of this tool in Section 2.4. Moreover, we use the definition in the IP Performance Metrics (IPPM) standard [24] to discuss jitter in Chapter 5.

Developed by the IPPM working group of the Internet Engineering Task Force (IETF), the `Surveyor` utility is capable of measuring end-to-end one-way delay [25],

packet loss [26], and route information along Internet paths [27]. This tool uses Jacobson's modified version of the `Traceroute` utiltiy [28] and a GPS corrected clock to measure one-way delay and loss. Data generated from `Surveyor` are archived into a repository server for further processing. In their website [27], the authors provide results on changes in the path of packets, asymmetry in routes, and the performance of high-speed research networks. Note that our software uses the method in the standards [25, 26] for our unidirectional experiments.

Based on the standards [29, 30, 31], the Regional Internet Registry (RIR) for the European region (RIPE NCC) developed a global network of measurement hosts known as *Test Traffic Measurement Service (TTMS)* project [32]. As described in their website, TTMS uses dedicated measurement devices (test boxes) that generates traffic with the `Ping` [33] and `Traceroute` [34] utilities. The data that is generated from these tools include round trip delay, packet loss rate, and delay variations (jitter). These data are further processed and displayed in graphs for monitoring of networks [35].

In addition, the TTMS project also provide access to a database of routing information that one may use to track changes in the path of packets. These data are also archived for users to be able to perform long term trend analysis. Note that the TTMS project later became the *RIPE Atlas* [36] project.

Starting with their paper on the proposal for a scalable Internet-wide architecture [37], Francis et al. developed the *distributed Internet measurement (DIMES)* project to measure and disseminate distance information on a global scale. On top of this, they provide open access to their data allowing content service providers to determine suitable locations on the Internet for the placement of content service provider servers.

This results in minimal latency between clients and content service providers. Other uses of the DIMES project include determining available bandwidth, link capacity, packets reordering, and queuing delay [38].

In an attempt to better understand the behaviour and the evolution of the Internet, the Center for Applied Internet Data Analysis (CAIDA) developed the

Microscopic project [39] to collect two types of data. Firstly, it collects Internet Protocol (IP) forwarding path information from `Traceroute`-like active measurements. Secondly, this project collects routing information from routing tables of the inter-domain Border Gateway Protocol (BGP).

To collect IP path information, the *skitter* [40]/*scamper* [41] tool is installed in the *Archipelago measurement infrastructure (Ark)* [42] to perform parallel `Traceroute`-like active measurements into a number of targeted IPv4 or IPv6 addresses. The scamper tool is also configurable to perform `Traceroute` with Internet Control Message Protocol (ICMP), TCP, or UDP packets and to discover paths with the maximum transmission unit (MTU) option [43].

In addition, the Microscopic project uses *RouteViews* [44] to collect information such as inter-domain Autonomous System (AS) paths from BGP routing tables. The BGP information (AS paths and IP addresses) are aggregated into a database that researchers can access for building Internet topologies.

Hosted at research institutions, routing centers (Internet2's Abilene backbone), and other Internet connected servers, the *PlanetLab* [45] project is based on a large number of volunteered hosts. These hosts execute a software known as MyPLC which manages activities such as bootstrapping of nodes, distribution of software updates, auditing of systems activity, and for managing user accounts.

The objective of the PlanetLab project is to support distributed virtualisation by enabling an application to run on a slice of PlanetLab's network-wide hardware resources. Once an application is scheduled on a slice, users can perform short-term experiments with a variety of planetary-scale services such as file sharing and network embedded storage [46], content distribution networks [47, 48], routing and multicast overlays [49], Quality of Service (QoS) overlays [50], scalable object location [51], anomaly detection [52], and network measurement [53].

On top of this, the PlanetLab project offers researchers real world experience with path characteristics such as congestion, link failures, and other link behaviour. The nodes can be configured to perform long term measurement with services such as CoDEEN [47], Coral CDN [48], the ScriptRoute network measurement service [54],

Chord [55] and OpenDHT [56] scalable object location services, and the PIER [57] network monitoring services.

In spite of the benefits provided by these studies/projects, we did not use them for the following reasons;

Firstly, the objective of most of these projects focus on tracing reach-ability of hosts and mapping the path of packets on a hop by hop basis. However, our objective is to find out the impact of jitter, packet loss and out-of-order arrivals on the performance of VoIP and other real-time applications.

Secondly, the use of most of the tools in these projects requires full administrative control of nodes to run their software. However we generally do not have this permission.

Furthermore, some of these projects employ the virtualisation concept (e.g.) PlanetLab uses virtualisation technologies that enable users to share hardware resources such as memory and real-time clock. This is a problem because the sharing of resources introduces delay when we require access to real-time clock for timestamping purpose. In addition, PlanetLab nodes run many other different experiments, and there is possibility for other experiments to interfere with our experiments.

In addition, most of these projects run on academic networks. For example, Claffy et al. [58] pointed out that a typical PlanetLab network consists of research networks where most of the packets travel through non-commercial backbone links. As a result, any information observed may not reflect the experience of packets on commercial networks.

Lastly, most of the nodes have limited resources. For example, a PlanetLab node sometimes use capped bandwidth, and limited processor time and storage capacity [58]. This may restrict the type of experiments that we can perform.

One thing that is common among many existing network measurement projects, is that they use a variant of `Ping` or `Traceroute` to perform active measurement. We presents the functionality of some of these tools in Section .

2.3 Limitations of Passive and Active Measurement Methods

In order to measure anything in any given scenario, there needs to be traffic to measure. Our research questions (which will be discussed in the next chapter) look at scenarios where two nodes are separated by long distance paths. Normally we would not expect traffic between those nodes to occur naturally. Even where such traffic occurs it may not be on a regular basis. Moreover, even if there is traffic on a regular basis it may be difficult to put probes on or near the nodes because we do not control most networks of interest. In the context of passive measurement, we would thus expect little or no traffic unless we generate any.

A common deployment scenario for passive measurement is a local network where one has control of the network infrastructure for installing/activating monitoring probes. Because there is no control over the generation of network traffic, users of passive measurement are unable to control the transmit time of packets, the time difference between the transmitting node and the monitoring probe, the total number of transmitted packets, and the intertransmission time of packets.

Without this information, one cannot use passive measurement together with the UDP protocol to determine characteristics such as latency, jitter, MOS, packet loss, and out of order arrivals. However path characteristics including some types of jitter, out of order arrivals, length of the path taken by packets, and buffer time/size requirements can be measured or estimated for the TCP protocol.

We thus decided to use active measurement for the following reasons: Firstly, it lets us place nodes in networks we have no control over, and the probe position in the network topology is not so important. Secondly, we can collect timing and other information at the transmitting node which we can use to determine characteristics including MOS, latency, jitter, packet loss, and order of packet arrival. In addition, we can probe the network with standardised TCP traffic to determine buffer time/size requirements.

One drawback of active network measurement is that the generated traffic may interfere with existing production traffic, which is a limitation for low bandwidth networks such as the satellite network we encountered in the Pacific. This prevented us from doing extensive TCP/NC download series experiments in some of the islands, e.g., Niue and Tuvalu.

2.4 Measurement Tools

- `Ping` [33]: `Ping` uses ICMP to determine the round trip time (RTT) and the number of hops between nodes. The source sends small packets of 64 bytes (56 bytes of data and 8 bytes protocol information) known as ICMP echo requests to the destination and awaits ICMP echo responses from the destination. If there is a response, the destination host is declared reachable. The source also obtains information such as *round trip time (RTT)*, packet loss, and the number of hops from the ICMP response.

- `Traceroute` [59]: `Traceroute` uses UDP, TCP, and ICMP to determine the path taken by packets and the routers that forward packets between hosts. It sends a sequence of packets from the source to the destination. The packets contain a *TTL* [60] value that the application gradually increases at every transmission, starting from 1.

  Every router reduces the TTL value by 1. When the application transmits the first packet, the TTL value will be 1. The first packet reaches the first router, the router decrements the TTL value to 0 and drops the packet.

  When a router drops the packet, it sends an *ICMP time exceeded* message back to the source. The source increases the TTL value to 2 and transmits the next packet. However this time the packet will be forwarded between the first and second router, where the TTL value reaches 0. The second router drop the packet and reports back to the source with an ICMP time exceeded message. The application repeats this process until one of the transmissions reaches the destination. The end result of this process is the mapping of the

likely path from source to destination, and the recording of the routers that have processed the transmitted packets.

- `Pathchar` [61]: `Pathchar` also uses UDP and ICMP to infer the characteristics of individual links along an Internet path by measuring the RTT of packets. The RTT here is the time it takes for packets to travel from source to a router along the path and back to the source. Like `Traceroute`, `Pathchar` takes advantage of the TTL value of packets (using the `ICMP time exceeded`), to determine how many links a packet can traverse before the TTL value reaches zero and a router drops the packet.

  `Pathchar` sends series of varying numbers of packets and varying packet sizes and measures the time until the source receives an `ICMP time exceeded` message from a router along the path [62]. By performing statistical analysis on the time until the ICMP message is received, `Pathchar` is able to determine latency and bandwidth on a link, the distribution of queue lengths and the probability that packets were dropped.

- **Clink** [63]: `Clink` uses UDP to estimate bandwidth and latency of links between source and destination. It sends large number of UDP packets and measures the RTT. `Clink` performs functions similar to `Pathchar`.

- `Pchar` [64]: `Pchar` uses UDP to estimate bandwidth, latency, and packet loss on links along an end-to-end path. It perform similar functions as `Pathchar` (open source implementation of `Pathchar`).

- `Nettimer` [65]: `Nettimer` uses TCP to estimate capacity in the form of bottleneck and available bandwidth, and actively probes the network using the packet-pair 'tailgating' technique. In the tailgating technique, the source sends two packets within a short time interval between nodes.

  While the packets travel from source to destination, they encounter delays from queuing and processing time in routers which change the interval between the packets. Upon arrival of the packets, the destination node records the interval between the packets to determine link capacity and bandwidth.

- `Pathrate` [66]: `Pathrate` uses UDP to estimate end-to-end capacity between nodes even on heavily loaded links. Network measurement with `Pathrate`

consists of three phases. In the first phase, the software discovers the maximum packet train length that the path can support.

The discovery phase involves sending a few packet trains of increasing length to detect multi-channel links (load balancing) and traffic shaping. The purpose of the first phase is to identify the maximum packet train length that the path between the source and destination can transfer, without causing buffer overflows at the routers or the receiving operating system.

After the first phase, the second phase involves generation of large number of UDP packet pairs (1000) of variable size and transmitting them into the link to uncover the local capacity modes (CM) of the underlying bandwidth distribution (the proper capacity of a link).

Finally, the third phase involves the estimation of link capacity by identifying the Asymptotic Dispersion Rate (ADR) [67] from the dispersion of long packet trains. The meaning of dispersion here refers to the time difference in the interval with which two packets were transmitted and the interval with which they were received.

Furthermore, `Pathrate` also establishes TCP connections known as *control channels* that among other functions, acknowledge packet pairs/trains that were received correctly.

- `Iperf` [68]: `Iperf` uses UDP to measure packet loss, and jitter. It also uses TCP to estimate bandwidth. Once a user defines the approximate bandwidth of a link, the software transmits a large amount of traffic to fill up the link to maximum capacity. In a TCP experiment, `Iperf` uses the *bandwidth delay product* [69] to determine bandwidth from the TCP window size.

  The bandwidth delay product is $B * RTT$, where B is the bottleneck bandwidth. For UDP experiments, `Iperf` determines packet loss using a buffer with a size that is either a default value or specified by the user. The calculation of jitter is based on the Real-time Transport Protocol (RTP) standard [70].

- `Pathload` [71]: `Pathload` uses UDP to estimate a range of available bandwidths for the end-to-end path between source and destination. Furthermore it estimates available bandwidth based on the impact of its own traffic on

the transmission rate of other traffic in the path. The idea of `Pathload` is to determine whether one-way delays (OWD) or latency of a periodic packet stream shows an increasing trend, when the stream rate exceeds the available bandwidth [72].

In an experiment, the source transmits periodic streams of packets with a transmit timestamp of t_i. Suppose that r_i is the receive time of the ith packet, OWD can be derived as $D_i = r_i - t_i$, where D_i is the absolute one-way delay from sender to the receiver.

The source continues to increase the transmission rate of streams, and the receiver uses the OWD to check whether the stream rate is greater than the available bandwidth. When the stream rate exceeds the available bandwidth, long queuing delay are expected so the available bandwidth is based on the transmission rate that causes the overloading of the link. This measurement method is known as Self Loading Periodic Stream (SLOPS) [72].

- `Pathchirp` [73]: `Pathchirp` uses UDP to estimate the available bandwidth with end-to-end probing of packets. `Pathchirp` sends a number of packet pairs from source to destination and performs statistical analysis at the receiver. It transmits successive packets at increasing inter-transmission rate to a point where packets suffer queuing delay from bursty traffic. The estimation of available bandwidth is based on the impact of queuing delays on spacing between packets.

- `OWAMP` [74]: `OWAMP` uses UDP to estimate path characteristics including OWD, loss probabilities, jitter, and the inter-arrival gap between packets. `OWAMP` uses two protocols: `OWAMP`-Control and `OWAMP`-Test. The control protocol initiates and controls sessions with TCP and receives results from peers. The test protocol exchanges UDP test packets [75].

  Prior to running an experiment, the source and destination node must agree on an experiment schedule to enable the destination node to keep track of lost packets. Note that `OWAMP` declares packets as lost using a timeout value. Upon the reception of packets at the destination, the test protocol records the sequence number, transmit time, receive time, and the TTL for IPv4 packets.

The control protocol is then responsible for transfer and archiving of results, as well as processing the results to determine path characteristics.

Note that there are active measurement tools that base their measurement on the injection of traffic modeled on a real use case and attempt to capture network effects on such packet sequence. A good example is the use of the Cisco IP SLA tool to generate voice traffic and to analyse the quality of VoIP applications on a network [76].

The remainder of the literature review will be provided in Chapter 10 where we discuss TCP/NC.

CHAPTER 3

Network Measurement

In this chapter, we describe the motivation for the development of our `beacon` *software, and follow this up with discussions on network measurement methods.*

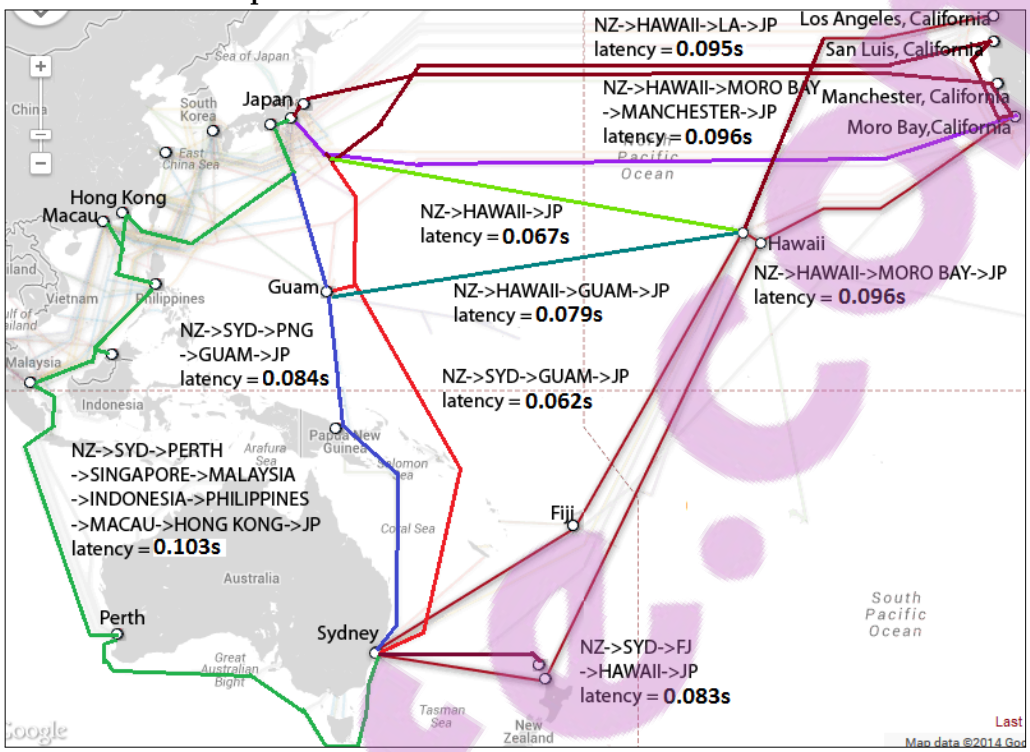
3.1 Motivation

The motivation for the development of our software arose from a series of tests conducted between the University of Tokyo and the University of Auckland in 2010. Using the `Traceroute` utility, the tests showed that packets took different routes from New Zealand to Japan depending on the time of experiment.

To describe the results of these tests, we use a map (see annotated screenshot in Figure 3.1) on submarine cables [1] together with the optical fibre latency estimator [77] to show the possible routes and latency of our packets between New Zealand and Japan.

Note that the latencies in the map on Figure 3.1 are based on the ITU-T G.652 standard [78]. In the map, there are multiple possible routes for packets to travel from New Zealand to Japan, most of which have latencies below 0.1 seconds. The

Figure 3.1: Screenshot taken from map in [1] showing possible paths between New Zealand and Japan



longest route from New Zealand via Sydney, Singapore, Malaysia, Indonesia, Philippines, Macau, and Hong Kong to Japan has a latency of 0.103 s, whereas shorter paths from New Zealand via Sydney, and Guam to Japan have latencies of 0.062 s.

As our test packets travel through the possible paths in Figure 3.1, they are delayed by long propagation times and the time spent in the queues of congested routers. Other undesirable effects are jitter and out-of-order delivery that is often associated with load balancing of packets across multiple paths.

In this thesis, we investigate the impact of these effects on VoIP and other real-time applications for high latency connections. There is also a strong aspect in this thesis that looks at connectivity in and out of the Pacific Islands, which often happens via satellite links.

To guide our investigation, we base our research on the following questions.

1. How stable is the latency between nodes?
2. How will the quality of Internet connectivity change over time with additional links and routers?
3. What can be done to improve the performance of VoIP and other real-time applications on high latency networks?

As already discussed in Chapter 2, the approach chosen for our experiments is an active measurement one. We want to be able to control the transmission time of packets with an active measurement utility and observe their arrival time at the destination. Only this approach will let us observe all desired parameters in sufficient quantities.

Note that the time available for this thesis is not necessarily long enough to answer these questions. However we will attempt to provide observations in Chapter 9 and Chapter 11. We hope that in approximately 10 years time, we may be able to collect a large enough amount of data to better answer these research questions.

3.2 Summary

In this chapter, we discussed the motivation for the development of the tool required for this research (`beacon` software). We will now discuss our `beacon` software in Chapter 4 with focus on:

- Functional requirements
- Non-functional requirements

Lastly, we present hardware requirements desirable for the operation of our software.

CHAPTER 4

Beacon Software Requirements

In this chapter, we outline the requirements for the `beacon` *software. We begin with the functional requirements and follow this with the non-functional requirements. Lastly, we present the hardware requirements essential for the operation of our software. The* `beacon` *software is our main tool that will assist us in answering the first two research questions, on the stability of latency between nodes, and long term quality changes in Internet connectivity.*

4.1 Functional Requirements

There are three sets of requirements that define the functionality of software: input, behaviour and output. Input is the information passed to software. Behaviour are the methods for processing this information, and output are the end results that are expected from a software.

In order to describe the input, behaviour and output of our `beacon` software, we base our discussion on the following type of experiments:

- Experiment 1: Simulate a unidirectional VoIP over UDP call.
- Experiment 2: Simulate a bidirectional VoIP over UDP call.
- Experiment 3: Simulate a unidirectional VoIP over TCP call.
- Experiment 4: Simulate a bidirectional VoIP over TCP call.
- Experiment 5: Simulate a unidirectional file download with TCP.
- Experiment 6: Simulate a bidirectional file exchange with TCP.

We repeat these experiments over time and each instances of an experiment is called the *run number*.

4.2 Definition

In this chapter, we use the following definition

Initiator	The node that initiate the transmission of packets in any experiment
Responder	The node that receives the packets from the initiator
Source	A node that transmit packets to another remote node
Destination	A node that receive packets from another remote node

4.2.1 Input

At the time of writing, we had deployed 33 nodes in 15 countries. Most of these countries use high speed fibre optic cable to connect to the Internet, but a significant number of ISPs in Pacific Island countries use high latency and low bandwidth satellites to connect to the global Internet.

To schedule an experiment between nodes, the following are mandatory parameters that must be configured.

- Source IP address
- Destination IP address
- Destination port number
- Inter-transmission time
- Mode of the experiment (bidirectional or unidirectional)
- Duration of the experiment

Note that the source port is dynamically allocated by the operating system.

As discussed in Chapter 1, our software is also capable of generating traffic for other real-time and streaming applications such as video. However, because of the limited bandwidth and experiment time slots on a significant subset of our nodes we restricted our experiments to simulating VoIP applications such as Skype.

In order to find out a suitable transmission rate for voice experiments, we decided to use the properties of Skype's SILK codec because at the time of writing this was the most widespread codec. A typical packet inter-transmission time is given in the SILK draft [79] as 20 to 100 milliseconds (ms). By using a flexible transmission rate, Skype can adjust to the quality of the network.

Despite the inter-transmission time range provided in the draft, there is no mention of the inter-transmission times that Skype uses in particular high or low bandwidth links. In order to determine the inter-transmission times, we conducted an experiment where we performed Skype calls between nodes located at the University of Auckland and the Pacific Island of Niue. Our findings indicate that Skype uses an inter-transmission time of 20 ms on high bandwidth or 60 ms on our low bandwidth path. Therefore, these are the inter-transmission times that we use as input into the configuration of our experiments.

In order to differentiate between the functionalities of a node in our experiments, we use mode numbers. For example, we use mode number 2 to configure a host as an initiator in our UDP bidirectional experiment. This node immediately transmits the

first packet into another node that we configured with mode number 3 (responder). Once the responder receives the first packet it immediately responds back to the initiator. Further technical details are described in [80].

In a simulated VoIP experiment, we input into our software the number of packets to be transmitted. By default, our software generates 10,000 UDP packets per experiment run. A typical phone call is generally taken as lasting three minutes [81, 82]. At an inter-transmission time of 20 ms, this corresponds to 9,000 packets. By using 10,000 packets, we allow for some packet loss which may occur if Network Time Protocol (NTP) synchronisation at one or both of the experiment endpoints fails. For TCP experiments, the TCP stack controls the transmission of packets, so we do not have this input requirement. We use an entire file as input to our software so that the TCP stack can break the file into snippets for transmission to the responder.

Since our software uses TCP for both simulated VoIP and file download experiments, we input into our software the character 'd' as indication of simulated file download and the character 'v' for simulated VoIP experiment.

4.2.2 Behaviour

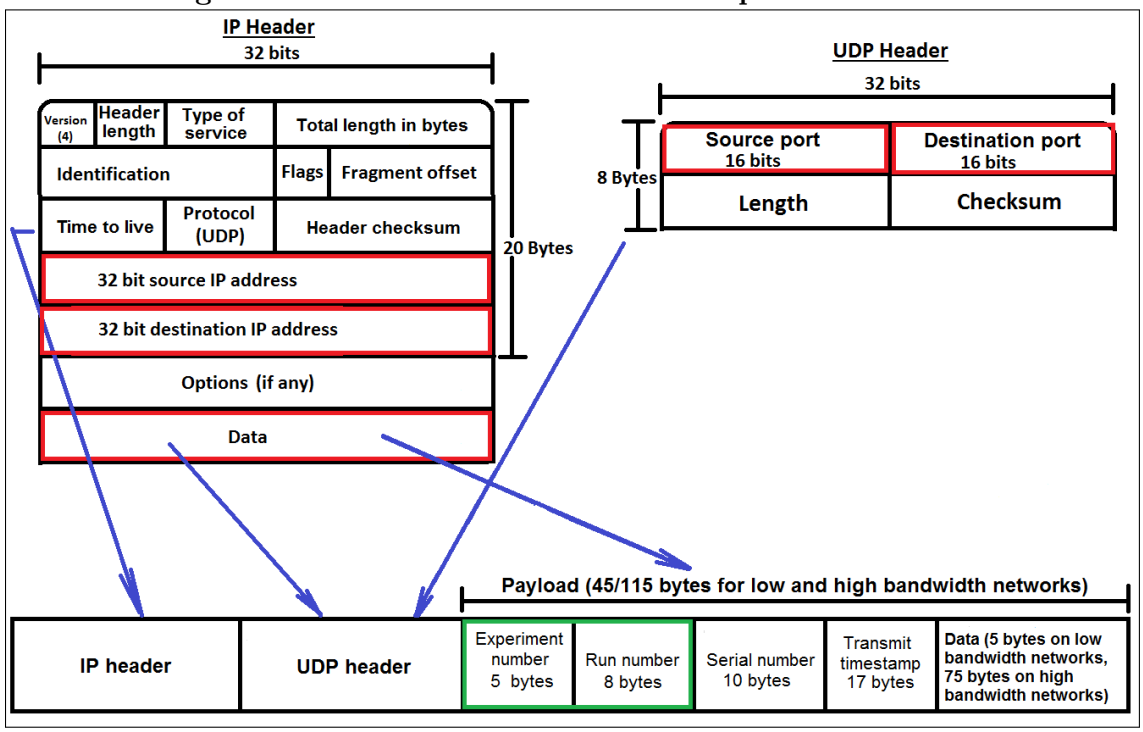
In our simulated VoIP experiment we read data from a text file and generate segments of typically 45 and 115 bytes of data to create UDP [83] packets. Figure 4.1 shows the packet structure for this type of experiment.

In the payload, we insert the experiment and run number in the first packet only. The experiment number uniquely differentiates experiments.[1]

At the transmitting host we count the number of packets already transmitted and insert that count into the payload of the next packet as a packet serial number. We also insert the time of transmission into the transmit timestamp field of the payload. At the responder, the serial number and timestamps are extracted from

[1]The run number was originally intended to differentiate instances of an experiment. Due to issues we encountered with duplicated run numbers, we now use timestamps for this purpose.

Figure 4.1: Simulated VoIP over UDP packet structure



Note: The fields marked in red corresponds to input parameters of our software. The experiment and run number fields are populated in every UDP packets.

payload and recorded. The rest of the payload contains padding (taking the rest of the space that the voice data would occupy in a real VoIP packet).

In the simulated voice data, we pad to 45 bytes per packet to simulate voice packets that are transmitted over low bandwidth networks and 115 bytes for high bandwidth networks, which corresponds to packet sizes typically observed with Skype. Since our application controls the transmit time of packets, we transmit 10,000 packets in each run of our simulated voice experiment per direction[2].

In a situation where UDP ports are blocked by firewalls, Skype reverts to using the TCP protocol for the transmission of voice information [84]. In cases like these, voice quality is expected to degrade because of the additional delays imposed by the flow and congestion control mechanisms of TCP.

[2]Note that Skype packets are not always exactly the same size and the values we mention here is an approximate size

To study the behaviour of VoTCP applications, we simulate voice data snippets and transmit them over a TCP connection. In this type of experiment, we mimic a situation where two nodes repeatedly exchange simulated voice traffic over TCP. The initiator sets up a TCP connection, and passes bytes of data to the TCP socket.

In low bandwidth networks, our application passes snippets of 45 bytes to the TCP buffer every 20 ms. In high bandwidth links, our application passes snippets of 115 bytes to the TCP buffer as per the requirement in [85]. At the responder, we observe the size of snippets and the time when snippets were received.

Moreover, we also do not control the number of TCP packets in an experiment. For example, when there is little to no congestion in the network there will be little to no packet loss, thus resulting in timely arrival of the snippets. However, the number of transmitted packets may increase in a congested link due to TCP retransmission and snippets may not arrive in a timely fashion.

As described earlier, we also use TCP to perform simulated file download experiments between a client and server. In this experiment, we emulate the behaviour of application protocols such as HTTP [86], FTP [87], SSH [88] and SMTP [89] where a test file is handed to the TCP stack to be transmitted to the responder.

In a file download experiment, our software instantaneously passes snippets of 1024 bytes into the TCP buffer to be transmitted. This is different from our VoIP over UDP experiments where we pass packets at regular interval of 20 ms into the transmitting UDP socket. At the responder, we observe the size of snippets and the time snippets were received.

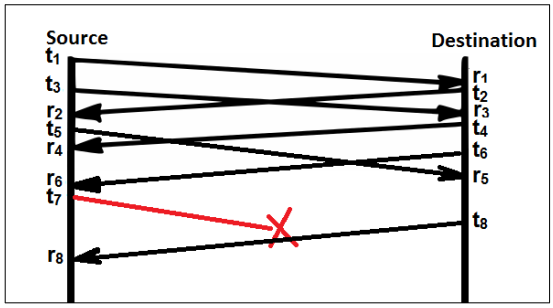
Figure 4.2: Unidirectional experiment between the initiator and responder node

Note: In this mode, the initiator can only send packets whereas the responder only receives packets.

In a unidirectional experiment, our application transmits packets in one direction

only from the initiator to the responder. At the responder node, timing information and other experiment observables are recorded into log files. As shown in Figure 4.2, the initiator transmits multiple packets during the time interval t_1 to t_4. Depending on the delays encountered by packets, they arrive between r_1 and r_4.

Figure 4.3: Bidirectional experiment between the initiator and responder node



Note: The initiator initiates the transmission of the first packet whereas the responder responds back to the initiator. Note that only the first packet from the initiator triggers the first packet from the responder. After the initial exchange of the first packets, the two nodes repeatedly exchange packets without waiting to receive packets from either side.

As presented earlier, the bidirectional experiment involves the exchange of the first packet between the initiator and responder. We refer to this process as the handshake process. In Figure 4.3, the initiator transmits its first packet at time t_1. Once received at time r_1, the responder completes the hand-shake process with its first packet at time t_2.

Once the handshake process is complete, the initiator and responder continuously transmit packets in both directions without waiting for the arrival of the packet at r_2. This is evident in Figure 4.3 where the packet transmitted at time t_3 leaves ahead of the arrival time r_2. Similarly, the responder may transmit the packet at time t_4 without waiting for the arrival of the packet at r_3.

Note that it is possible for packets from the initiator to be delayed. For example, the packet transmitted at time t_5 was received at time r_5 after the transmission of the packet at time t_6. There is also a possibility for packets to be lost. For example, the packet transmitted at time t_7 was lost but the responder continues to transmit the next packet at time t_8 even when there is packet loss.

4.2.3 Output

At the end of an experiment we either observe or extract information from packets and record it in log files. Depending on the role of the node (initiator or responder and mode), two types of log files are created: transmit *(Tx)* and receive *(Rx)* logs. Contained in the log files are two sections: header and data.

The header section contains information that we use to identify parameters in an experiment. This include port numbers, IP addresses and other configuration information (see [80]). The rest of the log file contains data that we record in an experiment. This includes transmit and receive time of packets and other information defined in [80].

In the data section of TCP transmit log file, we store data in columns. Since we do not have control over the transmission of packets in TCP, we do not record them in the data section of TCP transmit log files. The only information recorded in the TCP transmit log is the header information. At the receiver, we record the number of received data chunks in the *Rx sequence number* column. In addition we record the receive time of the data chunks in the *Rx timestamp*, and the size of the snippets in the *Rx size* column.

We also determine the length of the paths that our packets take. To do so, we extract the TTL values of the receive packets and record them in our TCP raw log files. This requires the use of the raw sockets interface. The path length of packets is the difference between the default TTL value of the Linux operation system (64) and the TTL value received.

In our UDP transmit log files we record two transmit timestamps: *Tx timestamp 1* the time at which the application decided that a transmission should happen, and *Tx timestamp 2* is the time at which the call to the UDP `send()` socket method returns. With this information we can determine the interval within which a packet was created and transmitted. We also record the size of data passed to the UDP socket and the sequence in which packets were transmitted. Note that we also insert *Tx timestamp 1* and the sequence number into the payload of our UDP packets.

At the receiver, we extract Tx timestamp 1 from the payload and record this information in the *Tx timestamp* column in our receive log. At the same time, we record the time at which packets were received in the *Rx timestamp* column. These are the information that we use to compute latency and jitter (see Chapter 5).

Moreover, we extract the serial number of packets from the payload (see Figure 4.1) and record it in the *Packet number* column. We also keep a tally of the number of packets that were received in the *Rx sequence number* column. Our UDP receive log file normally also contains the TTL value of the packet, which we directly extract from the IP header (see Figure 4.1) of UDP packets except for operating systems where we cannot extract this value for the IP header e.g., for the Berkeley Software Distribution (BSD) operating system.

4.3 Non-Functional Requirements

Our software runs on a global network of hosts that are connected via the Internet [8]. These hosts are managed in a number of ways. In a situation where there is no support from our research partners, we are provided with a user account to access a host in their network and to configure experiments.

However, in a situation where we cannot access our research partner's network, someone else manages the operating system and experiment configurations for us. We also have a situation where our research partner provided us with scripts to configure and they pull the configuration from our beacon server into their hosts.

Primarily, our software runs on the Ubuntu operating system, but we also have hosts that are running on MacOS and FreeBSD. It is for this reason that we developed our software to be compatible with several Unix operating systems.

As our experiments depend on the Internet for communication between hosts, we use two methods to configure Internet access. Firstly, we statically configure IP address information in the interface file (see Section A.2 of Appendix). This method is ideal for situations where a host is located in a demilitarised zone (DMZ) [90] and

there is no Dynamic Host Configuration Protocol (DHCP) server to provide IP address information to a host. If there is a DHCP server, we provide our research partner with the MAC address of our host for static assignment of an IP address. This ensures that our host always acquires the same IP address from the DHCP server.

Our application is not continuously active on our beacon hosts and so the ports associated with it are not always open. Instead we only execute the application at the time of an experiment. This presents a number of advantages. Firstly, long term stability is not an issue. Secondly, we make it more difficult for malicious parties to scan our application for vulnerabilities. However it is also a disadvantage especially when the difference between clocks of transmitting and receiving nodes is large. In addition, we designed our software so that only fixed fields are accepted from incoming packets/data. This feature safeguard our software against buffer overflow exploits.

When the clocks of the transmitter and receiver are unsynchronised, the transmitting socket may start transmitting packets before the receiving socket was executed. Similarly, the receiving socket may shut down early before the transmissions are complete. Regardless of whether there is early termination or late start of a socket, the end result is packet loss.

Currently, most of our hosts are configured with the domain name of public NTP servers. To resolve the domain name of NTP servers, our hosts must be configured with Domain Name System (DNS) information (see Section A.2 of Appendix). In most cases, we acquire this information from our research partner and configure it statically in their hosts.

When our host is located in a DMZ with no access to a DNS server, third party applications such as `unbound` [91] can be configured for domain name resolution. Note that there is a potential for our host to fall victim to a DNS distributed denial of service (DDOS) attack [92] if configured with incorrect DNS information. It is thus important that *unbound* is configured to resolve domain names using publicly available Domain Name System Security Extensions (DNSSEC) resolvers.

Furthermore the `hosts file` (see Section A.2 of Appendix) must be configured with a mapping of IP addresses to their hostname. This is required to ensure that our application uses the correct interface/IP address for the exchange of packets in an experiment.

In most networks, the purpose of a firewall is to protect against malicious traffic. However the default configuration of firewalls can often block some or all of the traffic destined to or originating from a network. To ensure that we can remotely manage experiment configuration and perform experiments, any firewalls must be configured to allow packets to and from ports required for experiments and management of hosts.

Third-party applications such as *rsync* [93] and *ssh* [94] must be installed on our beacon hosts to `back-haul` data to our central repository server. In addition, rsync must be configured with the compression and incremental backup options (*az*) to reduce the amount of traffic that our hosts back-haul data to our central repository server.

In order to determine the capability of the hardware to operate our software, we use an iterative counter to count the number of loop executions in the `beacon` software between packet transmissions. This gives us an idea as to how accurate our time-stamping on a beacon host is likely to be. A suitable value for the counter is greater than 200 corresponding to roughly 1 opportunity at every 0.1 ms if our inter-transmission time is 20 ms. Any value less than 200 may be caused by a host performing many other tasks or by hardware that is too slow for our experiments.

