Topologies and Coding Considerations for the Provision of Network-Coded Services via Shared Satellite Channels

Ulrich Speidel, Lei Qian
The University of Auckland
Auckland, New Zealand
ulrich@cs.auckland.ac.nz
lqia012@aucklanduni.ac.nz

'Etuate Cocker
Spark Digital NZ Ltd.
Auckland, New Zealand
Etuate.Cocker@spark.co.nz

Muriel Médard
RLE, MIT
Cambridge, MA, USA
medard@mit.edu

Péter Vingelmann, Janus Heide
Steinwurf ApS
Aalborg, Denmark
{peter|janus}@steinwurf.com

***Abstract*—Network traffic across shared bottleneck satellite channels using the Transmission Control Protocol (TCP) can suffer significant impairment due to TCP queue oscillation. In TCP queue oscillation, the input queue to the satellite uplink alternates between overflow and packet loss and subsequent exponential back-off. During back-off, the queue can drain completely and leave the link capacity idle and underused. Coding of such network traffic across multiple Internet Protocol (IP) packets allows packet loss to be masked from the senders to a certain degree. This lets TCP senders maintain larger congestion windows for longer, resulting in higher goodput rates. We argue that the concept of tunneling coded traffic across a satellite link is a flexible one and does not necessarily rely on a one-size-fits-all solution. This paper discusses a number of network topologies for the deployment of coding, from the perspective of satellite providers, Internet service providers (ISPs), end users and third-party entities, and looks at considerations surrounding code design, timing, and experiment methodology.**

Keywords–TCP; network coding; satellite Internet; queue oscillation

I. INTRODUCTION

The present paper is an extended version of [1], which investigates possible deployment scenarios for network-coded tunnel solutions to bridge the bottleneck given by the following scenario: An Internet Service Provider (ISP) on a small Pacific island receives its international connectivity via a geostationary (GEO) or medium earth orbit (MEO) satellite service. The capacity provisioned is in the range of several Mbps to several hundred Mbps, but always well below that of the networks connected at either end (assumed to be 1 Gbps or faster). The ISP services users on the island. The number of concurrently active client devices could be anywhere from a few dozen to a couple of thousand, and the ISP might observe up to a few thousand simultaneous TCP [2] flows. For the purposes of this paper, a TCP flow is a set of TCP packets travelling in one direction and is characterised by a unique combination of source and destination IP addresses and ports. Each flow belongs to a single TCP connection (i.e., a connection typically consists of two flows in opposite directions).

The flows across the link will typically be a heavily skewed mix: Most flows on the link will contain at most a few hundred bytes and will be too small and short to have their rate controlled by TCP flow control (also known as congestion control). Long flows, which are subject to flow control, contribute the majority of bytes on the link, however.

Satellite links of this type present a significant challenge to TCP: The long latency bottleneck makes it difficult for the TCP senders of sufficiently long flows to determine the correct congestion window [3], [4], [5], [6]. The root cause of this effect is the ACK-based feedback mechanism in TCP: A TCP sender interprets arriving / lost ACK packets as absence / evidence of congestion. However, in satellites, this feedback typically arrives with delays of over 500 ms on GEO and over 120 ms on current MEO links (at the time of writing, only a single MEO vendor existed, O3b, now part of SES [7], with orbital altitudes of around 8,000 km).

This delay gives each sender a rather outdated view of the current situation at the entrance to the bottleneck – the input queue at the satellite gateway for the uplink. In consequence, a sender may be encouraged by returning ACKs to increase its congestion window even though the input queue has since filled and is overflowing, and will now drop most additional packet arrivals from the sender. Similarly, a sender may be waiting for ACKs corresponding to data packets that were lost to a queue overflow event that has since resolved, and may needlessly reduce its congestion window. As the reduction in congestion window is an exponential back-off, multiple missing ACKs can quite radically reduce the goodput a sender is able to deliver.

Moreover, in our scenario, the link in question is *shared*, which means that a large number of simultaneous connections face exactly the same congestion situation and their packet round-trip-times (RTT) are dominated by the same latency. This causes all participating TCP senders to act more or less in unison when adjusting their congestion windows. Note that this is quite different from the situation at a "typical" Internet router, which sees connections with a wide mix of RTT values and senders that adjust their windows in either direction at any one time.

This effect on satellite links is known as *global synchronisation* [8], [9], [10] and can lead to *TCP queue oscillation*, where the input queue to the satellite link oscillates frequently between empty and overflow, causing link underutilisation when the queue is empty. Fig. 1 shows the effect of TCP

queue oscillation on the queue sojourn time of `ping` [11] packets sent at 100 ms intervals into a 120 kB input byte queue of a simulated 16 Mbps GEO link. The queue sojourn time is a measure of queue length and reflects the fast filling and draining of the queue. A well-dimensioned queue drains completely at regular intervals but does not remain empty for extended period of time. Similarly, the queue should not overflow for extended periods of time. The 120 kB queue deployed here meets this requirement reasonably well.

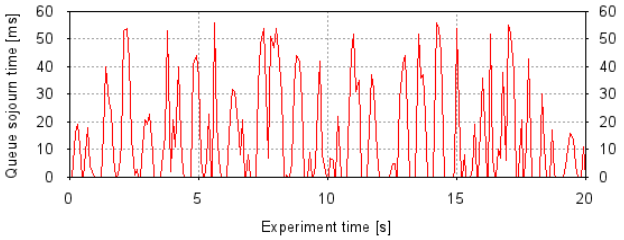

Figure 1. Queue sojourn time experienced by ICMP ping packets transmitted in 100 ms intervals across a simulated 16 Mbps GEO link experiencing TCP queue oscillation. Note the rapid rise and fall in queue sojourn time at regular intervals, which corresponds to the filling and draining of the link's input queue.

In addition to the work presented in [1], the present paper considers how tunnel solutions are constrained from a coding perspective, and how they may be evaluated experimentally. Section II provides a brief overview of related work. Section III explains the basic workings of the tunnel in a scenario where the ISP on the island provides the tunnel endpoint and where the coded traffic between the tunnel endpoints travels in the payload of UDP packets. On this basis, Section IV discusses the question as to where the off-island endpoint could be located and presents a case for having multiple endpoints. Section V describes a scenario in which a third-party entity operates the tunnel endpoint on the island. In Section VI, we look at the advantages and drawbacks of offering coding-as-a-service tunnels to individual end users on an island. Section VII then looks at various options for non-UDP communication between the tunnel endpoints. In extension of [1], Section VIII uses observations from real and simulated satellite links to explore criteria that the code design must meet. Section IX then looks at experimental approaches and introduces the Auckland Satellite Simulator facility, followed by our conclusion.

