isif asia

Project factsheet information

Project title	Realistic simulation of uncoded, coded and proxied Internet satellite links with a flexible hardware-based simulator
Grant recipient	The University of Auckland Private Bag 92019, Auckland, New Zealand +64-9-3737599 http://www.auckland.ac.nz
Dates covered by this report	27 – 09 – 2016 / 20 – 10 – 2017
Report submission date	31 – 10 – 2017
Country where project was implemented	New Zealand
Project leader name	Ulrich Speidel (ulrich@cs.auckland.ac.nz)
Team members (list)	Lei Qian (lqia012@aucklanduni.ac.nz)
Partner organizations	Massachusetts Institute of Technology Steinwurf ApS, Aalborg, Denmark The University of Ulm, Germany
Total budget approved	AUD 45,000.00
Project summary	Shared narrowband Internet satellite links are a staple in many islands of the South Pacific. They often underperform due to the difficulties that the dominant Internet transport protocol TCP faces in estimating the available capacity across the link. Our wider project aims to build a simulator capable of replicating the demand profile and other conditions encountered on such links in order to be able to study potential solutions to the problem, such as network coding or performance enhancing proxies. The specific objectives funded by the grant aimed primarily at the development of tools to automate the experimentation process. This includes scripts that configure the link emulator, the “island clients” and the “world servers” (a combined total of over 100 machines), any encoders, decoders, or performance enhancing proxies, start and stop experiments, and retrieve and analyse log files from around the system.

Table of Contents

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Background and Justification

Remote locations such as many Pacific Islands face a significant challenge when it comes to getting Internet connectivity: Laying submarine fibre cables [1] to such islands is expensive because of the huge distances (often many hundreds of kilometres) and depths involved. Small populations combined with huge distances and often low GDP make for bad business cases for the upfront cost of such cables. The only alternative at present are satellite links, with a number of providers of *geostationary* (GEO) and *medium earth or*bit (MEO) solutions vying for business. While the entry price for satellite solutions is lower, their relative cost for capacity provisioned is higher, typically in the hundreds of US dollars per Megabit per second (Mbps) of *capacity* per months.

Much to the chagrin of *Internet Service Providers* (ISPs) and users in such locations, such satellite links often fail to reach their nominal capacity by a considerable margin. Meanwhile, their users report slow download speeds. The blame for this lies neither with the ISPs nor with the satellite providers, however, but with the dynamic behaviour of the Internet's *Transmission Control Protocol* (TCP) [2][3]. TCP carries the bulk of the Internet's file transfers, including web traffic, software downloads and e-mail, making it an essential part of any Internet connection. Unfortunately, TCP was not designed with satellite links in mind.

What makes satellite links special? Two factors: Firstly, latency - the time it takes for a transmission to get from the sender to the receiver. In the case of a TCP data packet travelling via a GEO satellite, the latency is at least 250 ms (and often more if the "world" side of the connection is not located close to the satellite gateway). On a MEO satellite such as those operated by O3b Networks [4], this is around 190 ms lower, but still comparable to the latency on a fibre cable between Australia and the USA. The *round-trip-time* (RTT), the sum of the two one-way latencies in a TCP connection, determines the timeliness of the feedback that a TCP sender gets on its transmissions.

The second factor is capacity: At ISP level, cable-based communication now generally happens on networks using Gigabit Ethernet or faster. In comparison, some islands in the Pacific have to make do with inbound satellite capacities of 16 Mbps - the equivalent of an ADSL-based home Internet connection in New Zealand, or about 60 times below the capacity of the networks connected at either end of the satellite link.

TCP sends data in the form of packets - snippets of data - and keeps a running tab of their successful delivery by counting the bytes they contain and "crossing them off the list" when a TCP sender receives an acknowledgement (ACK) packet from the receiver. In order to make good use of the capacity offered by links, TCP needs to be able to have multiple data packets in transit to the receiver before it can expect the ACK for the first of these packets to arrive back. The number of packets that a sender should transmit before expecting an ACK is determined by the *bandwidth-delay product* of the connection between the sender and receiver. The delay in this product is the RTT, which is relatively easy to determine with reasonable accuracy. The bandwidth, however, is the available capacity on the links between sender and receiver that is not already taken up by other traffic. In the case of our satellite links, which are always shared between multiple connections, the million dollar question in this respect is: "How much capacity is left and how many data packets should I send?"

TCP is in principle designed to maximise the use of link capacity that it shares with other TCP connections by making this number adaptive [5][6]. This feature, known as *congestion control*, relies on the ACKs from the receiver to inform the sender whether its packets arrive or have fallen victim to queue drops as a result of congestion along the way.

Crudely speaking, the sender allows more packets to be in transit as long as it receives a contiguous sequence of ACKs, and curtails the allowable number in transit exponentially (*exponential backoff*) if ACKs become overdue. The number of packets that a sender will transmit before waiting for an ACK is known as the sender's *congestion window*. Connections that have a large bandwidth-delay product available to them thus benefit from a large congestion window, whereas connections with small bandwidth-delay products require small congestion windows. As large congestion windows carry the risk of causing congestion, TCP tends to err on the side of caution: It starts with a small congestion window (typically 10 packets) and then increases it as ACKs arrive.