4.4 Hardware Requirements

In some countries in the Pacific, power outages are an ongoing problem. This is a problem for two reasons. Firstly, it disrupts period of data collection. In other cases, electronic components are at risk of damage. As solution to this problem, a Uninterruptible Power Supply (UPS) is required to either provide back up power or provide graceful shutdown of hosts when there is a power outage.

One of the most rapidly developing technologies in server infrastructures is virtualisation. It enhances the administration of services such as mail and web. However, virtualisation is not suitable for time-critical network measurement: Multiple virtual machines on a physical host get executed by context switching, i.e., each virtual host gets to execute for a certain amount of time before the next virtual host takes over the resources. Therefore the time at which a particular host receives a packet or decides to transmit a packet may be later than the actual time that the virtual host should have timestamped the packet or transmitted a packet. This effects is triggered by the context switching found on virtual hosting system. In order to minimise disruption on the transmit or receive time of our packets, we recommend that our `beacon` software be installed in a dedicated physical host.

On a day to day basis, each node performs 6 experiments 3 times daily with each partner beacon, and a host can be configured to act as either an initiator/responder. Each experiment requires two peers to exchange packets and to observe and record the required information in log files. This task is mildly intensive and requires the hardware to be of a certain minimum grade for smooth running of our application.

A good system for hosting our software is a dedicated physical host with at least 20GB of hard disk space, processor speed of at least 500MHz, and memory of 256MB (RAM) to generate packets, transmit packets and to store them in log files. When a mini computer is the only option, we require a 16GB compact flash disk to store the operating system files, our `beacon` software and data from experiments.

4.5 Summary

This chapter discussed the requirements for hosting our software. We discussed functional, non-functional and hardware requirements. In the functional requirements, we discussed input parameters required in experiments, the way our software processes information and how information is written into log files. In the non-functional requirements, we discuss third party applications and operating system configurations that are required to support our software. Hardware requirements define the

hardware specification that is essential for hosting our software. In the next chapter, we discuss jitter and clock drift compensation.

CHAPTER 5

Jitter

In this chapter we discuss jitter, a measure for the stability of latency which we will observe over the long term. We begin with a review of jitter in the context of two nodes exchanging voice information. Then we look at the processes involved in the creation of IP packets, and present network processes that contribute to the formation of jitter. We follow this with various methods for jitter estimation and reflect on solutions to mitigate against jitter.

5.1 What is Jitter?

In the context of packet networks, jitter is the delay variation in the arrival time of packets. At the sending node, packets are sent out continuously, with constant time intervals between successive packets. Because of network congestion, and queuing, the time between packets becomes distorted, causing variations in the arrival time of packets [95]. In order to discuss the causes of jitter, we use the OSI model [96] to describe a context where two nodes exchange voice information with Skype.

Starting at the application layer, the transmitting host generates voice infor-

mation, e.g., with Skype and transmits it over the Internet to the receiving host. For VoIP and other real-time applications, timely arrival of voice information is a priority, and the best way of achieving this is to transmit small snippets of voice information at a time. So to achieve timely arrival, Skype breaks down voice information into small snippets and passes them to the transport layer. At this stage, the delay is application-specific (e.g., depends on delays from encoding/decoding of information with a specific codec).

At the transport layer, data snippets are encapsulated with source and destination UDP port numbers to form segments [97]. The source port is chosen from a list of dynamic ports and identifies the transmitting socket. The destination port identifies the receiving application [98] (i.e.,e.g.,Skype). The segments are then passed to the network layer. The delays in this layer is protocol specific (e.g., delays from use of UDP protocol depends on the readiness of the UDP segments to be handed over to the network layer. The delays from use of the TCP protocol depends on the congestion window).

The network layer encapsulates segments with source and destination IP addresses. The source address identifies the transmitter, the destination address enables routers to determine the path towards the destination. Once encapsulated with IP addresses, packets are ready for transmission.

At this stage, IP passes packets to the data link layer where they are converted into a frame for transmission on the physical layer. To transmit the frame, the data link layer checks the availability of the transmission channel and transmits only when the channel is free. The delays on the transmit time of frames depends on how busy the local network is.

At the physical layer, *propagation delay* is the main source of delay. This delay is fixed in connections that involve non-wireless media such as fibre optic, coaxial, and Ethernet cable. However in cases where the transmission channel is a satellite connection, there are two contributions to propagation delay. The first delay is fixed depending on the length of the physical cables that connect the end nodes to the ground stations. The second delay is always substantial due to the significant distance between ground stations and the satellite. This delay is also fixed if the

satellite is a geostationary (GEO) satellite, but varies slowly but nevertheless significantly over time for low Earth orbiting (LEO)/medium Earth orbiting (MEO) satellites.

While packets travel from source to destination, they are queued in the incoming buffer of routers. If packets arrive too fast for routers to process, the buffer will be filled with packets waiting to be processed. The time that packets spend in the buffer varies depending on processing time and the arrival rate of previous incoming packets, and this also contributes to jitter.

During processing, a router removes the data link header and extracts the destination address from the IP header of the packets. The router then uses the destination IP address together with a routing algorithm [99, 100, 101] to lookup the best path to forward packets towards the destination.

Sometimes the best path consists of multiple entries in the routing table. In situations like this, the router may choose the best path based on the network address, or the router may choose a path in a round robin fashion (per packet) [102]. This technique is known as load balancing and is often employed to increase network throughput by distributing network traffic among known routes.

As packets enter multiple exit queues, they experience variations in delay depending on the amount of time they spend in the exit queues before they enter the physical link. These variations in delay tend to accumulate over multiple routers. Regardless of the process that causes jitter, the end result is potentially poor performance for VoIP and other real-time applications.

5.2 Why Study Jitter?

Over the years, the inception and subsequent development of the Internet has revolutionised communication with the possibility of transmitting voice, video and data information over the same physical link. However, the Internet was developed as a best effort network, a type of network without guarantee of timely arrival for voice

information. As a result, voice is left to compete with other data for capacity on links.

Responding to the impact of best effort data on VoIP and other real-time applications, most Internet Service Providers (ISP) increase capacity by adding links and improve voice quality with use of techniques such as QoS [103], a collection of methods that let routers prioritise delay sensitive packets. However, such solutions are not always beneficial and can introduce additional delays when packets travel through extra hops to reach the final destination.

Moreover, QoS is not always available on the core infrastructure of the Internet because it requires direct control over routers.

Given the ongoing development of the Internet and its impact on the propagation of packets, this thesis proposes to investigate whether the addition of extra routers and links contributes to an increase in jitter or represents a long term strategy for improving the quality of voice communication. We therefore aim to monitor links for a significant number of years beyond the end of this thesis.

5.3 Notation

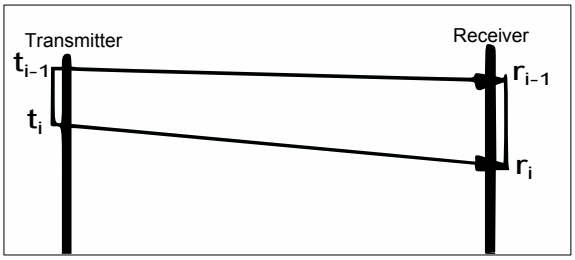
In this chapter, we use the following notation

$E_t(i)$	The expected arrival time of packet i
t_i	Time of transmission of the packet with serial number i according to the clock of the transmitting node
r_i	Time of receipt of the packet with serial number i according to the clock of the receiving node
t_T	Constant inter-transmission time
d_p	Estimated relative clock drift between the transmitter and receiver
$\phi(i)$	$\phi(i) = 1$ if the i'th packet was received and $\phi(i) = 0$ otherwise

5.4 Estimation of Jitter

The measurement of jitter in the context of packet networks is not uniquely defined (see, e.g., RFC3393 [24], RFC3350 [70]). In this section we begin by describing the concept of jitter and follow this with studies of jitter.

Figure 5.1: The time difference between the transmit and arrival times of packets



Note: The time t_{i-1} is the transmit time of the $i-1$'th packet, t_i is the current transmit time of the $i-1$'th packet. The time r_{i-1} is the receive time of the $i-1$'th packet, and r_i is the receive time of the i'th packet.

5.4.1 Transmit Time

In Figure 5.1, each packet leaves the transmitter at a particular time. For example, the i'th packet leaves at time t_i, the $i+1$'th packet at time t_{i+1}, and so on. Therefore t_i is the transmission time of the i'th packet. Note that this is when the packet gets transmitted, not the duration it takes for a packet to arrive at the final destination.

We can in principle measure this time in three ways:

- According to the transmitter's clock
- According to the receiver's clock
- According to any other clock we may choose, (for example) a virtual clock that "travels with the packet"

The difference between t_i and t_{i-1}, i.e., $t_i - t_{i-1} = t_T$ is the *inter-transmission*

time between packets $i-1$ and i. If this difference is constant regardless of i, then we have a constant packet transmission rate. Otherwise we have at best an average packet transmission rate.

5.4.2 Receive Time

The point in time at which the i'th packet of a flow arrives at the receiver is r_i. This is the *receive time* of the packet. Like the transmission time, we measure this using the clock of the node on which the action takes place i.e., the receiver's clock.

The time difference $r_i - r_{i-1}$ is the *inter-arrival time* between packets $i-1$ and i. The *arrival rate* of packets is simply the number of packets that arrive per second.

5.4.3 Latency

The time difference $r_i - t_i$ is the *latency* experienced by the i'th packet, but only if t_i and r_i have been measured by the same clock. When the two times are measured with different clocks (e.g., t_i is in transmitter clock time and r_i in receiver clock time), then $r_i - t_i$ is the latency plus the time offset between the clocks.

We could also call latency the *transit time*. However, this term is better reserved for the time a packet spends in a particular queue along the way. Note that in order to get an offset-free latency, we need timing data from both the transmitter and the receiver, measured with the same clock, or with highly synchronised clocks.

A constant inter-transmission time and constant latency result in a constant inter-arrival time. In practice, we will rarely see this though as latency across a large network is seldom constant. Therefore, one often looks at the *average latency*. This is derived by adding up latencies of all packets for all packet serial numbers i, then dividing by the total number of packets. This average latency is also the *expected latency*.

Instead of using the average latency, one can in principle also use the *median latency*. This can be derived by listing all latencies in increasing order, picking the value half-way down the list. If the latencies follow a symmetric distribution, such as a Gaussian, the median and the average latency will be almost the same.

A given transmission time, t_i plus the average latency yields the *expected latency* $E[r_i]$. This is the same as adding the departure time and expected travel time. Note that as the actual arrival time is t_i plus actual latency, $E[r_i]$ usually doesn't equal r_i. Another way of deriving $E[r_i]$ is by adding the median latency rather than the average.

With constant inter-transmission times, yet another way of arriving at $E[r_i]$ is from the previous arrival time r_{i-1}, adding the inter-transmission time to predict the current arrival time: $E[r_i] = r_{i-1} + t_T$.

Note that this way of computing $E[r_i]$ does not require any time information for the packets from the transmitter. It only requires the constant inter-transmission time t_T.

As we have already seen above, there is more than one way to get to an $E[r_i]$, and the values for $E[r_i]$ we obtain in these different ways may not be exactly the same. The difference $r_i - E[r_i]$ is usually the first step in formulating the *packet delay variation (PDV)*. Jitter is thus often expressed as an average or median over the $r_i - E[r_i]$.

5.5 Jitter

There are many approaches to jitter. Some jitter computations use methods that require times to be taken at the transmitter. An example is the standard for the RTP [70], where *feedback timing information* is provided to the source from the receiver to determine jitter for ensuring reliable delivery of voice information.

Another method uses a certain number of packets to determine the average or

median jitter. An example is the IPPM [24] standard for computing IP packet delay variation from two selected packets. Here, the timing information observed at the transmitter and the receiver is used to determine jitter.

Our methods also observe the transmit and receive time of packets and use this information to compute the average absolute deviation of the packet's observed latency from the mean latency. We begin with the observation that, in the absence of clock synchronisation and constant cumulative time difference e_t between the clocks of the initiator beacon and responder beacon, the arrival time of packets is given by:

$$r_i = t_i + D_i + e_t. \tag{5.1}$$

where D_i is the latency. Since the receive time r_i and transmit time t_i are known to us, we can rearrange the equation as:

$$D_i + e_t = r_i - t_i. \tag{5.2}$$

In most cases, the two beacons use unsynchronised clocks where $e_t \neq 0$. Consequently, we cannot predict a constant transmission time error e_t or determine the latency D_i. However we can still work with their sum as:

$$\begin{aligned} \sum_{i=0}^{n-1} \phi(i)(r_i - t_i) &= \sum_{i=0}^{n-1} \phi(i)(D_i + e_t) \\ &= n'e_t + \sum_{i=0}^{n-1} \phi(i) D_i, \end{aligned} \tag{5.3}$$

where n' is the number of packets that were actually received and n is the number

of packets that were originally transmitted. Even though D_i and e_t are not accessible to us, we can estimate their mean as:

$$\mathbf{E}[D_i + e_t] \approx \frac{1}{n'} \sum_{j=0}^{n-1} \phi(j)(r_j - t_j) \tag{5.4}$$

The deviation of the actual value for a given i from this estimate can be denoted as σ_i, and this turns the estimate into the following:

$$\mathbf{E}[D_i + e_t] \quad = \quad \sigma_i + \tfrac{1}{n'} \textstyle\sum_{j=0}^{n-1} \phi(j)(r_j - t_j). \tag{5.5}$$

Substituting this result for $D_i + e_t$ in Equation 5.2 and solving for σ_i, we get:

$$\sigma_i \quad = \quad r_i - t_i - \tfrac{1}{n'} \textstyle\sum_{j=0}^{n-1} \phi(j)(r_j - t_j). \tag{5.6}$$

Note that this no longer includes any unknowns which we cannot compute. So σ_i now denotes the deviation of the latency of the i'th packet from the average latency of all received packets. The *jitter* may be computed as the mean of the absolute values of the σ_i:

$$\begin{aligned} J_t &= \frac{\sum_{i=0}^{n-1} \phi(i) \left|\sigma_i\right|}{n'} \\ &= \frac{\sum_{i=0}^{n-1} \phi(i) \left|r_i - t_i - \frac{1}{n'} \sum_{i=0}^{n-1} \phi(j)(r_j - t_j)\right|}{n'}. \end{aligned} \tag{5.7}$$

Note there is no uniform way of computing jitter and there are alternative definitions e.g., we could use the root mean square (RMS) [104] approach. In this thesis we use the above definition as our jitter computation method. The main reason that we selected this jitter definition is that it gives us an average value of the absolute variation in arrival time we can expect, i.e., a time value that could inform the sizing of a jitter buffer, for example.

Also, since we have a large number of samples in each case and are more interested in trends rather than precise values, the choice of actual definition is not critical: We would expect most other definitions to yield values of the same order of magnitude for the same data, and one would expect strong correlation between them (so long-term trends would be evident from any). One advantage of our definition is that it is simple to compute: Unlike RMS- or variance-based methods (see below), most of the computation steps above involve additions rather than multiplications, so this is a fast way of computing a jitter value. In our case this is important because of the volume of measurement data involved.

Our definition differs from RFC3393 [24] because this standard only looks at the definition of the variation in packet delay between two selected packets. Essentially, if we selected successive packets, then RFC3393 would define IP Packet Delay Variation (IPDV) as the difference between the two successive latencies. What can be done with that difference is left open in RFC3393. Jitter is a property of the distribution of those differences, and RFC leaves open as to how to deal with distributions.

For example, one could compute an RMS value for the IPDV. To do so, we would have to square the difference for each packet, sum up the squares and divide the sum by the number of squares, then take the square root of the result. Note that this involves a multiplication for each packet, which is computing-intensive. Alternatively it is also possible to look at the distribution of the difference and compute a mean and variance, which is similarly computing-intensive. Instead, one could also take the absolute value of each difference, sum up the absolute values and divide by the number of differences, which yields something very close to the jitter above and is also less computing-intensive.

5.5.1 Compensation for Clock Drift

In an attempt to resolve the impact of unsynchronised clocks on our results, we estimate clock drift and use it to correct the clocks of the transmitter and receiver. Assuming that the cumulative time difference between the clocks of the initiator

beacon and responder beacon e_t is not constant, we can model the relative clock drift as a linear function such that:

$$e_{t,i} \quad = \quad e_{t,0} + i(d_p). \tag{5.8}$$

where d_p is the estimated relative clock drift between the transmitter and receiver. The value of d_p is thus positive if the receiver's clock runs faster than the transmitter.

Note that we cannot measure $e_{t,0}$. However we can estimate the relative clock drift d_p. For simplicity, assume that all n packets were received, and that the jitter was small compared to the total relative clock drift in an experiment. In a situation like this, per-packet relative clock drift is given as:

$$d_p \approx \frac{(r_{n-1} - r_0) - (t_{n-1} - t_0)}{n-1}. \tag{5.9}$$

We can also mitigate the effect of jitter on the above estimate by averaging d_p over $m > 1$ packet pairs as below:

$$d_p \approx \frac{\sum_{i=0}^{m-1}[(r_{n-m+i} - r_i) - (t_{n-m-i} - t_i)]}{m(n-m)}. \tag{5.10}$$

where the relative clock drift d_p is positive if the receiver clock ticks faster than the transmitter clock, and negative otherwise.

To apply d_p to the clocks of hosts, consider a situation where two beacons A and B exchange UDP packet streams such that beacons transmit packets at regular intervals. Consider a situation where each packet encounters a constant latency. If this condition is met: $d_p(A) > 0$ at A and $d_p(B) < 0$ at B, the clock at A runs faster than at B. Note that $d_p(A)$ is the measurement of d_p at A, whereas $d_p(B)$ is the measurement at B.

Table 5.2: Possible Results for Clock Drift Compensation

	$d_p(B) > 0$	$d_p(B) < 0$
$d_p(A) > 0$	congestion	Clock at A faster
$d_p(A) < 0$	Clock at B faster	not possible

If this condition is met: $d_p(A) < 0$ at A and $d_p(B) > 0$ at B, then the clock at B runs faster than at A.

Lastly, consider a situation where there is a bottleneck between A and B. As a result, packets accumulate and leave the bottleneck less frequently than they arrive. We may model this as a "first in, first out (FIFO)" queue whose processing rate is below the arrival rate. Now both A and B observe that the arrival times of packets are further apart. This results in positive d_p values at both ends. In this case, network congestion can be detected with: $d_p(A) > 0$ at A and $d_p(B) > 0$ at B.

In any situation, it is not possible for the estimated relative clock drift between the transmitter and receiver to be $d_p(A) < 0$ at A and $d_p(B) < 0$ at B.

5.5.2 Transit Jitter with Clock Drift Correction

In the first two situations that we have discussed in Section 5.5.1, we may attempt to compensate the clock drift estimate in our jitter estimate. As a result, Eq. 5.2 becomes:

$$D_i + e_{t,i} = r_i - t_i. \tag{5.11}$$

Thus

$$D_i + e_{t,0} = r_i - t_i - i(d_p). \tag{5.12}$$

Thus our clock drift corrected transit jitter is defined as:

$$J_{tc} = \frac{\sum_{i=0}^{n-1} \phi(i) \left| r_i - t_i - i(d_p) - \frac{1}{n'} \sum_{j=0}^{n-1} \phi(j)(r_j - t_j - j(d_p)) \right|}{n'} \tag{5.13}$$

When there are unsynchronised clocks at the initiator and responder beacon, we get packet loss, our jitter estimate thus becomes:

$$\begin{aligned} J_{tc} &= \frac{\sum_{i=0}^{n-1} \left| r_i - t_i - i(d_p) - \frac{1}{n} \sum_{j=0}^{n-1} (r_j - t_j - j(d_p)) \right|}{n} \\ &= \frac{\sum_{i=0}^{n-1} \left| r_i - t_i - i(d_p) - \frac{1}{n} \sum_{j=0}^{n-1} (r_j - t_j - d_p \frac{1}{n} \sum_{j=0}^{n-1} j) \right|}{n} \\ &= \frac{\sum_{i=0}^{n-1} \left| r_i - t_i - i(d_p) - \frac{1}{n} \sum_{j=0}^{n-1} (r_j - t_j) + d_p \frac{n-1}{2} \right|}{n} \\ &= \frac{\sum_{i=0}^{n-1} \left| r_i - t_i - d_p(\frac{n-1}{2} + i) - \frac{1}{n} \sum_{j=0}^{n-1} (r_j - t_j) \right|}{n}. \end{aligned} \tag{5.14}$$

5.5.3 Clock Offset in Bidirectional Experiment

We considered the case where the first packet received at the responder is not the first packet transmitted by the initiator. Similarly, the first packet received by the initiator may not be a response to the first transmitted packet from the responder.

The offset correction thus determines the relative offset between the initiator and responder clock as:

$$\begin{aligned} \Delta_T &= t_c(m') - (m' - m) \times 20 \text{ ms} \\ &\quad - \frac{r_b(m') - (m' - m) \times 20 \text{ ms} + t_a(m)}{2}. \end{aligned} \tag{5.15}$$

where m is the serial number of the first initiator packet seen by the responder, m' is the serial number of the first responder packet seen by the initiator, $t_a(m)$ is the time of transmission (in initiator time) of the first initiator packet m that is seen by the responder, $r_b(m')$ is the time of receipt of packet m' from the responder at the initiator (in initiator time), and $t_c(m')$ is the transmit time of that packet in responder time.

When Δ_T is positive, we interpret this as the responder clock runs ahead of the initiator.

5.6 Coping with Jitter

As discussed in Section 5.2, ISPs often use QoS to mitigate against the impact of jitter on VoIP quality. This technique prioritises voice packets to speed up their processing by routers. However the use of QoS requires Internet routers to be configured with this technique.

Unfortunately, this is not possible on many long distance international paths as the Internet consists of many networks that are not controlled by a single ISP or organisation. A more practical approach would be to use the jitter estimates that we discussed in Section 5.4 to determine the condition of links. This allows VoIP applications to adjust their buffer size or the packet transmission rate to suit the network load.

Such a functionality was investigated by the authors in [105] to determine the appropriate buffer size for VoIP applications. On top of this, the authors also proposed a feedback mechanism where the receiver estimates voice quality using a combination of the ITU-T perceptual evaluation of speech quality (PESQ) and E-model [106]. With the feedback information, VoIP applications can better adapt to the condition of links to improve performance.

In Section 5.2, we mentioned that routers use routing algorithms to compute the best path from source to destination. However, these algorithms uses concepts such

as shortest path to destination, speed of links, and estimated delay to determine the preferred path. The concept that is missing is the inclusion of jitter in the computation of the best path.

Without jitter, a router may use a busy link for the transmission of packets that leads to network congestion and packet loss. Li et al. proposed a jitter-aware algorithm for wireless networks [107], where routes in a routing table are assigned different weights. A route with low jitter is provided with a higher weight and therefore increases its chance of becoming the preferred route. By integrating jitter to improve the route decision of routers, we reduce the impact of network congestion on both jitter and non-jitter sensitive traffic. Additionally we improve the travel time for both jitter and non-jitter sensitive traffic in long distance paths.

It is also possible to improve the forward error correction (FEC) technique to reduce the unnecessary duplication of voice information [108]. A study by Feng et al. [109] shows that sending enough duplicate FEC packets over low bandwidth wireless links avoids unnecessary exchange of information. Even though this may be a problem when there is high packet loss, the reduction in the number of extra packets contributes to improvement in throughput for high latency wireless links.

Lastly, we can also improve throughput in links with TCP/NC. This is the technique that we will use in this thesis to provide FEC for transmissions over high latency satellite connections. Further information on TCP/NC is provided on Chapter 11.

5.7 Summary

In this chapter, we reviewed the processes involved in the creation of voice packets and we also described the network processes that contribute to the formation of jitter. Then we presented methods of jitter estimation and various solutions for jitter. In Chapter 6, our focus is on the method that we use to estimate network quality and voice quality.

CHAPTER 6

Estimation of Network Quality, Voice Quality, and Buffer Requirements

In this chapter we look at ways of assessing the quality of Internet connectivity over time. We put jitter in the context of network quality, followed by a brief history of complexity measures and entropy estimators, and an introduction to T-entropy. Next, we describe the mapping of inter-arrival times to strings, and the computation of T-entropy from such strings. Then we review the use of jitter in the ITU-T E-model to determine voice quality with an estimated MOS which can be computed from our `beacon` *data. Finally, we discuss the estimation of buffer requirements in our simulated Voice over TCP experiments.*

6.1 Motivation

Traditionally, jitter has been used by network engineers to determine the current state of a network. Essentially, high jitter is a sign of poor quality, whereas low jitter is a characteristic of a good quality network. Packets with high jitter, have travel times significantly affected by router queues whose lengths vary substantially over time.

However, jitter only tells us about the final variations in the arrival time of packets when they are received at the destination. The missing information is the event in the network that triggered the jitter. We may assume that packets were delayed by varying amounts of time in long router queues. Alternatively, packets may have taken different paths and therefore experienced variable latencies depending on the path they took. We refer to this latter type of jitter as *systematic jitter* where the inter-arrival times of packets form a pattern.

Motivated by the desire to determine the type of jitter, we employ T-entropy to find out the degree of patterning in the inter-arrival time of packets. By comparing jitter and patterns in the inter-arrival time, we can determine whether the jitter we observe is queue-induced or systematic jitter. In order to better understand T-entropy, we review complexity measures in the next sections.

6.2 Notation

In this chapter, the notations on complexity measures and T-entropy follow Günther's and Eimann's theses [110, 111]. We also reuse some of the notation that we discussed in Chapter 2. Other notation used in the following section is as follows:

M	A set of characters e.g., {A,B,C,D} for mapping the inter-arrival times of packets
T	List of inter-arrival time of packets in milliseconds
s	A string produced from the mapping of the inter-arrival time of packets
n'	Number of successfully received packets

m	Serial number of the first initiator packet seen by the responder
m'	Serial number of the first responder packet seen by the initiator
$t_a(m)$	Time of transmission (in initiator time) of the first packet to the responder
$r_b(m')$	Time of receipt (in responder time) of the first packet from the initiator

6.3 Complexity Measures

In compression techniques that follow the popular Lempel-Ziv parsing model, complexity estimators may be thought of as the engine that measures the number of elementary steps required to construct the input strings [111].

The best known complexity measure for strings is the *Kolmogorov-Chaitin complexity* [112]. This complexity deals with descriptions of strings using a fixed description language. However, Kolmogorov-Chaitin complexity is not computable [112], and this motivates the development of a number of computable complexity estimators.

Starting in 1976, Abraham Lempel and Jacob Ziv proposed a computable string complexity known as the *Lempel-Ziv production complexity*, *LZ production complexity* or simply as *LZ76* [113, 114]. Given an alphabet A and a string $x \in A^n$ (set of all strings of length n over alphabet A), this estimator parses x successively into a list S of substrings of x, such that each substring consists of a previously parsed part of x followed by an "innovation" symbol. Each list item in S is called a *production step* in the *production* (of x). The number of production steps (i.e., $\#S$) is the LZ production complexity.

However, LZ76 is $O(n^2)$, as every substring search for each production step starts from the first symbol of x. This becomes a limitation in the use of LZ76 as a parser for a compression algorithm. The subsequent development of the parser used in the LZ77 [115] compression algorithm addresses this issue. In LZ77, Lempel and Ziv modified their algorithm with the use of a sliding window of fixed size w [113]. This improves the search time to $O(wn)$, which permits the use of the algorithm on longer x.

The price for the improvements made in LZ77 is that the algorithm "forgets" previously seen patterns if they do not reappear at least every m symbols. This limitation of LZ77 led to the development of LZ78 [116] to perform a search for known prefixes p_i in a dictionary $D = \{\lambda, p_1, ..., p_k\}$ against an input string $x \in A^*$.

The limitation in the use of LZ78 is a restriction on its substring search. There are only certain positions in the string to start a search from and therefore some patterns may not be recognisable because previous occurrences of the pattern do not start at a location that LZ78 had previously started from. Despite the limitations of LZ77 and LZ78, they are the foundation on which popular compression applications such as *zip* and *gzip* are built.

Another complexity estimator, *T-complexity*, was developed by Titchener [117, 118, 119, 120] for the purpose of expressing the number of steps required to construct a string. The steps consist of distinct codewords from a series of prefix-free codes that are generated via a recursive copy-and-append process known as T-augmentation.

However, T-complexity is non-linear in the length of the input string, but it can be subjected to a linearisation via inverse logarithmic integrals (see next section). This linearised version of the T-complexity estimator is known as *T-information*. The gradient of the T-information is known as *T-entropy*.

Since this thesis focuses on the application of T-entropy to determine the degree of patterning in the inter-arrival times of packets; readers interested in the T-augmentation and T-decomposition algorithms are directed to Günther's and Eimann's theses [110, 111].

6.4 Entropy Estimators

Entropy estimators are another family of information measures. The term *entropy* was introduced by Rudolf Clausius in relation to thermodynamic processes [121]. Entropy was later adopted by Boltzmann in the field of statistical mechanics in

physics for the purpose of interpreting thermodynamics at a microscopic level [122]. As a result, Boltzmann was able to interpret microscopic systems and describe the state of particles in thermodynamic processes.

Using a similar concept to Boltzmann entropy [123], Shannon developed his own entropy as a measure of uncertainty in predicting subsequent outcomes of a discrete random variable [124]. The prediction is made for a set of known outcome probabilities $\{p_1, \ldots, p_n\}$.

Furthermore, Shannon explored the possibility of deriving p_i as an unknown variable from observations. Simple Shannon entropy is based on the assumption that symbols generated from an information source are not correlated. The problem with this assumption is that it does not apply to many real information sources (such as the English language, for example).

Shannon thus extended his method to include *n-gram* entropy [125], an entropy estimate where symbol strings are broken up into blocks $1 \ldots L$ of size n called *n-grams*. The frequency of occurrence of *n-grams* is used to estimate p_i for each possible *n-gram*.

In an attempt to be able to compare Shannon's entropy with T-complexity, Titchener proposed the inverse logarithmic integral $li^{-1}(x)$ as a function to "linearise" T-complexity. The T-information of a string x is thus defined as the inverse logarithmic integral of the T-complexity of x:

$$I_T(x) = li^{-1}(C_T(x)), \tag{6.1}$$

where the logarithmic integral $li(x)$ is defined in [126] as:

$$li(x) = \int_0^x \frac{dt}{\ln(t)}. \tag{6.2}$$

where $li(x)$ has a singularity at $x = 1$ and a positive zero at $x = 1.4513692\ldots$.

Table 6.2: Sample arrival times of Packets

Packet no. i	r_i
0	- (packet lost)
1	1348001461.691
2	1348001461.717
3	1348001461.738
4	1348001461.760
5	- (packet lost)
6	1348001461.821
7	1348001461.828
8	1348001461.848

Furthermore, Titchener defined average T-entropy as follows:

$$H_T(x) = \frac{I_T(x)}{|x|}. \tag{6.3}$$

The advantage in the use of T-entropy over Shannon's *n-gram* entropy is that the computation of T-complexity involves the parsing of a string progressively over a family of increasingly complex variable-length codes. These variable-length codes synchronise into the patterns in a given string. The synchronisation into patterns mean that they can be located anywhere in the string, whereas for *n-gram* entropy, the positions of patterns need to be multiples of n. In the next section, we discuss the use of T-entropy to determine whether there are patterns in the inter-arrival times of packets.

6.5 Mapping of Inter-Arrival Times

In order to provide us with a measure of "disorder", we map the inter-arrival times into letters, concatenate the letters into a string x and then we compute the T-entropy of x. In this section, we discuss how this is done.

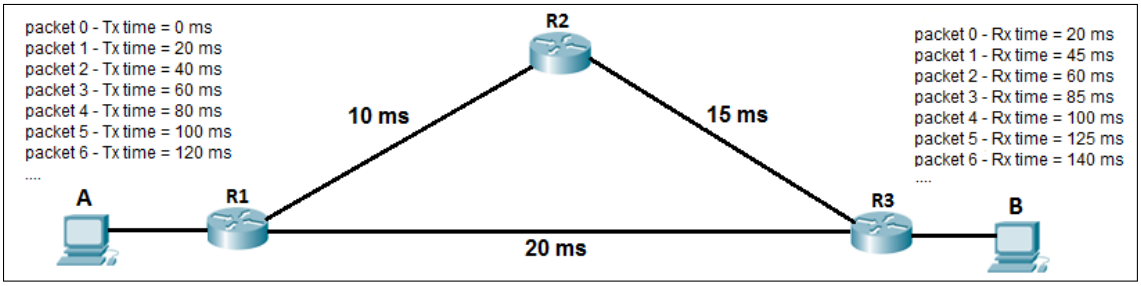
Firstly, we define bin sets for mapping the inter-arrival times of packets by choosing $b-1$ bin boundaries $= \beta_1, ..., \beta_{b-1}$ for b bins. Then we compute the values $I_i = r_i - r_{i-1}$ where $\Phi(i)\Phi(i-1) = 1$. With the inter-arrival times, we find k_i such that

$$k_i = \begin{cases} 0 & \text{if } I_i < \beta_1 \\ j-1 & \text{if } \beta_{j-1} \leqslant I_i < \beta_j \\ b-1 & \text{if } I_i > \beta_{b-1} \end{cases} \tag{6.4}$$

In Eq. 6.4, we concatenate k_i to yield x. Note that in our real experiments we use odd bin numbers $b = 3, 5, 7$, and 9 so the central bin can be centered around the mean.

We compute the entropy of the strings obtained from the mapping of the inter-arrival times and combine the entropy value with jitter to determine *systematic* and *queue induced jitter*.

Figure 6.1: Numbered packets that are load balanced alternatingly across links from R1 to R3 and from R1 to R2 and R3



Note: We use 20 ms as the inter-transmission time of packets

Systematic jitter typically occurs when packets are load balanced across links. Assume in Figure 6.1 that the router transmits even-numbered packets on the link between R1 and R3, while odd numbered packets are transmitted on the link from R1 to R2 and to R3.

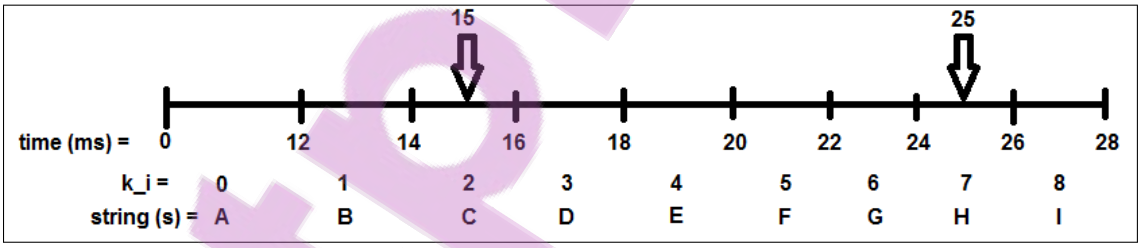
When the first packet (packet 0) is transmitted from host A to R1, the router sends this packet on the direct link to R3. The estimated travel time on this link is 20 ms and this is reflected on the arrival time of packet 0. After 20 ms of inter-

transmission time, the next packet (packet 1) is transmitted to R1. This time, it is transmitted on the direct link between R2 and R3, where the estimated travel time is 25 ms. The arrival time of packet 1 is thus at 45 ms. The rest of the packets also experience variable delays depending on their route.

At host B, we compute the inter-arrival time of the packets as follows:

- $r_1 - r_0 = 45 - 20 = 25$
- $r_2 - r_1 = 60 - 45 = 15$
- $r_3 - r_2 = 85 - 60 = 25$
- $r_4 - r_3 = 100 - 85 = 15$
- $r_5 - r_4 = 125 - 100 = 25$
- $r_6 - r_5 = 140 - 125 = 15$
- ....

Figure 6.2: Mapping inter-arrival times into strings using 9 bins



In Figure 6.2, we use $b = 9$ bins, so bins are represented by $k_i = 0, 1, 2, 3, 4, 5, 6, 7, 8$. Since, the inter-arrival times of our packets in Figure 6.1 are either 15 or 25, we get only symbols 2 and 7 for k_i. By mapping the inter-arrival times $I_0 = 15, I_1 = 25 ... I_5 = 25$ into string x we get:

$x = 272727....$

Since the above string forms a pattern, we get a low entropy. However in this case, jitter still remains high because of the variable length delays in the inter-arrival

time of the packets. In a situation where we detect high jitter and low entropy in our data, we interpret this as systematic jitter.