II. Related Work

The performance problem resulting from TCP queue oscillation has been studied in the context of satellite links for over two decades (see, e.g., [12], [13]) and remains essentially unsolved, despite the emergence of active queue management (AQM) techniques and improvements in the TCP congestion control algorithm itself [14], [15], [16].

In large parts, this is due to the fact that in our scenario, the senders overload the queue based on feedback from the receivers that is already enroute by the time that the queue shows signs of filling. Any feedback from explicit congestion notification (and even more so from random early drops) simply arrives too late to be useful. It is worth noting in this context that island ISPs do not normally control the TCP version used by the hosts on the island, and have no control whatsoever over versions used by senders on the rest of the Internet.

Network coding [17] offers a potential part-remedy here: By error-correcting packet losses that occur at the input queue, it is possible to prevent premature back-off by the TCP senders, allowing some of the lost capacity to be reclaimed. In order to do so, the packets that one wishes to protect by error correction coding must be encoded *before* the input queue: The conventional forward error correction that is a standard feature in many satellite terminals happens at the time of transmission to the satellite and can only code packets that have already made it into the queue – its function is to protect against errors from noise and fading. TCP performance under network coding has been investigated by other authors before [18], [19], but not in the context of satellite links. Similarly, network coding has been studied experimentally on satellite links, but not in the context of TCP [20].

The basic topology investigated in this paper is a tunnel [21], which operates across the link and both satellite input queues at either end. It accepts and delivers IP packets regardless of transport layer protocol involved, such that the end-to-end principle always remains intact. We have already demonstrated [21], [22] that such a tunnel solution can improve goodput for individual TCP connections, even in the presence of a majority of legacy TCP traffic on the same link.

III. Coded Tunnels

We begin our tour of the basic tunnel model (Fig. 2) by introducing our players and our components and following what happens during a TCP connection from an island end user client to a server off-island that the user wants to access:

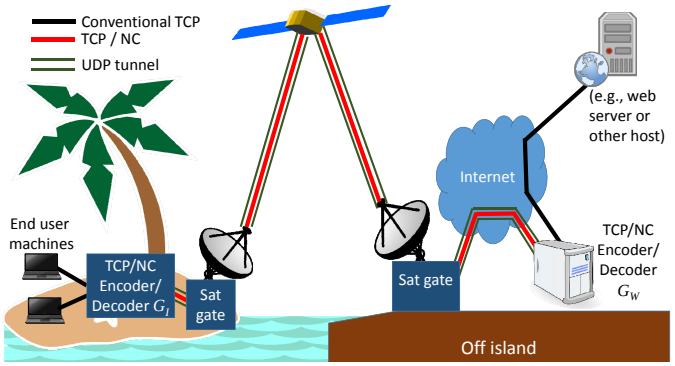

Figure 2. TCP/NC network topology in a scenario where the on-island ISP operates the local encoder/decoder. The off-island encoder/decoder may be at an arbitrary off-island location on the Internet.

The connection begins at an end user machine on the island. In most scenarios, we will assume that we have no control over this machine, i.e., that we cannot assume anything beyond the existence of a TCP/IP stack and some TCP client program on the machine. It means in particular that we cannot install software on the machine and that we cannot get the user to change settings on the machine. It is this machine that initiates the connection by sending a TCP packet to the off-island server, with the SYN flag set. The TCP/IP stack encapsulates this packet inside an IP packet whose source

address is that of the client machine. Its destination address is that of the off-island server.

On its way to the world, IP initially forwards the packet to a local gateway router on the island, and from there possibly along further gateway routers in the direction of the on-island satellite gateway. In our case, we replace one of these gateway routers by our on-island encoder/decoder G_I. Since our packet is heading off-island, we use its encoder functionality here.

The encoding works as follows: G_I captures the original IP packet and prevents it from travelling further. Instead, G_I forms sets of n successive IP packets it has captured. Each such set is called a *generation* and n is the *generation size*. In random linear network coding (RLNC), G_I now creates $n + \omega$ byte-wise linear combinations of all packets $p_1, p_2, \ldots, p_n$ in the generation, using randomly chosen coefficients $c_{i1}, c_{i2}, \ldots, c_{in}$. That is, the i'th linear combination r_i that G_I produces is given by:

$$c_{i1}p_1 + c_{i2}p_2 + \ldots + c_{in}p_n = r_i. \quad (1)$$

The $n + \omega$ combinations thus produced form an overdetermined system of linear equations whose solution is the set of original packets $p_1, p_2, \ldots, p_n$. In doing so, G_I codes all bytes of the incoming packets, including the headers with the IP addresses of the island and off-island end hosts involved. Note that in pure RLNC, the length of r_i is that of the largest of the packets $p_1, p_2, \ldots, p_n$.

G_I now communicates this system, one equation at a time, to the decoder G_W, located somewhere on the Internet on the off-island side of the satellite link. For this purpose, G_I sends $n + \omega$ UDP packets to G_I, with its own IP address as source address and the IP address of G_W as destination address. Each UDP packet contains the equation for a particular i in the form of the $c_{i1}, c_{i2}, \ldots, c_{in}$ and r_i. In pure RLNC, this makes each UDP packet a little larger than the largest of the original packets – one of the reasons to use systematic coding, which we will discuss in Section VIII-E.

These UDP packets now travel via the satellite link and the off-island Internet to G_W, which solves the system of linear equations. The solution consists of the original packets $p_1, p_2, \ldots, p_n$, of course, which G_W then forwards to their off-island destinations. Note that G_W generally only needs any n of the $n+\omega$ UDP packets in order to decode the $p_1, p_2, \ldots, p_n$. The remaining ω UDP packets are not required and can safely be dropped along the way – for example, at the input queue to the satellite gateway. The important point here is that we can leave it up to the input queue to decide which packets to drop.

Our SYN packet has now arrived at its off-island server destination, and the server wishes to send a SYN+ACK in response. At this point, the network topology becomes critical: In most island scenarios, all island hosts including the satellite gateways at either end of the link belong to a single IP subnet. From the world's perspective, the off-island satellite gateway is also the IP gateway to this subnet. In this scenario, the SYN+ACK response from the server (and any subsequent packets from the server) are routed straight to the off-island satellite gateway, entirely bypassing G_W. This is unacceptable, of course, since most data flows in the direction to the island and it is important that we encode this direction in particular.

The solution is to split the subnet: End hosts in the islands become part of a new subnet A (this could also be several subnets), whereas the on-island satellite gateway is placed in a disjoint subnet B. One then configures routing such that traffic to A is routed to G_W as gateway, whereas traffic to B is routed to the off-island satellite gateway.