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

When ACKs fail to arrive, TCP detects packet loss, and decreases the congestion window exponentially. A number of algorithms exist for this purpose.

Enter our satellite links: We now have a large number of TCP senders trying to have just the right number of packets in transit, but ACKs may not return for hundreds of milliseconds. Moreover, along the way, we have the capacity bottleneck of the satellite link. So what happens? Our TCP senders accelerate as they receive their ACKs from their earlier transmissions at lower packet rates. This increase in traffic causes the queue at the input to the satellite link bottleneck to grow, and eventually overflow. The senders now lose packets there, but still receive ACKs for older packets back from the far end. By the time the ACKs stop arriving, the senders have lost a lot of packets, causing them to radically reduce the number of packets in transit. The queue clears and the satellite link sits idle for a while, until the cycle repeats.

This phenomenon, called TCP *queue oscillation*, has been known for the best part of three decades [7], but has not been solved. It accounts for both the link underutilisation reported by the ISPs, and the slow download speeds reported by their users. In effect, it means that Pacific Island ISPs are paying for satellite capacity they cannot use.

Since TCP's large worldwide installation base precludes wholesale replacement of the protocol, one needs to look for other approaches to a solution. We are often asked whether active queue management techniques such as RED [8] or CoDel [9] are the way to go here. Sadly, this is not the case since these techniques also rely on feedback affected by the same latency: There are simply no low latency connections on a satellite link.

So, what works? The classic approach has been to reduce the latency by using performance-enhancing proxies (PEPs) [10][11]. PEPs generally split the TCP connection at or near the off-island satellite gateway, effectively operating two connections in series, each of which has a lower individual latency than an end-to-end connection. However, most comparative literature on PEPs - and there is not much - looks at scenarios where the PEPs handle multiple parallel large downloads. This is quite different from the mix in TCP data flows encountered on real links in the Pacific, where download range from a few bytes to into the hundreds of megabytes (MB). PEPs also suffer from problematic failure modes when the split connections do not terminate cleanly [12].

A more recent approach has been to use network codes (or, more generally, error-correcting codes) [13]. While forward error-correcting codes (FEC) have long been a staple in space communication, they are generally applied on the space segment between the satellite gateways only. In our case, however, the point of loss sits literally at the entry point to the FEC encoder - meaning that the protection of the codes does not extend to the packets lost there.

The fundamental idea of coding TCP is to change this, so that the encoding happens *before* the packets reach the input queue to the satellite modem. In principle, this allows *all* TCP packets to be protected against loss and thus lets the senders sustain higher average packet rates.

In a precursor project funded by ISIF Asia, we investigated the performance of such network coding software when used on a small number of individual large TCP transfers on actual satellite links in the Pacific (Rarotonga, Niue, Tuvalu, and Aitutaki). The project found that on the links that showed evidence of queue oscillation (Rarotonga and Tuvalu), the coded transfers could access some of the unused capacity and achieve goodput rates of typically at least twice those of uncoded transfers [14][15][16].

As the precursor project used production links that we could not disrupt, there were a number of questions it could not answer:

- How feasible would it be to code all traffic to an island? It was always clear that gains would be restricted to the unused capacity less coding overhead, but how much capacity could we claw back?
- How well would coding work compared to a PEP? Could one combine coding and PEPs?
- Would other coding schemes offer potentially better performance?

Because we could not investigate these questions in the islands, we decided to head back to the lab and build a simulator that would emulate a Pacific Island ISP, its users, and the TCP senders in the world that they draw

data from. We kick-started the project with a number of existing machines, university CAPEX and OPEX, and a generous donation from a research account courtesy of Brian Carpenter, who needs no introduction in Internet circles. Internet New Zealand came to the party with a grant for further equipment. This is where the story of the present project starts.

Project Narrative

The platform

At the start of the project, most of the project hardware that we required immediately was in hand:

- 17 Super Micro servers: 14 servers emulating the "servers of the world" that send data to the "island", three servers to emulate satellite links and provide for an encoder/decoder/PEP/observation point on either side of the link.
- 84 Raspberry Pis to act as clients receiving the data on the "island side" of the simulator.
- 10 Intel NUC machines to complement the Raspberry Pis.
- 2 Raspberry Pis to act as special purpose machines on the world & island sides of the simulator.
- A command and control desktop which doubles as the project's file storage.

Most of these had already been configured and installed at the time of application, and we had confirmed via baseline experiments that all components "talked" and were able to reach unthrottled capacity across the simulator using artificial traffic designed to maximise load and flush out any unexpected issues. A network topology of the simulator in its configuration at the time was submitted as part of the interim report.