On the other hand, queue-induced jitter occurs during the transit of packets through routers. During this time, routers receive packets on the input queue, process them to determine the next hop and forward packets to the output queue for transmission to the next hop. The overall time that packets spend in the input and output queues contribute to the queue-induced jitter.

The advantage of using entropy to investigate inter-arrival times is that we do not need to know in advance *which* patterns to look for: as long as there are repeating patterns, they will reduce entropy.

6.6 Adaptive Entropy

In the previous section, we generate an arbitrary number of bins in advance before we look at the inter-arrival times of packets. The problem with this method is that the boundaries are fixed between different experiments. When the jitter is small, our inter-arrival times almost always end up in the center bin. For large jitter, our inter-arrival times end up predominantly in the first and the last bin. In both cases, we get the same symbol and we interpret this as systematic jitter.

For example, assume that we use 3 bins where we map inter-arrival times to strings based on the bin boundaries: $\beta_0 = 19$ ms, $\beta_1 = 21$ ms. Yet it is also possible to use any other set of boundaries (other than 19 and 21) with respect to the mean of 20 ms.

To address this issue, we assume that the inter-arrival times are normally distributed and use the quantile function of the Gaussian distribution [7] to dynamically allocate β_j for our bins. We use the following steps to compute our adaptive entropy:

1. Compute the mean of the inter-arrival times.

$$\mu = \frac{\sum_{i=1}^{N}(r_{i+1} - r_i)}{N} \tag{6.5}$$

2. Compute the standard deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(r_{i+1} - r_i - \mu)^2}{N-1}} \tag{6.6}$$

3. Use the quantile function of the normal distribution with mean μ and standard deviation σ [7] to compute b boundaries β_j

$$F^{-1}(p) \quad = \quad \mu + \sigma\sqrt{2}erf^{-1}(2p-1), p \in \left(\frac{1}{b}, \frac{2}{b}, ..., \frac{b-1}{b}, \right). \tag{6.7}$$

   Here, *erf* is the error function defined in [7]. With the quantile function, we can derive boundaries for b bins (i.e., $\beta_j = F^j(p)$).

4. Compute the k_i

5. Compute the entropy of the resulting string of concatenated symbols.

When the inter-arrival times of packets are normally distributed, adaptive entropy will be high because in this case the inter-arrival times will be uniformly distributed across our bins, which we associate with queue-induced jitter.

In contrast, low adaptive entropy is an indication that our packets encountered systematic jitter. Note that adaptive entropy is a *single* observable that we can use to determine the occurrence of queue-induced and systematic jitter. This is different from our previous method, where we compared jitter together with a fixed size bin entropy to distinguish the two jitter types.

6.7 Estimated Mean Opinion Score (MOS)

Trying to deal with the challenges of jitter and latency, some authors proposed improvements in QoS, to improve the performance of VoIP applications (e.g., [127, 128]), while others focused on improvements for voice codecs (e.g., [129]). However, any such solutions require ways to verify their effectiveness.

One way of assessing the quality in the use of VoIP applications is the MOS. This method rates the user's experience of voice quality on a scale of 1 to 5 [130]. MOS is widely used as a measure in the improvement of QoS techniques and voice codecs. Measuring MOS requires collecting actual opinions from a large number of individuals in a controlled environment. Alternatively, it can be estimated from jitter, latency and packet loss [131]. In this thesis we demonstrate that we can use the data from our beacon network, or more specifically from our bidirectional UDP experiments, to estimate MOS.

As described in Chapter 4, the underlying design of our beacon network is made up of two nodes: *initiator* and *responder*. In a bidirectional experiment, the initiating node sends its first packet to the responder node. Upon reception of the first packet, the responder performs two tasks:

First, the responder retrieves timing information from its local clock and records the receive time r_m of the first packet. Secondly, the responder retrieves the transmit timestamp t_m and the serial number from the payload of the received packet and records them.

Responding to the first packet, the responder transmits a packet with the same format back to the initiator. When packets from the responder arrive at the initiator, the process of recording the transmit and receive time is repeated there.

Our calculation of the estimated MOS is based on [132] and it starts with average

latency as follows:

$$\bar{L} = \frac{\sum_{i=0}^{n-1} \phi(i)(r_i - \Delta_T)}{n'} \tag{6.8}$$

where t_i and r_i are transmit and receive timestamps for the i'th packet, Δ_T is the clock offset correction, n is the number of packets transmitted and n' the number of successfully received packets, as defined in Section 5.3 $\phi(i) = 1$ if the i'th packet was received and $\phi(i) = 0$ if not. Note that in Section 5.5.3, we concluded that a positive Δ_T means that the responder clock runs ahead of the initiator.

The jitter we use for the MOS estimate is the average absolute difference between successive latencies l_{i-1} and l_i [132]:

$$J = \frac{\sum_{i=1}^{n-1} \phi(i)\phi(i-1)|l_i - l_{i-1}|}{n''} \tag{6.9}$$

where n'' is the number of successive packet pairs received, i.e.,

$$n'' = \sum_{i=1}^{n-1} \phi(i)\phi(i-1). \tag{6.10}$$

Based on the formula in [132], we compute the effective latency as:

$$L_{\mathit{eff}} = \bar{L} + J \times 2 + 10 \tag{6.11}$$

Then, we use the recommended values in [132] to obtain an R-value for different latencies based on our effective latency:

$$R = 93.2 - \frac{L_{\mathit{eff}}}{20} \tag{6.12}$$

for $L_{\mathit{eff}} < 80$ ms and

$$R = 93.2 - \frac{L_{\mathit{eff}} - 60}{5} \tag{6.13}$$

for $L_{\mathit{eff}} \geq 80$ ms. The final step before the MOS conversion of the R-values accounts for packet loss:

$$R' = R - \frac{(n - n') * 250}{n}. \tag{6.14}$$

We then estimate the MOS with the R'-factor conversion formula in [132].

$$MOS = 1 + 0.035 \times R' + 0.000007 \times R' \times (R' - 60) \times (100 - R'). \quad (6.15)$$

We will be using this method for our MOS estimation results in Section 9.2.2 of Chapter 9.

6.8 Voice over TCP

Studies of Internet traffic distribution indicate that TCP is the most widely used protocol on the Internet [133]. In contribution to the uses of TCP, VoIP applications such as Skype switch to TCP when a firewall blocks their UDP ports. However, TCP uses congestion control mechanisms to reliably deliver information. For VoIP applications, these mechanisms can delay the transmission of voice information and introduce extra jitter.

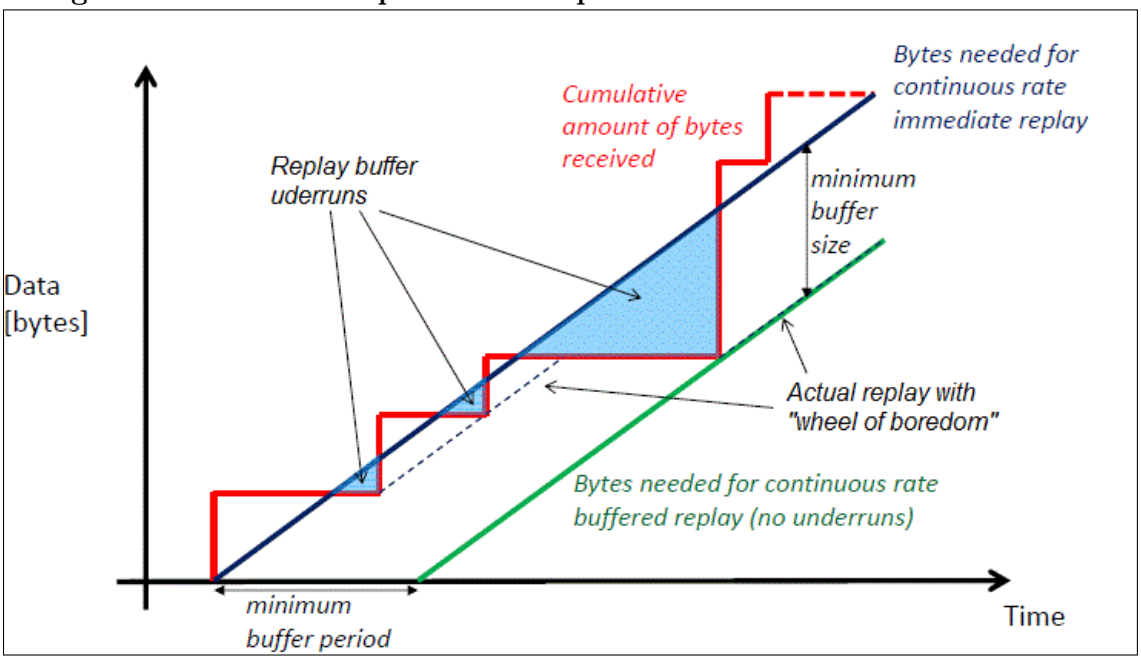
As a solution to this problem, VoIP applications use a FIFO-style queue (*replay buffer* or *jitter buffer*) to temporarily hold chunks of voice data. The use of the buffer enables VoIP applications to temporary hold voice information and replay it to the end user at a constant rate.

However, there are two problems with the replay buffer. When the buffer size is too small, there is a risk for the buffer to run empty, causing the VoIP application to await the arrival of the next sets of voice chunks for replay (*buffer underrun*). This can lead to breaks in voice replay or "wheel of boredom" events in video. On the other hand, large buffer sizes may cause VoIP applications to delay replay unreasonably.

Figure 6.3 shows such buffer underruns in the form of the blue triangles where the red graph (amount of data received) crosses the dark solid blue line (amount of data required for continuous replay).

To avoid breaks or the *wheel of boredom*, the source must transmit enough data

Figure 6.3: Relationship between required buffer size and available data



Note: This diagram was reproduced from Figure 1 in [134]. The red graph (staircase) is the cumulative amount of data that has been received, the vertical parts of the red graph are the times at which chunks of data are received. The linear solid dark blue graph is the number of bytes that the replay consumes, assuming that we replay from the moment that the first packet is received. The green graph is the amount of data that the receiver must have received when using a buffer in order to guarantee continuous replay without buffer underruns.

at a rate that enables the destination node to continuously process information. For VoIP applications such as Skype, the typical snippet size is approximately 115 bytes at an inter-transmission interval of 20 ms for communication over high speed networks. On a low bandwidth network, Skype uses smaller data snippets and lower transmission rates (e.g., 45 bytes every 60 ms [79]). Our simulated VoIP over TCP experiment investigates the performance of such traffic on links between nodes. We will be presenting results on this experiment in Section 9.2.4 of Chapter 9.

As mentioned in Chapter 1 and Chapter 4, our software is capable of simulating other real-time and streaming applications. To simulate other applications, our software require application properties such as the transmission rate and size of data, which are configurable input parameters.

6.9 Summary

In this chapter, we introduced two uses of jitter to determine network and voice quality: Firstly, in conjunction with entropy, to determine whether there is systematic or queue-induced jitter. We also propose adaptive entropy as an alternative method for distinguishing between systematic and queue-induced jitter.

In addition, we outlined the use of jitter as part of estimated MOS and looked at the method we use to estimate buffer requirements for VoIP applications that have to use the TCP protocol.

In the next chapter, we will now discuss the challenges and limitation in the development of our software, with focus on organisational policy, security considerations, software requirements, and limited resources. In addition, we will also present the limitations of our software.

CHAPTER 7

Challenges and Limitations

This chapter is divided into two parts. In the first part, we review the challenges on the implementation of our software. These challenges are based on organisational policy, security considerations, software requirements, and availability of resources, and are important for our ability to run long term experiments. In addition, we also reflect on the steps we took to overcome these challenges. We discuss the limitation of our software, and the methods that we use to determine path characteristics such as latency, path of packets, and voice quality.

7.1 Organisational Policy

In Chapter 4, one of our requirements is to open certain ports on firewalls to allow packets destined to and from our `beacon` software. In most deployments, this is challenging when there are organisational policies that restrict external access to hosts inside our research partners' networks.

In situations like these, the troubleshooting and installation of our software de-

pends on the individuals that support our research *from the inside* of these organisations. Depending on the time it takes to resolve an issue, we can miss periods of data collection. Our solution to this problem is to provide our software as an installation package for Ubuntu and BSD as well as instructions on the installation process of our software. This makes it easy to install our software and to troubleshoot problems without the need for additional support.

Furthermore, our software requires root/administrative permissions to run the raw socket component of our software. This is a problem because it is also possible to use root permissions for launching attacks against internal hosts in our research partners' networks. The challenge presented in this problem is that we can only convince our partners to assist by gaining their trust.

We achieved some of this trust through face to face meetings via Skype and by attending conferences, and in some cases we had to compromise in the range of functionality we were able to implement. In some cases, we visited our research partners. For example, we visited the University of Tokyo, University of Johannesburg, MIT. In a number of organisations that we are working with, the people we partner with in research are not necessarily the people that control systems and networks. In cases like these, we also need to build trust with the engineers that complete the configuration for us.

7.2 Software and Hardware Issues Surrounding Deployments

In Chapter 4, we mentioned that our software is executed only at the time of an experiment. Furthermore, we also mentioned that the local clocks of hosts must be synchronised so that experiments are executed at the scheduled time. In the case where the clocks remain unsynchronised, the transmitter or receiver may not be ready at the scheduled time to perform the experiments. Consequently we get "packet loss" at the beginning or end of experiments which we must be careful not to interpret as network packet loss.

In Section 5.5.1, we discussed the issue of synchronising the local clock of hosts with publicly available NTP servers. For hosts located in low latency networks, the delays from synchronising their time to the closest accurate NTP server are usually in the sub-millisecond range. However, hosts located in high latency networks do not have NTP servers that are synchronised with atomic clocks, so they depend on remote publicly available NTP servers for clock synchronisation. This becomes a challenge because the high delays from satellite links cause errors in the time information exchange between NTP servers and hosts in high latency networks.

Ideally, we would want to deploy atomic clocks in locations where there are high latency satellite connections. However, atomic clocks are prohibitively expensive. An alternative presented itself in the form of a project at the University of Auckland to develop a highly accurate time server based on the Global Positioning System (GPS) [135]. As a result, we were able to procure a small number of low cost NTP server solutions that run on a *Raspberry Pi* [136]. At a later stage of this project, we deployed some of these NTP servers near some of our nodes.

Previously in Chapter 4, we mentioned that one of our requirements is a dedicated physical host for the installation of our software. We also mentioned that a physical host provides applications with direct access to a real-time clock for time-stamping purposes. While such a requirement is affordable for some of our partners, others find it hard to provide us with a dedicated physical host because they do not have the financial resources.

As an alternative platform, we procured a number of mini PCs [137] and configured them with Ubuntu's server operating system. On top of this, we installed our software and third party applications including `ssh`, `rsync` and `unbound` for supporting our application. These mini PCs are lightweight and affordable, and can be shipped via standard air mail to some of our research partners.

7.3 Limited Resources

As in-kind contributions, some of our partners provided us with physical machines and/or a space in their network for hosting our software. However these networks change from time to time, e.g., triggered by an upgrade of their gateway router, blocking of certain ports, or installation of new devices. As a result, there is a need for us to actively monitor hosts and to advise our partners of any connectivity issues.

Host monitoring has been a challenging task for us because we depend on the information contained in the headers of log files and, in some cases, on the number of packets received to detect issues. Instead of only reading the content of files, we use the reporting time of beacons to determine connectivity issues. In an experiment, a node will terminate our software and write the observed information into log files. These log files are then reported back to our server one minute after the experiment.

By monitoring the reporting time of a beacon, we can determine whether the beacon is overdue and therefore potentially offline or unreachable. From the last reporting time, we can also determine the status of beacon hosts. Once a host is identified as unreachable, we use the header of the log files to determine the problem and to recommend an appropriate solution to our partners.

In Chapter 4, we mentioned that we require an Internet connection for the exchange of packets, management of experiments, and for back-haul of data after an experiment. While this requirement may not be an issue for hosts located in an area with a high speed Internet connection, some of our hosts in developing countries use satellite Internet connections which have high latency and low bandwidth. This became a challenge because we needed to ensure that our software was not a burden for the hosting network.

Responding to this challenge, we made the following changes to the configuration of experiments and the way we back-haul data: As described in Chapter 4, we configured VoIP style experiments in low bandwidth networks with packet sizes of 45 bytes and inter-transmission times of 60 ms rather than the usual 115 bytes and inter-transmission times of 20 ms. By reducing the packet size and increasing

the transmission rate, we reduced the number of packets and the amount of data that our beacon application transmits over the network. On top of this, such a configuration makes our software less bursty and friendlier to other transmissions on high latency links. At the time of writing, our beacons in Kiribati, Tuvalu, and the Solomon Islands ran under such non-standard configurations.

Furthermore, we also reduced the amount of data that we back-haul from hosts. In Chapter 5, we mentioned that we activated the compression and incremental back up options of `rsync` to save bandwidth, and to ensure that we back-haul only new data from experiments.

7.4 Limitations of the Software

Naturally, our software has its limitations. For example, there are limitations in the methods that we use to compute latency, identify changes in the path taken by packets, and for estimating voice quality with an estimated MOS.

The biggest problem we encountered with our computation of latency is the fact that we do not know the exact clock difference between the two endpoints. Even assuming that we solved this problem, we still do not know how much of the delay is caused by packets spending time in router queues. Moreover, as we cannot determine latency on a per-router basis, our data cannot be used to determine specific locations in the path between nodes where our packets encountered the most delay.

The functionality of determining per hop latency is already integrated into some active measurement utilities such as `Pingplotter` [132], `Ping` [33], and `Traceroute` [138]. These utilities use ICMP to determine per hop latency from the RTT of packets. In our software, we did not consider this functionality because we were more interested in variations of overall delay (jitter) and the performance of real-time applications rather than the reachability of hosts. On top of this, if we had added ICMP to our software, it would have added complexity to the configuration of experiments.

Furthermore, we also discussed in Chapter 4 the use of the TTL field in the IP header to document the changes in network infrastructure between nodes. With the TTL value of packets, we can detect changes in the path of packets. The limitation of this method is that the TTL value on its own is not enough for us to determine where in the path the change occurred. There are many events that could trigger changes: These include load balancing, removal of a router, or disconnection of a route that triggered packets to be rerouted using an alternative path.

However, we are not interested in the individual causes of path changes, but rather in how often the path changes and how the path changes relate to jitter, out-of-order arrival and our estimated MOS.

As discussed in Section 2.2, the `Traceroute` utility uses ICMP to map the likely path of packets between nodes. We did not integrate this functionality, simply because we are not interested in the precise path. The data that we collect with our `beacon` software will enable us to determine how often the path between nodes changes over time.

Again in Chapter 4, we described our UDP experiment as mimicking a situation where two nodes repeatedly exchange VoIP calls over an extended period of time. The limitation of our simulated VoIP experiments is that we ignore the delay that is introduced when a VoIP application uses a codec to compress voice information.

Furthermore, we did not consider the behaviour of voice codecs in the development of our software for the following reasons. Firstly, the delays from voice codecs are additional to the latency that the network produces. Secondly, we are interested in the minimal latency that packets experience. Our experiments does not look at possible additional delays from the use of voice codecs.

Also, if we had considered voice codecs in our application we would had to increase the number of experiments to ensure that we include different voice codecs. In addition, voice codecs change over time and we cannot simply reconfigure experiments to record new data every time there is a new codec or codec version. Moreover, we run a number of experiments with research partners in the Pacific and we cannot ask them to donate much more capacity for our experiments because of

the limited capacity of their satellite connections.

On top of this, we would also run out of experiment time slots if we were to include different codecs for experiments between a beacon pair. As a result we would end up configuring beacons to exchange packets with at most two other beacons. However we wanted to ensure that the number of partners for a host can be larger than 3.

7.5 Summary

In this chapter, we have reviewed the challenges that we encountered in the deployment of our `beacon` software. Furthermore, we also reflect on the limitations of our software and the reasons that we decided to accept these limitations. In the next chapter, we will discuss the collection of data in log files and the processing of log files to determine path characteristics.

CHAPTER 8

Backup and Processing of Log Files

In this chapter we describe the methods that we use to process our log files. We begin by reviewing the experiments we perform with our `beacon` *software. Then we reflect on the network design of the servers that we use for collecting and archiving our data. Lastly, we present the methods for processing data to determine path characteristics such as jitter, average latency, packet loss, entropy, estimated Mean Opinion Score (MOS), out-of-order arrivals, path length, and buffer requirements for voice over TCP.*

8.1 Data Backup Topology

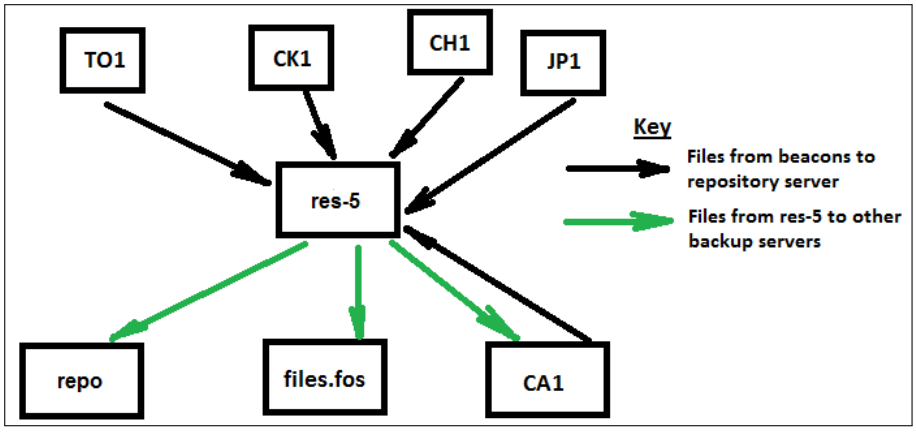
To date, we have installed our `beacon` software on 33 hosts located in 16 countries. In Chapter 4, we mentioned that we pair hosts to perform 6 types of experiments. As an example, we configured our third node in Tonga (TO3) as a beacon server to perform experiments with our first node in New Zealand (NZ1). Based on our

description of experiments in Chapter 4, the first experiment involves NZ1 initiating a unidirectional (one-way traffic flow) VoIP-like experiment with TO3 using UDP. In this experiment, NZ1 transmits packets and TO3 receives and records them.

The second experiment is a bidirectional VoIP-style experiment with UDP. In the bidirectional experiment, both NZ1 and TO3 transmit packets concurrently, with NZ1 initiating the transmission of the first packet. The third and fourth experiments are similar to the first and second experiments, except that this time we use TCP to perform VoIP-style experiments (see Chapter 4). Lastly, we use TCP to perform unidirectional file downloads in experiment 5 and bidirectional file exchanges in experiment 6.

By configuring hosts to perform experiments with many more peers, the amount of data that we collect increases proportionally to the number of configured experiments (number of configured peer pairs times 6 types of experiments). In an attempt to ensure sufficient disk space for storage of data and to make redundant copies of our data, we developed a network of backup servers for this purpose. The two basic tools that we use for data back-haul are the `rsync` [139] and `cron` [140] utilities. We use `rsync` to copy data to our servers and we `cron` to schedule the back-haul of data.

Figure 8.1: Design for back-haul from beacons to the main repository server (res-5) and the backup of data to other servers



At the University of Auckland, we installed two file servers (see res-5 and repo in Figure 8.1). We use one of the servers as a landing server (res-5) that receives copies of our data from the beacons, and sends them to our repository server. Furthermore,

we created directories on the landing server for backup of our data. We use one of the directories to receive newer files from hosts and keep an archive of our data in another directory. This server serves as the first line backup of our data.

Moreover, we configured our second server as a repository server (repo). It receives files from the landing server. On this server, we created another two directories. One of the directories was designed to be small and lightweight for us to quickly scan for newer files. The other directory contains all log files that we collect over time. Further details on file processing will be discussed on the next section.

We also store our data on two other servers. The first server is a shared network drive (files.fos). We attach the shared drive as a network attached disk to the landing server and transfer newer files from the landing server to the shared drive.

The other server was provided to us by our research partners at Simon Fraser University in Canada (CA1). The original purpose of the server was to host our `beacon` software there and to perform experiments. As a result, we scheduled the backup and experiment times to occur at different times. In doing so, the bursty traffic from the backup of data does not influence our experiments on the beacon at Simon Fraser University. Similar to the shared disk at the University of Auckland, we created a directory and we back up data to it from the landing server.

8.2 Log File Processing

As files land on the repository server at the University of Auckland, they are scanned regularly and processed to create CSV files containing various observables including jitter, average latency, entropy, packet loss and buffer requirements for voice over TCP. We store data on our repository server, and we make them available via a publicly accessible web interface.

8.2.1 UDP Experiments

8.2.1.1 Jitter

As defined in Chapter 5, jitter is the variation in the travel time of packets. To estimate jitter, there is a need to capture the transmit and receive time of packets. Also in Chapter 5, we mentioned that we did not estimate jitter for TCP applications because the TCP stack controls the transmission time of packets. As a result, our application cannot determine the transmit time and therefore we do not have the information required to compute jitter. However, UDP applications are able to control the transmission time of packets and therefore we can estimate jitter with data from our UDP experiments.

To compute jitter, we scan through the *Tx_timestamp* and *Rx_timestamp* of receive log files and compute latency as the time difference between these timestamps. However our calculation of latency includes errors from the clock difference between the beacon pair. There are two potential causes of error:

Firstly, no two clocks run at exactly the same speed, the clocks of a beacon pair drift during experiments. To deal with this situation, we use Equation 5.13.

Also in Section 5.4.3, we mentioned that when either the source or destination node loses synchronisation, the execution time of either the receiving or transmitting socket occur at different times. As a result, we may miss packets. To compensate for this situation, we use Equation 5.14. We insert the results of Equation 5.13 and Equation 5.14 as clock drift compensated and non-clock drift compensated jitter, respectively, into our CSV files.

8.2.1.2 Average Latency

Our implementation of average latency is based on the RTT of packets. When two beacons exchange packets, A as initiator and B as responder, the RTT that we compute is based on the travel time of packets from A to B and back to A. To

compute the RTT from our log files, we restrict the data that we scan to log files of UDP bidirectional experiments. The reason for this restriction is that we can only record the travel time of packets from A to B and back to A in this type of experiment.

Assume that t_i is the transmit time of the ith packet at A, r_i is the receive time of the ith packet at B, t_k is the transmit time of the kth packet at B, r_k is the receive time of the kth packet at A. We therefore compute the average latency as:

$$L = \frac{\sum_{i=1}^{n-1} \phi_A(r_i - t_i) + \sum_{k=1}^{n-1} \phi_B(r_k - t_k)}{N} \tag{8.1}$$

where n is the number of transmitted packets, and N is the number of packets that were successfully received at A and B. The result from this equation is the average latency that we insert into the CSV files.

8.2.1.3 Entropy Estimate

In Section 6.5, we introduced a technique for mapping the inter-arrival times of packets into a string x. Then we compute entropy from string and make the following observations from the result. Firstly, when the entropy of our string is low, we may conclude that the arrival times of our packets would have been predictable. In other words, our packets arrive more or less on time and as a result we are able to see regular patterns in our string.

However, when there is random delay in the inter-arrival time of packets, we observe fewer patterns in our string. This is reflected in a higher entropy. Our result is a single entropy value per experiment number. As discussed in Section 6.5, we use a range of fixed sized bin boundaries to determine the normal distribution.

Then in Section 6.6, we introduced adaptive entropy and describe the use of flexible bin boundaries derived from the cumulative probability function for the normal distribution. We insert the result from the use of fixed and flexible bin boundaries into the CSV files.

8.2.1.4 MOS Estimate

One of our objectives was to monitor probable VoIP quality on the links between beacon pairs. In Section 6.7, we use the ITU-T E-model to compute an estimated MOS of our data. We begin with Equation 6.8 to compute the average latency of our data. Next we use Equation 6.9 and Equation 6.14 to compute jitter and packet loss. Lastly, we use Equation 6.15 to compute our estimated MOS. We insert the result of this calculation into our CSV files.

8.2.1.5 Out-of-Order Arrivals in UDP

The impact of out-of-order arrivals on the performance of VoIP applications is well known [141]. For this reason, we decided to compute the order of arrival of packets with the following method. As mentioned in Section 4.2.2, we extract the serial number of packets and record them in one of columns in the receive log files (see [80]).

In order to detect out-of-order packets, we scan through our UDP receive log files and compare the *Packet number* with *Rx sequence number* column. When there is a mismatch between the expected sequence and the packet number, we detect this as an out-of-ordered packet and use a counter to keep track of these. Note that the out-of-ordered packet were received but at a later time. After scanning through the log file and detecting all out-of-order packets, we insert the value of the counter into our CSV files.

8.2.1.6 Path Length of Packets

Sometimes, the changes in the path of packets in an experiment are a contributing factor to out-of-order arrivals. In order to detect path changes, we extract the TTL value of packets from the IP header and insert the minimum and maximum TTL value for each experiment run into our CSV file. Note that the initial TTL value of packets varies between operating systems. In our experiments we use the Linux/Unix initial value of 64 [142].

8.2.1.7 Packet Loss

One of the issues we encountered with the processing of log files is inaccuracy in the calculation of packet loss. There are two causes of packet loss in our experiments.

Firstly, when links are congested with packets, router queues fill to full capacity. A common technique employed by routers in such circumstances is to perform tail drop on packets. This causes some of our packets to be lost in transit. For this type of packet loss we simply compute a percentage with the following equation:

$$P = \left(\frac{N}{n}\right) * 100 \tag{8.2}$$

where N is the total number of packets that were received and n is the number of transmitted packets (i.e 10,000). We derive N from the maximum number in the `Rx Sequence Number` column of the receive log file.

The other type of packet loss originate from the way we perform experiments (see Section 5.4.3). As we configure beacon pairs (initiator and responder) to perform experiments, we use the cron utility to schedule the execution time of experiment runs (see Section A.1.4 of Appendix). However, the cron utility depends on the real-time clock of the beacons to activate its jobs. On top of this, the clock between pairs must be synchronised so that the sending and receiving component of our software are activated at the same time in the two beacons.

The problem with our existing setup is that the local clock of beacon pairs are not necessary fully synchronised and this may cause the initiator or responder to execute our software at different times. When either the initiator or responder is executed ahead of the scheduled experiment time, the end result is packet loss because one of the hosts will not be ready to receive and/or transmit packets. However such packet loss is not a network effect but simply an issue with the execution time of our software during an experiment.

In order to minimise the impact of this type of packet loss on our results, we scan through our log files to determine if there were large missing series of packets right

at the beginning or end of log files. When there is such packet loss, we subtract the observed packet loss from n in Equation 8.2. The end result from this equation is the percentage of true network-related packet loss, which we insert into our CSV files.

8.2.2 TCP Experiments

Apart from the use of TCP in web applications such as Facebook and YouTube, other VoIP applications such as Skype use TCP when UDP is blocked by firewalls [143]. However, fewer studies has been done to understand issues with VoIP communication over TCP [144, 145, 146].

Note that we can determine the following TCP path characteristics from our data (see [147]).

- Minimum buffer time requirements (as explained on page 91 in Chapter 6)
- Minimum buffer size requirements (the product of the minimum buffer time and the average payload data rate)
- Number of chunks: the number of chunks that the receiving socket passes to the receiving application. Without congestion, we would expect each transmitted snippet to travel in its own TCP packet and end up in its own chunk at the other end, therefore this number may be equal to the number of transmitted snippets. When there is congestion, several consecutive snippets may end up in the same chunk, so the number of chunks decreases.
- Percentage of congested chunks: the percentage of chunks that the receiving socket passes to the receiving application that are larger than the snippet size passed to the transmitting socket
- Percentage of congested bytes: the percentage of bytes that the receiving socket passes to the receiving application in chunks larger than the snippet size passed to the transmitting socket

In this thesis, we will only focus on minimum buffer time requirements and percentage of congested chunks. Readers interested in total number of chunks, minimum buffer size, and congested blocks measures may find more details in Ghazzi's thesis [147].

8.2.2.1 Buffer Time Requirements

In our paper on the measurement of buffer requirements for TCP [134], we mentioned that our VoIP over TCP style experiments generate data such as segment size and the receive time of segments in receive log files. We scan through our TCP Rx log files and keep a tally on segment size to get the cumulative amount of data received.

Next we compute the average receive rate of our segments by taking the total number of received bytes and divide this by the time difference between the first and the last received chunks. The result from this calculation is a single value that we use as buffer time requirement for buffering voice packets. We insert this value into our CSV file for further analysis.

8.2.2.2 Congestion Percentage

Furthermore, we also compute the percentage of TCP chunks that were affected by network congestion. In Chapter 4 we mentioned that the standard size for VoIP over TCP experiments is 115 bytes. At the receiver we compare the size of our received chunks with 115 bytes. If the size of chunks are smaller or greater, we use a counter to keep track of congested segments. Then we compute the percentage of congested chunks by dividing the number of congested chunks by the total number of received chunks and multiply this with 100. We also insert this value into the CSV file for further analysis.

8.3 Summary

This chapter described the method that we use to back up our data and various methods for processing log files into CSV files. We will now discuss in Chapter 9 the results from our experiments focusing on methods of accessing our results, relationship between path length and average latency. Furthermore, we also discuss voice quality (MOS) in terms of entropy, latency, jitter, and packet loss.

CHAPTER 9

Results

In Chapter 8, we discussed the various observables that we extract by processing our log files. In this chapter, we review methods of accessing our data followed by an interpretation of sample results. We reflect on results from our UDP experiments and follow this with a selection of results from our TCP experiments. We show that our results can contribute towards our first and second research questions on stability of latency and long term changes in connectivity quality.

9.1 Data Access Methods

There are two methods for accessing our data. The most convenient is to generate graphs with the graph generation tool provided on our website [148]. On this website, we use the `gnuplot` utility [149] to read the content of our CSV files and render the data in the form of graphs.

Depending on user preferences such as experiment type (UDP or TCP), responding beacon (e.g., NZ1), initiating beacon (e.g., TO3), and the type of path

characteristics to plot (e.g., jitter, average latency, entropy, MOS estimate, TTL), our software executes the corresponding gnuplot commands to render graphs based on the user's choice. The website also makes the CSV files and the underlying log files publicly available for download and customised offline processing. The next section discusses a number of sample results produced with our graph generation tool.

9.2 Results

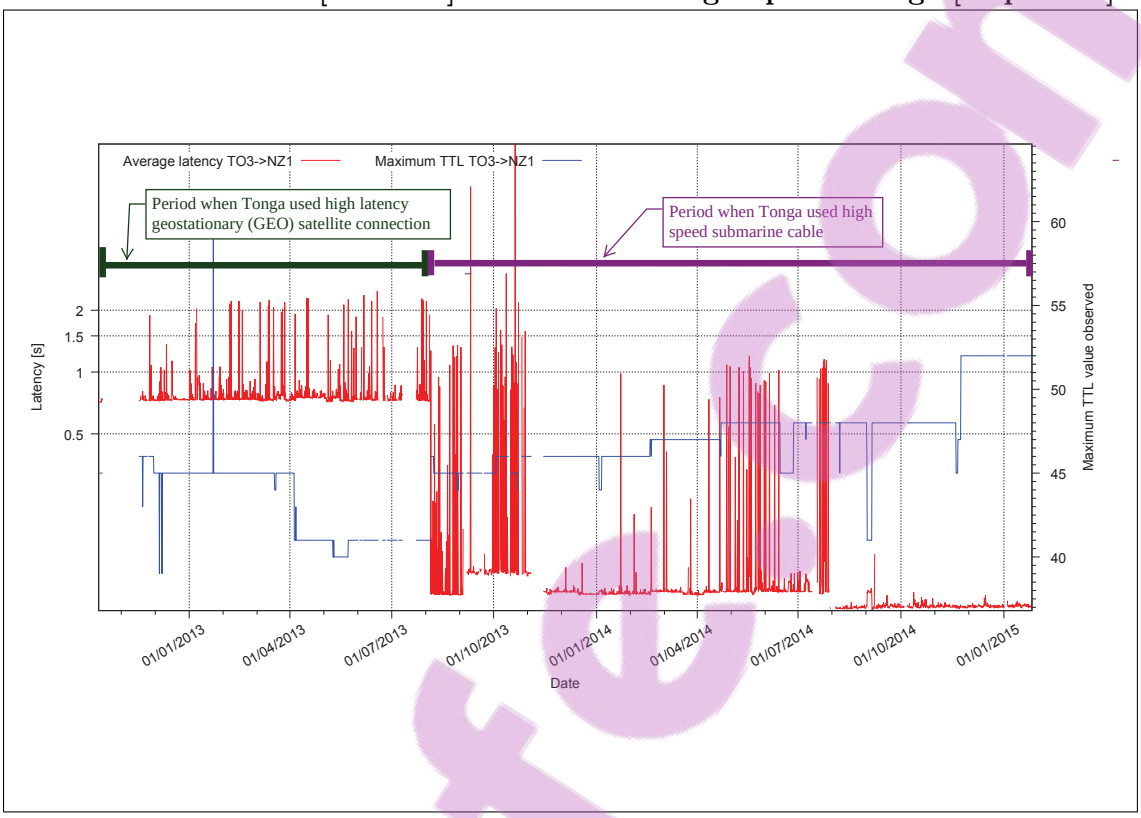
At the time of writing, we had collected 491 gigabytes of log file data. As we have too much data to reproduce in this thesis, the following sections discuss a few selected results as representative examples of the observables we can derive.