In this scenario, the off-island server receives the SYN packet with a source address from network A, and thus responds by forwarding the SYN+ACK to G_W. There, G_W encodes the packet in the same way G_I encodes packets in the opposite direction. It then forwards the coded packets inside UDP to G_I for decoding and release to the island end user machine, which completes the round-trip handshake. Further packets between the hosts follow the same path. That is, the packets travel trough a coded UDP *tunnel* between G_I and G_W and vice versa.

This scenario requires the ISP on the island to either operate G_W off the island, or contract an off-island entity to operate G_W on their behalf. In many cases, it will also be desirable to make at least network A an autonomous system (AS) for routing purposes, in which case G_W needs to be duplicated for redundancy. The current experimental software that we have been working with is capable of supporting two instances of G_W.

In the next sections, we will consider variations of this base scenario.

IV. Tunnel Endpoints and Their Locations

Our basic tunnel scenario above assumes that G_W is located at an arbitrary location on the Internet. As long as the tunnel that it spans with G_I covers the satellite link, it can fulfill its purpose of masking packet loss at the satellite gateway input queues. There are however good reasons to consider the placement of G_W carefully. The following options may deserve consideration:

- G_W could be placed in the path between the Internet at the off-island satellite gateway (Fig. 3), or even be a part of the satellite gateway hardware itself. In this case, network B could use private IP addresses, and G_W simply acts as a gateway for network A as in the previous scenario. That is, all machines on the island could be in the same subnet of an upstream provider's network, save the off-island facing interface of the on-island satellite gateway. For the ISP and/or their upstream provider, this removes the cost of maintaining a separate block of public IP addresses or even a separate AS.
  However, a placement at the satellite gateway requires the competent cooperation of whichever party controls the off-island satellite gateway: They need to install and assist in commissioning G_W or permit VSAT terminals with equivalent built-in functionality to be installed. In practice, one encounters a variety of scenarios, however: One ISP owns and controls both satellite gateways, another ISP owns and operates the island side only and contracts to a satellite provider

and upstream ISP off-island, and yet another buys a turnkey solution from a satellite provider who also controls and services the on-island satellite gateway. We note in this context that especially in the latter case, satellite providers often provide WAN accelerators with network memory, parity packets and various other optimisation functions – inserting G_W as part of such a solution would thus not be without precedent.

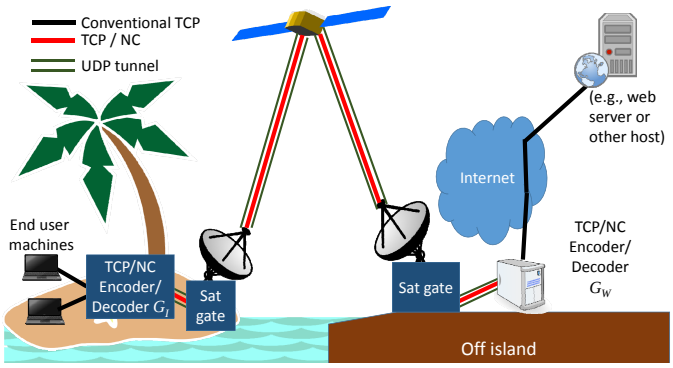

Figure 3. TCP/NC network topology in a scenario where the on-island ISP operates the local encoder/decoder, and the off-island encoder/decoder is inserted in the path between off-island satellite gateway and the Internet.

- G_W (or several instances thereof, labelled G_{W1}, G_{W2}, etc.) could be placed close to the known primary sources of bulk data content sought by island clients (Fig. 4). The advantage of such a placement would be that it would protect a longer portion of the paths between servers and clients by coding and bridge other potential sources of loss. However, there is an obvious drawback: G_W is no ordinary server – it needs a significant amount of network configuration in its environment to work. For example, the network that G_W is placed in has to advertise a route to network A in order to draw traffic for subnet A from the bulk data content sources. Hence, placing G_W in such a site potentially far away from both ISP and satellite provider premises requires the cooperation of a third party that is not only able to host G_W, but also able to arrange for its routing needs. Such partners could potentially be difficult to recruit for an ISP based on a remote island.
- G_W could be placed at the premises of a specialised off-island provider, who could also own and operate the device and sell its encoding/decoding services to the ISP on the island. An obvious advantage of this model is that it allows a provider to specialise in this type of service and host the G_W for multiple island installations, achieving some economies of scale. A potential disadvantage is added latency: The latency between off-island satellite gateway, G_W and off-island data sources may be much higher than that between data sources and satellite gateway alone. N.B.: This problem can be exacerbated by a failure to peer near the off-island satellite gateway. The authors are aware of a Pacific Island ISP whose off-island gateway is located in Hawaii. While the island has close cultural and economic links to New Zealand, lack of peering in Hawaii at the time of writing meant that all traffic between the island and New Zealand also has to travel between Hawaii and the U.S. mainland and back.

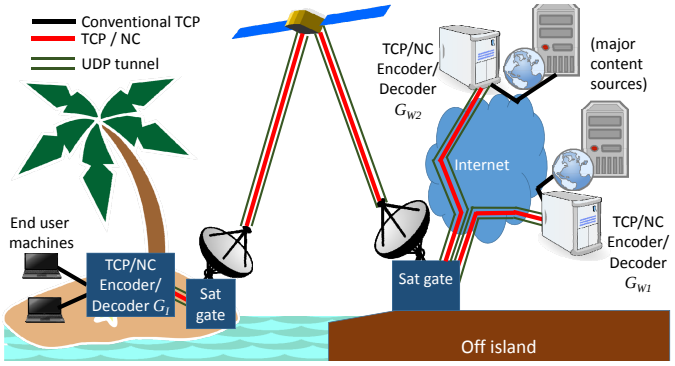

Figure 4. TCP/NC network topology in a scenario where the on-island ISP operates the local encoder/decoder, and off-island encoders/decoders are placed close to the servers on the Internet from which most of the download content originates.

V. Tunnels not Involving ISPs

In all our scenarios so far, the island ISP has played a core role as the operator of G_I, if not G_W. However, in principle there is no reason why G_I cannot be operated by another party on the island. Assuming for the moment that the ISP and satellite provider will pass UDP in both directions, any of the ISP customers within the island network can operate a G_I to tunnel to some G_W located off-island. This customer can then spawn their own network (Fig. 5) or – in the case of individual rather than institutional customers – simply run G_I on their own host or local NAT box.