In 2017, we were offered a significant capital expenditure injection by the University of Auckland's Department of Computer Science, within which we operate. This allowed us to acquire an additional 15 Super Micro servers and another 12 Pis. This opportunity and would not have been available to us had it not been for this ISIF Grant.

One of the new Super Micros replaces the student desktop used as command, control and storage machine, a further 8 boosted our pool of world servers. A further two machines now allow us to separate coding and PEP functionality – so we can now run coded PEP traffic, yet investigate which influence each technology has on the result. Another two machines with large drives now act as dedicated traffic observers on either side of the link, supported by a fleet of five copper taps. The remaining two new Super Micros act as a spare "world server" and as a special purpose server, which conducts active measurements and is also the vantage point from which we inject UDP traffic into the link. A current topology is attached at the end of this report.

Our first task in 2016 was to test hardware and software together, and to learn to "drive" the simulator so it would yield usable data.

Learning to drive

…our own software:

At the time of application, we had also developed prototype versions of our experiment client and server software. This is special software and operates a little different from conventional textbook client/server applications: Each physical client runs the client software, which maintains a configurable number of "channels". Each channel runs a loop that opens a TCP connection to a randomly selected server on the "world" side, downloads whatever data the server has to offer, gets disconnected, and immediately repeats the process until the end of the experiment. The total number of channels across all physical clients gives us a pretty precise "knob" with which we can increase or decrease the "demand" on the link.

The server side acts largely like a normal TCP server, in that it accepts connections from clients and sends them data. The special feature here is that it determines the amount of data sent from a configurable distribution. Our default distribution is a flow size distribution collected at Bluesky Cook Islands in Rarotonga. This allows us to get a realistic flow size mix - and it is a very strongly skewed distribution, too: Its median flow size is a few

hundred bytes (read: half the flows are smaller, the rest larger), its mean flow size (total number of bytes in all flows divided by the number of flows) almost 100 times larger, and the longest flow is over 800 MB. This had a very direct impact on our progress, and for this reason we will return to this topic a little later.

When testing the software prototype, we noticed that no physical client seemed to be able to handle more than around 20 channels. That is, the client would be able to open connections on any number of channels, but only the first 20 or so would receive significant amounts of data, and be able to complete and renew their connections. Beyond that number, the channels would connect at most a couple of times and receive almost no data compared to the others. Mighty strange!

Moreover, this effect seemed to be independent of platform (NUCs and Pis were similarly affected) and several major restructuring attempts (separate processes for each channel, threaded, active sockets serviced in a loop) made no difference whatsoever. Twenty or so it was. Problem: With only 94 physical clients, this effectively limited our ability to reach our target of being able to deploy well over 2000 channels. We reached out to the most experience network hacks we could find - and drew a blank. Even the good people at CAIDA at USCD had never seen this happen.

It then dawned on us that we were doing something rather unusual: Sure, many people had written applications that run multiple sockets in parallel. Browsers do, for example. However, these applications do not normally replace sockets with new ones after the old ones have completed their task. We eventually traced the problem to a quirk in POSIX-based operating systems (read: all common operating systems), which always assign the lowest available socket file descriptor when a new socket needs to be create. Unfortunately, they also service these file descriptors in ascending order, starting at the bottom. This means that sockets with lower numbered file descriptors have their buffers copied with priority. They finish first, and bequeath their file descriptors to the next sockets off the production line. As a result, sockets with higher numbered file descriptors never get service. We solved the problem by artificially limiting the rate at which the servers write to individual connections - not a problem in a satellite context, but it lets us handle 40 channels and more per physical client.

Lesson learned: Some socket file handles are created more equal than others.

Learning to drive…a simulated satellite link:

The next task was to configure the “satellite link”, or more specifically the latencies and capacity constraints we needed. A bit of light googling yields any number of recipes as to how one might implement such an “impaired link”: The general recommendation is to use Linux **`tc`** traffic control with a **`netem`** delay qdisc for the latency and a token bucket filter of sorts to enforce the satellite’s bandwidth constraint. Sounds easy, and sure enough it appears to be that way. Test with **`ping`** and you get the expected round-trip time. Test with **`iperf3`** and the bandwidth constraint appears spot on. Test with both simultaneously and you get round-trip times that none of the buffers involved will explain, and eventually your interfaces block completely.

Lessons learned after a lot of trial and error: Do not try to chain a token bucket filter and a **`netem`** delay on the same interface. Ensure that your **`netem`** delay is configured with sufficient buffer to allow it to hold the entire bandwidth-delay product of your satellite link.

In order to automate the task of configuring the simulated link (which involves the removal of any previous configuration, the configuration of two network interfaces with netem delays and ingress redirects, and of two “intermediate function block devices” with token bucket filters), we wrote a **`bash`** shell script that we pass link bandwidth, queue capacity and a number of other parameters. This script runs whenever we change link configuration between experiments and forms part of our results with respect to **Objective 2** (*Automating experiment runs in non-coded configurations*) and **Objective 3** (*Automate experiment runs in coded configurations*).