9.2.1 Average Latency and Path Length

Previously in Section 3.1, we introduced our research questions. Two of these questions identify the need to determine the stability of latency between our beacons, and the long term impact of changes in Internet topology on the quality of Internet access. In Section 8.2.1.2, we describe our method for computing average latency with Equation 8.1. In this section, we show how we can use average latency to provide answers to these research questions. Note that we use the TTL value of packets to describe the changes in the path length of packets. The TTL is based on the default Linux value of 64. By taking the difference between the default TTL and that of received packets we get the number of hops that processed our packets. We refer to the number of hops as the length of path taken by the packets.

Starting with the stability of latency between nodes, we deployed some of our nodes in countries that transitioned from high latency GEO satellite to high speed submarine cable in 2013 or to low latency MEO satellites. An example is our node (TO3) in Tonga that migrated from GEO satellite connection to submarine cable in 2013. In Figure 9.1, our beacon TO3 was implemented in late 2012 during the time

Figure 9.1: Average latency and maximum TTL value between NZ1 in Auckland in New Zealand [initiator] and TO3 in Tongatapu in Tonga [responder]



Note: The data shown here was generated from UDP bidirectional experiments. The gaps between lines are periods where we did not collect data. The lower the TTL value, the greater the number of hops between our beacons. Note how the average latency drops with the introduction of the fibre optic connection at the beginning of August 2013.

when the average latency between Tonga and New Zealand was still over 700 ms.

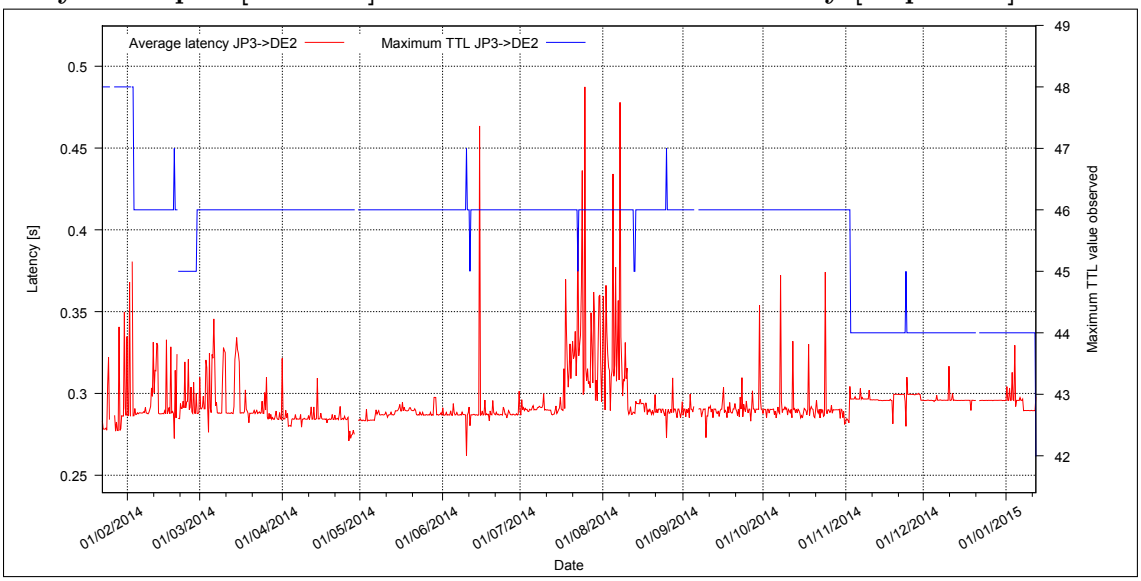
Based on the information we derived from using `Traceroute` to determine the path of packets between New Zealand and Tonga, Tonga's GEO satellite connection terminated in Canada. To reach Tonga, our packets were routed via submarine cable from New Zealand to Hawaii, from Hawaii to the mainland U.S. and finally to Canada. This represents additional latency on top of the propagation delay from the satellite connection.

At the beginning of August 2013, Tonga migrated to high speed submarine cable [150]. With the new connection, packets from New Zealand are now routed

through the Southern Cross cable [1] to Australia, Fiji and finally to Tonga. This improves the propagation delay significantly because packets travel through shorter links between New Zealand and Tonga.

As shown in Figure 9.1, average latency fell from 700 ms to less than 100 ms. At the same time, the change in connectivity increased the TTL value from 41 to between 45 and 46. That is a decrease in the number of hops that processed our packets from 23 to 19 or 20. Although a reduction in the number of hops reduces the number of queues that processes our packets, the reduction in physical latency (i.e., shorter path and high capacity links) is the main factor that improves latency.

Figure 9.2: Average latency and maximum TTL value between nodes JP3 in Tokyo in Japan [initiator] and DE2 in Berlin in Germany [responder]



Note: The data shown here was generated from UDP bidirectional experiments. The overall trend shows a reduction in TTL value from 48 to 44, which is an increase in the number of hops from 16 to 20. This is accompanied by a slow increase in baseline average latency from 280 ms to above 290 ms.

In contrast, we generated Figure 9.2 from experiments between Japan (JP3 at the University of Electro-Communication) and Germany (DE2 at Acticom). We selected these countries because they are interconnected via multiple alternative high speed fibre optic cable paths. The following discussion is based on the assumption that we expect correlation between latency and TTL for each path in a multiple path connection.

Firstly, in February 2014 there was a drop in TTL value from 48 to 46. That is an increase from 16 to 18 hops in the path between Japan and Germany. This was accompanied by a small increase in baseline latency from 275 ms to about 280 ms.

Then in November 2014, TTL value dropped again from 46 to 44 (increase from 18 to 20 hops). At the same time, the baseline average latency increased from 285 ms to 298 ms. Even though there was little correlation between average latency and TTL value between July and the first week of August 2014, the majority of our results show additional delay when the number of hops increases in the path between Japan and Germany. This is the opposite of our results in Figure 9.1, where the TTL value of packets incrementally increases from 41 to 52 (decrease from 23 to 12 hops) while latency decreases to below 100 ms.

Moreover there is no significant change in latency to indicate that there were changes in the international transit route between the two countries. Based on the above observations, we can conclude that the physical path between our beacons were the same throughout the experiment but the routing at the local/metro/regional area network changed. This means that the structure of the local/metro/regional network at either the University of Electro-Communication in Japan or Acticom in Germany changed.

Obviously, there are reasons for why individual paths are becoming faster or slower in terms of average latency over time. However our interest is on the path characteristics of individual paths over a long period of time. At this point, we have only been able to measure bidirectional experiments. In the future we hope to deploy more GPS corrected NTP servers close to all of our nodes so that we can record precise timing information in unidirectional experiments as well.

9.2.2 Latency, Jitter, Packet Loss and Mean Opinion Score (MOS)

One of the goal of this thesis is to identify solutions for improving VoIP quality for high latency satellite networks (see Section 3.1). However, finding the appropriate

solution requires better understanding of VoIP quality in links between countries.

In his article on VoIP services over satellites, Kim described three issues with VoIP communication over satellite [151]: latency, jitter, and packet loss. As per the ITU-T G.114 recommendation, the allowable two way latency (RTT) for a voice call is 300 ms [82]. Latency higher than the recommended level may make it hard to understand a voice conversation.

In addition, high jitter above 100 ms results in choppy speech, which contributes to difficulty in understanding voice conversations [152]. Also, packet loss can cause snippets of a voice communication to be missed and may eventually even cause the conversation to be dropped. Since we compute a MOS estimate in Chapter 6 from latency, jitter and packet loss, we wanted to find out whether low MOS in our experiments was caused by a single or several of these observables.

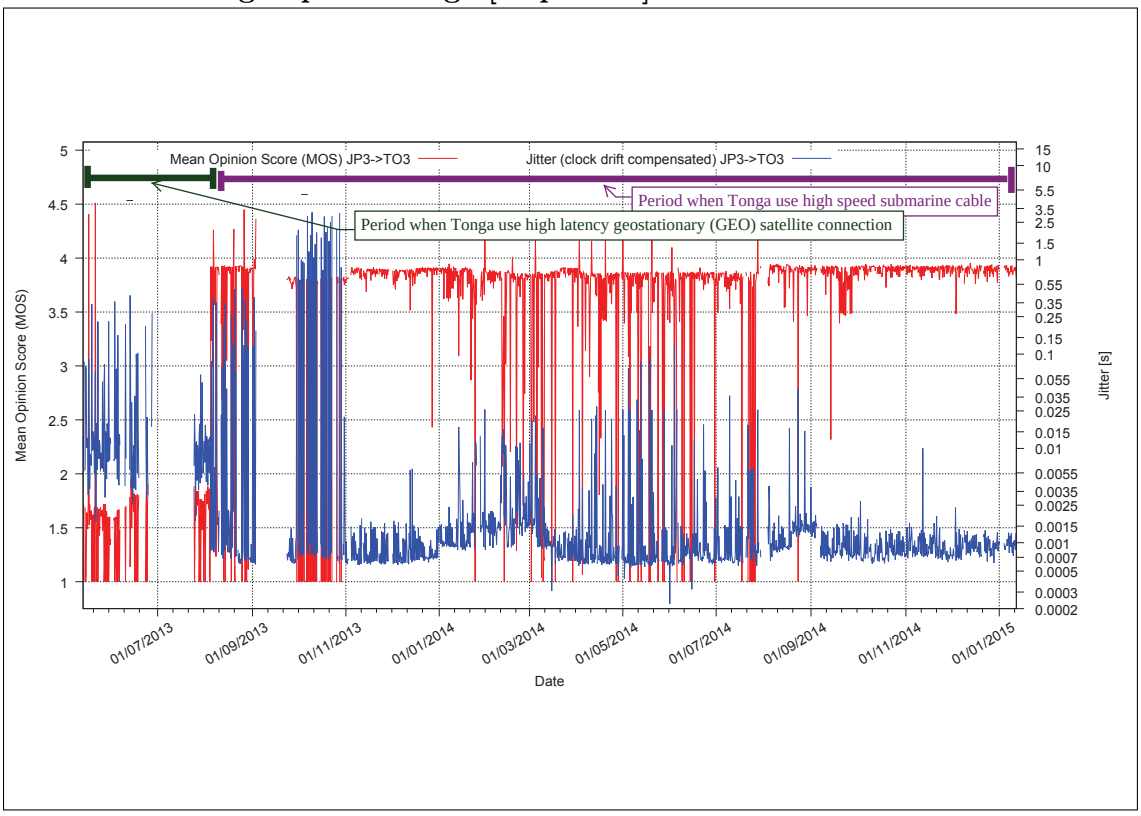
In this section, we use our results on path characteristics to describe their impact on voice quality in the form of estimated MOS at different times over our observation period.

In Section 8.1, we mentioned that Tonga transitioned from high latency satellite connection to high speed submarine cable between August and September 2013. In Figure 9.3, the transition period shows fluctuations in jitter between 1 ms and 25 ms and estimated MOS between 1 to 3.85.

Then in October 2013, MOS again fluctuates between 1 and 3.85, corresponding to fluctuations in jitter between 7 ms to just below 1 second. From November 2013 to January 2015, MOS increases in two small steps from 3.8 to 3.9. Between January and September 2014, jitter fluctuates at generally low levels with occasional spikes accompanied with falls in MOS.

Although the majority of our results in Figure 9.3 show correlations between jitter and MOS, there are sections in the graph that show no correlation. This occurred between August and the first week of September 2014 where jitter increases from 8 ms to 17 ms while MOS was stable at 3.9. Note that 17 ms is still relatively small jitter compared to the allowable 150 ms and therefore we would not expect

Figure 9.3: Jitter and MOS estimate between JP3 in Tokyo in Japan [initiator] and TO3 in Tongatapu in Tonga [responder]



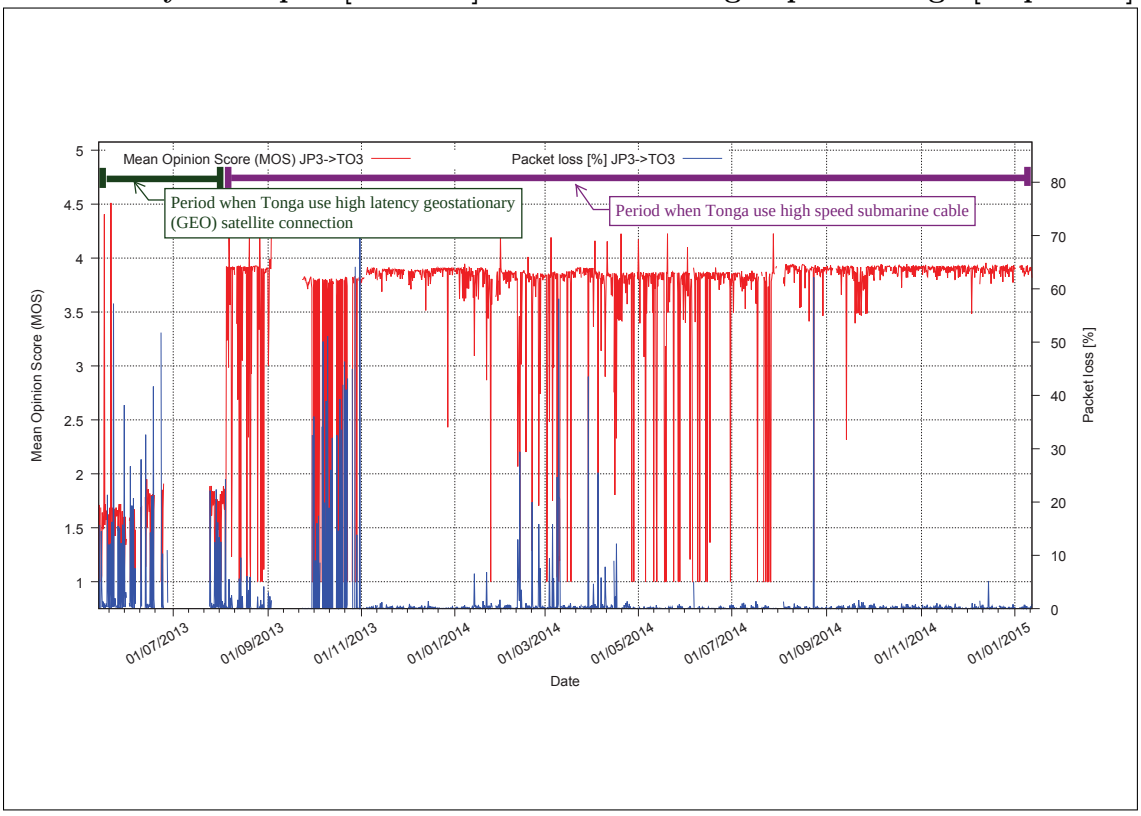
Note: The data presented here was generated from UDP bidirectional experiments.

this level of jitter to have a serious impact on MOS.

Moreover, there were times where jitter remained low at times of poor MOS. This occurred in late June 2013, prior to July 2013 where jitter fluctuated between 55 ms and around 600 ms while MOS fluctuated between 1 and 1.7. As we have described earlier, links with jitter values below 100 ms are sufficient to achieve good quality VoIP. However in this case, there were times when jitter was below 100 ms but our MOS value indicates poor VoIP quality. This means that we cannot blame the jitter component of the estimated MOS.

As discussed earlier, packet loss and latency are the other characteristics that impair the performance of VoIP applications. Since jitter was not the cause of low MOS value in late June 2013, we generated Figure 9.4 using the same experiment

Figure 9.4: Packet loss and MOS for the bidirectional experiments between JP3 in Tokyo in Japan [initiator] and TO3 in Tongatapu in Tonga [responder]



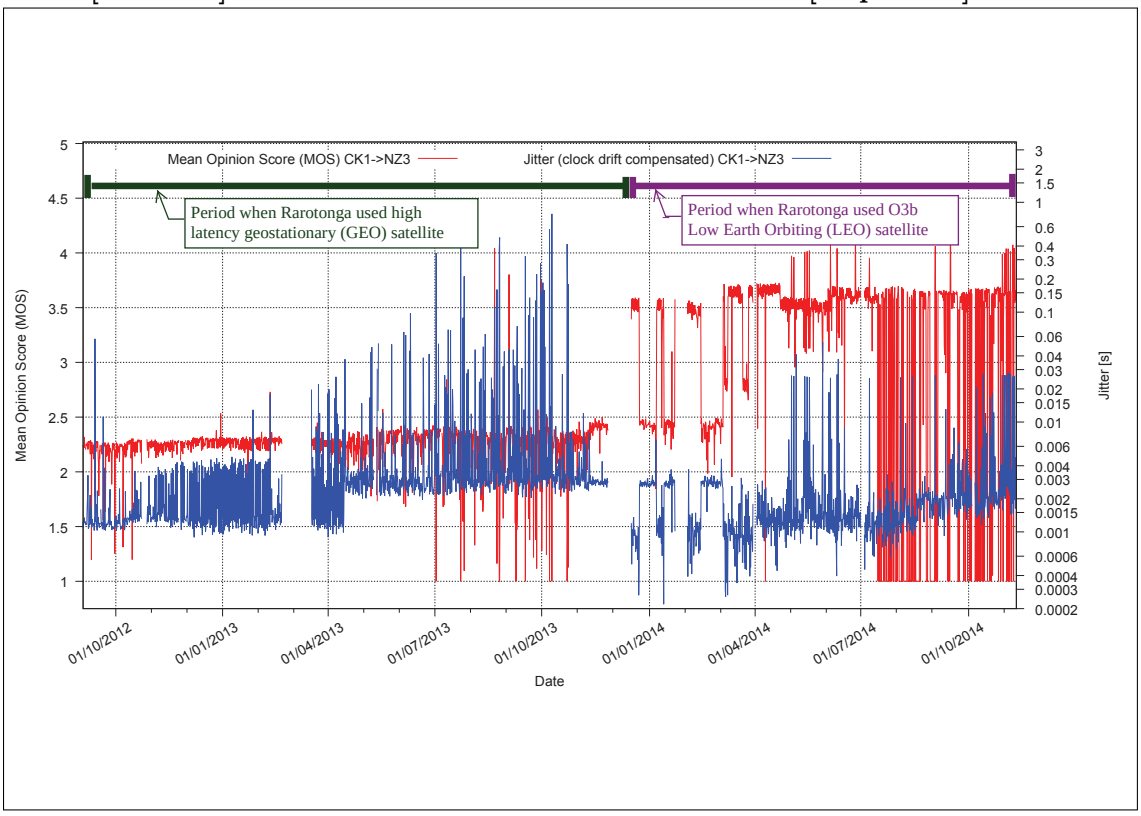
as in Figure 9.3. During this period we have strong correlation of high packet loss with low MOS value.

Tonga was not the only site to transition to a faster connection: The Cook Islands also transitioned, in this case from a GEO satellite to the O3b MEO satellite system. In Figure 9.5, the occurrence of MOS below 2.5 between October 2012 and late 2013 corresponds to the time when Rarotonga in the Cook Islands still used the GEO satellite.

During the same time period, the vast majority of jitter values were below 100 ms, sufficient to support good quality VoIP. This is similar to the situation encountered above in Figure 9.3 when both jitter and MOS value were low.

In an attempt to better understand this situation, we generated Figure 9.6. In this graph, the latency between October 2012 and late 2013 was almost 600 ms, the

Figure 9.5: Jitter and MOS between CK1 in Rarotonga in the Cook Islands [initiator] and NZ3 in Auckland in New Zealand [responder]



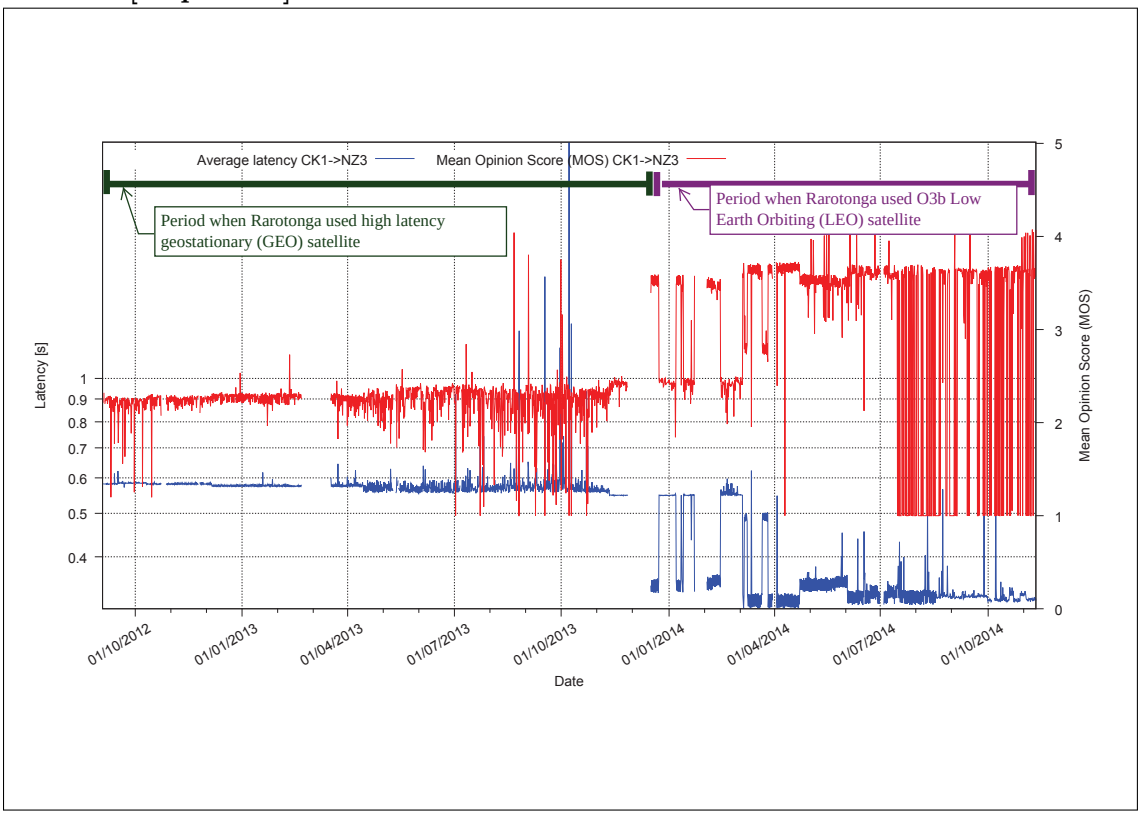
main cause of low MOS during this period.

Note that high latency does not always correlate with high jitter. The reason for this is that variable delays in the arrival time of packets depend on a number of factors, including the time that packets spend in router queues but also the propagation delays if packets travel across different links (further discussion on jitter types is provided in the next section).

Between December 2013 and April 2014, Telecom Cook Islands migrated from GEO to MEO satellites [153]. The migration process involved a series of tests with the new satellites. The transition period is reflected in Figure 9.5 when jitter flip-flops between 10 ms and 30 ms while estimated MOS flip-flops between between 2.5 and 3.5.

Moreover, similar evidence of the migration period is shown in Figure 9.6 where

Figure 9.6: Latency and MOS for bidirectional experiments between CK1 in Rarotonga in the Cook Islands [initiator] and NZ3 in Auckland in New Zealand [responder]



latency flip-flops between 200 ms and 600 ms in January and February 2014. In March 2014, O3b became the stable communication link [154]. This corresponds to a mostly stable MOS above 3.5 with jitter generally below 40 ms. In Figure 9.6, latency is now mostly stable between 200 ms and 300 ms.

Since there was low latency and jitter during this period, the cause of low MOS is now packet loss. This coincides with the observation in Chapter 10 made during the network-coded TCP deployment in Rarotonga, where we noticed that the satellite queue was oscillating [10, 11, 12]. This was the probable reason for packet loss from July 2014 onwards. Further discussion on queue oscillation is provided in Chapter 11.

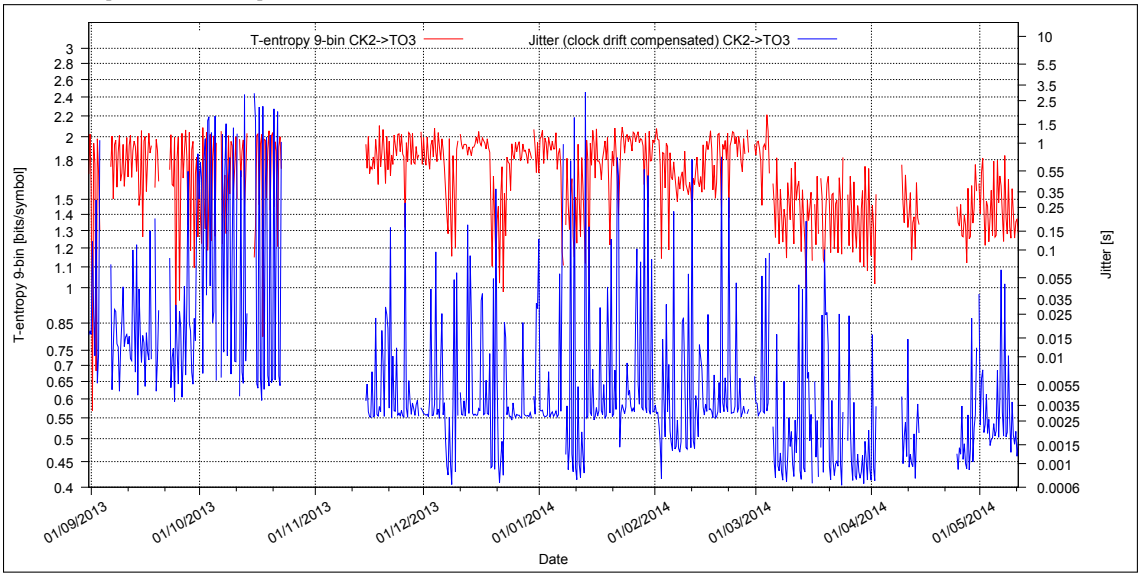
To conclude, our results show that changes in jitter can have a huge impact on VoIP quality in practice. However, there were occasions where changes in MOS did

not correlate with jitter. We were thus able to demonstrate that there is no single cause of poor VoIP quality and that latency, jitter and packet loss all have a fair share in degrading VoIP quality.

9.2.3 Entropy

Chapter 6 discussed the mapping of the inter-arrival times of packets to strings. Moreover, we also discussed the computation of entropy from such strings and the difference between systematic and queue induced jitter. In this section, we use entropy together with jitter to identify events when systematic and queue-induced jitter occur.

Figure 9.7: Jitter and 9-bin entropy for bidirectional experiments between CK2 in Rarotonga in the Cook Islands [initiator] and TO3 in Tongatapu in Tonga [responder]



Note that the 9-bin entropy in Figure 9.7 is based on the mapping of the inter-arrival time of packets to nine different symbols.

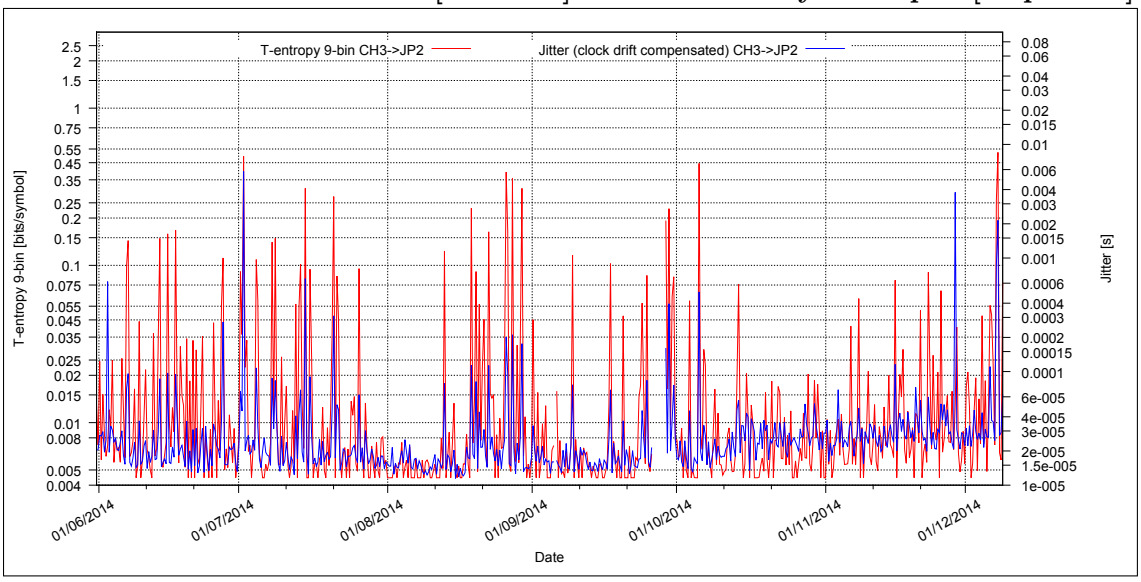
As described in Chapter 6, systematic jitter occurs when there are regular patterns in the inter-arrival times of packets. Basically, we look for periods with high jitter and low entropy. In Figure 9.7, this occurred during the first 22 days of Octo-

ber, the last week of November 2013, and the second week of January 2014 - three periods when there is clear evidence of systematic jitter.

There were also occasions where jitter and entropy peaked together. This occurred during the first week and middle of December 2013, for example. These are times when the queue-induced jitter dominated (see Chapter 6 for details on queue induced jitter). On the 20th of December 2013 and between the last week of March and the beginning of April 2014, jitter and entropy both fall. When there is little activity in the network, router queues are not overly loaded and so we also expect patterns in the inter-arrival time of packets (resulting in low entropy).

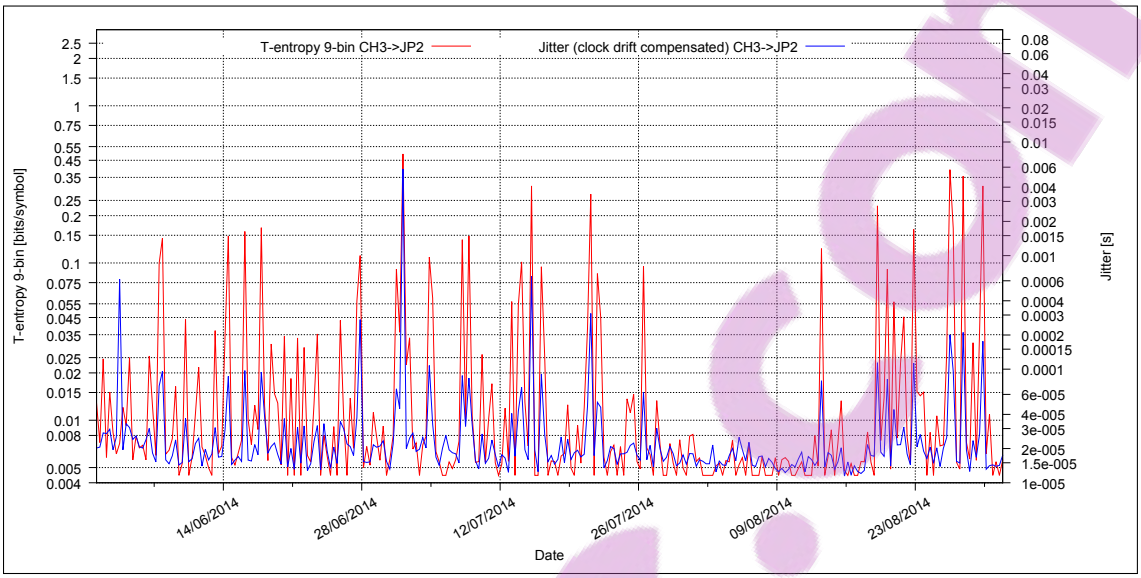
Another example is Tonga's transition to submarine cable between August and September 2013. The change in connectivity improved minimum jitter from 5.5 ms to around 2.5 ms while maximum jitter dropped from just under a second to less than 55 ms. Jitter continues to improve from 2.5 ms to 1 ms in the first week of March 2014. Occasionally we see drops in entropy which correspond to drops in jitter.

Figure 9.8: Jitter and 9-bin T-entropy for bidirectional experiments between CH3 in Zurich in Switzerland [initiator] and JP2 in Tokyo in Japan [responder]



In our discussion of queue-induced jitter in Chapter 5, we described the occurrence of this jitter type as a random pattern in the inter-arrival time of packets. We

Figure 9.9: A snapshot of Figure 9.8 taken from the months June to August 2014



also mentioned that the variable length of router queues is the main contributing factor to queue induced jitter; some routers may have long queues while others have shorter queues.

Consequently, this causes variable delay in the travel time of packets, which subsequently causes the variation in the inter-arrival time of packets. To detect this type of jitter, we look for times when there is high jitter combined with high entropy.

In Figure 9.8, evidence of queue induced jitter reoccurs throughout the graph where jitter and entropy increase together. This is the case for example between June and August 2014 (Figure 9.9 shows a closer plot of that period) where entropy and jitter increases together.

There were also times when there was little to no activity in the network such that router queues were not particularly loaded. As discussed earlier, these are the times when jitter and entropy fall together. This occurred in Figure 9.8 between the last week of July and the first week of August 2014 on the days around the 12th of September.

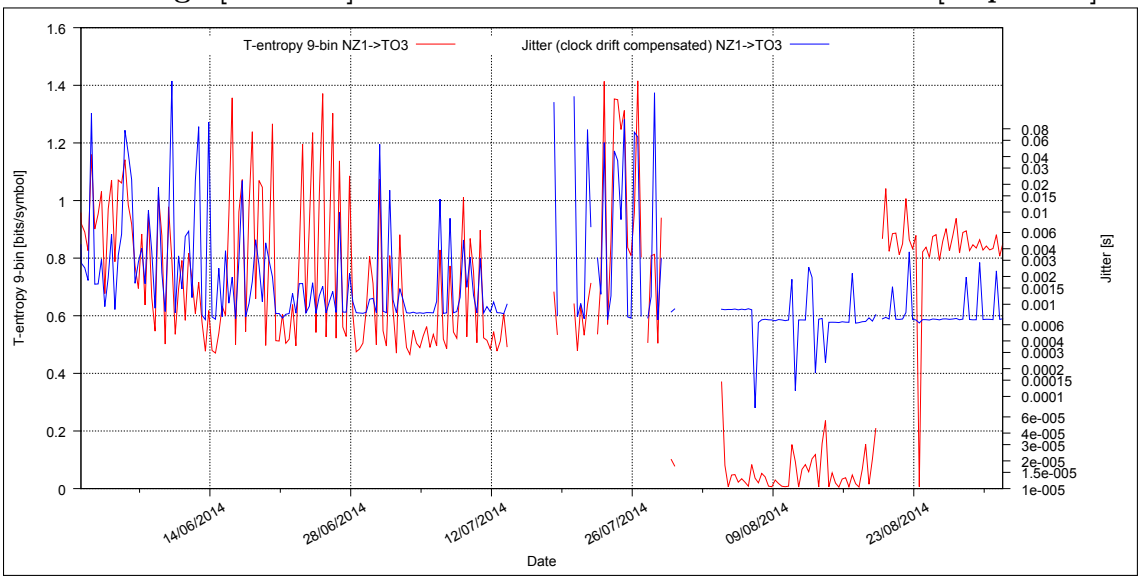
Comparing our results in Figure 9.7 and Figure 9.8, long distance high speed

connectivity between Japan and Switzerland is more likely to be affected by queue induced jitter.