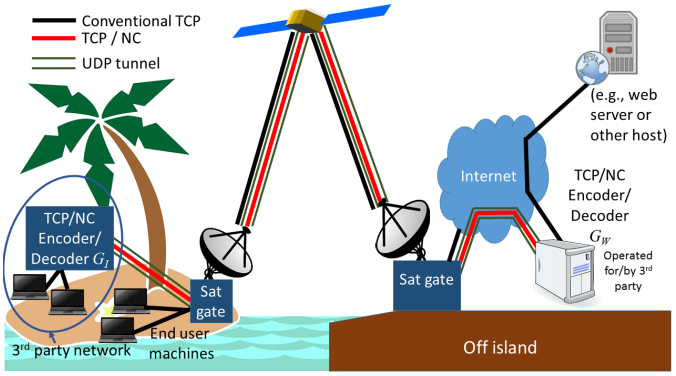

Figure 5. TCP/NC network topology in a scenario where an on-island encoder/decoder and the off-island encoder/decoder on the Internet are operated by third parties.

In the case of institutional customers, the corresponding G_W could be located at an organisation's off-island data centre or at the premises of a specialised third party off-island provider as discussed in the previous section. In the case of individuals, there could also be the option of G_W being provided on a subscription or pay-as-you-go basis by an off-island entity – an option that the next section explores.

Any such arrangement has a number of drawbacks, however. Firstly, it almost inevitably means that only some of the traffic on the link will be coded traffic, with the remainder being (mostly) conventional TCP. This residual uncoded traffic may still cause queue oscillation. While the coded traffic would be – at least to an extent – be protected from the associated packet loss and slow-down, the coding scheme involved would nevertheless have to provision sufficient overhead in order to cope with the potentially lengthy burst errors that queue oscillation causes. This would further increase to the load on the link.

Secondly, any overhead transmitted or received by a G_I under customer control increases that customer's data usage. In cases where the ISP on the island applies volume charges (a very common scenario in the Pacific), this results in additional cost for the customer. This may however be outweighed by data volume savings at the application layer as customers have to repeat fewer unsuccessful downloads.

VI. Coding-as-a-Service Tunnels

Another possible scenario is to absorb G_I into a virtual network interface on the end user machine and provision G_W off-island on a subscription or pay-per-coded-volume basis (Fig. 6). In this case, the end user would download an application which implements the client-side solution with G_I and interfaces with G_W off-island. The end user machine would then use two IP addresses: that assigned by the ISP, which appears in the header of the UDP packets between the machine and G_W, and an IP address assigned by the off-island provider of G_W, which belongs to the off-island provider's network and is not visible on the island to any host except G_I (which of course operates on the machine itself). This address is the source of IP packets departing G_W in the direction of off-island servers on behalf of the end user machine, and the destination of any packets that these servers send in response. In this respect, the service operates in a very similar fashion to a tunnelled VPN connection, except that the traffic across the tunnel is encoded rather than encrypted (it may of course also be encrypted in addition to the encoding).

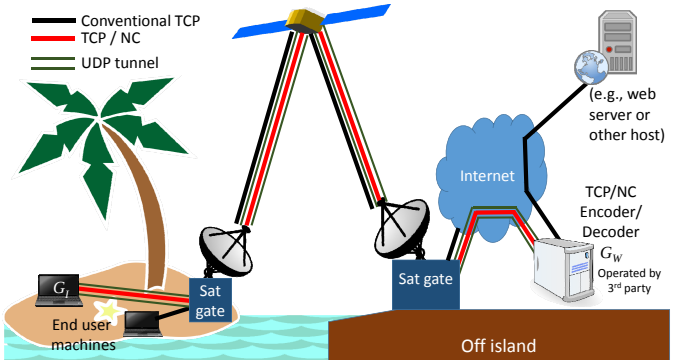

Figure 6. TCP/NC network topology in a scenario where an on-island encoder/decoder is integrated into an end user machine on the island, and the off-island encoder/decoder is provided by a third party on the Internet as a service, e.g., for a fee.

An obvious advantage of this approach is that there is no need for dedicated on-island infrastructure, the ISP does not have to expend or upskill personnel resources (or even be aware of the tunnel operation), and there is no need for equipment or personnel to be sent to the island to install or support the system. These are significant factors as many Pacific islands with satellite connection are difficult to reach – air services may be infrequent or non-existent, and intervals between ship visits may be lengthy and freight is expensive. Similarly, many island ISPs struggle to hire and retain qualified personnel.

Naturally, there are also a number of drawbacks, which start with those discussed in the previous section. In addition, there is now an additional challenge from the location perspective: As discussed in Section IV, the latency between satellite gateway, G_W and data sources may be significantly higher than the latency between satellite gateway and data sources alone. If the specialised off-island provider of G_W implements a coding-as-a-service scenario at scale, it will inevitably find its client software used in multiple island locations, with satellite gateways in geographically dispersed locations: An island in the western Pacific can have its off-island GEO gateway in Canada, whereas an island in French Polynesia might opt for space segment terminating in Australia.

A further challenge is the diversity in consumer operating systems. To be able to serve a large majority of users, the off-island provider would need to supply the application implementing G_I on multiple popular operating systems such as Windows, MacOS, IOS and Android. This represents significant additional effort compared to a tunnel application on a single operating system of the implementor's choice. It also carries the risk of leaving the end user machine disconnected: The software needs to modify the network configuration of the machine. There could be unintended consequences if, in doing so, the software interferes with any of the myriad of network configuration managers, tools and utilities which commonly inhabit these ecosystems. Given that the off-island provider has no control over what else may be installed on the end user's machine, this risk could be substantial. Another question that arises in this context is how an island user would pay the off-island provider: Not every islander has access to a credit card.

VII. Connecting the Tunnel Endpoints

On many islands, ISPs and/or satellite providers block UDP to keep traffic off their satellite link that does not back off under congestion. This can backfire, however, as many applications that have higher bandwidth efficiency using UDP do sense congestion and will switch to less efficient TCP when UDP is blocked. It is worth noting in this context that the UDP carrying our coded packets *will* back off as well: If too many packets of a generation are lost, the generation as a whole will become undecodable and the TCP packets it contains are lost as well, causing the contributing TCP senders to back off, too.

However, the communication between G_I and G_W need not rely on UDP. There are several options for this, two of which are discussed below:

A. Spoofing TCP

One option is to pass the coded combinations as TCP packets without actually running TCP at G_I or G_W. The only differences to the UDP variant are as follows:

- The packets carry a TCP header instead of a UDP header, with nominal sequence and acknowledgment numbers
- G_I and G_W acknowledge any packet received but do not attempt to retransmit any packets not received (and in fact ignore any ACK received)
- The first and second packet from G_I to G_W have their SYN and SYN+ACK flags set, respectively, and the first packet G_W to G_I correspondingly has SYN+ACK set.
- Either end ignores flags and ACK numbers upon receipt and concentrates on the packet payload instead.