Learning to drive …multiple machines at the same time:

The Unix utility we had selected for this purpose was **`pssh`** (parallel ssh). This tool allows scripts to open parallel secure shell (SSH) connections to multiple machines - just what we needed for the script development part of the project! So we happily built an extensive script that would start up the servers, start packet capture on both sides of the link, start the clients, run the experiment, backhaul the results from across the simulator to command and control server and analyse them there. As part of the analysis, we wrote a script that would automatically generate per-client and per-server statistics and check them to ensure that all of the servers and clients configured had actually partaken in the experiment and were seen on the link on a regular basis.

This script alerted us to the fact that some clients seemed to arrive late on the job. In fact, not just randomly late but by almost exactly 60 and 120 seconds, respectively. On closer inspection, it turned out that **`pssh`** has a hardwired limit of 32 connections at a time - any further connections are deferred by a minute in batches of 32. Since we could not easily override that limit, we wrote a script that load balances the required number of channels across the physical clients and then starts multiple instances of **`pssh`** for up to 32 identically configured clients each. Problem solved! Lesson learned: Expect the unexpected - not everything is documented.

Learning to drive …the poltergeists in the simulator away:

The same analysis script that had alerted us to the **`pssh`** problem flagged the intermittent "absence" of a number of the "world servers". We then noticed that these servers would also only respond intermittently to connection attempts on the simulator-internal network for the experiment traffic when we tried to contact them via this network. As all of the servers also have an interface on the university network, we could check that they were indeed operating as intended - they just seemed to take their occasional senior moments on the experiment interfaces.

It did not take long to find the problem: an IP address conflict. We demonstrably had several machines on the experiment network that responded to the same IP addresses as the legitimate servers. Moreover, their hardware addresses were all issued by Super Micro… but we knew the hardware addresses of our servers - and they did not match any of them. The initial hunt for "that idiot grad student or cleaner who must have plugged the wrong cable from another machine into our switch" yielded no results. Only our machines were on the switch, and so were the offending addresses. Eventually, we only had two machines on the switch, and still an offender among them. At this point, we started to suspect that there was more to the Super Micros we had inherited. Indeed: Someone on a previous project had activated their embedded IMPI servers - small embedded computers that run a little web server with which one can remotely monitor and maintain the main server machine.

These IPMI servers often have their own network connectors. On the Super Micros, they share the same physical connector. That someone had set the IPMI's IP addresses to private IP addresses in the same range our servers used, such that the servers became "unresponsive" if an IPMI beat them to an ARP (address resolution protocol) response.

Lesson learned: Never assume that your gear doesn't have a nice feature just because you didn't pay for it.

These and other gremlins out of the way, we could finally proceed to experimentation in September 2016.

Preparing for uncoded baseline simulations

Uncoded baseline simulations are not to be confused with our baseline experiments carried out at the time of application, whose purpose was to demonstrate that the simulator hardware did not contain any hidden performance bottlenecks that would have prevented us from using the hardware for the simulation. The uncoded baseline simulations actually use the equipment as intended, i.e., as a satellite simulator, and simulate using the full complement of physical clients and servers responding according to our empirical flow size distribution.

Our goal was now to be able to show under which uncoded scenarios TCP queue oscillation would occur. This naturally meant investigating a large number of scenarios, and **Objective 2** (Automating experiment runs in non-coded configurations) was key to our ability to do this efficiently. As the possible combinations of satellite capacity, demand and input queue capacity are practically infinite, this required a bit of planning:

- We had to select representative scenarios for investigation. We settled on eight different base scenarios: GEO links with 8, 16, 32 and 64 Mbps capacity, and MEO links with 32, 64, 160 and 320 Mbps capacity, respectively. As the behaviour of a link under load depends on both the input queue capacity and the load level, we needed to run experiments for a large number of combinations, guided by literature on queue capacity and observations from experiments as we completed them in order to determine recommendable queue capacities on which we could then base further experiments.

  As a rule of thumb, the larger the queue capacity at the input to the satellite link, the wider the demand range a link can handle before TCP queue oscillation slows long transfers down. That is, a large queue capacity is good for large TCP downloads. At the same time, a large queue capacity means a potentially long queue sojourn time, which impairs real-time protocols such as VoIP. We also needed to determine the load levels at which queue oscillation set in, and at which levels standing queues would form in the queue buffer. High loads result in high link utilisation, but prevent any sizeable TCP download from completing. This involved experimenting with a range of load levels and queue capacities for each link scenario.

- Choosing an experiment protocol. This was a relatively complex task, requiring a number of experiments in its own right. What exactly did we need/want to observe, and how would we get at these observables? We wanted to know: goodput, throughput, loss, but also practical information such as how long a download of a given size would take in a certain scenario, and how the input queue would behave during the experiment. We would also like to know how many parallel TCP flows operated at any one time, and how many flows in total an experiment covered.