This is not necessarily obvious given the fact that there are numerous topological options for multiple paths between Japan and Switzerland. However, our results shows that systematic jitter is more common in the path between Tonga and the Cook Islands. This also means that there probably was load balancing somewhere in the path between Tonga and the Cook Islands, but probably not between Japan and Switzerland.

9.2.3.1 Entropy levels

Figure 9.10: A snapshot of Figure 9.1 for 9-bin entropy and jitter between TO3 in Tonga [initiator] and NZ1 in Auckland in New Zealand [responder]



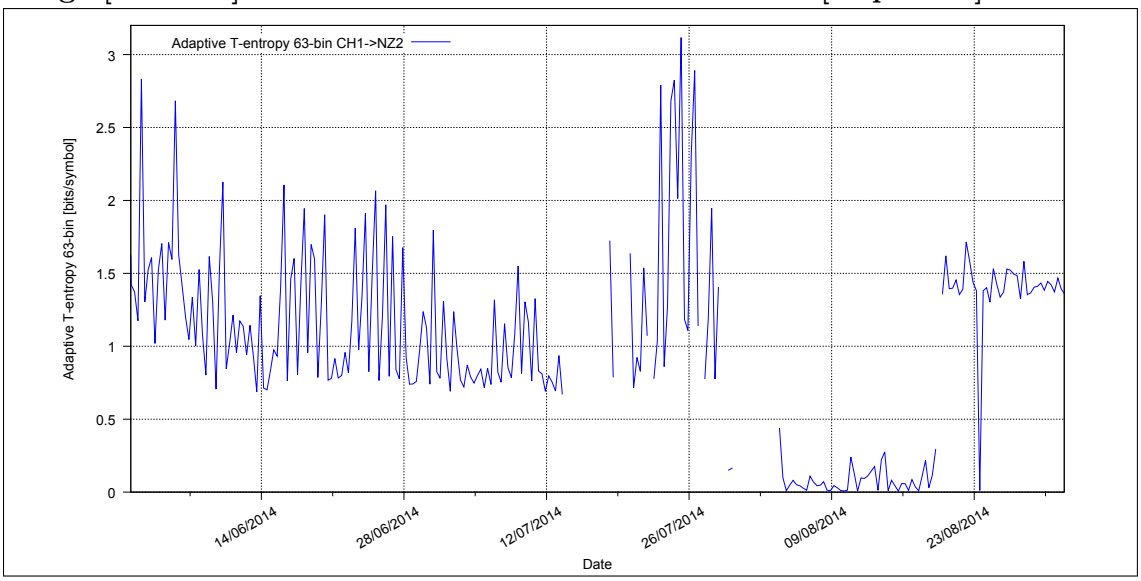
In Chapter 6, we discussed entropy and described the reason for introducing adaptive entropy. In the last section of this present chapter, we discussed patterns in our results that we use to identify the occurrence of systematic and queue induced jitter. Basically, we look for positive or negative correlations between jitter and entropy.

Adaptive entropy tells us the extend for which our result agrees with a hy-

pothesis. We start with the hypothesis that the inter-arrival times are normally distributed (i.e. a high adaptive entropy corresponds to a normally distributed queue-induced jitter whereas a low entropy corresponds to systematic jitter). Note that in our fixed size bin entropy we had to compare our results with jitter to determine queue-induced or systematic jitter. With adaptive entropy we use a single figure to determine these jitter types.

To demonstrate the effectiveness of our adaptive entropy method, we generated Figure 9.11 and compare our results with Figure 9.10.

Figure 9.11: A snapshot of Figure 9.1 for adaptive entropy between TO3 in Tonga [initiator] and NZ1 in Auckland in New Zealand [responder]



In Figure 9.10, jitter and entropy increases together between 14th and 28th of June 2014, and between 24th and 26th of July 2015. This is an indication of queue-induced jitter which also correlate with the time when Tonga was still using GEO satellite connection. We can verify queue-induce jitter with the high spikes in adaptive entropy in the same time line in Figure 9.11.

Between the 5th and 20th of August 2014 in Figure 9.10, jitter remained high but entropy was low. This is an indication of systematic jitter. After consulting the local ISP, we were told that Tonga transitioned to submarine cable but kept using the satellite connection for smooth transition.

With the load balancing of traffic between connections, we get patterns in the inter-arrival time of packets and this is the reason for low entropy. In Figure 9.11, we can verify systematic jitter with the low entropy in the same time line.

To conclude, our result shows that we can use adaptive entropy as an alternative method for distinguishing systematic and queue induced jitter.

9.2.4 Simulated VoTCP

This section discusses examples of our results on experiments with traffic simulating VoTCP and describes results from collaborative work with Firas Ghazzi. A more detailed discussion of these results is provided in his thesis [147]. Note that the concept of minimum buffer requirement in the context of buffer underrun is discussed in Chapter 4.

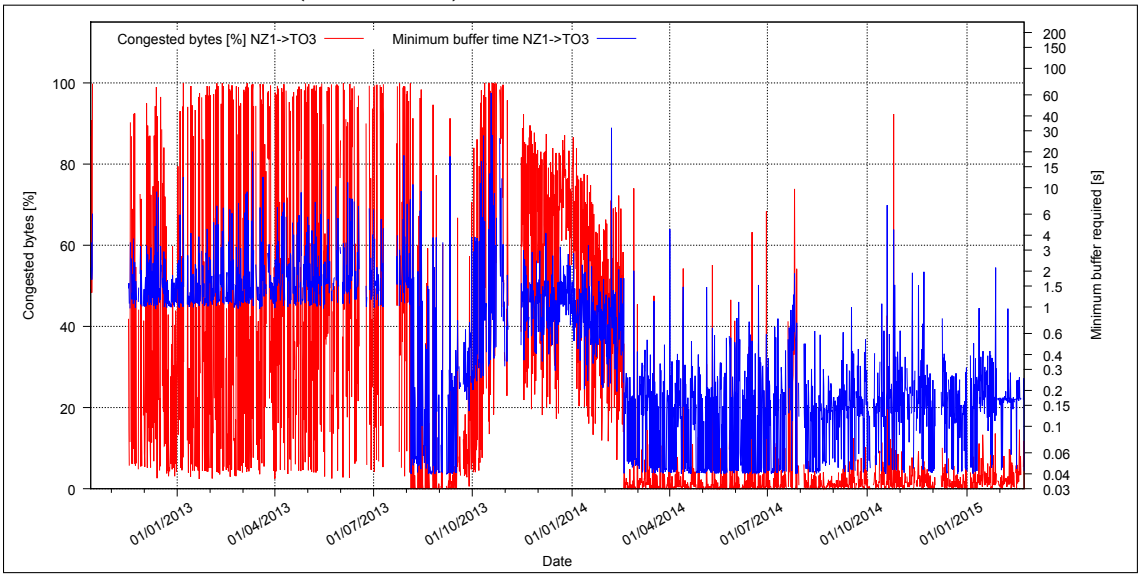
As already mentioned, VoIP applications often use TCP to continue operation in a network where UDP is blocked. The main concept that we attempt to demonstrate in this section is that VoTCP communication is problematic over high latency satellite connections

As discussed in Chapter 4, there are two TCP-related path characteristics that we record in our CSV files: minimum buffer requirement and the percentage of congested bytes. Figure 9.12 shows the results of the simulated VoTCP experiments between Auckland and Tonga, which used voice-style snippets of 115 bytes.

Between January and August 2013 in Figure 9.12, the minimum buffer time requirement started at 1 second and above. This is the time that VoIP applications would have had to buffer voice packets for smooth replay. As discussed earlier, this is the period that Tonga still used the high latency GEO satellite connection.

When Tonga transitioned to the fibre connection in September 2013, minimum buffer times dropped to values typically below 50 ms and seldom exceeding 500 ms. The percentage of congested bytes also fell before rising back to roughly the original values in October 2013.

Figure 9.12: Minimum buffer time requirement and percentage of congested bytes for bidirectional experiments between beacon pair NZ1 (initiator) in Auckland and TO3 (responder) in Tonga



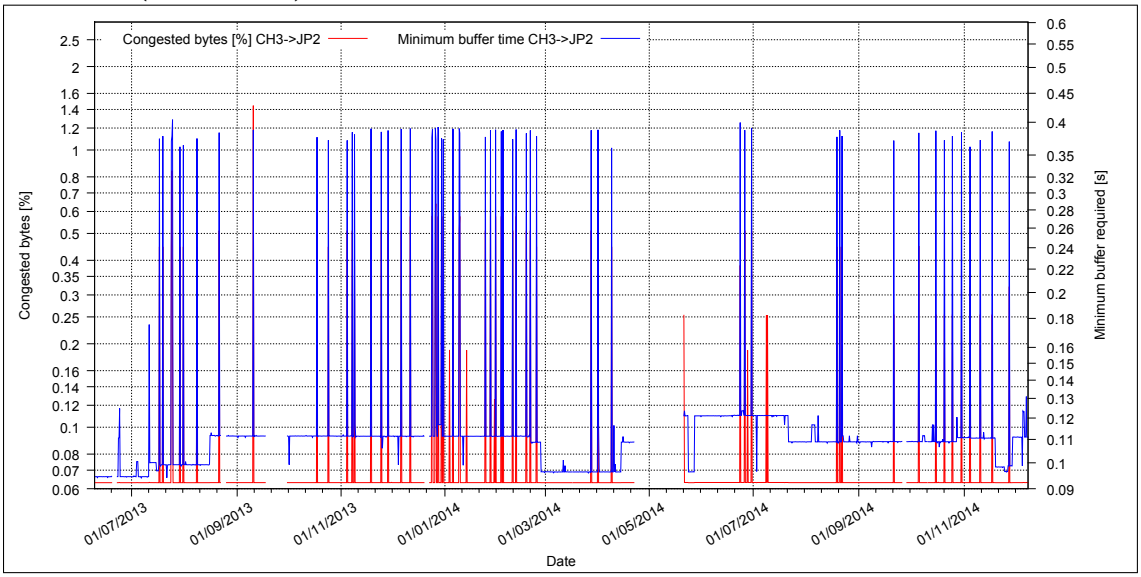
Then in March 2014, minimum buffer time again settled between 50 ms and 250 ms. At the same time, the percentage of congested bytes was typically well below 10%. With the improvement in our results we observe that, as expected, high latency satellite connections make VoIP communication with TCP practically impossible. Note that a 1 second buffer time significantly exceeds the allowable 250 ms delay.

In contrast, the low latency fibre optic connection improves the amount of time that VoIP applications need to buffer when using TCP. Figure 9.12 also shows that the percentage of congested bytes improved considerably after the arrival of high speed connectivity.

In an attempt to verify the claim that high speed fibre optic cable is favorable for VoTCP communication, we generated Figure 9.13. The data in this graph were derived from experiments involving the exchange of 115 byte snippets that is passed into the TCP socket at 20 ms intervals. Note that there are multiple possible high speed fibre optic connection paths between Switzerland and Japan.

Compared to Figure 9.12, buffer time requirements in Figure 9.13 were consis-

Figure 9.13: Minimum buffer time requirement and percentage of congested bytes for bidirectional experiments between beacons CH3 (initiator) in Zurich and JP2 (responder) in Tokyo



tently below 390 ms. This is still greater than 300 ms which is not perfect, but the vast majority of values are less than 100 ms and the percentage of congested bytes was also below 0.35 % throughout.

To conclude, our result in Figure 9.13 proves that high bandwidth connections provided by the international fibre backbone network are generally good enough to even permit VoTCP, even under long latencies.

9.3 Summary

In this chapter, we discussed the methods we use to access our data followed by interpretation of sample results. The data presented in this chapter is just a small subset of the data collected. The sample result shown here demonstrate the kind of analysis and observations possible. Here, we have shown that:

- Switching from high latency GEO satellite to high speed submarine cable improves the propagation delay and also improves voice quality (high MOS in

Figure 9.1).

- Improvements in jitter from switching to high speed connectivity does not always correlate with better voice quality (shown in Figure 9.3). Our result in Figure 9.4 indicate that low packet loss is another reason for better voice quality.

We will now discuss solutions for high latency connections in Chapter 10 with particular focus on TCP/NC.

CHAPTER 10

Solution for High Latency Satellite Networks

In this chapter, we turn to our last research question: how to improve the performance of VoIP and other real-time applications on high latency networks. We begin by reflecting on the motivation of this chapter, and follow this with possible solutions for improving throughput. Then we use references to previous work on TCP/NC to describe the theoretical background of this solution. Lastly we reflect on design requirements for the deployment of the solution we investigated, a TCP/NC tunnel across satellite links to ISPs in Pacific Islands.

10.1 High Latency and Low Bandwidth Networks

In Chapter 4, we described experiments that we perform with our `beacon` software, and in Chapter 9, we plotted our data in graphs and interpreted trends in path characteristics from the data we collected. Based on our results, we conclude that data generated from satellite-connected countries in the Pacific indicates poor voice quality and high jitter.

Although this is not surprising given the fact that many Pacific island countries use high latency satellite connections, the question we have not addressed is whether there is a way to improve throughput for these countries.

A common way of improving throughput is to transition from geostationary satellite to high speed fiber optic connections. Although this is the preferred long term solution, countries such as Niue have low population sizes and cannot afford such a project. Even though Niue is approximately 600 km from Tonga, the two countries are divided by the Tonga trench which would complicate a project to connect with the Southern Cross Cable [155].

Other countries such as Tuvalu and Kiribati are very remote and require much longer cables and thus large funds to establish fiber connectivity. This is not feasible because the gross domestic product of these countries is so small that fiber optic connection is beyond reach.

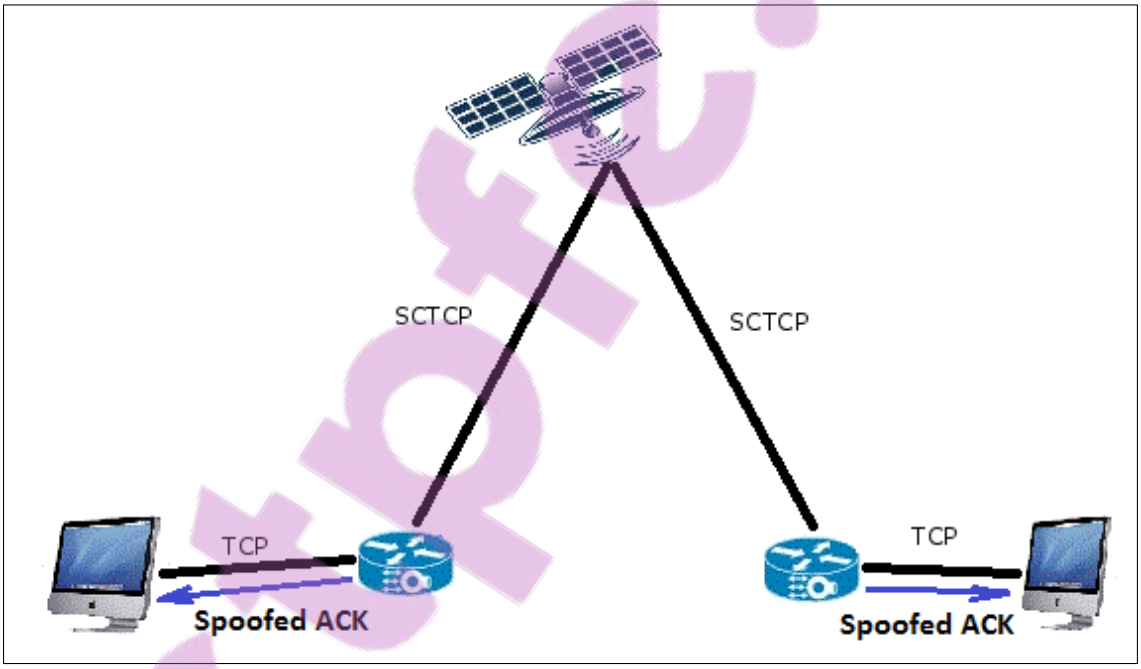
Alternatively, countries in the Pacific can improve throughput by changing the type of satellite they use. For example, as mentioned before, Rarotonga in the Cook Islands recently migrated from a high latency geostationary satellite to the O3b Medium Earth Orbiting (MEO) satellites [154]. This migration reduces the orbital height from over 35,000 kilometres to just over 8,062 kilometres and improves round trip latency from at least 480 ms to below 150 ms [156].

Although O3b is a relatively accessible solution for improving throughput, the

installation cost is still too high for many islands because the communication between base station and satellite requires the installation of at least two steerable antennae that track the satellites [157]. An alternative approach would be to deploy Kacific satellites to provide multiple parallel connections. However, Kacific was not operational at the time of this research project.

A less expensive solution would be to deploy Performance-Enhancing Proxies (PEP) [158] in countries that use GEO satellites. An example of such a device is the Cisco network capacity expansion (NCE) service module [159] that uses techniques such as TCP spoofing to overcome the effects of TCP slow start in high latency networks. Figure 10.1 is a simplified version of Figure 1 in [159] showing a satellite link between two PEP boxes.

Figure 10.1: Spoofing of TCP ACKs in a PEP



Note: PEP can be performed with a single box. However better performance can be achieved with two boxes on the two ends of the satellite connection.

In Figure 10.1 there are three sessions involved in TCP spoofing, the first session occurs when host A sends segments of data to host B. The segments are intercepted and the session terminates at PEP box 1. At the same time, PEP box 1 initiates a second session with Stream Control Transport Protocol (SCTCP) [160] or another

variant of TCP to send the data on to PEP box 2. The SCTCP session terminates at PEP box 2 and executes a third session using TCP to transmit the data from PEP box 2 to host B. This means that PEP box 1 pretends to be host B when talking to host A while PEP box 2 pretends to be host A when talking to host B. Depending on the PEP design, different techniques can be activated to further improve throughput. As an example, NetPerformer [161] uses TCP spoofing together with the following techniques to improve the performance of TCP over high latency satellite networks.

1. Window scaling option of TCP [162] to support large data sizes.
2. Time-stamping option of TCP [162] to perform accurate reading of RTT in order to better estimate TCP timeouts and retransmissions:
3. Selective negative acknowledgments (SNACK) to notify the sender of segments that were lost or corrupted. This improves throughput because TCP no longer requires acknowledgment of every packet but only acknowledges missing segments for retransmission.
4. Congestion avoidance and control to manage TCP connections across satellite links with the TCP congestion control [163], fair share and dynamic right-sizing algorithms. These algorithms enable the sender to perform immediate retransmission of data upon negative acknowledgment, and to use RTT and available bandwidth for making buffer decision on the appropriate size of the congestion window. In doing so, PEP devices can allocate any unused bandwidth to other TCP connections that require more bandwidth.
5. Netperformer uses the fast checksum calculation standard [164] to perform 32 bit calculations of checksums instead of the standard 16 bits. As a result, Netperformer can perform incremental updates rather than having to recalculate the whole checksum of a received segment. Such an approach speeds up the processing of segments at the destination.

In addition to the above techniques, a PEP can be configured with additional services such as compression, filtering of rogue Internet traffic, traffic shaping with QoS, data suppression, optimization of particular application protocols, and header

compression, as well as lossy compression techniques in applications [165] to further improve throughput.

However, there are limitations to the use of PEPs. Firstly, this technology depends on the visibility of the TCP header to be able to modify the behaviour of TCP. This is not possible when network layer encryption applications such as IPSec completely hide the header of TCP segments from the PEP. In such cases users are forced to use encryption applications that works over TCP or proprietary solutions such as the Selective Layer Encryption (SLE) [166] that enable IPSec over satellite. On top of this, encryption renders the payload of TCP segments incompressible, which undermines the potential for compression.

Despite the availability of additional services and the use of TCP enhancement techniques for improving throughput, PEPs are uncommon in the Pacific because they are still costly. The cost of PEP devices increases in proportion with additional services required by an ISP.

In some cases, satellite providers are reluctant to install additional devices on their end, which complicates the adoption of PEP. Even though there is an open sender version, PEPSal [167], many ISPs in the Pacific cannot commit sufficient human resources to its complex configuration.

10.1.1 Other Transport Protocols for the Space Segment

Since PEPs use a lightweight version of TCP over the satellite connection, it might be more logical to use another variant of TCP that can perform well over satellite links. For example, TCP Cubic and TCP Hybla [168, 169] manage their congestion window independently of the RTT of a link and thus improve the overall performance of TCP.

In particular, TCP Hybla was developed for the purpose of reducing multiple losses, inappropriate timeouts, and burstiness of a link. It modifies the standard

rules for increasing the congestion window to achieve high throughput in heterogeneous networks [170]. However, there is some evidence that TCP Hybla does not perform as well as TCP Cubic on satellite links with high burst error rate (BuER) [171]. Despite their differences, both TCP types suffer greatly on high BuER links.

On the other hand, H-TCP [172] performs fast recovery after congestion. It uses RTT scaling to manage the window size and to improve throughput for high speed long distance networks [172]. Note that both H-TCP and Hybla were TCP variants designed for wide-band scenarios which aren't what we typically encountered in the Pacific. Most of the satellites we have worked with are narrow-band.

A more loss-tolerant TCP (LT-TCP) was also developed to counter the problem of BuER [173]. However LT-TCP requires the use of explicit congestion notification (ECN) and Reed-Solomon codes [174] to reduce the impact of BuER. The problem with this requirement is that there is the possibility of decoding errors which can significantly degrade the performance of TCP.

10.1.2 Problems With Use of Satellite

As an alternative, ISPs could deploy TCP/NC [175]. To better understand TCP/NC, we briefly revisit problems associated with the use of TCP over high latency satellite networks. As mentioned in [161], there are two aspects of a satellite network that impact the performance of TCP: packet loss, and latency.

There are two causes of packet loss on satellite links. One cause is bit errors from low signal-to-noise ratio. Most often, atmospheric effects such as rain fade [176] causes weak signal to be received from satellite. Another cause of low signal-to-noise ratio is excessive noise that interferes with the signal. The other potential cause of bit errors is queue drops at the satellite modem at the uplink from queue oscillation.[1]

[1]Note that a detailed description of queue oscillation is provided in Section 11.2.1.

The impact of atmospheric effects is uncommon compared to the impact of queue oscillation because atmospheric effects come and go with the weather, while what we see is that our packet loss percentage increases with the load and is highest during peak traffic hours. The higher the load the slower it is for packets to be processed and transmitted through the satellite link. As a result, packets are held up in long router queues that may be subjected to tail drops when the queue overflows.

Currently, the way TCP handles lost packets is to use a timeout period at the sender to determine when packets were lost. When the timeout period elapses and the sender receives no acknowledgment (ACK) from the receiver, lost segments are retransmitted by the sender. In other cases, when the sender receives duplicate acknowledgments from the receiver, it regards the link as congested and reduces the window size, resulting in a reduction in the transmission rate which eventually degrades throughput.

Furthermore, the distance between the satellite and base station determines the latency of packets. For GEO satellites, the two-way RTT is upward of 480 ms and this excludes any coding delays [177]. The way TCP determines throughput is with the formula: "window size divided by RTT". Depending on the available throughput, TCP increases or decreases the rate of transmission to avoid network congestion.

Initially in a satellite link, the window size will be small because of the slow start mechanism of TCP. The window size slowly increases when there is timely arrival of acknowledgments at the sender. Because of high propagation delay in most satellite links, there is a high chance that the acknowledgments are not received on time or may not arrive at all. As a result, TCP decreases the window size and this contributes to low throughput.

10.2 Network Coding (NC)

Started as theoretical applications for improving the speed and reliability of data communication [178, 179], the practical implementation of NC in the context of

TCP/IP has moved into focus over the last few years.

10.2.1 TCP/NC

For clarity, the author would like to emphasise that we did not contribute to the theory of NC or its implementation. However, our contribution is the practical deployment of the TCP/NC software, and the design information that led to the implementation of this technology as a tunneling solution on a production network environment.

The work by Ho on distributed random linear network coding first provided the idea of translating theory into practical applications [180]. Additional work by Katti et al. on the embedding of coding coefficients in the header of packets was a major improvement on the coding method [181]. This led to the first practical system, with the implementation of the opportunistic unicast network coding [182].

It was the application of network coding to TCP (TCP/NC) by Sundararajan et al. [183] that advanced the development of network coding towards the TCP/IP stack. In their paper, they proposed TCP/NC as a solution for improving the performance of TCP over high latency lossy wireless networks. As discussed before, TCP communication involves the sender reading data and sending them as segments to the destination node. At the destination, received segments are acknowledged back to the sender. As long as acknowledgments arrive on time, the sender can adapt the congestion window size to transmit the next segment(s) of data.

As described earlier, packet loss and high latency are frequent events in satellite links. Responding to delays or loss in the acknowledgment (ACK) of packets, TCP reduces the window size and this leads to low throughput. With TCP/NC, two solutions were proposed by Sundararajan et al. to solve this problem [183].

Firstly, instead of transmitting segments from the sender and acknowledging them one by one at the receiver, the sender captures a number of incoming TCP packets. It uses a random number generator and seed to generate random coefficients. The coefficients are multiplied with the intercepted packets to form coded

packets. The header of coded packets are embedded with coefficients. The coded packets are then added together to form linear combinations of packets [183]. The linear combinations are transmitted to the destination. This process is repeated until all data are transmitted by the sender. Note that the number of packets to be coded is adjustable to suit the condition of a link and that each linear combination correspond to an equation.

At the destination, the receiver retrieve the coefficients and keeps track of the number of received coded packets. When there are enough information to restore the original data, the receiver uses the coefficients together with the coded packets to solve a system of linear equations. The end result is the original data.

While the coding of packets is beneficial for error correction, it became a problem when maintaining the behaviour of TCP in the ACK of segments. In TCP, segments are cumulatively acknowledged in the order they were received. However the coding of packets with TCP/NC means that coded packets do not have an implicit order.

Instead of acknowledging the receipt of individual segments, Sundararajan et al. introduced the concept of *seen packets* [184]. Upon the reception of coded packets, the receiver scans through them to find out if they contain newly received original packets. When there are new packets, the receiver immediately acknowledges them as an additional degree of freedom (DOF) without the need to decode the coded packets. This is different from TCP where segments must be processed to determine their sequence number before they are acknowledged.

The idea behind the use of linear combinations of packets and DOF is to overcome the slowing down of TCP when there is packet loss. In their paper on modeling TCP/NC [175], Kim et al. describe the way TCP uses packet loss to detect congestion. Assume that the sender sends 6 packets numbered 1, 2, 3, 4, 5, 6 to the destination as part of the same window and the 4th packet is lost. Since TCP requires cumulative ACK, the receiver repeatedly acknowledges the last known received packet (packet 3) until the arrival of packet 4.

When the fourth packet is received and acknowledged, the sender updates its window size and transmits the next packet. In our case, the duplication of packets

causes the sender to receive acknowledgements for packets 1, 2, 3, 3, 3, 3. When the sender receives these 4 duplicate acknowledgements, its TCP misinterprets the link as congested and the sender reduces the congestion window size to transmit fewer packets.

In Section 10.1, we mentioned that TCP uses a timeout period to determine if there is a need for retransmission of lost segments. In a congested network, TCP slowly increases the timeout period and retransmits a single packet to determine if the receiver can receive and acknowledge that single packet. It is only when an ACK is received that TCP decreases the window size and timeout period again.

As described in [183], the use of a large size Galois field for the generation of random linear packets increases the chance that the next unseen packet will be seen in the order it was expected. Even when some of the coded packets are lost in transit, the next expected packet will eventually be seen when the subsequent coded packets are received and decoded. This functionality of TCP/NC improves throughput for lossy links in two ways. Firstly, TCP/NC overcomes the problem with duplicate acknowledgements because the sender simply continues to transmit coded packets when there is packet loss. Secondly, there will be fewer retransmissions because any lost packet is recoverable from subsequent coded packets.

The problem with continuous transmission even when there is packet loss is the possibility for coded packets to cause congestion. Since there is limited bandwidth in lossy wireless networks, TCP/NC requires a congestion control mechanism to ensure that transmission of coded packets will not contribute to congestion. The paper by Sundararajan et al. [183] describes the use of the TCP Vegas [185] congestion control algorithm to control the transmission rate. Unlike TCP Reno [186] that uses packet loss to determine congestion, TCP Vegas determines the size of the TCP buffer even before packet loss. When there is congestion, the buffer starts to fill up and this delays the transmission of packets. The delay is also reflected in the increase of RTT, the congestion indicator [185]. However the TCP Vegas congestion control was developed specifically for the cumulative ACK of TCP.

In order to adapt the congestion control of TCP Vegas for TCP/NC, Sundararajan et al. proposed a solution whereby the sender matches the last received DOF

with the previously transmitted linear combined packet [183].

In an effort to make the TCP/NC algorithm more efficient and robust, Kim et al. developed *coded TCP (CTCP)* with the following features [175]:

1. Network coding in user space is implemented at the application layer. CTCP uses the transport layer in the form of UDP to send coded packets. This led to more efficient coding methods and provides room for improvement in the congestion control algorithm.

2. CTCP is more adaptive to network conditions because it estimates packet loss p and dynamically adjusts the redundancy factor $R = \frac{1}{1-p}$. This is an improvement from TCP/NC which uses a known average end-to-end packet loss rate p to determine the redundancy in coded packets R. In the natural flow of traffic, packet loss fluctuates and so the use of known average packet loss does not reflect dynamic changes in the network.

3. CTCP uses systematic block coding to manage delay and complexity. There are advantages and disadvantages in the use of the two methods. As Kim et al. require a method that reduces the decoding overhead, the systematic coding approach is the preferred choice in this case [175]. When the link is lossless ($p = 0$), the systematic approach assigns redundancy factor $R = 1$. In this case, CTCP transitions to normal TCP-like functionality.

4. CTCP uses tokens rather than a congestion window to control transmission rate. The congestion control signal is the rate at which tokens are generated and destroyed.

Earlier in this section, we mentioned that TCP sends duplicate acknowledgements for the last known packet received in sequence until the expected cumulative ACK can be sent. The end result of duplicate acknowledgements is repeated back-offs, which prevent the congestion window from filling the link to full capacity. This behaviour is well known; an example is provided in [187] where the congestion window decreases under packet loss.

In an attempt to solve this problem, Sundararajan et al. [184] modified the ACK to use the concept of degrees of freedom and the sliding window approach to increase throughput even when there is packet loss. Although the sliding window approach allows for better management of coded packets and better performance, Kim et al. note that the decoding of a file requires the reception of *all* corresponding packets [175]. The impact of the sliding window approach may not be noticeable in the transfer of small files. However it makes a huge difference when transferring big files or when using real-time applications such as Skype.

Because of this limitation in the sliding window approach, Kim et al. decided to use systematic coding and *tokens* to transmit packets [175]. The slow start behavior is identical to the traditional TCP where the reception of ACK increases *cwnd* by 1, and increases the *cwnd* by $1/cwnd$ when there is no ACK. When an ACK is received later than the maximum round trip time (RTM), the algorithm resets the slow start phase with initial tokens value. When there is congestion, the received ACK change tokens in two ways.

If the ACK is a response to the next data (ACK is a response to the last received data), token is increased by its reciprocal. When the ACK is a response to an unexpected data (lost data), tokens is scaled by a factor of RTT_{min}/RTT [175]. Note that RTT_{min} is the lowest observed round trip time and RTT is the current observed round trip time.

When the link is not congested, RTT will be close to RTT_{min} and tokens will be scaled by a factor close to 1. The scaling of tokens adapts the systematic coding approach to be suitable to changes on the condition of the link so that the source performs normal TCP functionality when there is low packet loss.

Contributing to the development of approaches to congestion control algorithm, Cloud et al. compared the performance of CTCP with different variants of TCP (HTCP, Cubic, Hybla, Reno, Veno, and Westwood) over satellite links [188]. Their experiments indicate that the "additive increase" functionality of CTCP's congestion control algorithm is the reason that CTCP performs poorly under low packet error rates (PER). This study led to the conclusion that CTCP could overcome the challenge of long RTT by increasing the congestion window in a manner similar to

H-TCP [189].

10.2.2 Network-Coded Proxies and Tunnels

Unfortunately, there is a naming collision in the literature. There are two types of TCP over Network Coding (TCP/NC). One uses degrees of freedom while the other is a tunnelling solution that does not use a feedback mechanism. In this section, we refer to TCP/NC as the proxy version by Sundararajan et al. [183].

So far, we reviewed CTCP as potential solution to overcome the challenge of long RTT in satellite links. However, CTCP is an end-to-end solution that requires the installation of the software in both the sender and receiver. This is not practical in a large scale deployment because we do not have control over all clients and servers on the Internet. The devices that we can control from an island network operators perspective are designated machines located on either end of a satellite connection. More specifically we typically have control over a host.

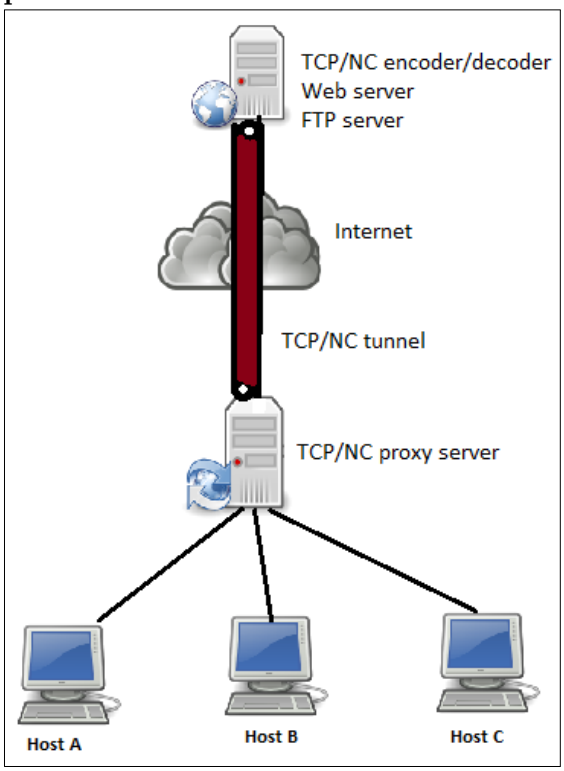
10.2.2.1 TCP/NC Proxy Solution

Attempting to overcome this challenge of deploying network coding, the user space SOCKSv5 proxy application was developed to tunnel coded packets across satellite links between two endpoints with either side acting as the proxy. Despite the experiments conducted with simulations [188], Steinwurf [190] developed a proxy application that could tunnel coded packets between two end points. At the time we started this project with Steinwurf, the proxy application were tested using simulations and has not been tested across high latency low bandwidth satellite links.[2]. Attempting to install the software on Internet connected nodes, we performed tests with a node configured as TCP/NC proxy and server.

As shown in Fig 10.2, we installed the TCP/NC software in two nodes (proxy and

[2]Note that a similar implementation of the user space proxy is available as open sender software in TCP Performance-Enhancing proxy (TCPeP) [191]. However this version is based on TCP/NC by Sundararajan et al. [183]

Figure 10.2: Network topology for the first deployment of the TCP/NC SOCKSv5 proxy application



Note: The main requirement for TCP/NC is for traffic to be redirected through the tunnel. The TCP/NC proxy encodes packets and sends them through the tunnel to the TCP/NC server where coded packets are decoded.

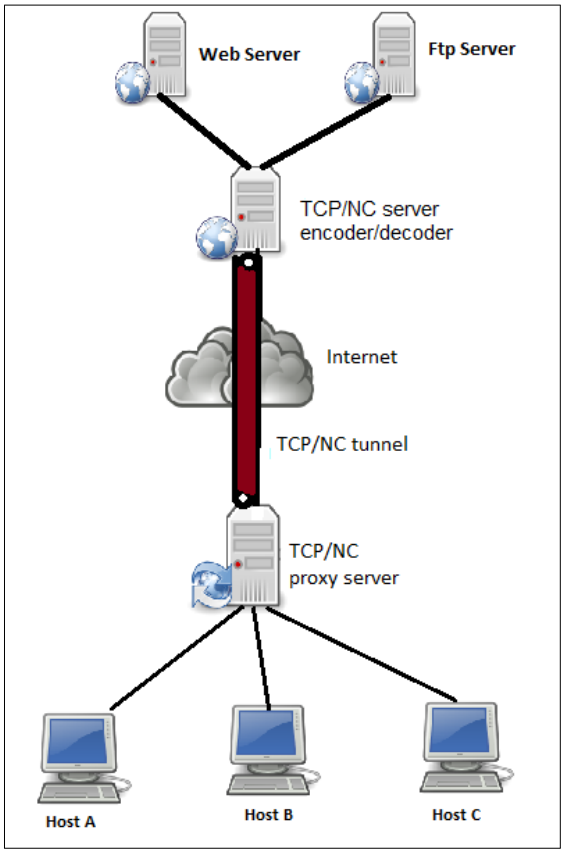
server) and established a tunnel between them. The reason for the establishment of the tunnel is to ensure that our coded packets are exchanged between two end points capable of encoding and decoding packets so the end hosts don't have to do it.