To an outside observer, such flows are almost indistinguishable from genuine TCP and practically impossible to detect or block on a firewall with stateful inspection. Even in a real TCP connection, an observer somewhere along the path may not get to see all packets of the connection due to load balancing and asymmetric paths. The disadvantage of this approach is that it is a hack and, from the ISP's perspective, could be considered improper use.

B. Multiple TCP Connections

Another option would be to open multiple TCP connections between G_I and G_W at the outset and communicate only a small number of linear combinations (or even just one) per generation as data across each connection.

In scenarios where G_I and G_W are the only significant users on the satellite link, this has the advantage of replacing what would otherwise be a mix of TCP flows of varying lengths by a fixed number of TCP flows with infinite length and more or less equal data rate. Since the arrival of each combination is now ensured by TCP, one could also set $\omega = 0$ and thus reduce overhead to zero. However, TCP also adds its own overhead. It is also possible to use TCP variants optimised for long latency networks, such as Hybla [14] or H-TCP [15].

One potentially significant problem occurs at G_W (and possibly G_I, too), however: In the UDP or spoofed TCP scenarios, data arrives at G_W at full Gbps network rates and leaves in the direction of the sat gate at the same high rate in encoded form. So G_W does not need to buffer or concern itself with keeping any form of state once the coded packets of a generation have left. If we connect G_W and G_I via TCP, we transfer at least a significant part of the sat link bottleneck and its associated queue to G_W. Since TCP sockets cannot queue drop, G_W would need to implement this functionality *before* the linear combinations are written to the TCP connections with G_I.

VIII. Coding Considerations

The scenarios presented thus far do not consider under which circumstances one might achieve a goodput gain from coding. This section discusses a number of basic constraints that a scenario needs to satisfy in order to result in a gain.

A. Shared high latency bottlenecks

For TCP queue oscillation to occur at the satellite link, the link must represent a high latency bottleneck. If the link bandwidth does not represent a bottleneck, no queue can form at the input. Similarly, the fact that the satellite latency is shared between all flows is an essential ingredient for queue oscillation: It ensures that *all* senders respond with a significant minimum delay. Note that queues and queue drops also occur as part of normal TCP operation in terrestrial networks with links of varying latencies and bandwidths. However, where there are bottlenecks, there is often no significant latency that all flows share, and TCP senders accelerate and back off with a distribution of response times governed by network latency *away* from the bottleneck. Where there is significant latency, such as on transoceanic submarine fibre cables, there is often no bottleneck.

B. Flow size distribution

Real Internet traffic is highly heterogeneous. As a rule of thumb, most flows carry only a few packets' worth of bytes, as shown in Fig. 7, but most bytes in transit are found in large flows that contain many packets, as shown in Fig. 8. The beneficial effect from coding mostly accrues to TCP flows whose congestion windows can remain open wider for longer. This means that such flows must be long enough in order to see ACKs from the receiver arrive before their last packets have left the sender. Large TCP flows already slow down significantly at packet loss level of around 0.1% – a level at which only a fraction of small flows would be affected at all. Expending coding overhead on very small flows takes up capacity but yields no tangible benefit at all.

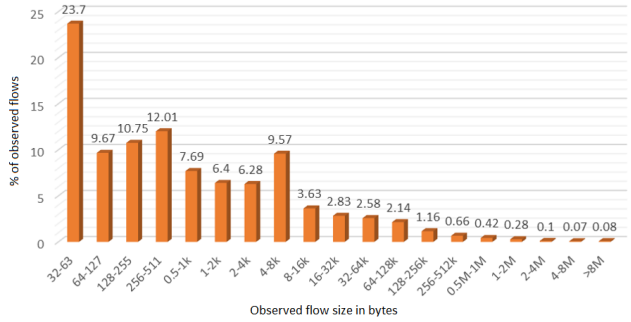

Figure 7. Percentage of flows of certain size classes in TCP traffic, observed on a real MEO link to Rarotonga, Cook Islands.

A typical default congestion window size in Linux is 10. At a maximum transmission unit (MTU) of 1500 bytes, this lets the sender transmit up to 15,000 bytes without having to wait for a first ACK. This rules out between 86 and 90% of all flows, but less than 3% of bytes on the link shown in Fig. 7 and Fig. 8.

One would thus expect coding to work best in situations where the share of large flows in the flow size distribution is high. Alternatively, it would be desirable to aim coding exclusively at large flows. However, this would require identifying such flows in advance, e.g., based on IP address and TCP port combinations known to produce large flows.

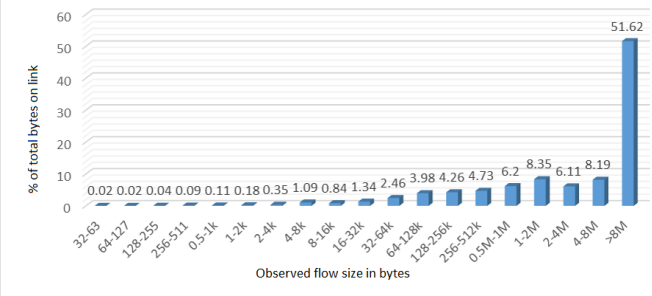

Figure 8. Distribution of bytes across flows of certain size classes in TCP traffic, observed on a real MEO link to Rarotonga, Cook Islands.

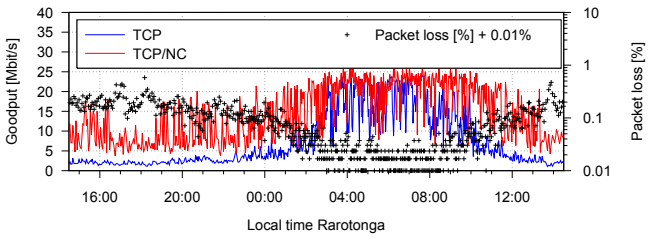

Figure 9. Goodput of uncoded (blue) and coded (red) TCP transfers between Auckland (New Zealand) and Rarotonga (Cook Islands) via a real MEO satellite link. Note that the goodput is quite comparable during the low demand hours (late night and early morning) with low packet loss from queue oscillation. During peak times, coded goodput significantly exceeds uncoded goodput. Peak link utilisation on this link was around 50%.

C. Demand considerations

A less obvious ingredient in TCP queue oscillation is the demand that is placed on the link. A TCP sender increases its congestion window when it receives ACKs from the receiver. The latencies involved on a satellite link make for a rather slow growth in congestion window. Consider a link whose queue is currently empty. If the number of flows simultaneously trying to increase their congestion windows across this link is small, the queue grows slowly as well: Each flow's contribution to the increase in arrival rate is small; their number limits the compound effect on the arrival rate. A slowly growing queue overflows more slowly, allowing for a more graceful back-off by the senders. Coding makes little sense in this context as most of the link underutilisation is a result of lack of demand rather than unreasonably harsh back-off.