  As a result, we capture packet data both as the packets enter the first of the three link chain machines and as they exit the last. A ping from the world special purpose Pi to its island counterpart at the beginning and end of each experiment acts like a referee whistle and lets us synchronise the packet traces from both ends of the link. Comparison of the packet traces allows us to determine loss, throughput and goodput. We also start a dedicated large download shortly after the beginning of each experiment to determine the download time. Last but not least, we send a series of ping packets at short intervals. Because of their small size, they are practically always admitted into the input queue, and give us an accurate picture of the queue sojourn time.

- Part of choosing the experiment protocol was to determine how long an experiment would run for. With a highly skewed flow size distribution, we need to run experiments sufficiently long - especially at low loads - to ensure that the experiment contains enough of the rare large flows in order to yield an average flow size that resembles that of the distribution we observed in Rarotonga. Think of it this way: Lottery wins represent a highly skewed distribution. Small wins are far more probable than large wins. Playing Lotto just a few times will usually yield wins below the average win only. Playing Lotto a few million times is likely to give us a jackpot or two, and get us closer to the average win.

  Running our experiments for very short periods leaves us with results that are possibly not representative. Running them for longer periods takes time, especially when may experiments are involved to cover a large number of configurations. At a given load level, experiments with higher link capacities produce more flows during a given time, and MEO experiments produce more flows than GEO experiments. While that means that we can shorten the measurement period for the same degree of convergence, the faster links also result in larger trace files and longer analysis times, which largely compensates for any gains in the measurement period itself.

  In total, we settled on measurement periods of 600 seconds for individual 8 Mbps GEO experiments. At the other end of the scale, a 320 Mbps MEO link requires only about 90 seconds (and its results

converge much better during that period). In total, however, each experiment took around 20 minutes to complete.

With these design decisions made, we could finally start experimenting. Under **Objective 2 and 3**, we wrote a bash script **`run-exp.sh`**. In its present version (14 kB / 460 lines), this script:

1. Ensures that there are no other experiments running.
2. Creates a log file directory for the experiment run
3. Opens the result log file for the run
4. Records server, link and (where applicable) coding configuration to the log
5. Tests the status of all machines and links in the simulator before the start of an experiment run. This includes invoking a script that checks that any ping round-trip times to interfaces configured with latency and possibly jitter are within the allowable range. This also functions as part of the quality control for Objective 4 (*Automate the configuration of the "world Internet" on the simulator*)
6. Determines the required experiment duration, guard times, and download sizes
7. Starts an iperf3 server to prepare for an individual TCP data transfer
8. Starts packet captures. We always start and stop captures progressively such that each capture at an observation point includes the entire capture period of all captures taken at observation points closer to the world side. This ensures that we cannot “lose” packets heading to the island as a result of a packet passing through a subsequent capture point that is not yet listening.
9. Starts packet capture on the satellite link when required. This feature is useful when the traffic across the satellite is coded.
10. Flushes the path maximum transmission unit (PMTU) cache on the servers to ensure that any fragmentation of IP packets reflects the current link setup.
11. Starts the “world” servers which provide the bulk data that is transmitted across the link.
12. Starts the island clients. This gets the bulk traffic across the link flowing.
13. Sends a ping from the special purpose Pi on the world side to its counterpart on the island side. This “whistle” ping appears in all packet captures and helps us synchronise the events in the two traces.
14. Starts a large iperf3 data transfer from the world to the island side.
15. Starts a series of ICMP ping packets from the special purpose Super Micro server on the world side to a machine on the island. This lets us measure the queue sojourn time at the satellite link input.
16. Waits for the measurement period to conclude.
17. Sends a second “whistle” ping from the special purpose Pi to act as an experiment end marker in all packet captures.
18. Shuts down the various components in the correct order.
19. Converts and retrieves the capture files and iperf3 and ping logs and submits them to other scripts for analysis.
20. Retrieves decoder statistics for coded experiments.
21. Triggers the routine analysis of the captures.
22. Writes the results to the log file.

In these tasks, **`run-exp.sh`** is supported by around 40 other scripts, which take care of the individual jobs, such as starting or shutting down individual experiment components. These scripts in turn have in most cases counterparts on the local machines that carry out these tasks.

We have further written scripts to:

- Automatically configure latencies and jitters on the “world” servers, pushed from the command and control machine. This goes part-way towards **Objective 4** (*Automate the configuration of the "world Internet" on the simulator*)
- Start up and shut down an open source PEP solution (PEPsal) on either side of the link.
- Analyse the capture files for total throughput, goodput, TCP payload data loss, quality (presence of servers and clients), iperf download times and goodput.

- Analyse the capture files for TCP flows and produce flow statistics to assist us in quality control and give us an idea of the kind of traffic that develops on the simulated link: Total number of flows, average number of concurrent flows (which should increase with and be limited by the number of channels), average flow size, average number of packets per flow, average time between packets in a flow.
- Analyse the ping logs for round-trip-time (RTT) latency. This latency is the sum of satellite link latency in both directions, the latency on the world server involved (both known since they are part of our configuration), and the queue sojourn time - the observable of interest here since it tells us how full the input queue to the satellite link was at the time each ping packet arrived at the queue.
- Analyse the capture files for throughput and goodput trends over time.
- Summarise and automatically assess result files for multiple experiment runs.
- Automatically generate plots of throughput, goodput and ping RTT.