In the proxy server, we use IPtables to redirect traffic from host A to C to the TCP/NC proxy application. The proxy application encodes TCP based requests (e.g. Web, FTP) from host A, host B, or host C and sends them as UDP packets through the TCP/NC tunnel to the TCP/NC server. Upon reception of the encoded packets, the TCP/NC server decodes and forwards the connection request to services installed in the same host (Web or FTP).

When these services acknowledge the TCP connection and send packets back

to host A to C, the TCP/NC server encodes and sends the return packets as UDP packets through the tunnel to the proxy server. The proxy server receives the coded return packets, decodes them to yield the original return packets and forwards them to the sender. Of course, this only works as long as the server runs other services such as a Web or FTP server. However, this is not the way the Internet works and we cannot simply ask companies such as Google or Facebook to install our software in their Web server for improving throughput.

Figure 10.3: Network topology for the second deployment of the TCP/NC SOCKSv5 proxy application



Note: The difference between this diagram and Figure 10.2 is the separation of Internet services from the TCP/NC server. The main requirement here is for the return traffic from these services to be received by the TCP/NC server for encoding and transmission to the TCP/NC proxy.

In order to solve this problem, we use the TCP/NC server to encode/decode packets from the proxy server, and route the decoded packets to servers on the Internet. As shown in Figure 10.3, we remove the server services from the TCP/NC

encoder/decoder and dedicate the functionality of the TCP/NC server to encoding, decoding and routing of decoded packets. However, this only solves the forwarding of packets *to* the Internet, but the return traffic *from* the Internet must also be returned to the TCP/NC server for re-encoding and transmission to the proxy server. The reason that this is not an easy task is because the encoder/decoder is not the gateway for host A to C for traffic to be routed naturally from the Internet services to the hosts. Therefore the return traffic from server services on the Internet may not be routed back to the TCP/NC server to be encoded/decoded but may be routed directly to host A to C.

A simple way around this problem is to perform network address translation (NAT) at the TCP/NC server. With NAT translation, the TCP/NC server replaces the original source address of the packets from host A to C with the TCP/NC server's IP address. As a result, services such as FTP and Web see the IP address of the TCP/NC server as the source address, and so any return traffic will be forwarded back to the TCP/NC server for encoding/decoding.

Although our design in Figure 10.3 is more compatible with the way the Internet work, we noticed that the software encountered delays. Firstly, NAT requires IPtables [192] to be configured to translate between addresses. Secondly, the TCP/NC software requires IPtables to perform port redirection where incoming TCP traffic from host A to C are forwarded to the TCP/NC proxy application to be encoded/decoded. The use of IPtables for NAT and port redirection are additional delays that renders any throughput improvements with TCP/NC unnoticeable.

10.2.2.2 TCP/NC Tunneling Solution

Attempting to solve this problem, Steinwurf ApS [190] developed a TCP/NC tunnelling solution at our request, which operates at the kernel level of the operating system. This version of TCP/NC intercepts a certain number of IP packets. This set of packets is known as a generation and the number of packets in a generation as the *generation size*. It then uses coefficients to generate coded packets as linear combinations of the intercepted IP packets. Note that the coefficients are embedded

into the header of the coded packets. The number of coded packets generated is that of the original packets plus a number of additional ones, referred to as the *overhead*. The set of coded packets represents an overdetermined system of linear equations. We transmit the coded packets to the other endpoint of the tunnel as UDP.

At this point, the source and destination IP address of the linear combined UDP packets is the address of the encoder and decoder. This also means that we encode the entire original IP packet including the source and destination IP address of host A to C and server services (see Figure 10.3) inside the coded UDP packets. As such, this version works without the need for NAT or port redirection and this is a major performance improvement as compared to the proxy version.

At the decoder, coded UDP packets are scanned through and when there is enough packets to recover the original IP packets, the receiver extracts both the coefficients and overhead and attempt to solve a system of linear equations. Because of the overhead, we can afford to lose packets on a lossy link and still manage to decode the original information.

In order to adjust the functionality of the kernel version to be suitable for high latency low bandwidth satellite links, Steinwurf performed tests using hosts at the University of Auckland and Rarotonga. Their tests concluded that it is practically impossible to use degrees of freedom. Their comments on the experiments showed that the delays render any feedback unusable because the time it takes to receive feedback is too long.

For example, take an 8 Mbps MEO link with 125 ms RTT and generation size of 60, where the size of each packet is 1500 bytes. Thus the entire generation is 90 kB (720 kb). On this link, the entire generation takes less than 1/8th of a second to transmit. However, the first degrees of freedom feedback arrives only after the whole generation has already been transmitted. At this point, the actual DOF at the encoder could be anywhere between 1 and 60, which means the feedback is meaningless. This problem gets worse with higher bandwidths and higher latencies (i.e., it applies to all satellite links under consideration).

We also contributed to the development of the tunnel version in the following

ways: We used our experience with deployments in Figure 10.2 and Figure 10.3 to provide network design information for the development of the kernel module. In addition, we designed two network topologies for the deployment of TCP/NC, which we discuss in the next section.

10.3 Deployment of TCP/NC

We designed two network topologies for the implementation of the TCP/NC kernel module, and propose to use them for deployment of the software. Before discussing the details of our designs, we review the differences between the user space and the kernel version of TCP/NC.

Previously in Section 10.2, we mentioned that the user space application was developed to perform end-to-end encoding/decoding of packets. We also mentioned that there is a need to install the TCP/NC application together with Web or file transfer services. In order to make end-to-end encoding and decoding of packets work, end users and server operators need to install TCP/NC on their machines if they are going through a proxy. As a result, this means that TCP/NC in an end-to-end model will not work without integration into every client and server software.

The user space version of the proxy had two shortcomings. Firstly, it encoded the payloads but not the headers meaning that any encoder and decoder needed to be on the path between the two end hosts. Secondly, user space versions require copying of packet data, which slows processing down. This issue has also been raised in the context of Performance Enhancing Proxies (PEP) [191]. What was needed was a TCP/NC tunnel that encoded packets across the satellite link, and accepted and output unencoded traffic at the tunnel endpoints, and a network topology that allowed the encoding/decoding tunnel endpoints to be anywhere as long as they were on different sides of the satellite link.

Our colleagues at Steinwurf considered the issues with the user space version and developed a kernel module to permit network coding over the satellite link via

a tunnel without having to integrate TCP/NC with the endpoints. At the time of writing, we have installed the kernel module in four sites in three countries in the Pacific (two in the Cook Islands, one in Niue, and one in Tuvalu). As shown in Figure 10.4, we installed and configured the software in the TCP/NC server, and the on-island encoders/decoders.

In Figure 10.2, the execution of the user space TCP/NC application established a tunnel between the TCP/NC proxy server and the TCP/NC server. Once established, TCP traffic was redirected to the TCP/NC application where it was encoded and forwarded through the tunnel. In order to adapt our topology in Figure 10.4 for packets to be transmitted through the tunnel, we borrow a small portion of the University of Auckland (UoA) network address space, and configured hosts on the island with an UoA IP address. In Figure 10.4, the gateway for these hosts is the on-island encoder/decoder. Note that we also arranged for the border router at UoA to route traffic for the borrowed subnet to our off-island encoder/decoder.

The reason that we use a borrowed subnet from UoA is so that return traffic can be routed back to the University of Auckland, for our off-island TCP/NC encoder/decoder to code it. In Figure 10.4, when hosts on the island use TCP to perform Google searches or YouTube video streaming, the connection request is sent to the on-island encoder/decoder to be encoded and encapsulated into UDP packets for tunnel transport. When the encoded UDP packets arrive at the other end of the tunnel, the off-island TCP/NC encoder/decoder strips off the tunnel encapsulation and attempt to decode the original TCP packets. Once decoded, the TCP/NC server forwards the original IP packets to the respective server (Google or YouTube) on the Internet.

When there is return TCP traffic from servers on the Internet, the sender IP address becomes the new destination address. In this case, the return traffic from the Internet server is forwarded to the border router at UoA. The border router routes our borrowed subnet to the TCP/NC server. When the return traffic arrives at the TCP/NC server, it is encoded into UDP packets and forwarded to the on-island encoder/decoder. Here, the UDP packets are decoded to regenerate the original TCP return traffic, and then forwarded on to the host that initiated the original

traffic.

10.3.1 Network Topology for Production Deployment

Although our design in Figure 10.4 worked for our experiments, it is not ideal for use in production environments for the following reasons. Firstly, most ISPs that may use the software have a large address space and so there is no need to borrow a subnet. Secondly, ISPs in the Pacific may prefer to install the off-island encoder/decoder at the other end of the satellite link and in this situation, there would be no need for packets to pass through a more remote encoder/decoder in Auckland.

Our topology for the deployment in a commercial setting is slightly different from Figure 10.4. Instead of borrowing subnets, ISPs can delegate one of their existing subnets to their upstream service provider to announce to the world via BGP [193]. As shown in Figure 10.5, assuming that the ISP had installed the off-island encoder/decoder as well as delegates one of its subnets to its upstream provider, the ISP can establish a tunnel between the on-island and off-island encoder/decoder.

In Figure 10.5, when the on-island host initiates a Google search, the host sends TCP traffic to the on-island encoder/decoder where they are encoded into UDP packets for transmission through the tunnel. At the far end of the tunnel, the off-island encoder/decoder decodes the coded packets to reconstruct the original TCP traffic and forwards the traffic to the Google server on the Internet. Note that this is the same process that we described in Figure 10.4.

Since the on-island host uses an address from the delegated subnet, the return traffic from Google will be transmitted to the upstream provider's core router because this is the network where the on-island host are visible to the Internet.

At the upstream provider, the core router routes traffic for the announced subnet to the TCP/NC server. Once received, the off-island encoder/decoder encodes the

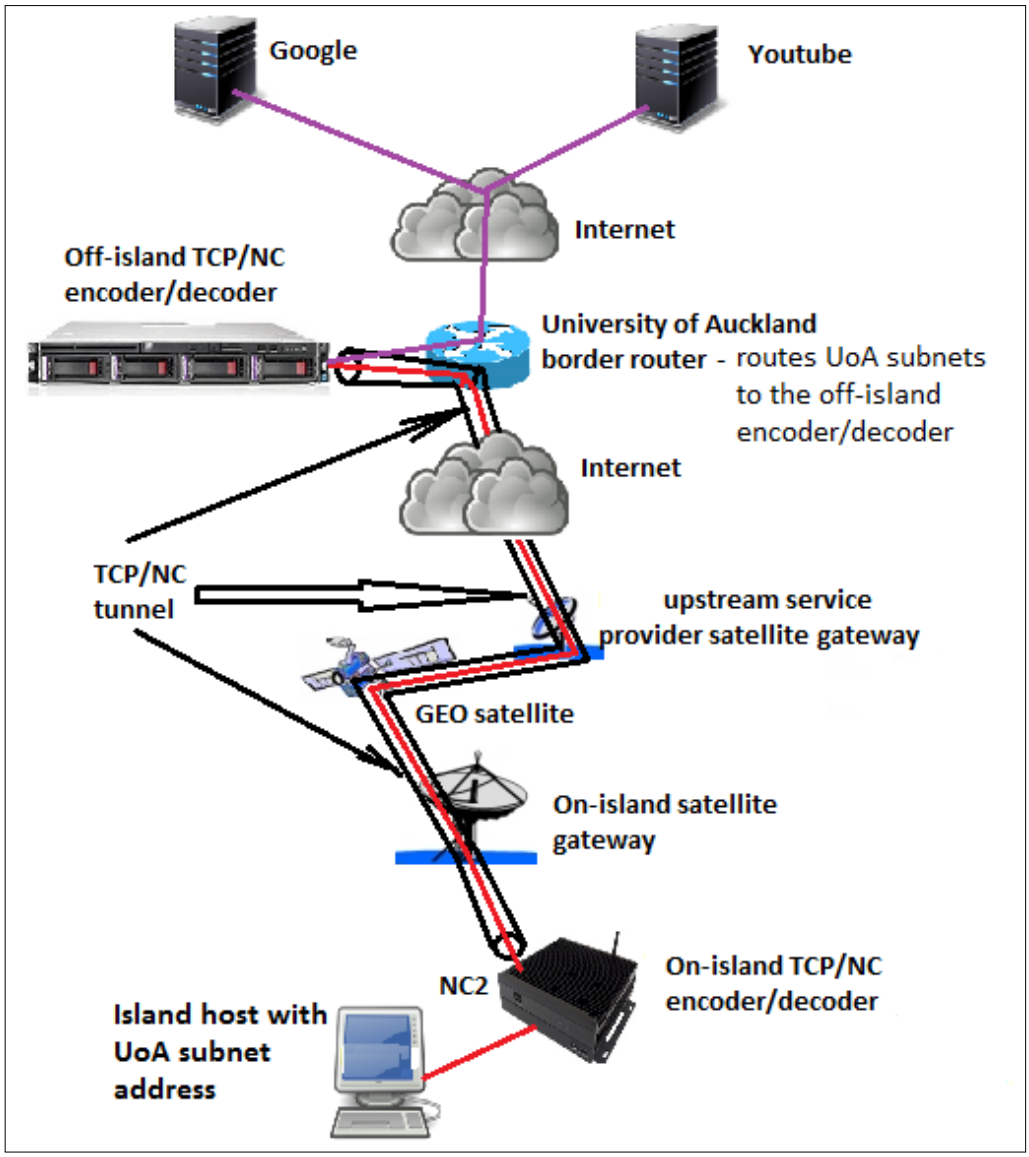
TCP return traffic into UDP packets and transmits to the on-island encoder/decoder. Once received, the on-island encoder/decoder decodes them to reconstruct the original TCP return traffic, and forwards the return traffic to the on-island host.

As shown in Figure 10.5 we include in our design a redundant TCP/NC server. The idea here is for the on-island encoder/decoder to perform BGP multi-homing [194, 195], with the server located at the upstream provider and the Steinwurf TCP/NC server. For this to work, ISPs that uses this solution will require an autonomous system number (ASN) [196] that we can use to perform BGP peering. With the ASN, the ISP may configure BGP to prefer the link between the on-island encoder/decoder and the upstream service provider and only when the upstream service provider server is down, that traffic is routed to the Steinwurf TCP/NC server. Of course there will be no redundancy if there is link failure in the satellite connection, but this problem is beyond the scope of this thesis.

10.4 Summary

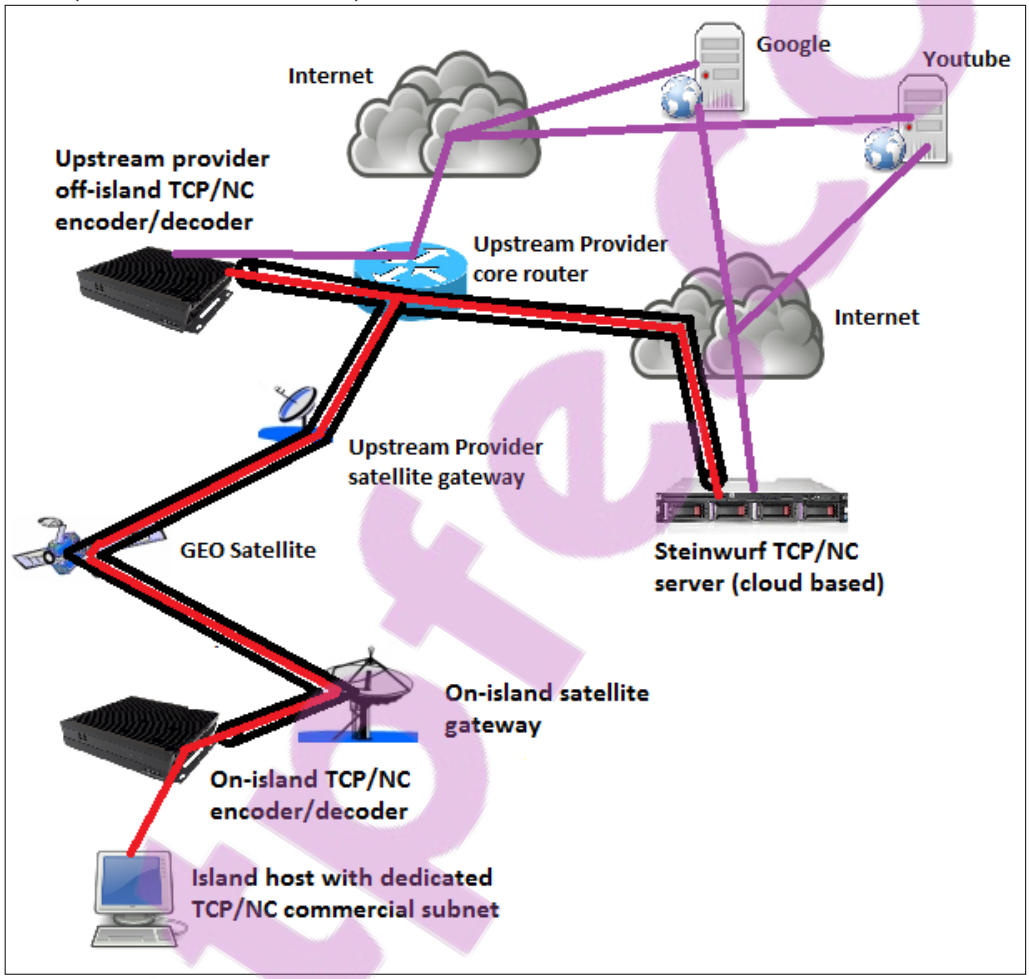
In this chapter, we reflected on possible solutions for improving throughput in high latency and low bandwidth networks. In the last section, we looked at possible topologies for the implementation of TCP/NC. We will now discuss the results of our TCP/NC experiments in Chapter 11, focusing on implementation challenges and the deployment of TCP/NC, issue of queue oscillation and how TCP/NC is a solution to this problem, and results from TCP/NC experiments.

Figure 10.4: Network topology for the deployment of the TCP/NC tunneling solution



Note: The red line is the TCP traffic. The black line is the TCP/NC tunnel. The purple line marks the end of the tunnel and the connection between our TCP/NC server and servers located on the Internet. Note that the tunnel is not restricted to the transmission of TCP traffic but can be used to transmit any kind of protocol (i.e. UDP, ICMP).

Figure 10.5: Network topology for the deployment of the TCP/NC kernel module (commercial license)



Notes: Similar to Figure 10.4, the red line that is encapsulated inside the black tunnel is the TCP/NC tunnel. The purple line marks the end of tunnel and shows the return traffic from Internet servers to the TCP/NC server. The difference between this topology and Figure 10.4 is the redundancy between the upstream service provider and Steinwurf's cloud-based TCP/NC server.

CHAPTER 11

TCP/NC Implementation Challenges and Results

In this chapter, we present challenges that we encountered with the deployment of the TCP/NC kernel module. We follow this with a review of our results, where we compare the performance of some of TCP variants with TCP/NC.

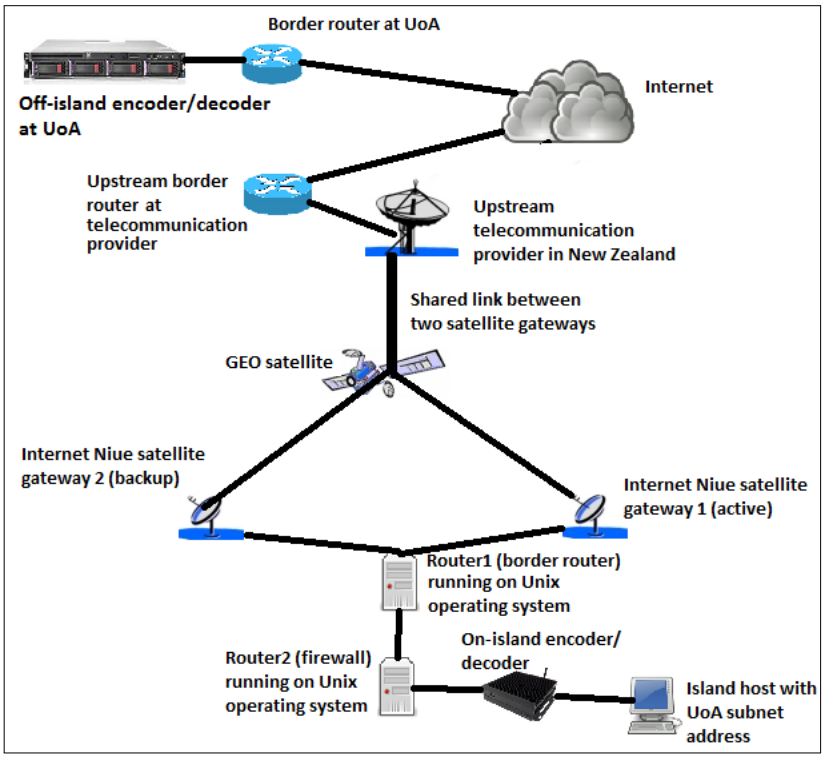
11.1 Implementation Challenges

In Chapter 10, we discussed the deployment of the TCP/NC kernel module at five sites: the off-island TCP/NC encoder/decoder at the University of Auckland in New Zealand, two on-island encoder/decoder in the Cook Islands (Rarotonga and Aitutaki), on-island encoder/decoder in Niue, and on-island encoder/decoder in Tuvalu (Funafuti). The fundamental structure of our deployment was the same in all five sites. We installed the TCP/NC kernel module in all five sites in order to create tunnel endpoints (on-island encoder/decoder) there. Note that the latest version of the software is able to support a small number of tunnel endpoints.

Referring to Figure 10.4 of Chapter 10, when one of the hosts connected to the island endpoint requests information from a server such as (e.g. YouTube), the kernel module on the island intercepts IP traffic from the respective host and encodes it into network-coded UDP packets. Once encoded, the packets are transmitted as UDP tunnel packets to our off-island encoder/decoder in Auckland.

At the off-island encoder/decoder in Auckland, the kernel module decodes the coded packets and forwards the decoded IP packets to the YouTube server via the Internet. When return packets from the YouTube server arrive at the off-island encoder/decoder, they are encoded and transmitted as coded packets to the respective tunnel endpoint on the island. The kernel module at the island decodes the response and sends the decoded packets to the host on the island that initiated the original packets.

Figure 11.1: Our initial setup of the on-island encoder/decoder at Internet Niue



Note: This is a simplified diagram of Figure 10.4 in Chapter 10.

11.1.1 Deployment Issues in Niue

Problem can arise if there are firewalls involved in the passing of packets between the TCP/NC encoder and decoder. For example, in Niue the on-island encoder/decoder was installed behind the border router and firewall (see Figure 11.1). After using the kernel module to establish the tunnel between the on-island encoder/decoder and the off-island encoder/decoder in Auckland, we were unable to ping from the island host to Internet hosts. Similarly, we were unable to ping from hosts on the Internet to the on-island host.

If the network design for the deployment of the TCP/NC kernel module is similar to Figure 11.1, we normally recommend that the on-island encoder/decoder be placed closer to the border router to eliminate filtering effects from firewalls. However, in our deployment in Niue, we had placed this node close to the border router but the problem still persisted.

In the Niue deployment, we found out that their border firewall (Router1) was using an older version of `Pfsense` [197]. As a result, packets forwarded between interfaces with unusual MTU were dropped by the firewall in Router1. Pfsense had already fixed this problem, and we thus recommend that networks using Pfsense as router/firewall update the firmware to version 2.2 or later.

11.1.2 Deployment Issues in Tuvalu

Another issue that we encountered during our deployment in Tuvalu was that part of their network equipment was not managed and controlled by the local ISP, but by their upstream satellite service provider. While this was not a fundamental problem, it meant having to liaise with two different entities to accomplish the installation of our tunnel endpoint. Such organisational arrangements increase the effort involved in deployment and therefore represent to an extent an engineering cost.

The upstream provider also installed a wide area network (WAN) accelerator device, Silverpeak NX3700 [198], that intercepts traffic from users. There were two

such devices installed, one at Telecom Tuvalu, the other at the off-island satellite gateway. The devices would then modify the traffic passed between the on-island and off-island networks.

The Silverpeak device can provide basic FEC through parity packets and also provides network memory, so that identical packets are not retransmitted across the satellite link. Although the details on the configuration of this device were not available to us, we were able to infer from the network traffic passing through the Silverpeak device that some of these functions were activated, including forward error correction and network memory.

Other optional features of the Silverpeak device include packet reordering, coalescing of smaller packets into larger ones, IP header / payload compression, and TCP and other protocol acceleration. However it was unclear which of these features were activated because we do not have access to the Silverpeak device. After establishing the TCP/NC tunnel and testing connectivity between the off-island encoder/decoder in New Zealand and our hosts behind the tunnel end point in Tuvalu, we received no response.

This suggested that the Silverpeak device was interfering with the TCP/NC traffic. In a situation like this, it is best to test the connectivity without the tunnel. In our case, we tested the connection without the tunnel and found out that there was a missing route from Tuvalu to our off-island encoder/decoder in New Zealand. We alerted the upstream provider to this problem and they applied the appropriate route.

After establishing the tunnel with the off-island encoder/decoder, we were still unable to send traffic through the tunnel.

We eventually traced this problem to a Cisco ASA firewall [199] in the path between Telecom Tuvalu and the Internet. As it turned out the checksum algorithm used by the kernel module was raising suspicion flags in the ASA firewall. Consequently, the firewall dropped our packets. Our software supplier Steinwurf supplied a solution to this problem which involved modifying the TCP/NC kernel module to use the UDP magic checksum method [200].

Once again, our experience in Tuvalu showed that having complex firewall logistics between two tunnel endpoints can be a potential source of problems. When deploying TCP/NC, there is a need to identify firewalls between end points and additional configurations may be needed to ensure that TCP/NC packets are not dropped.

11.2 Preliminary Observations

In most of deployment sites, we were able to obtain link utilisation data either provided by our research partners or through our own measurement with NTOP and Nprobe. In Rarotonga and Tuvalu in particular, the inbound utilisation was significantly below the theoretical link capacity. In Rarotonga, the typical utilisation was between 90 and 100 Mbps, with a maximum peak of 124 Mbps; this compares with a total link capacity of 160 Mbps at that time.

In Tuvalu, we were unable to determine the actual installed and provisioned capacity of the link, but our own measurements indicate that the link capacity was around 16 Mbps. We also found out from our measurements that conventional TCP over Silverpeak was only able to utilise up to about 3 Mbps of the total bandwidth. This raises the question as to why it is not possible to achieve maximum throughput over such a satellite link.

During the time that links are underutilised, the queue of the satellite modem is empty and there is nothing transmitted from the queue and therefore the link becomes idle. An idle link is not good because the ISP in the island is paying for capacity that they are not using.

However, our observations show that we also get burst packet losses on the link between New Zealand and Tuvalu. We do not get the same effects on links between Europe and New Zealand that use high speed fibre optic cables.

11.2.1 Queue Oscillation

What do these observations mean for a satellite connection? When a large number of TCP connections are initiated from different hosts across the satellite gateway, the connections all follow a slow start phase while the sending hosts await the arrival of acknowledgements from the receiving hosts. Assuming that the acknowledgements are received, TCP increases the congestion window size of the sending hosts to transmit more data.

When multiple hosts transmit large amounts of data, the queue at the gateway becomes longer and longer and eventually overflows. When the queue overflows, the gateway drops packets because there is no way of accommodating packets in the queue. Packets that are dropped are not acknowledged and so the sender remains unaware of this situation until the timeout period for receiving acknowledgements elapses.

When queue overflows in a long latency environment such as satellite links, a large number of packets are already on route from the sender to the satellite gateway. Because of the overflow, the gateway drop packets causing multiple ACK to be missing which resulted in burst packet losses.

Subsequently, TCP's congestion control algorithm on the TCP senders slows down traffic. This causes the arrival rate of packets at the satellite gateway to decrease sharply from exponential back-off. As a result, the queue at the satellite gateway clears and at this point the satellite gateway is underutilised. When the queue is empty, the senders detect low packet loss in the link and retransmit lost packets. When packets arrive at the destination, the receiver generate ACKs and the whole process repeats; i.e. the queue becomes longer again, causing the gateway to drop packets and the senders to slow down once more.

This is a symptom of *queue oscillation* [10, 11, 12] where the queue runs dry and subsequently overflows again when the TCP senders pick up the speed of transmission again. This is a well known effect and one can determine queue oscillation when there is under utilisation of a link, and burst packet losses. Note that we

did not observe the effects of queue oscillation in Niue because the link was busy transporting 80 % to 90 % of goodput. Although we observed under utilisation of the link in Aitutaki, it was not because of queue oscillation but a lack of demand for Internet connectivity. However we witnessed queue oscillation in Rarotonga in the Cook Islands.

Therefore the question that arises is whether we can fix the problem of queue oscillation. TCP/NC is a possible answer to this question because it hides the packet losses from the TCP sender. This means that the TCP sender will not slow down during burst packet losses, thus enabling TCP to achieve higher goodput even under burst packet losses that are smaller than the overhead.

11.3 Results

As discussed in the previous section, TCP/NC can in principle mask the packet losses and therefore prevents TCP sender from slowing down its packet transmission rate under burst packet losses.

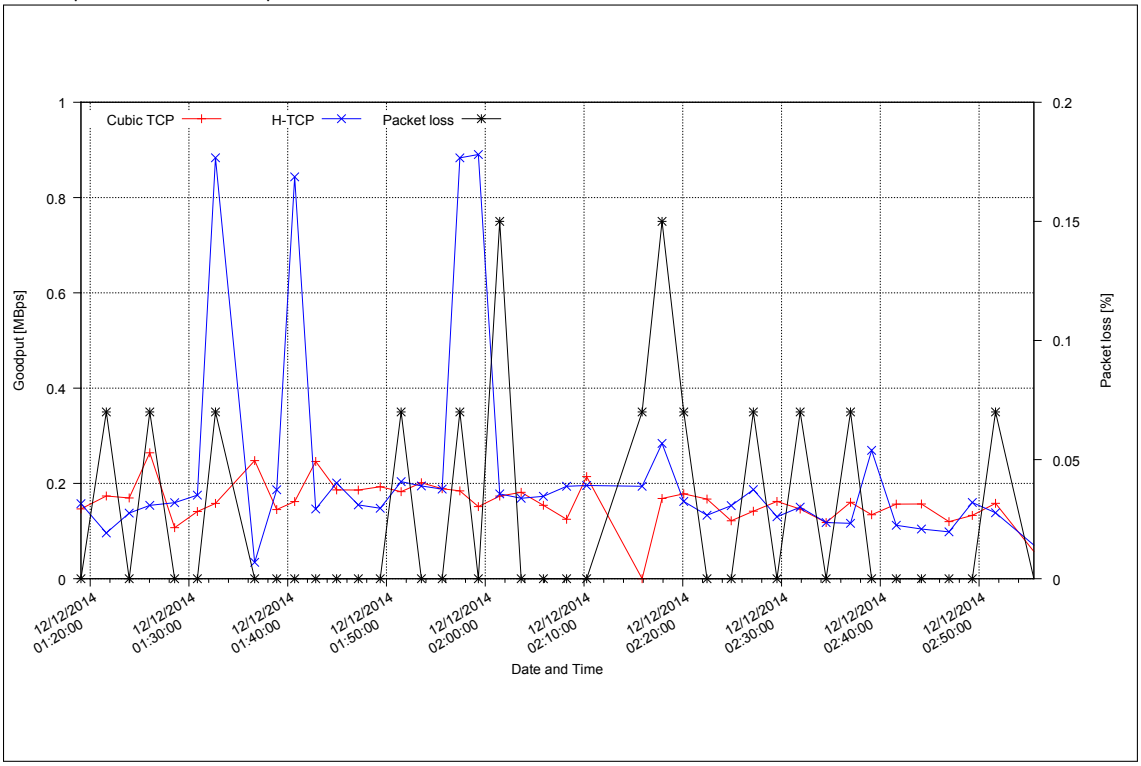
In this section, we look at the comparative performance of TCP/NC and a number of classical TCP variants: Cubic TCP, H-TCP, and Hybla. We have already discussed these TCP variants in Section 10.1.1 of Chapter 10.

From the list of TCP variants, Cubic was the most widely deployed TCP variant and the default TCP in Ubuntu Linux distributions at the time of writing. Therefore, we used Cubic as the benchmark for our experiments. Figure 11.3 shows that Cubic generally outperforms Hybla most of the time.

In Figure 11.2, all packet losses were below 0.2% and the relative performance of H-TCP and Cubic shows very little correlation with packet loss. At times H-TCP can perform vastly better than Cubic. However, most of the time the performance is comparable, and in some cases Cubic TCP significantly outperforms H-TCP.

This does not present a clear case for the replacement of Cubic TCP with Hybla

Figure 11.2: Goodput and packet loss for experiments between the off-island TCP/NC encoder/decoder in New Zealand and the tunnel end point in Niue



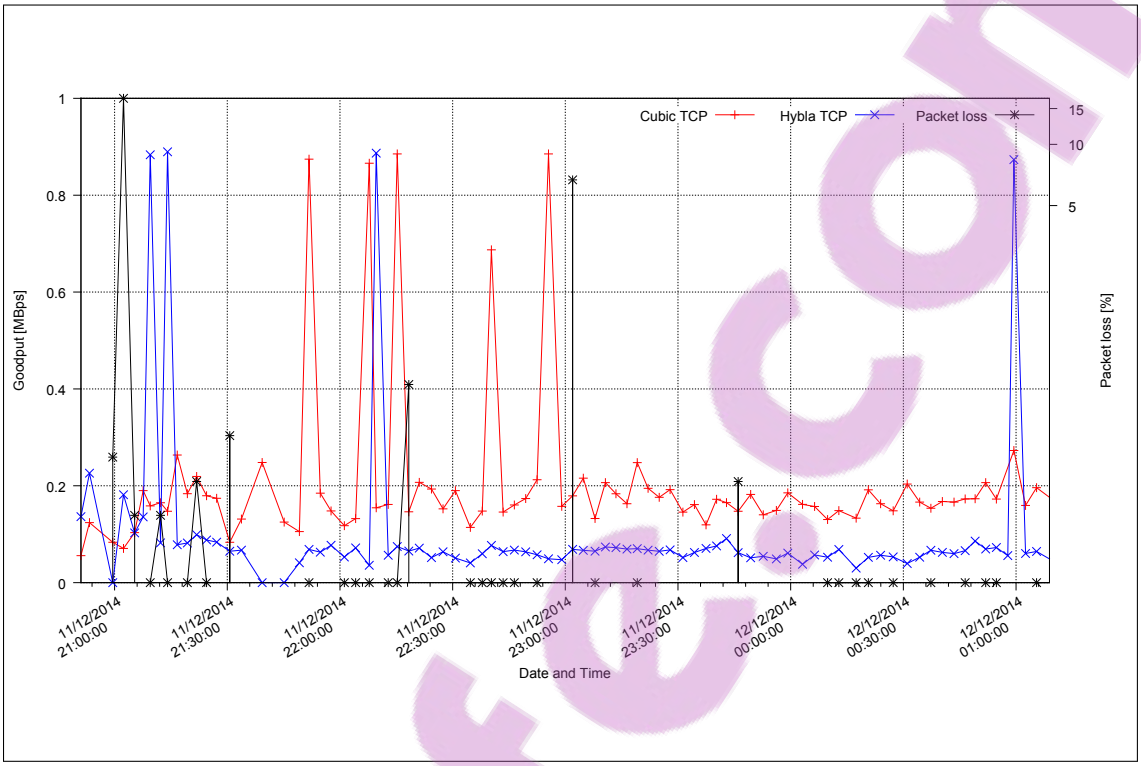
Notes: In our experiments, we upload a file with iperf from the off-island encoder/decoder to the tunnel end point in Niue. We performed our experiment at a time when the network was not overly busy (off-peak). Note that we turned off the TCP/NC tunnel at the time of this experiment. The time axis is in Niue time (NUT).

or H-TCP, however, because Hybla and H-TCP were specifically designed for wide band links and not the high latency narrow band links we encounter in the Pacific.