As the number of competing large flows increases, their compound effect even under slow window growth results in fast queue fill. Similarly, the concerted back-off response of these flows results in a drastic drop in queue arrival rate. This is the demand region in which we expect coding to yield benefits, and where we have been able to observe significantly better goodput in practice. Fig. 9 shows a comparison between the goodput rate of coded and uncoded 20 MB TCP transfers via the otherwise uncoded MEO link above over a 24 hour period in 2015. The comparison shows that low TCP goodput and coding gain correlate with packet loss, which in turn has a strong diurnal pattern governed by demand. Even at peak demand times, the average link utilisation was only around 50% – a typical symptom of TCP queue oscillation. On this occasion, packet loss was determined by sending a sequence of large UDP packets, followed by a standard TCP transfer and then the coded TCP transfer, with whichever background traffic happened to be present.

At even higher demand, the rate at which new flows appear that have yet to engage with flow control can grow so large that packets from such flows alone can maintain a full queue. Large flows thus experience packet loss, no matter how much they back off, and consequently slow to a crawl. Fig. 10 shows this effect on a simulated 16 Mbps link using the flow size distribution above: As load grows, the link saturates with goodput from small flows. However, the long transfer cannot exploit the spare capacity; its goodput rate drops to about 2 Mbps at a load of 60 simultaneously active client sockets, even though the link still has over 6 Mbps of spare capacity. The fact that the transfer is able to achieve 8 Mbps at a low demand level shows that this is not a TCP slow start effect.

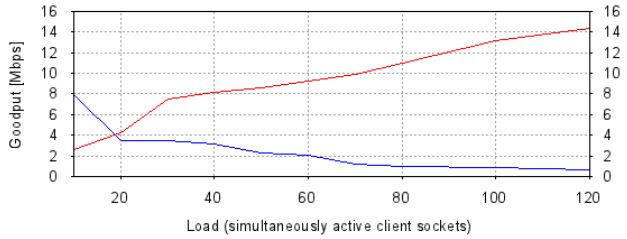

Figure 10. Goodput of uncoded TCP at different demand levels on a simulated 16 Mbps GEO link. The red curve shows the average goodput rate achieved by all flows on the link combined (red), the blue curve shows the goodput rate achieved by a single 40 MB transfer.

In the high demand regime, coding can at best displace some of the new flows but has no spare capacity on the link that could accommodate overhead and coding gains for all flows on the link. The authors of this study observed such a scenario on an (at the time) 8 Mbps GEO link into Niue, where high link utilisation simply saw coded TCP flows eat into the capacity taken up by conventional TCP traffic. The only benefit that coding yields at this demand level is that it can prevent individual large coded transfers from stalling – at a price.

D. Accommodating the overhead and gains

In principle, the more overhead ω we add, the larger the packet loss that we can tolerate. However, any overhead that makes it into an otherwise unmanaged input queue not only contributes to filling the queue, but subsequently makes it onto the link, and thus reduces spare capacity there. Compared to the uncoded case, the link's spare capacity must be able to accommodate both this overhead and the goodput gain we hope to achieve from coding [23]. There is thus a limit on how much overhead can be deployed before it starts to displace any goodput gains made. Quite where this point lies is a matter of ongoing research, especially as the question is further complicated by the timing issues discussed below, as well as by the type of tunnel.

Consider coding only a (small) subset of the (sufficiently large) TCP flows across a satellite link, as is the case for

satellite links with tunnels that do not involve ISPs or do not aim at coding all such flows, e.g., in the coding-as-a-service scenario. Then this subset of flows has, in principle, the link's entire spare capacity to expand into. In such cases, one can use comparatively large amounts of overhead and still accommodate potentially substantial gain over the base performance of these flows. In fact, this is what we have been able to observe in our actual island experiments, which coded only a small part of the traffic.

On the other hand, if the tunnel handles all (suitable) flows on the link, the link's spare capacity must be shared between the flows. All else being equal, we are now dealing with more flows, so the total capacity occupied by the associated overhead is larger, and the spare capacity available for gain per flow is smaller. However, because coded flows are less prone to rapid TCP queue oscillation, a scenario that codes all flows may require less overhead per protected flow volume.

E. Timing issues

In our scenarios, we can still expect the vast majority of packets to make it into the queue and subsequently reach the receiver. Using entirely random c_{ij} means that we need to collect all n incoming IP packets at the encoder before we can send our first coded UDP packet to the decoder, i.e., we have a decoding delay that is more or less proportional to n. Therefore, our coding approach does not choose all c_{ij} randomly. Rather, it uses *systematic coding*, which for $i, j \leq n$ sets $c_{ij} = 1$ if $i = j$ and $c_{ij} = 0$ if $i \neq j$. This allows the first UDP packet to be sent immediately after the first IP packet arrives at the encoder. Only the c_{ij} for $i > n$ are randomly chosen.

One may thus think of the first n UDP packets (*systematic packets*) as being merely encapsulated original packets, while the remaining ω UDP packets (*coded packets*) carry coded "spares" in case any of the first n packets are lost.

1) Coding and decoding delay: In this context, we need to remember that TCP itself also provisions such "spares" in the form of retransmissions after ACK timeouts. Coding gain is only possible if the spares from coding arrive at the receiver before any retransmissions. As retransmissions will not arrive for at least one RTT, any coded packet that arrives within about one RTT after the original packet(s) represents an improvement over TCP's own mechanism.

We also generally want the coded packets to contain redundancy for all of our n packets, i.e., we want only non-zero c_{ij} for $i > n$. This prevents us from sending coded packets until all n incoming packets of the generation have been received. This means that the first systematic packet and the first coded packet are at least n packets apart, and the associated delay is the time between the arrival of the first and the last packet of the generation at the encoder. As we need this delay to be below one RTT, this imposes a limit on n.

Similarly, we also have a limit on ω arising from the fact that the last coded packet would be useless if it took longer to arrive than any TCP retransmission for the last of the systematic packets.

2) Overhead timing: While the coded packets must be delivered within a certain maximum time limit, it is also important that they are not delivered too quickly. The only occasion when the decoder needs the coded packets is when systematic packets are lost during a queue overflow event. If we transmit the coded packets immediately after the last systematic packet, we risk losing them to the queue overflow as well.

Another consideration in this context applies to links with particularly low transmission rates, especially in the scenario where the encoder G_W sits between the Internet and the off-island satellite gateway and has a Gigabit Ethernet (GbE) connection to the latter. Here, we need to consider that IP traffic headed to the gateway will generally have data rates *in the same order of magnitude* as the transmission rate of the link. However, this may still be one or two orders of magnitude below the GbE rate at which G_W can send overhead. A well-dimensioned queue at the satellite gateway will be designed to buffer at arrival rates in the order of the link rate. If we let G_W fire its overhead at the satellite gateway at GbE rates, we risk almost instantaneous queue overflow.