Uncoded baseline simulations

These started in September 2016, with the first month or so of output taken to bring the results up to our quality control standards. To date, we have run many hundreds of these, for all eight exemplary links and for a range of queue capacities and channel numbers. This has allowed us to focus on “sweet spot” regions representing workable compromises, i.e., queue capacities and demand levels that

- allow TCP transfers (as measured by `iperf3`) to complete within reasonable time frames
- cause as little queue oscillation as possible
- achieving good link utilisation
- do not lead to the formation of significant standing queues

Initially, we ran only a single experiment for each combination of queue capacity and demand levels, with each experiment taking around 20 minutes as discussed. A significant portion of these needed repeating in the first few months as a result of some of the aforementioned problems that our quality assurance flagged.

We then focussed on the sweet spot regions and ran additional baselines with parameters in these regions. For some of the best parameters, we ran multiple experiments with identical parameters to obtain more representative samples. To an extent, the uncoded baselines are not yet complete as we continue to run / repeat additional baseline samples with parameters of interest in order to improve the statistical reliability of our existing observations: The random selection of servers and flows from the distribution causes significant spread especially for the lower capacity GEO link scenarios.

Sweet spot regions

The sweet spot regions allow us to select suitable queue sizes for further experimentation with coded traffic and guide us to the load levels at which “the going gets tough”. Here is a (necessarily slightly fuzzy) description of our findings to date:

- For GEO links, there is really no amount of queue capacity that is truly acceptable in terms of queue sojourn time as voice communication should not exceed 300 ms one-way latency and the satellite link itself takes up 250 ms of this, with many if not most terrestrial onward links consuming the rest. Assuming somewhat arbitrarily that an additional 100 ms of queueing delay are “marginally acceptable”, we observe that:
    - At 8 Mbps, 100 ms allows us to have queues of up to 100kB capacity. Here, the sweet zone is at a load of between 20 and 30 client channels. At 20 channels, utilisation is just over 50% and a 20 MB download takes around 40 seconds with very few queue overflows and no sign of standing queues. At 30 client channels, utilisation is just over 70%. The download time has shot up to 100 s, and we see evidence of mild oscillation during the download and some formation of standing queues during the download (which is one of the longest, if not the longest flow one would expect in such a scenario in a 10 minute time frame).

  - At 16 Mbps, the same argument allows us queues of up to 200kB. The sweet zone here is in the range of 100kB to 150kB with loads around the 50 to 60 channel mark. We chose 120kB for our further experiments.
  - At 32 Mbps, queue capacities around 250 kB seem to work best with loads of between 100 and 150 channels.
  - At 64 Mbps, queue capacities around 600 kB with loads of around 250 kB worked best.
- For MEO links, we have residual one-way latencies upwards of about 60 ms, meaning that we could, in principle, afford up to another 240 ms in queue sojourn time before exceeding the 300 ms maximum delay for voice. Note that a single channel represents a higher load here than in the GEO case: Firstly, the connection establishment phase at the beginning of each connection puts no load on the link because the data does not start flowing until it is complete. In the GEO case, this phase is over four times longer than for MEO links, so channels on MEO links spend more of their time actually transferring data. The lower round-trip time also means that the TCP congestion windows can grow much faster, so the transfers spend less time in TCP "slow-start" mode. Overall, this means that each channel can complete many more connections on MEO during the same time period. Alas, our investigations show that…
- 32 Mbps: A 200 kB queue supports 50 client channels with no queue oscillation and virtually no standing queues, an 80 MB transfer took around 90 seconds. Larger queue capacities lead to standing queues, and smaller capacities to queue oscillation.
- 64 Mbps: 400 kB queues seem to work best with loads up to around 100 channels.
- 160 Mbps: Queue capacities around 1 MB can support up to around 180 channels.
- 320 Mbps: Queue capacities around 2 MB seem to work for up to 300 channels.

We still need to re-examine the 160 Mbps and 320 Mbps MEO baselines here to verify that simulator constraints were not an issue. Because of the large amounts of packet capture data that accrue in these experiments, the duration was cut to a point where the iperf3 transfer had to be constrained in volume to be able to complete, and may at this stage not be making its way out of slow start mode anymore. We are now able to address this question with the additional hardware of the upgraded simulator and will do so in due course.

Caveat: With only one experiment for most combinations, and significant steps between parameters investigated, the above figures should be taken as an indicator of recommended magnitude only as they reflect combinations investigated. Capacities deviating up to +/-50% from those stated may work equally well if not better in some cases, and some of the given capacities may support slightly higher loads as well. In practice, load cannot really be controlled and link queue configuration will generally base itself on an estimate of demand anyway.