11.3.1 Results: TCP/NC and Other TCP Variants

Ideally, we wanted to investigate the impact of high packet loss on the performance of TCP/NC and TCP. As shown in Figure 11.4, the moment there is significant packet loss between 8 pm on the 26th and 5 am on the 27th of January 2015, the performance of conventional TCP degrades.

Figure 11.3: Goodput and packet loss for experiments with Cubic and Hybla TCP



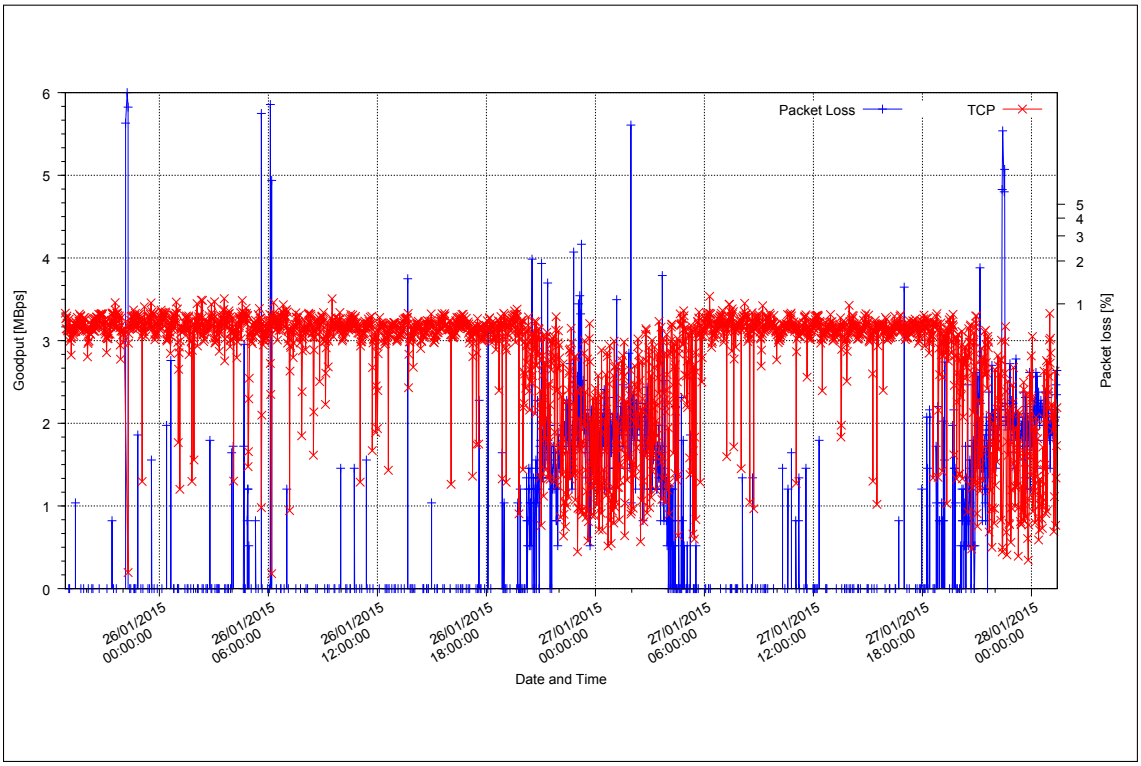
Note: As in the experiments in Figure 11.2, we uploaded a file from New Zealand to Niue for each measurement.

However, this is not always the case and it is possible to observe low performance in conventional TCP without significant packet loss. An example is provided in Figure 11.4 between 2 pm and 4 pm on the 27th January 2015, when there was a fall in goodput to as low as 1 MBps but there was no significant packet loss.

Furthermore, Figure 11.4 shows a number of occasions when there is high packet loss but no impaired performance of conventional TCP. This occurred between 10 am and 11 am, 2:50 pm on the 26th January 2015 and at 5 pm on the 27th January 2015, for example.

In December 2014, the O3b satellite connection to Rarotonga had a 160 Mbps downlink and a 40 Mbps uplink. We performed tests with iperf and found that burst packet losses were large enough to require a network coding generation size of 30 and an overhead of 50% to recover packets from burst losses and for TCP/NC

Figure 11.4: Goodput for Cubic TCP and packet loss for experiments between our off-island encoder/decoder in New Zealand and the tunnel endpoint in Rarotonga

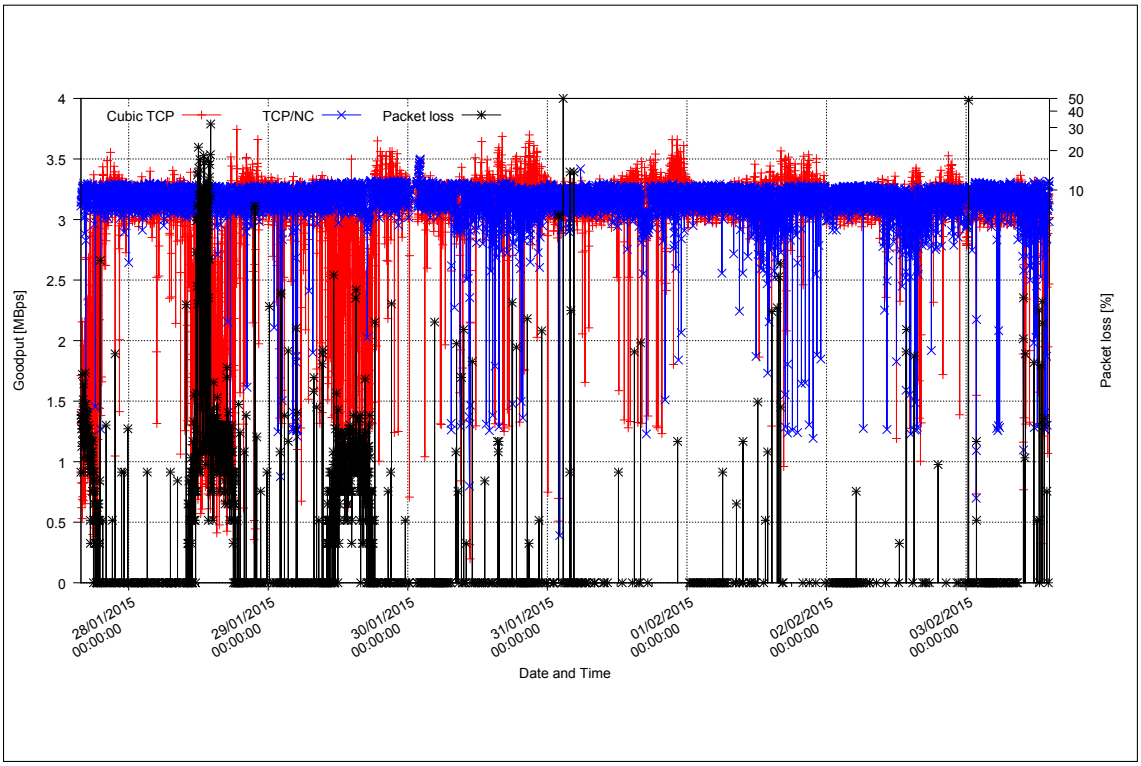


to outperform Cubic TCP.

Revisiting the link in January 2015, we found out that using the same generation size and overhead, Cubic TCP outperformed TCP/NC at most times. Although we were unable to confirm this with Telecom Cook Islands on the new available capacity on the link, we observed that the characteristics of the link had changed and we now got low packet loss. A possible cause could be unnotified increase in link bandwidth by their upstream provider O3b.

In an attempt to improve the performance of TCP/NC, we reconfigured the tunnel between the server in New Zealand and the tunnel end point in the Cook Islands with a reduced overhead of 20% in January, that is, a generation size of 30 and an overhead of 6. This configuration improved the performance of TCP/NC to be comparable with conventional TCP.

Figure 11.5: Packet loss and goodput for TCP/NC and Cubic TCP between the TCP/NC server in New Zealand (in Figure 10.4 of Chapter 10) and the tunnel end point in the Cook Islands



Note: This graph was reproduced from Figure 2 in [9]. In this experiment, we configured the TCP/NC kernel module to use a generation size of 30 and overhead of 6. Note that TCP/NC outperforms conventional TCP during times when conventional TCP performance is impaired by packet loss and other effects.

As shown in Figure 11.5, TCP/NC matches the goodput of Cubic TCP of between 3 and 3.3 MBps. However, when there are significant packet losses, TCP/NC performs better. This is shown in Figure 11.5, when packet loss increases to above 18% between 9:00 am and 6:00 pm on the 28th of January 2015. Correspondingly, TCP degrades to between 0.6 MBps and 2.5 MBps while TCP/NC maintains high goodput between 3 and 3.3 MBps.

The above measurement series shows a large number of features that we consider worthy of further investigation. Note that investigating these artifacts requires generating traffic, recording information from experiments and transferring the results to our repository server in New Zealand. This presents a significant load on our re-

search partners' networks. In order to minimise the impact of our experiments, we should therefore only perform a small number of experiments. This is particularly the case for the deployments in Niue and Tuvalu which have narrow band satellite connection.

11.3.2 Results in our Niue Deployment

Niue uses a GEO satellite to connect to the Internet. This connection has a throughput of 8 Mbps downlink into Niue and 4 Mbps uplink. The satellite capacity is almost fully utilised during peak times and, in order to avoid flooding the satellite gateway with our coded TCP packets, we performed most of our experiments during off-peak times.

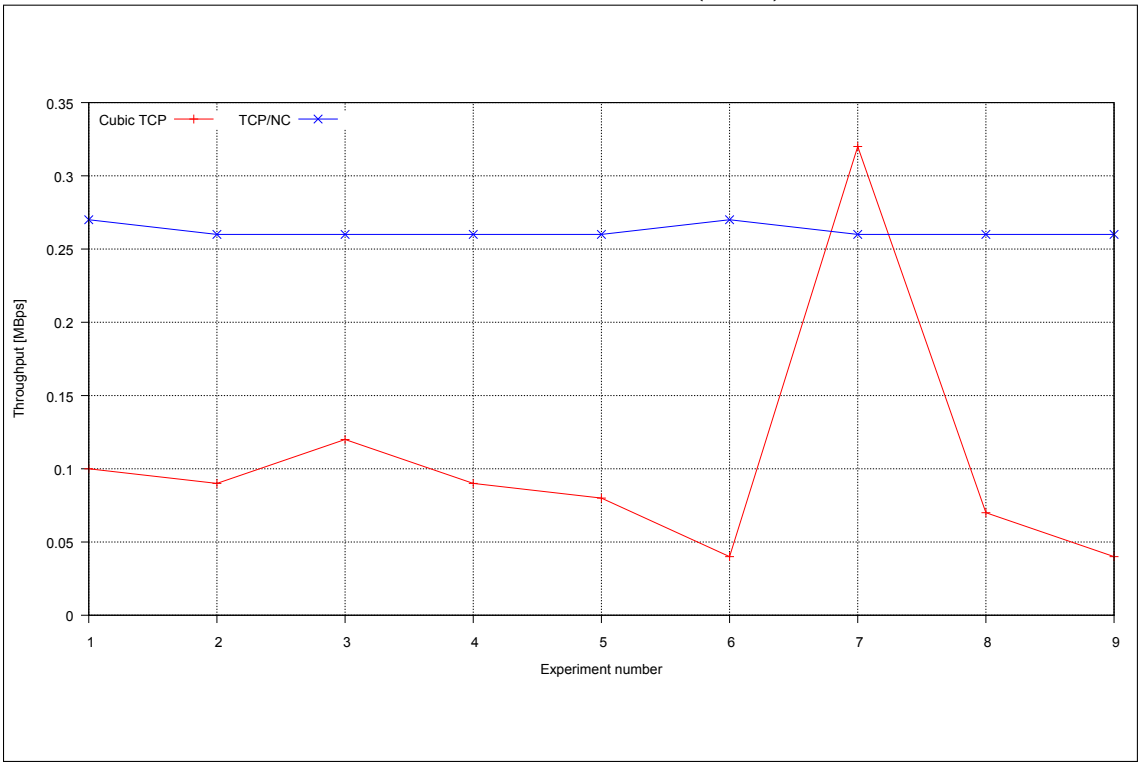
In Figure 11.6, we configured the tunnel with 50 percent overhead, that is a generation size of 30 and an overhead of 15. TCP/NC almost consistently yielded a higher throughput of at least 0.25 MBps, while Cubic barely exceeded the 0.1 MBps mark. However there was one occasion in experiment number 7, where Cubic achieved a higher throughput of 0.33 MBps while TCP/NC remained at 0.26 MBps. Although we did not record packet loss in Figure 11.6, it is possible that the spike in TCP performance corresponded to low packet loss during the time that we performed the experiments.

When TCP/NC achieves higher throughput, the coded UDP packets takes up most of the free space in the link thus squeezing out conventional TCP. As a result, conventional TCP gets less link capacity and users may experience delays.

As mentioned in Section 11.1.2, the Silverpeak NX3700 device in Tuvalu intercepts all TCP traffic to and from the Tuvalu Telecom network. In addition to the techniques we mentioned in Section 11.1.2, this device uses High Speed TCP (HSTCP) [201] to improve throughput [202].

As shown in Figure 11.7, the performance of TCP over the Silverpeak NX3700 device shows higher throughput than TCP/NC of about 0.7 MBps to 1.8 MBps between the 17th January 2015 and the 19th January 2015 at 6:00 am. On many of

Figure 11.6: Throughput for Cubic TCP and TCP/NC from a number of experiments that we performed with iperf between the TCP/NC server in New Zealand and the tunnel endpoint in Niue (NC2)



Note: We performed 9 experiments with iperf between 8pm and 9pm Niue time (NUT). Because of the extremely constrained bandwidth in Niue, we performed only a small number of experiments.

these occasions, the packet loss was low. During these times, we could have relaxed the generation size and overhead of TCP/NC to achieve higher throughput with TCP/NC.

However, we can see higher throughput with TCP/NC on the 19th of January 2015, roughly between 10 am and 6 pm. It is during busy periods (business hours) when there are burst packet losses which degrade HSTCP throughput while TCP/NC remains stable.

To conclude, our results show that we can achieve higher goodput with TCP/NC during times when there is packet loss. However, the TCP/NC kernel module must be aware of the condition of the links to adjust its overhead. One option for providing

Figure 11.7: Throughput from experiments conducted with iperf between the TCP/NC server (New Zealand) and the tunnel end point in Tuvalu (NC3)

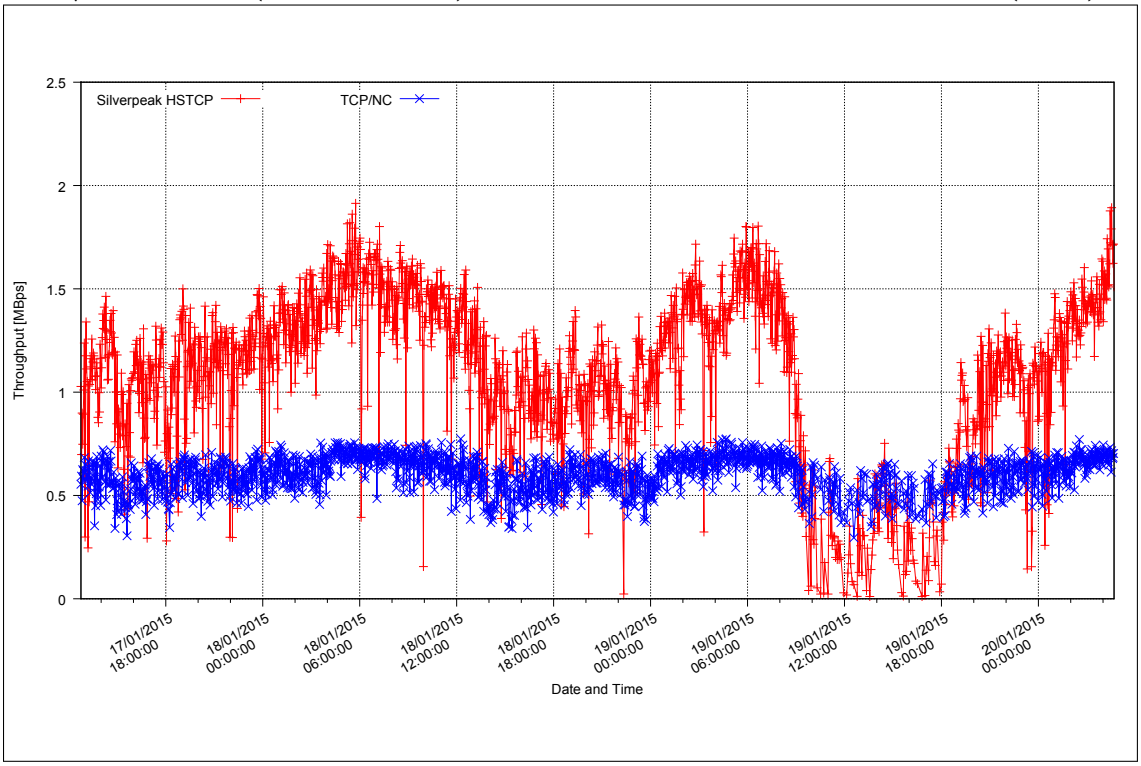


Figure 11.8: *Note: The comparisons here is between TCP/NC and HSTCP used by the Silverpeak NX3700 device in Tuvalu.*

feedback to the sender in TCP/NC is to feed back the degrees of freedom to the sender. However we did not use this technique in the kernel module because the high delay from the satellite propagation delays the arrival time of feedback with degrees of freedom.

11.4 Summary

In this chapter, we discussed challenges that we encountered in the implementation of the TCP/NC kernel module in Rarotonga and Aitutaki (Cook Islands), Niue and Funafuti (Tuvalu). Furthermore, we also compared results for a number of TCP variants and TCP/NC. We will now conclude this thesis in the next chapter.

CHAPTER 12

Conclusion

In the course of this thesis, we developed an active measurement utility that we refer to as our `beacon` software. While a number of active measurement utilities exist and were presented in Chapter 2, we use our software and the beacon network to make the following contributions to our research questions.

Research question: How stable is the latency on the Internet?

1. At the time of writing we had deployed the `beacon` software on 33 hosts located in 16 countries worldwide.
2. We performed experiments emulating VoIP over UDP and TCP and TCP file download *over an extended period* of time.
3. We generated a *large data repository* of currently 491 gigabytes worth of log files spanning a period of over 4 years of measurement for some of nodes.
4. In Chapter 6, we *applied information theory* to network measurement. We use T-entropy to determine the occurrence of systematic and queue induced jitter in our data.

Research question: How will the quality of Internet connectivity change over time with additional links and routers?

1. We used estimated MOS to track the expected quality of VoIP applications over time.

2. We tracked TCP jitter buffer time/size requirement.

Furthermore, our results in Chapter 9 show that experiments conducted using high latency satellite connections are associated with high jitter, packet loss and poor estimated MOS. In particular, this is the case in many Pacific Island countries where GEO satellites are the common mode of Internet connectivity.

In order to improve connectivity in the Pacific, we discussed potential solutions for improving goodput for high latency satellite connection in Chapter 10. In particular we outlined the possibility for TCP over Network Coding (TCP/NC) to improve goodput on satellite connections.

In an attempt to use the TCP/NC software in our research, we commissioned a TCP/NC kernel module from network coding vendor Steinwurf and installed the software in nodes located in Tuvalu, Niue and the Cook Islands. Using our experience from the deployment of the software, we contributed to the following research question.

Research question: What can be done to improve the performance of networks with high latency and low bandwidth?

1. We demonstrated that TCP/NC tunnels can improve goodput on some satellite links into the Pacific.

2. We demonstrated that TCP/NC tunnels cannot improve goodput for everyone all the time (e.g. Niue).

3. We developed network topologies for the use of TCP/NC tunnels in experimental and commercial production deployments.

We also presented TCP/NC related results in Chapter 11. Our results with fixed overhead show that TCP/NC outperforms TCP during times of high packet loss.

However, TCP outperforms TCP/NC at times of low packet loss. In the original design of TCP/NC, the purpose of the degrees of freedom was for the receiver to provide feedback to the source so that the overhead from TCP/NC could be adjusted to be suitable for the current conditions on the network.

However, the use of degrees of freedom is not feasible here because the feedback from the receiver to the source would arrive too late to be useful. The required overhead is mostly a function of the time of day and since the conclusion of our experiments, Steinwurf have implemented a version of TCP/NC that feeds back the minimum number of excess combination packets arriving at the receiver across a number of generations, allowing the transmitter to adapt overhead to slowly changing conditions. This seems to allow both better performance in difficult conditions and equal performance to standard TCP during quieter times.

The author hopes that the results presented here will be useful for both ISPs and over the top service providers.

12.1 Open Problems

Since the version of the TCP/NC module that we used does not feed back the degrees of freedom, there is a non-negligible probability that our packets force the satellite gateway to drop other packets. At present, we therefore perform experiments in extremely low bandwidth environments mostly at times when the network is not overly busy.

Despite the effort to improve clock synchronisation for our nodes, we only managed to deploy a limited number of GPS time-corrected NTP servers. The majority of our nodes in the beacon network still uses publicly available NTP servers to synchronise their time. For our experiments, as long as there is timely arrival of timing information into our nodes, the data we record can be corrected with our clock drift compensation method (see Section 5.5.1). However this is not always guaranteed, and there is significant potential for inaccuracy in the timing information in our nodes.

12.2 Future Work

The insights from this thesis generate the following recommendations for future research.

At the time of writing, we had collected four years worth of beacon data. This is an ongoing process and our beacons continue to collect data. It would be interesting to revisit our data collection at a later time and compare our results to developments in our research partners' networks.

As described in the previous section, we deployed a limited number of GPS time corrected NTP servers in some of our nodes to counter the delays in clock synchronisation. In the future, we hope to deploy NTP servers in all of our nodes. Analysis of more comprehensive time-corrected data may generate greater insights into development of our software and beacon network.

As we collected data over an extended period of time, we often encountered the problem of limited disk space. Although attempts were made to minimise disk usage by storing our data in a centralised database, the rate at which our beacon application generates data is too high for SQL database inserts. One future project might involve the development of a database that involves both fast storage and packet level retrieval of data.

In Chapter 7, we mentioned that we do not consider the delays encountered during the encoding and decoding of voice packets. One could integrate this delay into the method we use to determine the total latency from the time voice packets are encoded and transmitted to the time they are received and decoded.

Our experiment uses a predefined inter-transmission time (default: 20 ms) to exchange simulated VoIP and file download packets. A future development for our `beacon` software would be to detect the condition of links and adjust the inter-transmission time accordingly.

Although our experiments showed that the use of TCP/NC improves goodput for high latency and low bandwidth networks, we did not load the software with

network traffic for the entire island. One could route all the network traffic for an entire island into the on-island encoder/decoder and observe if we are still achieving high goodput.

As indicated in the first section of this chapter, our network coding software supplier Steinwurf added an adaptive overhead as the congestion control algorithm for the TCP/NC software. However, future work is required to fine tune this algorithm and to determine the appropriate generation sizes for a particular island deployment scenario.

Finally, the current deployment of the TCP/NC kernel module requires complex skills and dedicated hardware to ensure that the software works. A future development for the TCP/NC software is to redesign it as a module that is easily enabled on satellite gateway/routers.

CHAPTER A

Appendix

This appendix provide details on the implementation of our beacon software. Readers interested in the functionality of our software may also refer to Chapter 4. This appendix is divided into four main sections. The first section discusses third party applications that we use to support the operation of our software. This is followed by a discussion on the operating system files that must be configured to support our software. Then we discuss configurable options of the beacon software and follow this with sample configurations of our experiments. Lastly we explain the structure of our log files.

A.1 Third Party Applications

In this section, we will use the following IP addresses to describe some of the tools that we use in this research project.

- Repository Server Address: 192.168.0.1/24
- Source IP Address: 10.0.0.1/24
- Destination IP Address: 10.0.0.2/24
- Gateway Address: 10.0.0.254/24

- DNS Address: 172.16.0.1/24

A.1.1 Secure Shell (ssh)

Secure shell (`ssh`) [203] is an encrypted network protocol for establishing secure communication with a remote machine. We use `ssh` to remotely configure experiments and to secure the transfer of data from our research nodes to the repository server at the University of Auckland. The configuration options for this utility can be found in [94].

A.1.2 Rsync

`rsync` is a utility found on Unix based operating systems for transferring files between machines. We use `rsync` together with `ssh` to export data from our research nodes to our repository server. The configuration options for this utility can be found in [204]. The following is an example of how we combine `rsync` and `ssh` to export data from our research nodes to the repository server.

```
rsync -azvv -e ssh /Users/Experiments/TCP/logs/ etuate@192.168.0.1:
/home/etuate/BeaconData/NZ1/TCP
```

A.1.3 SendEmail

`sendEmail` [205] is a lightweight command line SMTP email client. We use `sendEmail` to alert us on errors in our experiment. The alerts are sent into an email account that we check to identify and resolve errors. The configuration options for this utility can be found in [205].

A.1.4 Cron

`cron` is a time based job scheduler in Unix based operating systems [206]. We use `cron` to schedule experiments (jobs) on our nodes. The following are examples of such `cron` jobs. Note that we presented the different type of experiments in Chapter 4.

```
#Japan Beacon 2
#Running experiments between Japan Beacon 2 as transmitter and
#initiator and Japan Beacon 1 as receiver and responder.

01 2,10,18 * * * /opt/Experiments/UDP/beacon -n JP2 -o JP1 -m 0 -p
8088 -f /opt/Experiments/UDP/beacon.txt -a 10.0.0.1 -e 1 -t 10000

11 2,10,18 * * * /opt/Experiments/UDP/beacon -n JP2 -o JP1 -m 2 -p
8088 -f /opt/Experiments/UDP/beacon.txt -a 10.0.0.1 -e 2 -t 10000

21 2,10,18 * * * /opt/Experiments/TCP/beacon -n JP2 -o JP1 -m 4 -p
8088 -f /opt/Experiments/TCP/beacon.txt -a 10.0.0.1 -e 3 -r d -l 120

31 2,10,18 * * * /opt/Experiments/TCP/beacon -n JP2 -o JP1 -m 4 -p
8088 -f /opt/Experiments/TCP/beacon.txt -a 10.0.0.1 -e 4 -r v -l 120

41 2,10,18 * * * /opt/Experiments/TCP/beacon -n JP2 -o JP1 -m 6 -p
8088 -f /opt/Experiments/TCP/beacon.txt -a 10.0.0.1 -e 5 -r d -l 120

51 2,10,18 * * * /opt/Experiments/TCP/beacon -n JP2 -o JP1 -m 6 -p
8088 -f /opt/Experiments/TCP/beacon.txt -a 10.0.0.1 -e 6 -r v -l 120

#Japan Beacon 1
#Running experiments between Japan Beacon 1 as receiver and
#responder and Japan Beacon 2 as transmitter and initiator.
```

```
00 2,10,18 * * * /Users/Experiments/UDP/beacon -n JP1 -o JP2 -m 1
-p 8088 -f /Users/Experiments/UDP/beacon.txt -a 10.0.0.2 -e 1
-t 10000 -l 300

10 2,10,18 * * * /Users/Experiments/UDP/beacon -n JP1 -o JP2 -m 3
 -p 8088 -f /Users/Experiments/UDP/beacon.txt -a 10.0.0.2 -e 2
-t 10000 -l 300

20 2,10,18 * * * /Users/Experiments/TCP/beacon -n JP1 -o JP2 -m 5
-p 8088 -f /Users/Experiments/TCP/beacon.txt -a 10.0.0.2 -e 3 -l 120

30 2,10,18 * * * /Users/Experiments/TCP/beacon -n JP1 -o JP2 -m 5
-p 8088 -f /Users/Experiments/TCP/beacon.txt -a 10.0.0.2 -e 4 -l 120

40 2,10,18 * * * /Users/Experiments/TCP/beacon -n JP1 -o JP2 -m 7
-p 8088 -f /Users/Experiments/TCP/beacon.txt -a 10.0.0.2 -e 5 -l 120

50 2,10,18 * * * /Users/Experiments/TCP/beacon -n JP1 -o JP2 -m 7
-p 8088 -f /Users/Experiments/TCP/beacon.txt -a 10.0.0.2 -e 6 -l 120
```

A.1.5 Ufw

`ufw` is the default firewall for Ubuntu Linux distribution. We use this tool to manage the IPtables in our research nodes. For example, we allow only specific hosts (beacon pairs) to communicate using the TCP/UDP protocol with our software. The configuration options for this utility can be found in [207].

A.1.6 Ftd

`ftd` is a program that computes T-complexity, T-information, and T-entropy of

a string. This tool was developed by Speidel and Yang [208]. `ftd` is an enhanced version of `tcalc` which was developed by Titchener and Wackrow [209]. `ftd` operates in $O(n \log n)$ time. In this thesis, we use this tool to compute the entropy of strings from the mapping of inter-arrival times.

A.2 Operating System Configuration

In the course of this thesis, we configured the following files to support the operation of our software, starting with the network configuration of our research nodes, in

/etc/network/interfaces

The configuration of this file depends on the installation of our node. For example, if our node is deployed in a network where there is a DHCP server, we use the following configuration:

```
# The primary network interface
auto eth0
iface eth0 inet dhcp
```

When there is no DHCP server, we configure our node with static IP address information as follows.

```
# The primary network interface
auto eth0
iface eth0 inet static
address 10.0.0.1
network 10.0.0.0
netmask 255.255.255.0
```

We configure our nodes to use publicly available NTP servers for synchronisation of the local clock. As this functionality depends on the DNS lookup process to

determine the IP address of NTP servers, we use two methods to configure nodes. If there is a DNS server on the network, we configure the resolver file as follows:

/etc/resolv.conf

```
nameserver 172.16.0.1
search test.dns
```

When there is no DNS server, we install the `unbound` utility [91] and configure the resolver file with the entry `nameserver 127.0.0.1`. This instructs the node to use `unbound` as domain name resolver.

To ensure that our software uses the correct interface, we configure the hosts file as follows.

/etc/hosts

```
127.0.0.1 localhost
10.0.0.2 NZBeacon2
```

where `10.0.0.2` is the IP address of an Internet reachable interface.

A.3 Beacon Software Configuration

The beacon software can be configured to perform unidirectional or bidirectional experiments. The following lists the options that are configurable in our software.

```
Usage:

beacon [-h help] [-l listening time] [-f file] [-p port] [-m mode]
[-b name] [-n transmission]

Options:

-h: help
-H: static assignment of hostname/FQDN for IP resolution
-l: socket listening time (in seconds, default is 240~seconds)
-a: IP address of destination node (if initiating, default is
    127.0.0.1)
-A: public IP address  (if sitting behind a NAT router)
-b: total number of bytes in each transmission (in payload but not
    IP/UDP/TCP headers, default is 115~bytes)
-c: encryption setting for sendEmail to use when sending failure
    notifications, default is transport layer security (TLS)
-d: minimum time between transmissions (in milliseconds, default
    is 20~ms)
-e: experiment number (if initiating, no default)
-f: file from which to take padding bytes for transmissions,
    (default is beacon.txt)
-g: email and port number for sendemail to use when sending
    failure notifications, (default is smtp.gmail.com and 995)
-i: from address for sendEmail to use when sending failure
    notifications, (default is uoabeacon@gmail.com)
-j: to address for sendEmail to use when sending failure
    notifications, (default is ecoc005@aucklanduni.ac.nz)
-m: mode to operate this beacon process in, possible modes are:

0: beacon will immediately transmit UDP packets upon starting the
   beacon process. The experiment number must be specified, run
 number will be generated automatically based on the last
```

```
 successful run
1: beacon will listen and receive UDP packets. Experiment and run
   number will be extracted from the first packet received
2: beacon will transmit and receive UDP packets. It will act as
   initiator, i.e., it will transmit packets immediately.
 The partner beacon process should start first, in mode~3
3: beacon will transmit and receive UDP packets. It will act as
   responder, i.e., it will not transmit packets until one is
   received from the remote initiator beacon, which
 should be started after this process in mode 2
4: beacon will initiate a TCP connection and send data
5: beacon will accept a TCP connection and receive data
6: beacon will initiate a TCP connection and transmit and
   receive data
7: beacon will accept a TCP connection and transmit and
   receive data

-n: name of local beacon (default is LOC)
-o: name of partner beacon (default REM)
-p: port to use for listening and destination (default
    port is 5000)
-r: for TCP experiments: rate mode. Options are:

v: VoIP mode (supply constant medium data rate)
d: Download mode (supply maximum possible data rate)

-s: suppress e-mail notifications to beacon administrator.
-t: number of transmissions to be made. In UDP, this is the
    number of packets to be transmitted. In the case of TCP
this is the number of data chunks that are passed to
the TCP socket for transmission.
-u: user name for sendEmail to use when sending failure
    notifications, (default user name is uoabeacon@gmail.com)
```

```
-w: password for sendEmail to use when sending failure
    notifications, (default password is pre configured)
-N: NTP server address, (default is localhost)
```

The following are descriptions of the various modes for identifying a node as transmitter, initiator, receiver, or responder. Note that a node may act as a transmitter/receiver in a unidirectional experiment. In a bidirectional experiment, a node may initiate the transmission of packets (initiator) or respond to a transmission from another node (responder).

```
Experiment 1 mode 0, (UDP): Configured at the transmitter to
simulate a UDP unidirectional VoIP call.

Experiment 1 mode 1, (UDP): Configured at the receiver to
simulate a UDP unidirectional VoIP call.

Experiment 2 mode 2, (UDP): Configured at the initiator to
simulate a UDP bidirectional VoIP call. The initiator transmits and
receives packets concurrently.

Experiment 2 mode 3, (UDP): Simulate UDP bidirectional VoIP call.
Configured at the responder to receive the first packet from
the initiator. The responder awaits the arrival of the first packet
from the initiator and responds by transmitting and receiving
packets concurrently.

Experiment 3 mode 4, option r = d (TCP): Configured at the
transmitter to simulate a TCP unidirectional file download.

Experiment 3 mode 5, option r = d (TCP): Configured  at the
receiver to simulate a TCP unidirectional file download.

Experiment 4 mode 4, option r = v (TCP): Configured at the
```

```
transmitter to simulate a TCP unidirectional VoIP call.

Experiment 4 mode 5, option r = v (TCP): Configured at the
receiver to simulate a TCP unidirectional VoIP call.

Experiment 5 mode 6, option r = d (TCP): Configured at the
initiator to simulate a TCP bidirectional file download.
The initiator transmits and receives data concurrently.

Experiment 5 mode 7, option r = d (TCP): Simulate a TCP
bidirectional file download. Configured at the responder to
receive the first packet from the initiator. The responder
awaits the arrival of the first packet from the initiator and
responds by transmitting and receiving data concurrently.

Experiment 6 mode 6, option r = v (TCP): Configured at the
initiator to simulate a TCP bidirectional VoIP call.
The transmitter transmits and receives data concurrently.

Experiment 6 mode 7, option r = v (TCP): Simulate TCP
bidirectional VoIP call. Configured at the responder to receive
the first packet from the initiator. It then responds to the
initiator by transmitting and receiving data concurrently.
```

A.4 Log Files

The specific format of the log files is described in [80]. Readers interested in the structure of log files may refer to this document.

REFERENCES

[1] TeleGeography, "Submarine cable map." http://tinyurl.com/3cbkyma (last visited on December 28th, 2015).

[2] E. Ogg, "Internet van helped drive evolution of the Web." http://tinyurl.com/n9u7x2u (last visited on December 28th, 2015), November 2007.

[3] Central Intelligence Agency (CIA), "The world factbook." http://tinyurl.com/2q9qyq (last visited on December 28th, 2015), 2012.

[4] The World Bank, "Internet users (per 100 people)." http://tinyurl.com/2f2yeq4 (last visited on December 28th, 2014), 2013.

[5] Cisco, "Cisco IOS voice, video, and fax configuration guide." http://tinyurl.com/gs9qqbh (last visited on December 24th, 2015), 2001.

[6] M. R. Titchener, "Generalized T-Codes: an extended construction algorithm for self-synchronizing variable-length codes," in *IEE Proceedings on Computers and Digital Techniques 143(3)*, pp. 122–128, June 1996.