The following "all-of-island coding" experiment illustrates this: Consider sending a 32 Mbps UDP data stream into an encoder with $n = 60$ and $\omega = 10$. The encoder thus outputs around 37 Mbps, which is fed into a simulated 16 Mbps satellite link with 120 kB input queue capacity, i.e., overloading the link by a factor of at least 2.3. Note that this implies a permanently full queue at the satellite gateway. The ratio between systematically coded packets and overhead packets as they leave the encoder is n/ω, i.e., 6:1 in this case, and all overhead follows the systematically coded packets without delay on a Gigabit Ethernet link. On the other side of the link, the statistics collected by the decoder record the number of packets received of each of the two types.

One may now naively assume that the decoder should also see six times as many systematically coded packets as overhead packets. However, in the actual experiment, this was not so: The ratio between the packet types observed at the decoder was approximately 56:1. This demonstrates that overhead packets hitting the queue at a 1000 Mbps rate were largely dropped, with only a few getting the chance to fill whatever occasional spare capacity there may have been in the queue. When the experiment was repeated at rates below the link rate instead of at 37 Mbps, it did not overload the queue and all packets arrived.

F. Sizing n and ω – and choosing what to code

Beyond the timing considerations above, the choice of n and ω has other implications. For the same fraction ω/n of overhead, larger n provide higher robustness against the burst losses from queue overflow events. However, this carries a cost in terms of decoding complexity and decoding delay, as solving an $n \times n$ system of linear equations is $O(n^3)$. Systematic coding mitigates somewhat against this but still leaves an $O(\omega^3)$ residual complexity.

Larger ω add robustness but also contribute towards the development of standing queues at the input to the satellite link. Last but not least, it pays to remember that one should not attempt to correct for *all* packet losses at the input queue.

This would simply lead large flows to open their congestion windows to the point where the *average* data rate of the flow becomes unsustainable given the link capacity. Packet loss at queues is a necessary feature of TCP. The only goal of coding can be to delay the onset of packet losses at the receiver and to aid in the recovery from loss events where a large RTT prevents this from occurring in a more timely fashion.

The topologies presented in this paper open up several choices when it comes to deciding *what* to code. The choice in principle is between coding individual flows and coding all flows together. In the first case, all n packets of each generation belong to the same TCP flow. In the second case, a generation usually contains packets from multiple flows.

Coding individual flows requires the encoder and decoder to maintain state, i.e., they must be able to detect when a TCP flow starts (SYN or SYN+ACK packet) and when it ends (FIN, FIN+ACK, or as a result of a variety of timeouts), which adds complexity. It also implies that a generation of n packets will typically cover a larger time interval. With traffic naturally interleaved, it requires the encoder and decoder to maintain as many active generations as there are active flows. However, as all flows contribute to the packet loss during a queue overflow event, this limits the number of packets an individual flow loses – and this number determines sensible values for ω. Our observations in Rarotonga (shown above) and Tuvalu confirm that this approach can work even if all other flows on the link remain uncoded.

Coding all flows together means that the encoder and decoder only need to maintain one active generation at a given time, at least in principle, and do not need to maintain flow state. Generations fill faster, but burst losses from queue overflow now generally require more overhead per generation to correct, because the losses now no longer spread across multiple generations for multiple flows. Note however that a lost generation in this scenario generally still only translates into a small number of lost packets per flow, so the damage done to the flow's congestion window remains limited. Systematic coding further reduces this problem as systematically coded packets received do not require the presence of a whole generation of n coded packets to decode.

IX. Experimental Options

When investigating coding techniques over satellite links, one has the choice between four approaches:

1) Physical on-site experiments with a real production link. The authors of this paper used this approach in four Pacific Island locations [21], [22], and were able to obtain very encouraging results when coding small numbers of mostly large flows. The advantage of this approach is that the link uses real equipment, carries real traffic, and is subject to all phenomena a real link encounters. There are a number of disadvantages, too: Working on site is expensive, time-consuming, and logistically complex. Some of this can be addressed by only deploying equipment and working remotely, but this depends on the cooperation of locals who may need to assist in troubleshooting. Another challenge is that most islands of interest will only have one satellite link, which is often their only real-time link to the outside world. Loading this link with measurement traffic (e.g., TCP transfers for timing purposes or UDP streams for determination of packet loss) comes at an expense to the locals. Similarly, inserting an encoder/decoder into the path on either side of all-of-island coding requires reconfiguration and a (brief) service disruption, an operation with considerable commercial and health & safety risk: Typically, at a minimum, the local hospital, bank(s), airline(s) and government rely on the link.
2) Laboratory experiments using dedicated actual space segment. This option is the most costly, especially if one wants to be able to investigate all types of satellite links one encounters in practice. Another challenge here is to generate appropriate load.
3) Software-based simulation. The authors of this paper used this approach in [22], but it became apparent pretty quickly that it had serious drawbacks. Both TCP queue oscillation and coding are time-sensitive processes. Software network simulators can generally not simulate the networks of interest here in real time, so questions arise as to how the simulators handle the parallel generation of traffic, the chaotic interleaving of traffic before it arrives at the satellite gateway queue, TCP timeouts etc. Another question is how to integrate coding software written for a real TCP stack into a simulator. Even if this is all taken care of, simulating many hundreds of TCP clients and a large number of servers simply takes a very long time.
4) Hardware-based simulation/emulation: This is the approach we currently take.

Figure 11. The Auckland Satellite TCP/IP Traffic Simulator at the University of Auckland. The two racks on the left contain the "island" machines, the next rack to the right accommodates (from bottom) capture servers, copper taps, the satellite chain, and "world" servers on the top. The rack on the right holds the remaining "world" servers, a spare, and a special purpose server.

Fig. 11 shows the current setup of our simulator [24]. It uses 96 Raspberry Pis and 10 Intel NUCs to simulate up to several thousand "island clients", which are served with data from 22 "off-island" Super Micro servers operating with a variety

of "terrestrial" latencies. A dedicated Super Micro server emulates the satellite link itself with its input queues, delays and bandwidth constraints. Two further servers operate as encoders/decoders; two more can run performance-enhancing proxies such as PEPsal [25] or TCPEP [19], and two standalone capture servers give passive listening access to any part of the network. Two further Raspberry Pis and another server are used for signaling and active measurements during experiments, and a central command, control and storage server orchestrates the other machines, wherever possible via an external, independent harness network. This approach has several advantages: It is real-time, uses real components (except for the satellite link itself), and by serving TCP data that follows a real size distribution, it is possible to control the load via the number of simultaneously active client sockets. The simulations shown in this paper in Fig. 1 and Fig. 10 were produced with the previous version of this simulator.