Coded baseline simulations

We started a few coded baseline simulations for a 16 Mbps GEO link in October 2016, using the existing network coding software version that had given us such good gains during the island deployments. In doing so, we used the same **`run-exp.sh`** script that we developed for the uncoded runs, and sandwiched it between two dedicated scripts that configures and start the network-coded tunnel on either side of the satellite link. These scripts in turn invoke local helper scripts on the encoder and decoder, completing our deliverables for **Objectives 2** and **3**.

These simulations provided less overall goodput than their uncoded counterparts and downloads also took a bit longer, regardless of code configuration.

Investigation showed that all coded packets were maximal size (1500 bytes IP), even if the packets they were coding were much smaller. In our island deployment, we had only coded large downloads, where almost all packets were maximal size anyway. Now, we were coding small packets as well as large ones, and since network coding generates at least one coded packet for each uncoded one (and then some), coding small packets as full size packets led to a significant increase in overall coded traffic volume. This was clearly not helping, so we contacted Péter Vingelmann at Steinwurf, who declared it a bug and fixed it within days.

This improved results significantly, but still did not represent an improvement over uncoded communication, and we needed to figure out what held the coding back.

Remember that the basic idea behind coding is a trade-off: From a certain load level onwards, uncoded traffic causes queue oscillation, which slows down large TCP transfers and leaves us with spare capacity on the link. Coding attempts to use some of this spare capacity to add protective overhead that will compensate for the packet loss of the oscillation, which in turn will allow large transfers to maintain higher goodput rates. That is, the spare capacity before coding must accommodate both the protective overhead *and* any gains in goodput. The less overhead we can get away with, the more spare capacity there will be for gain, and vice versa.

For example, if we have uncoded throughput of 10 Mbps on a 16 Mbps link, and need to configure 30% overhead in order to get higher throughput, then our final throughput (less overhead) will be capped at 16 Mbps / 130% = 12.3 Mbps. At 50% overhead, any gains would be below 660 kbps.

This was one part of the insight: We needed to keep unnecessary overhead down.

The second part was once again that we were now no longer dealing just with coding long transfers with large packets, but to a good extent with short transfers, many of whom completed within less that one RTT. Transfers of this duration do not really benefit much from coding: The ACKs for their first packets normally arrive back at the sender after the last data packet has already left there. Even if a packet or two are lost at first transmission, the sender just retransmits them, but their loss does not really have much effect on the sender's future packet transmission rates because there are simply no future packets to transmit as part of the transfer. Coding packets from such transfers represents a cost without benefit: Unnecessary overhead.

The third part of the insight was that short transfers very often meant short packets. Since each coded packet carries a fixed size coding header, the relative cost of encoding short packets such as SYNs, ACKs, or FINs is still substantial, even if we do not inflate them to full size. Short packets are also less likely to be rejected at the tail of a byte queue: When the remaining queue capacity is too small to accommodate another full-size (1500 byte) packet, there is usually still space for small packets. E.g., our small (84 byte) ping packets that we send during our experiments in order to measure queue sojourn time are practically never lost.

The fourth part of the insight came from the predecessor project in the islands: When coding our long transfers on a 16 Mbps link, we coded 30 packets of our long transfer at a time (called a *generation*) and had to supply up to 15 coded "spares" for each such generation in order to compensate for the packet losses the flow sustained during queue overflows. That meant that up to 15 packets from some generations were lost in overflow events. However, in the islands, the coded transfer had only been one of many packet flows that used the link at the time of overflow, and the other flows lost packets as well. Now that we were coding all flows, we needed to ensure that we covered these losses as well. That meant either more relative overhead - not a good option as we already knew - or larger generation sizes. The latter is something the kernel module cannot do due to limitations in the kernel data structures it uses. Note that larger generations also mean that overhead cannot be used for error correction until almost all of the generation and some overhead have been received at the decoder - this time delay needs to be below that with which TCP itself can retransmit. This also places a limit on the largest possible generation size.

Using loss data from the uncoded baseline experiments, we looked at the sort of generation size and overhead that would be required in order to meet both of these constraints and still give better throughput on a 16 Mbps link. This yielded generation sizes in the several hundred - well beyond the 60 packets the software can currently pack into a generation.

We discussed these issues with Steinwurf and, with a number of budget savings, were able to commission a number of modifications to the software:

- Addition of an interleaver: Rather than adding successive incoming packets to a single generation, the software now adds them to a series of generations in a round-robin fashion. This is a well-known technique used in burst error correction, which has been used in consumer electronics since the days of

the compact disc (CD). In our case, it helps us to circumvent the first of our constraints on generation size, as each of the generations can contain up to 60 original packets. This still does not get us around the second constraint, but still allows us to correct significantly larger packet loss bursts than before.
- A size threshold for packets below which we do not encode. This avoids spending unnecessary and inefficient overhead on traffic with a low risk of packet loss.
- An ability to code slightly larger packets as a single packet without the need to fragment.
- Modifications to the coding header.
- An ability to defer overhead by one generation. Overhead packets were sent immediately after the coded original packets. If these were transmitted into a queue that had just started overflowing, we lost the overhead as well.