[7] Wikipedia, "Normal Distribution." http://tinyurl.com/m2gx6 (last visited on December 28th, 2015).

[8] E. Cocker, U. Speidel, N. Rebenich, S. Neville, A. Gulliver, R. Eimann, K. Nisar, S. Hassan, Z. Aziz, M.-C. Dong, and V. Wong, "Measurement of packet train arrival conditions in high latency networks," in *Information,*

Communications and Signal Processing (ICICS) 2013 9th International Conference on, (Singapore), pp. 1–5, Dec 2013.

[9] U. Speidel, E. Cocker, P. Vingelmann, J. Heide, and M. Médard, "Can network coding bridge the digital divide in the Pacific?," in *2015 International Symposium on Network Coding, NetCod 2015, June 22-24, 2015*, (Sydney, Australia), pp. 86–90, 2015.

[10] J. Kim and I. Yeom, "Reducing queue oscillation at a congested link," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, pp. 394–407, March 2008.

[11] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, (Tel Aviv, Israel), pp. 1157–1165 vol.3, Mar 2000.

[12] S. Suthaharan, "Reduction of queue oscillation in the next generation Internet routers," *Computer Communications*, vol. 30, no. 18, pp. 3881 – 3891, 2007. Optical Networking: Systems and Protocols.

[13] P. Calyam, D. Krymskiy, M. Sridharan, and P. Schopis, "Active and passive measurements on campus, regional and national network backbone paths," in *Proceedings of the 14th International Conference on Computer Communications and Networks (ICCN), October*, (San Diego, U.S.A), pp. 537 – 542, 2005.

[14] D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing Simple Network Management Protocol (SNMP) management frameworks." http://www.ietf.org/rfc/rfc3411.txt (last visited on December 17th, 2015), Dec. 2002. Updated by RFCs 5343, 5590.

[15] Y.-C. Chen and I.-K. Chan, "SNMP GetRows: An effective scheme for retrieving management information from MIB tables," *Int. J. Netw. Manag.*, vol. 17, pp. 51–67, Jan. 2007.

[16] G. Gardikis, G. X. K. Sarsembagieva, and A. Kourtis, "An SNMP agent for active in-network measurements," in *4th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), October*, (St. Petersburg, Russia), pp. 302 – 307, 2012.

[17] A. Cardigliano, L. Deri, J. Gasparakis, and F. Fusco, "vPFRING: Towards wire-speed network monitoring using virtual machines," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC'11, (New York, USA), pp. 533–548, ACM, 2011.

[18] L. Deri, "nProbe: An open source NetFlow. Probe for gigabit networks.." http://luca.ntop.org/nProbe.pdf (last visited on December 18th, 2015), May 2003.

[19] Cisco, "Introduction to Cisco IOS NetFlow - A technical overview." http://tinyurl.com/36ho6v (last visited on December 24th, 2015), May 2012.

[20] Cisco, "Catalyst switched port analyzer (SPAN) configuration example - Cisco Systems." http://tinyurl.com/q7kzedw (last visited on December 24th, 2015), July 2007.

[21] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the exchange of flow information." http://www.ietf.org/rfc/rfc7011.txt (last visited on December 17th, 2015), Sept. 2013.

[22] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)." http://www.ietf.org/rfc/rfc4656.txt (last visited on December 17th, 2015), Sept. 2006.

[23] K. Hedayat, R. Krzanowski, A. Morton, K. Yum, and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)." http://www.ietf.org/rfc/rfc5357.txt (last visited on December 17th, 2015), Oct. 2008. Updated by RFCs 5618, 5938, 6038.

[24] C. Demichelis and P. Chimento, "IP packet delay variation metric for IP Performance Metrics (IPPM)." http://www.ietf.org/rfc/rfc3393.txt (last visited on December 17th, 2015), Nov. 2002.

[25] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way delay metric for IPPM." http://www.ietf.org/rfc/rfc2679.txt (last visited on December 17th, 2015), Sept. 1999.

[26] G. Almes, S. Kalidindi, and M. Zekauskas, "A one-way packet loss metric for IPPM." http://www.ietf.org/rfc/rfc2680.txt (last visited on December 17th, 2015), Sept. 1999.

[27] S. Kalidindi and M. Zekauskas, "Surveyor: An infrastructure for Internet performance measurements." http://tinyurl.com/hxvywwk (last visited on December 28th, 2015), 1997.

[28] S. Leinen, "Original van jacobson/unix/lbl traceroute." http://tinyurl.com/ohujl5h (last visited on December 28th, 2015), 2006.

[29] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for IP Performance Metrics (IPPM)." http://www.ietf.org/rfc/rfc2330.txt (last visited on December 17th, 2015), May 1998.

[30] J. Mahdavi and V. Paxson, "IPPM metrics for measuring connectivity." http://www.ietf.org/rfc/rfc2678.txt (last visited on December 17th, 2015), Sept. 1999.

[31] G. Almes, S. Kalidindi, and M. Zekauskas, "A round-trip delay metric for IPPM." http://www.ietf.org/rfc/rfc2681.txt (last visited on December 17th, 2015), Sept. 1999.

[32] Regional Internet Registry for European IP Networks (RIPE NCC), "Test Traffic Measurement Service (TTMS)." http://tinyurl.com/zzt9bqm (last visited on December 28th, 2015), 2010.

[33] J. Postel, "Internet Control Message Protocol (ICMP)." http://www.ietf.org/rfc/rfc792.txt (last visited on December 17th, 2015), Sept. 1981. Updated by RFCs 950, 4884, 6633, 6918.

[34] C. Pignataro and F. Gont, "Formally deprecating some IPv4 options." http://www.ietf.org/rfc/rfc6814.txt (last visited on December 17th, 2015), Nov. 2012.

[35] Regional Internet Registry for European IP Networks (RIPE NCC), "Statistics -RIPE labs." https://labs.ripe.net/statistics (last visited on Oct 21th, 2014), 2010.

[36] Regional Internet Registry for European IP Networks (RIPE NCC), "What is RIPE Atlas?." https://atlas.ripe.net/about/ (last visited on Oct 21th, 2014), 2010.

[37] Y. Shavitt and E. Shir, "Dimes: Let the Internet measure itself," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 71–74, Oct. 2005.

[38] M. Allalouf, E. Kaplan, and Y. Shavitt, "On the feasibility of a large scale distributed testbed for measuring quality of path characteristics in the internet," in *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009. 5th International Conference on*, (Washington D.C., USA), pp. 1–6, April 2009.

[39] Center for Applied Internet Data Analysis (CAIDA), "Macroscopic topology measurements." http://tinyurl.com/pdmpdhz (last visited on December 28th, 2015), 1998.

[40] Center for Applied Internet Data Analysis (CAIDA), "Skitter." http://tinyurl.com/qbx4st7 (last visited on December 28th, 2015), 1998.

[41] Center for Applied Internet Data Analysis (CAIDA), "Scamper." http://tinyurl.com/on6odcx (last visited on December 28th, 2015), 2007.

[42] Center for Applied Internet Data Analysis (CAIDA), "Archipelago measurement infrastructure." http://tinyurl.com/yyk3cnr (last visited on December 28th, 2015), 1998.

[43] M. Luckie, "Scamper: a scalable and extensible packet prober for active measurement of the Internet," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC)*, (Melbourne, Australia), pp. 239–245, 2010.

[44] University of Oregon, "Route views project page." http://www.routeviews.org/ (last visited on Oct 26th, 2014), 2005.

[45] PlanetLab, "About PlanetLab." http://tinyurl.com/gl735f2 (last visited on December 28th, 2015), 2007.

[46] M. Beck, T. Moore, and J. S. Plank, "An End-to-End approach to globally scalable network storage," Tech. Rep. PDN–02–007, PlanetLab Consortium, November 2002.

[47] L. Wang, R. Pang, V. Pai, L. Peterson, and K. Park, "CoDEEN - A Content Distribution Network(CDN) on PlanetLab." http://codeen.cs.princeton.edu/ (last visited on Oct 21th, 2014), 2003.

[48] M. Freedman, D. Mazires, E. Freudenthal, and K. Shanahan, "The Coral Content Distribution Network (CDN)." http://www.coralcdn.org/ (last visited on Oct 22th, 2014), 2004.

[49] A. Nakao, L. Peterson, and A. Bavier, "A routing underlay for overlay networks," Tech. Rep. PDN–03–012, PlanetLab Consortium, April 2003.

[50] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz, "OverQoS: Offering Internet QoS using overlays," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 11–16, Jan. 2003.

[51] M. Balazinska, H. Balakrishnan, and D. Karger, "Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery," in *Pervasive Computing* (F. Mattern and M. Naghshineh, eds.), vol. 2414 of *Lecture Notes in Computer Science*, pp. 195–210, Springer Berlin Heidelberg, 2002.

[52] B. N. Chun, J. Lee, and H. Weatherspoon, "Netbait: A distributed worm detection service." http://tinyurl.com/h5oypzb (last visited on December 28th, 2015), 2003.

[53] N. Spring, D. Wetherall, and T. Anderson, "Scriptroute: A public Internet measurement facility," in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, (Berkeley, CA, USA), pp. 17–17, USENIX Association, 2003.

[54] N. Spring, D. Wetherall, and T. Anderson, "Scriptroute network measurement." http://tinyurl.com/gow9tnw (last visited on December 28th, 2015), 2003.

[55] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, (New York, USA), pp. 149–160, ACM, 2001.

[56] S. Rhea, "OpenDHT: A publicly accessible DHT service." http://tinyurl.com/yenc6h2 (last visited on December 28th, 2015), 2005.

[57] B. Chun, J. Hellerstein, R. Huebsch, S. Jeffery, and B. Loo, "Querying at Internet scale." http://tinyurl.com/h5dvyke (last visited on December 28th, 2015), 2004.

[58] K. Claffy, M. Crovella, T. Friedman, C. Shannon, and N. Spring, "Community-Oriented Network Measurement Infrastructure (CONMI) workshop report," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 36, pp. 41–48, Apr 2006.

[59] G. Malkin, "Traceroute using an IP option." http://www.ietf.org/rfc/rfc1393.txt (last visited on December 17th, 2015), Jan. 1993. Obsoleted by RFC 6814.

[60] J. Postel, "Internet Protocol." http://www.ietf.org/rfc/rfc791.txt (last visited on December 17th, 2015), Sept. 1981. Updated by RFCs 1349, 2474, 6864.

[61] R. Matthieu, "Pathchar." http://tinyurl.com/ydnsurw (last visited on December 28th, 2015), May 1997.

[62] A. B. Downey, "Using pathchar to estimate internet link characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 29, pp. 241–250, Aug. 1999.

[63] A. Downey, "Clink: A tool for estimating Internet link characteristics." http://tinyurl.com/ju7esv4 (last visited on December 28th, 2015).

[64] B. Mah, "pchar: A tool for measuring Internet path characteristics." http://www.kitchenlab.org/www/bmah/Software/pchar/ (last visited on January 21th, 2014), February 2005.

[65] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link, bandwidth," in *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, USITS'01, (Berkeley, CA, USA), pp. 11–11, USENIX Association, 2001.

[66] C. Dovrolis and R. Prasad, "Pathrate." http://tinyurl.com/gqozxe3 (last visited on December 28th, 2015), January 2004.

[67] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, (Anchorage, Alaska, USA), pp. 905–914 vol.2, 2001.

[68] O. Olvera-Irigoyen, A. Kortebi, L. Toutain, and D. Ros, "Available bandwidth probing in hybrid home networks," in *IEEE Workshop on Local and Metropolitan Area Networks (LANMAN), October*, (Chapel Hill, NC, USA), pp. 1–7, 2011.

[69] French forum for Iperf, "Iperf - The TCP/UDP bandwidth measurement tool." http://iperf.fr/ (last visited on July 05th, 2014).

[70] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications." http://www.ietf.org/rfc/rfc3550.txt (last visited on December 17th, 2015), July 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[71] C. Dovrolis and M. Jain, "Pathload: A measurement tool for the available bandwidth of network paths ." http://tinyurl.com/pbetaln (last visited on December 28th, 2015), March 2006.

[72] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions on Networking*, vol. 11, pp. 537 – 549, August 2003.

[73] S. Suthaharan and S. Kumar, "Measuring available bandwidth: pathChirp's chirp train structure remodeled ," in *Australasian Telecommunication Networks and Applications Conference (ANTAC 2008), December*, (Adelaide, Australia), pp. 379 – 384, 2008.

[74] K. Stangherlin, R. Filho, W. Lautenschlager, V. Guadagnin, L. Balbinot, R. Balbinot, and V. Roesler, "One-way delay measurement in wired and wireless mobile full-mesh networks," in *IEEE Wireless Communications and Networking Conference (WCNC), March*, (Cancun, Quintana Roo, Mexico), pp. 1044 – 1049, 2011.

[75] H. Veiga, T. Pinho, J. L. Oliveira, R. Valadas, P. Salvador, and A. Nogueira, "Active traffic monitoring for heterogeneous environments," in *Proceedings of the 4th International Conference on Networking - Volume Part I*, ICN'05, (Berlin, Heidelberg, Germany), pp. 603–610, Springer-Verlag, 2005.

[76] Cisco, "IP SLAs RTP-based VoIP operation." http://tinyurl.com/nno9rfc (last visited on September 13th, 2015), February 2006.

[77] K. Miller, "Calculating optical fiber latency." http://tinyurl.com/od63bcv (last visited on December 28th, 2015).

[78] ITU-T, "G.652 : Characteristics of a single-mode optical fibre and cable." http://tinyurl.com/jps8tk4 (last visited on December 28th, 2015), November 2009.

[79] K. Vos, S. Jensen, and K. Soerensen, "Silk speech codec - draft-vos-silk-00.txt." http://tools.ietf.org/html/draft-vos-silk-00 (last visited on July 12th, 2014), July 2009.

[80] U. Speidel and E. Cocker, "Hosting a beacon." http://tinyurl.com/poo5fau (last visited on December 28th, 2015), September 2013.

[81] P. T. Brady, "A statistical analysis of on-off patterns in 16 conversations," *The Bell System Technical Journal*, vol. 47, pp. 73–91, Jan 1968.

[82] ITU-T Recommendation, "G.114 : One-way transmission time." http://www.itu.int/rec/T-REC-G.114-200305-I (last visited on February 17th, 2015), May 2003.

[83] J. Postel, "User Datagram Protocol (UDP)." http://www.ietf.org/rfc/rfc768.txt (last visited on December 17th, 2015), Aug. 1980.

[84] E. Freire, A. Ziviani, and R. Salles, "Detecting Skype flows in web traffic," in *IEEE Network Operations and Management Symposium (NOMS), Salvador, April*, (Bahia, Brazil), pp. 89 – 96, 2008.

[85] S. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, (Barcelona, Spain), pp. 1–11, April 2006.

[86] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." http://www.ietf.org/rfc/rfc2616.txt (last visited on December 17th, 2015), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[87] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)." http://www.ietf.org/rfc/rfc959.txt (last visited on December 17th, 2015), Oct. 1985. Updated by RFCs 2228, 2640, 2773, 3659, 5797, 7151.

[88] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) transport layer protocol." http://www.ietf.org/rfc/rfc4253.txt (last visited on December 17th, 2015), Jan. 2006. Updated by RFC 6668.

[89] J. Klensin, "Simple Mail Transfer Protocol (SMTP)." http://www.ietf.org/rfc/rfc2821.txt (last visited on December 17th, 2015), Apr. 2001. Obsoleted by RFC 5321, updated by RFC 5336.

[90] B. Mitchell, "DMZ - Demilitarized Zone." http://tinyurl.com/8jgqu (last visited on December 28th, 2015).

[91] W. Wijngaards, "Unbound: howto setup and install." http://tinyurl.com/pygzwkl (last visited on December 28th, 2015), October 2008.

[92] ICANN Security and Stability Advisory Committee (SSAC), "SSAC advisory SAC008 DNS Distributed Denial of Service (DDoS) attacks." http://tinyurl.com/gtmvvw4 (last visited on December 28th, 2015), March 2006.

[93] K. Zahed, P. Rani, U. Saradhi, and A. Potluri, "Reducing storage requirements of snapshot backups based on rsync utility ," in *First International Conference on Communication Systems and Networks and Workshops (COMSNET), January*, (Bangalore, India), pp. 1–2, 2009.

[94] Berkeley Software Distribution (BSD), "Linux man page." http://linux.die.net/man/1/ssh (last visited on December 28th, 2015), April 2013.

[95] Cisco, "Understanding Jitter in packet voice networks (Cisco IOS platforms)." http://tinyurl.com/m6pcmt8 (last visited on December 24th, 2015), Feb 2006.

[96] J. Postel, "Internet official protocol standards." http://www.ietf.org/rfc/rfc1610.txt (last visited on December 17th, 2015), July 1994. Obsoleted by RFC 1720.

[97] J. Reynolds and J. Postel, "Assigned numbers." http://www.ietf.org/rfc/rfc1700.txt (last visited on December 17th, 2015), Oct. 1994. Obsoleted by RFC 3232.

[98] M. Larsen and F. Gont, "Recommendations for transport-protocol port randomization." http://www.ietf.org/rfc/rfc6056.txt (last visited on December 17th, 2015), Jan. 2011.

[99] C. Hedrick, "Routing Information Protocol (RIP)." http://www.ietf.org/rfc/rfc1058.txt (last visited on December 17th, 2015), June 1988. Updated by RFCs 1388, 1723.

[100] R. Ogier, F. Templin, and M. Lewis, "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)." http://www.ietf.org/rfc/rfc3684.txt (last visited on December 17th, 2015), Feb. 2004.

[101] Y. Rekhter and P. Gross, "Application of the Border Gateway Protocol (BGP) in the Internet." http://www.ietf.org/rfc/rfc1268.txt (last visited on December 17th, 2015), Oct. 1991. Obsoleted by RFC 1655.

[102] Cisco, "How does load balancing work?." http://tinyurl.com/kcg2n2f (last visited on December 24th, 2015), August 2005.

[103] Cisco, "Catalyst 3550 multilayer switch software configuration guide." http://tinyurl.com/oy2snup (last visited on December 24th, 2015), March 2003.

[104] D. Babic and J. Plombon, "Relationship between root-mean-square and peak-to-peak Jitter measurements," *Microwave and Optical Technology Letters*, vol. 39, no. 4, pp. 323–326, 2003.

[105] C.-C. Wu, K.-T. Chen, C.-Y. Huang, and C.-L. Lei, "An empirical evaluation of VoIP playout buffer dimensioning in Skype, Google Talk, and MSN Messenger," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '09, (New York, NY, USA), pp. 97–102, ACM, 2009.

[106] L. Ding and R. Goubran, "Assessment of effects of packet loss on speech quality in VoIP," in *Haptic, Audio and Visual Environments and Their Applications, 2003. HAVE 2003. Proceedings. The 2nd IEEE Internatioal Workshop on*, (Ottawa, Ontario, Canada), pp. 49–54, Sept 2003.

[107] S. Li, R. K. Neelisetti, C. Liu, and S. Kulkarni, "An interference-aware routing algorithm for multimedia streaming over wireless sensor networks," *Int'l Journal of Multimedia & Its Applications*, vol. 2, no. 1, pp. 10 – 30, 2010.

[108] P. Frossard and O. Verscheure, "Joint source/FEC rate selection for quality-optimal MPEG-2 video delivery," *Image Processing, IEEE Transactions on*, vol. 10, pp. 1815–1825, Dec 2001.

[109] J. Feng, C. Xuefen, P. Li, W. Yining, L. Guan, and X. Wang, "Adaptive FEC algorithm based on prediction of video quality and bandwidth utilization ratio," *Journal of Ambient Intelligence and Humanized Computing*, vol. 1, no. 4, pp. 309–318, 2010.

[110] U. M. Günther, *Robust Source Coding with Generalised T-Codes*. PhD thesis, Department of Computer Science, University of Auckland, 1998.

[111] R. E. A. Eimann, *Network event detection with entropy measures*. PhD thesis, Department of Computer Science, 2008.

[112] A. N. Kolmogorov, "Three approaches for defining the concept of information quantity," *Problems of Information Transmission*, vol. 1, pp. 3–11, 1965.

[113] A. Lempel and J. Ziv, "On the complexity of finite sequences," in *IEEE Transactions on Information Theory*, vol. IT-22, pp. 75–81, IEEE, January 1976.

[114] J. Yang, *Fast String Parsing and its Application in Information and Similarity Measurement*. PhD thesis, Department of Computer Science, University of Auckland, 2005.

[115] A. Lempel and J. Ziv, "A universal algorithm for sequential data compression," in *IEEE Transactions on Information Theory*, vol. IT-23, pp. 337–343, IEEE, May 1977.

[116] A. Lempel and J. Ziv, "Compression of individual sequences via variable-rate coding," in *IEEE Transactions on Information Theory*, vol. IT-24, pp. 530–536, IEEE, September 1978.

[117] M. R. Titchener, "A deterministic theory of Complexity, Information, and Entropy," in *Proceedings of IEEE Information Technology Workshop*, p. 80, February 1998.

[118] M. R. Titchener, "Deterministic computation of String Complexity, Information and Entropy," in *Proceedings of IEEE International Symposium on Information Theory*, p. 326, August 1998.

[119] M. R. Titchener, "A novel deterministic approach to evaluating the Entropy of language texts," in *Proceedings of 3rd Conference in Information-Theoretic Approaches to Logic, Languages, and Computation*, (Hsi-tou, Taiwan), June 1998.

[120] M. R. Titchener, "A measure of information," in *IEEE Data Compression Conference (DCC 2000)*, (Salt Lake City, USA), March 2000.

[121] Wikipedia, "Second law of thermodynamics." http://tinyurl.com/ykrm4e (last visited on December 28th, 2015), October 2002.

[122] Wikipedia, "Boltzmann's Entropy formula." http://tinyurl.com/nva7plk (last visited on December 28th, 2015), February 2007.

[123] D. Petz, "Entropy, von Neumann and the von Neumann Entropy." http://tinyurl.com/gvraf2w (last visited on December 28th, 2015), February 2001.

[124] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October 1948.

[125] C. E. Shannon, "Prediction and Entropy of printed english," *The Bell System Technical Journal*, vol. 30, pp. 50–64, 1951.

[126] E. W. Weisstein, "Logarithmic Integral." http://tinyurl.com/y4oxy6 (last visited on December 28th, 2015).

[127] R. Goleva, D. Atamian, S. Mirtchev, D. Dimitrova, and L. Grigorova, "Traffic shaping measurements and analyses in 3G network," in *Proceedings of the 2nd ACM Workshop on High Performance Mobile Opportunistic Systems*, HPMOSys '13, (New York, NY, USA), pp. 67–74, ACM, 2013.

[128] J. Matta and A. Takeshita, "End-to-end Voice over IP quality of service estimation through router queuing delay monitoring," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, (Taipei, Taiwan), pp. 2458–2462 vol.3, Nov 2002.

[129] K. Kim and Y.-J. Choi, "Performance comparison of various VoIP codecs in wireless environments," in *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '11, (New York, NY, USA), pp. 89:1–89:10, ACM, 2011.

[130] N. Unuth, "Mean Opinion Score (MOS) - A measure of voice quality." http://tinyurl.com/j2b3upb (last visited on December 28th, 2015), December 2013.

[131] ITU-T, "G.107 : The E-model: a computational model for use in transmission planning." http://www.itu.int/rec/T-REC-G.107-201402-P/en, Dec 1998.

[132] Nessoft, "How is MOS calculated in PingPlotter Pro." http://www.nessoft.com/kb/50, November 2005.

[133] D. Lee, B. Carpenter, and N. Brownlee, "Observations of UDP to TCP ratio and port numbers," in *Fifth International Conference on Internet Monitoring and Protection (ICIMP), May*, (Barcelona, Catalonia, Spain), pp. 99 – 104, 2010.

[134] E. Cocker, F. Ghazzi, U. Speidel, M.-C. Dong, V. Wong, A. Han Vinck, H. Yamamoto, H. Yokoo, H. Morita, H. Ferreira, A. Emleh, R. McFadzien, S. Palelei, and R. Eimann, "Measurement of buffer requirement trends for real time traffic over TCP," in *High Performance Switching and Routing (HPSR), 2014 IEEE 15th International Conference on*, (Vancouver, British Columbia, Canada), pp. 120–124, July 2014.

[135] ORA, "Economic and cheap NTP server." http://tinyurl.com/zvllj5z (last visited on December 28th, 2015).

[136] D. Taylor, "Building a Raspberry-Pi Stratum-1 NTP server." http://tinyurl.com/aqg8typ (last visited on December 28th, 2015), June 2014.

[137] PC Engines, "PC engines alix2d2 product file." http://www.pcengines.ch/alix2d2.htm (last visited on July 24th, 2014), 2012.

[138] K. White, "Definitions of managed objects for remote Ping, Traceroute, and Lookup Operations." http://www.ietf.org/rfc/rfc2925.txt (last visited on September 20th, 2013), September 2000.

[139] W. Davidson, "Rsync." http://rsync.samba.org/ (last visited on August 7th, 2013), September 2013.

[140] J. Valencia, "Using rsync and cron to automate incremental backups." http://tinyurl.com/oeynujv (last visited on December 28th, 2015), February 2011.

[141] Sevcik and Wetzel, "Out-of-order packets trash voice and video." http://tinyurl.com/pqpvthc (last visited on December 28th, 2015), March 2009.

[142] N. Davids, "Initial TTL values." http://noahdavids.org/self_published/TTL_values.html (last visited on September 9th, 2013), November 2011.

[143] H. Yang, K. Lee, and S. Ko, "Communication quality of voice over TCP used for firewall traversal," in *International Conference on Multimedia and Expo, April*, (Hannover, Germany), pp. 29 – 32, 2008.

[144] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol." http://www.ietf.org/rfc/rfc3261.txt (last visited on December 17th, 2015), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878.

[145] K. Kim, D. Niculescu, and S. Hong, "Coexistence of VoIP and TCP in wireless multihop networks," *Communications Magazine, IEEE*, vol. 47, pp. 75–81, June 2009.

[146] T. Bu, Y. Liu, and D. Towsley, "On the TCP-friendliness of VoIP traffic," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, (Barcelona, Catalunya, Spain), pp. 1–12, April 2006.

[147] F. Ghazzi, "TCP timing quality measurements for VoIP applications and services," Master's thesis, Department of Computer Science, January 2015.

[148] U.Speidel and E.Cocker, "International Internet Beacon Experiment (IIBEX) website." https://iibex.auckland.ac.nz/ (last visited on December 28th, 2015).

[149] H. Gavin, "Gnuplot 4.2 tutorial." http://people.duke.edu/~hpgavin/gnuplot.html (last visited on August 07th, 2014), Dec 2002.

[150] Tonga: Ministry of Information, "Tonga celebrates arrival of the fibre optic cable." http://tinyurl.com/ow7u2rb (last visited on December 28th, 2015), June 2013.

[151] G. Kim, "Exede and HughesNet launch US satellite VoIP services." http://tinyurl.com/onzllh9 (last visited on December 28th, 2015), June 2013.

[152] Cisco, "Quality of Service (QoS) - VoIP." http://tinyurl.com/nd4woju (last visited on December 24th, 2015), June 2001.

[153] H. Fletcher, "Telecom sells Cooks telco stake." http://www.nzherald.co.nz/business/news/article.cfm?c_id=3&objectid=11230840 (last visited on August 07th, 2014), April 2014.

[154] S. Collar, "Telecom Cook Islands - O3b." http://tinyurl.com/zapqbfs (last visited on December 28th, 2015), March 2014.

[155] Southern Cross Cable, "Southern Cross Cable Network website." http://www.southerncrosscables.com/ (last visited on December 24th, 2015).

[156] O3b Networks, "Why O3b." http://tinyurl.com/jtz7yk7 (last visited on December 28th, 2015).

[157] O3b Networks, "O3b maritime 2.2m orbit datasheet TRx 7-500." http://tinyurl.com/nqscr7a (last visited on December 24th, 2015).

[158] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies (PEP) intended to mitigate link-related degradations." http://www.ietf.org/rfc/rfc3135.txt (last visited on December 17th, 2015), June 2001.

[159] Cisco, "Cisco accelerated Internet over satellite solution." http://tinyurl.com/owvcskz (last visited on December 24th, 2015).

[160] R. Stewart, "Stream Control Transmission Protocol (SCTCP)." http://www.ietf.org/rfc/rfc4960.txt (last visited on December 17th, 2015), Sept. 2007. Updated by RFCs 6096, 6335, 7053.

[161] Memotec Inc, "17-TCP Acceleration Option." http://tinyurl.com/pmuh4u5 (last visited on December 28th, 2015), April 2011.

[162] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance." http://www.ietf.org/rfc/rfc1323.txt (last visited on December 17th, 2015), May 1992.

[163] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control." http://www.ietf.org/rfc/rfc2581.txt (last visited on December 17th, 2015), Apr. 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[164] R. Braden, D. Borman, and C. Partridge, "Computing the Internet checksum." http://www.ietf.org/rfc/rfc1071.txt (last visited on December 17th, 2015), Sept. 1988. Updated by RFC 1141.

[165] C. Younghusband, P. Slade, and J. Weaver, "PEP Deployment and Bandwidth Management Issues," in *Personal Satellite Services*, vol. 15 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 77–84, Springer Berlin Heidelberg, 2009.

[166] Encore Networks, "High performance VPN solutions." http://tinyurl.com/pr9sgxs (last visited on December 28th, 2015).

[167] D. Lacamera, "Pepsal." http://tinyurl.com/zg2p55u (last visited on December 24th, 2015), June 2005.

[168] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, July 2008.

[169] C. Caini and R. Firrincieli, "TCP Hybla: A TCP enhancement for heterogeneous networks," *International Journal of Satellite Communications and Networking*, vol. 22, pp. 547–566, 2004.

[170] M. Park, M. Shin, D. Oh, B. Kim, and J. Lee, "TCP Hybla: Making TCP more robust against packet loss in satellite networks," in *Computational Science and its Applications - ICCSA 2011* (B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, and B. Apduhan, eds.), vol. 6785 of *Lecture Notes in Computer Science*, pp. 424–435, Springer Berlin Heidelberg, 2011.

[171] S. Trivedi, S. Jaiswal, R. Kumar, and S. Rao, "Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under

low error conditions," in *Internet Multimedia Services Architecture and Application(IMSAA), 2010 IEEE 4th International Conference on*, (Bangalore, India), pp. 1–5, Dec 2010.

[172] D. Leith, "H-TCP: TCP congestion control for high bandwidth-delay product paths." http://tinyurl.com/z85ex5j (last visited on December 28th, 2015), April 2008.

[173] B. Ganguly, B. Holzbauer, K. Kar, and K. Battle, "Loss-Tolerant TCP (LT-TCP): Implementation and experimental evaluation," in *Military Communications Conference, 2012 - MILCOM 2012*, (Orlando, Florida, USA), pp. 1–6, Oct 2012.

[174] M. Riley and I. Richardson, "Reed-solomon codes." http://tinyurl.com/d8wqepn (last visited on December 28th, 2015), 1998.

[175] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Network coded TCP (CTCP)," *CoRR*, vol. abs/1212.2291, 2012.

[176] T. Hanada, K. Fujisaki, and M. Tateiba, "Average bit error rate for satellite downlink communications in Ka-band under atmospheric turbulence given by Gaussian model," in *Microwave Conference, 2009. APMC 2009. Asia Pacific*, (Singapore), pp. 1092–1095, Dec 2009.

[177] C. Caini, R. Firrincieli, M. Marchese, T. d. Cola, M. Luglio, C. Roseti, N. Celandroni, and F. Potort, "Transport layer protocols and architectures for satellite networks," *International Journal of Satellite Communications and Networking*, vol. 25, no. 1, pp. 1–26, 2007.

[178] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, pp. 1204–1216, Jul 2000.

[179] R. Koetter and M. Medard, "An algebraic approach to network coding," *Networking, IEEE/ACM Transactions on*, vol. 11, pp. 782–795, Oct 2003.

[180] T. Ho, *Networking from a network coding perspective*. PhD thesis, Massachusetts Institute of Technology, Dept. of EECS, May 2004.

[181] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Medard, "Practical Network Coding for Wireless Environments," in *Proc. Forty-Third Annual Allerton on Communication, Control, and Computing*, (Allerton, IL), September 2005.

[182] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, "XORs in the Air: Practical Wireless Network Coding," *Networking, IEEE/ACM Transactions on*, vol. 16, pp. 497–510, June 2008.

[183] J. K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation," *Proceedings of the IEEE*, vol. 99, pp. 490–512, March 2011.

[184] J. K. Sundararajan, D. Shah, and M. Medard, "ARQ for network coding," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, (Toronto, Canada), pp. 1651–1655, July 2008.

[185] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, SIGCOMM '94, (New York, USA), pp. 24–35, ACM, 1994.

[186] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm." RFC 6582 (Proposed Standard), Apr. 2012.

[187] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno Performance: A simple model and its empirical validation," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 133–145, Apr. 2000.

[188] J. Cloud, D. Leith, and M. Medard, "Network Coded TCP (CTCP) Performance over Satellite Networks," in *SPACOMM 2014 (ISBN: 978-1-61208-317-9)*, (Nice, France), pp. 53–56, Oct. 2013.

[189] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks." http://www.hamilton.ie/net/htcp3.pdf (last visited on December 18th, 2015), 2004.

[190] Steinwurf, "About Steinwurf." http://steinwurf.com/about/ (last visited on December 18th, 2014), 2011.

[191] G. Delannoy, "Performance-Enhancing Proxy (PEP) for TCP over lossy links." http://tinyurl.com/n9rq7se (last visited on December 28th, 2015), May 2013.

[192] R. Russell, "Man page for IPtables." http://ipset.netfilter.org/iptables.man.html (last visited on November 09th, 2015), June 2015.

[193] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)." http://www.ietf.org/rfc/rfc4271.txt (last visited on December 17th, 2015), Jan. 2006. Updated by RFCs 6286, 6608, 6793.

[194] Cisco, "Load sharing with BGP in single and multihomed environments: sample configurations." http://tinyurl.com/ky4ltal (last visited on December 24th, 2015), August 2005.

[195] Cisco, "Sample configuration for BGP with two different service providers (Multihoming)." http://tinyurl.com/q8sxksw (last visited on December 24th, 2015), Aug 2005.

[196] Asia Pacific Network Information Center (APNIC), "What is an Autonomous system (AS)?." http://tinyurl.com/o6holpg (last visited on December 24th, 2015).

[197] PfSense, "Pfsense." https://www.pfsense.org/ (last visited on January 03th, 2015).

[198] Silverpeakworks, "Appliance manager operators guide, VXOA 6.2." http://tinyurl.com/jbmcxhk (last visited on December 28th, 2015).

[199] Cisco, "Firewalls - Cisco." http://tinyurl.com/j3br6cn (last visited on December 28th, 2015).

[200] Free Electrons, "Linux cross reference." http://tinyurl.com/hwwwdkj (last visited on December 28th, 2015).

[201] S. Floyd, "HighSpeed TCP for large congestion windows." http://www.ietf.org/rfc/rfc3649.txt (last visited on December 17th, 2015), Dec. 2003.

[202] Silverpeakworks, "Silver peak NX-3700." http://tinyurl.com/obngw2p (last visited on December 28th, 2015).

[203] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture." http://www.ietf.org/rfc/rfc4251.txt (last visited on December 17th, 2015), Jan. 2006.

[204] S. Sheppard, "Rsync man page." http://ss64.com/bash/rsync.html (last visited on June 23th, 2015), June 1999.

[205] B. Zehm, "An email program for sending SMTP mail from a command line." http://caspian.dotconf.net/menu/Software/SendEmail/ (last visited on November 21th, 2015).

[206] Wikipedia, "Cron." http://tinyurl.com/64xqopa (last visited on December 28th, 2015), February 2004.

[207] B. Visel, "UFW - Community Ubuntu Documentation." https://help.ubuntu.com/community/UFW (last visited on August 14th, 2013), April 2013.

[208] Y. Jia and U. M. Speidel, "An Improved T-Decomposition Algorithm," in *4th International Conference on Information, Communications and Signal Processing (December 2003)*, (Singapore).

[209] M. R. Titchener and S. Wackrow, "T-Code Software Documentation (Tamaki T-code project series)," tech. rep., Department of Computer Science, University of Auckland, 1995.

Index