X. Conclusion

Network-coded tunnels carrying TCP/IP traffic in coded form across lossy bottlenecks in satellite networks have been shown to be able to improve goodput under TCP queue oscillation conditions even in the presence of a majority of flows using legacy TCP. The core insight that underpins the tunnel concept is that packet losses occur by queue drop at the input queue to the satellite link. As long as one can protect traffic against data loss at this location, the remaining system topology is a question of who will or can provide the tunnel service, how cooperative the local ISP and satellite provider are, and how much gain one requires from the coding to make the effort worthwhile.

As a general rule, topologies in which these two players are not involved (or even actively oppose the use of coded tunnels) should be less effective: The presence of legacy TCP connections forces coded traffic to use more overhead, so any parties on the island with coded traffic consume more data and bandwidth than necessary. Those not using coding are also put at a potential disadvantage as this may eat into their bandwidth as well. Active involvement of satellite providers and/or local ISPs thus seems advantageous.

On the other hand, coding all traffic for an ISP poses a number of additional challenges: One needs a more careful code design, goodput gains distribute over a larger number of flows, and timing of overhead is more critical. Avoiding the coding of small and inflexible flows becomes more important.

At the time of writing, only experimental implementations of coded tunnels are available. These are based on a Debian/Ubuntu Linux kernel module. While they do not cater for the subscription model discussed in Section VI at this point in time, they nevertheless represent a proof of concept for the remaining scenarios.

Current work aims to demonstrate that the technology scales to whole-of-island coding. For this purpose, we have built the hardware-based Auckland Satellite TCP/IP Traffic Simulator, which is capable of running island scenarios with up to around 4000 simultaneously active client sockets.

Acknowledgment

The research reported on in this paper would not have been possible without the generous support of many parties: APNIC/ISIF Asia and Internet NZ supported our work with multiple grants, and Brian Carpenter kindly donated a large residual balance in his research account to us. The Pacific Chapter of the Internet Society have been strong supporters throughout, and none of the practical work in the islands would have been possible without Telecom Cook Islands (now Bluesky Cook Islands), Internet Niue, and Tuvalu Telecom opening their doors and equipment racks to us. Meitaki ma'ata, fakaue lahi, fakafetai, thank you!

References

[1] U. Speidel, 'E. Cocker, M. Médard, Janus Heide, and Péter Vingelmann, "Topologies for the Provision of Network-Coded Services via Shared Satellite Channels", Ninth International Conference on Advances in Satellite and Space Communications (SPACOMM2017), Venice, Italy, 2017.

[2] J. Postel, "Transmission Control Protocol", RFC793, 1981, https://tools.ietf.org/html/rfc793 (accessed 24 November 2017).

[3] V. Jacobson and R. Braden, "TCP Extensions for Long-Delay Paths", RFC1072, 1988, https://tools.ietf.org/html/rfc1072 (accessed 24 November 2017).

[4] V. Jacobson, R. Braden and D. Borman, "TCP Extensions for High Performance", RFC1323, 1992, https://tools.ietf.org/html/rfc1323 (accessed 24 November 2017).

[5] D. Borman, R. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance", RFC7323, 2014, https://tools.ietf.org/html/rfc7323 (accessed 24 November 2017).

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", RFC2018, 1996, https://tools.ietf.org/html/rfc2018 (accessed 24 November 2017).

[7] –, SES web site, https://www.ses.com (accessed 24 November 2017).

[8] B. Braden et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC2309, 1998, https://tools.ietf.org/html/rfc2309 (accessed 24 November 2017).

[9] B. Briscoe and J. Manner, "Byte and Packet Congestion Notification", RFC7147, 2014, https://tools.ietf.org/html/rfc7147 (accessed 24 November 2017).

[10] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management", RFC7567, 2015, https://tools.ietf.org/html/rfc7567 (accessed 24 November 2017).

[11] J. Postel, "Internet Control Message Protocol", RFC792, 1981, https://tools.ietf.org/html/rfc792 (accessed 24 November 2017).

[12] J.-M Jouanigot et al., "CHEOPS dataset protocol: an efficient protocol for large disk-based dataset transfer on the Olympus satellite", International Conference on the Results of the Olympus Utilisation Programme, Sevilla, CERN CN/93/6, 1993.

[13] J. Kim and I. Yeom, "Reducing Queue Oscillation at a Congested Link", IEEE Transactions on Parallel and Distributed Systems, 19(3), 394–407, 2008.

[14] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks", Int. J. of Satellite Communications and Networks, 22, 547–566, 2004.

[15] D. Leith, "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths", Internet Draft, IETF, April 7, 2008. https://tools.ietf.org/html/draft-leith-tcp-htcp-06 (accessed 24 November 2017).

[16] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant", ACM SIGOPS Operating System Review, 42(5), 64–74, 2008.

[17] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation", Proc. IEEE, 99(3), 490–512, 2011.

[18] J. Hansen, J. Krigslund, D.E. Lucani, and F.H.P. Fitzek, "Sub-Transport Layer Coding: A Simple Network Coding Shim for IP Traffic", IEEE VTS Vehicular Technology Conference (VTC), 1–5, 2014.

[19] G. Delannoy, "Design and Implementation of a Performance-Enhancing Proxy for connections over 3G networks", Dublin City University, May 27, 2013, https://github.com/GregoireDelannoy/TCPeP/blob/master/Final_Report.pdf (accessed 24 November 2017).

[20] H. Bischl, H. Brandt, and F. Rossetto, "An experimental demonstration of Network Coding for satellite networks", CEAS Space Journal 2.1-4, 75–83, 2011.

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

[22] U. Speidel, L. Qian, 'E. Cocker, P. Vingelmann, J. Heide, and M. Médard, "Can Network Coding Mitigate TCP-induced Queue Oscillation on Narrowband Satellite Links?", International Conference on Wireless and Satellite Systems, Springer International Publishing, 301–314, 2015.

[23] U. Speidel, S. Puchinger, and M. Bossert, "Constraints for coded tunnels across long latency bottlenecks with ARQ-based congestion control", IEEE International Symposium on Information Theory, Aachen, Germany, 271–275, 2017.

[24] Auckland Satellite TCP/IP Traffic Simulator, The University of Auckland, https://sde.blogs.auckland.ac.nz/ (accessed 24 November 2017).

[25] C. Caini, R. Firrincieli, and D. Lacamera, "PEPsal: a Performance Enhancing Proxy designed for TCP satellite connections", IEEE 63^{rd} Vehicular Technology Conference, pp. 2607–2611, 2006.