We took delivery of the last of these modifications just before the interim report date and saw modest improvements in TCP transfer rates. However, further experiments showed that we were still losing a large portion of our overhead packets – the one-generation deferral only represented a relatively small mitigation.

PEP experiments

We have also run a number of performance-enhancing proxy baselines for the 120 kB / 16 Mbps case using the open source PEP software PEPsal, a fully connection-breaking PEP. This resulted in significantly faster TCP transfers over uncoded baseline - even better than our coded baselines. However, overall goodput dropped by around 2%.

We intend to repeat these experiments in due course with a tool other than iperf3: Currently, we send the TCP data from an iperf3 client on the world side to an iperf3 server on the island side of the simulator. This is a little problematic as the iperf3 rate reporting uses information at the application layer, i.e., the server sends regular reports back to the client, where we capture the final report the client receives at the time it completes its transmission. However, the client determines this point in time based on its own view of the connection. In the case of a connection-breaking PEP, the client considers the job done once it has offloaded all data to the PEP - which is a comparatively quick job given the smaller latency to the world--side PEP and the large network capacity on this part of the path. Little does the client know that the PEP still has much of the data in cache, trying to squeeze it through the satellite bottleneck!

Investigation of codes other than network codes

This is part of **Objective 5** (*Prepare for automated experiment runs in different coded configurations and combinations*), which resulted from a visit in January 2016 by Martin Bossert from the University of Ulm, who suggested the use of Partial Unit Memory (PUM) codes as an alternative. I paid him two extended counter-visits during my research and study leave, in October and December 2016, respectively. He introduced me to his very capable PhD student Sven Puchinger, who simulated the performance of various coding schemes, including a number of PUM codes, based on loss data from the uncoded baseline experiments discussed above. This also resulted in a novel tweak to PUM codes. Our results (entirely theoretical at this point) give the new PUM codes a leg up. A paper summarising our results [17] was presented by the primary investigator at the IEEE International Symposium on Information Theory (ISIT) 2017 in late June.

Impact up to interim report time

In our interim report, we were reported that our work had already had a number of beneficial impacts for us. As the simulator's hardware extent grew, we became progressively concerned about its accommodation in a student lab without after-hours air-conditioning as well as about its power load on the sockets there.

We successfully lobbied for a dedicated network lab, with strong support from both Brian Carpenter and Nevil Brownlee (who did a lot of the lobbying himself while I was on sabbatical). In January 2017, the simulator moved to its new home, now amply supplied with power, and with capacity to grow. Our faculty facilities manager applied for minor capital works approval to partition off the area with the simulator racks and fit it with its own

dedicated heat pump air conditioner to support and supplement the business hours-only building cooled air supply over weekends.

We still can't wait to see this happen as Lei often works on weekends where temperatures in the room soar into the high 20s. Unfortunately, this enhancement has only recently been progressed to a proper planning stage, as the discovery of asbestos during the renovation of another building caused workload issues at committee level and staffing shortages at faculty level.

In a separate development, our Head of Department, Prof. Robert Amor, negotiated a significant increase in departmental CAPEX, and we were able to enhance the simulator hardware considerably beyond what was envisaged when we applied for the grant we are reporting on here. We have already described the hardware impact of this in our Platform section at the beginning of this narrative and will describe the flow-on effects on our experiments in the next section below.

The simulator is also proving to be attractive to students. We now have Fuli, a talented Samoan MSc student working on a PEP that does not fully break connection, and I am still in discussions with a number of potential PhD students. Sadly, money is proving an obstacle for many applicants – full scholarships are worth around NZ$35,000 annually, very hard to get, and even those lucky enough to get them often still need savings, family support, or external jobs in order to survive in Auckland. Courtesy of a residual balance in one of Brian Carpenter's research accounts, we now have small amounts of up to $5000 p.a. available, however this is still only a small drop into a very big bucket.

Simulator upgrade

As a result of the aforementioned CAPEX injection, we acquired additional hardware in June/July 2017. In addition to the machines mentioned at the beginning of this narrative, we also acquired an extra rack to accommodate them, as well as a number of switches and a large number of cables. Most of the equipment was ably configured and integrated into the simulator by Lei and a very capable secondary school intern volunteer, Liam Scott-Russell, who sacrificed a good part of his July school holidays for this unpaid activity.

Following the hardware upgrade, we also needed to reconfigure the simulator network itself: The separation of PEP and coding onto dedicated machines required two further networks along the satellite chain that connect the PEPs and coding machines on either side of the simulated satellite link. We also needed to integrate the interfaces on the newly inserted copper taps: Each of these interfaces has a nominal IP address, even though it only listens to traffic passing through the tap.

We then set upon upgrading our experiment scripts. This became necessary for several reasons: Firstly, the extra hardware meant that some functions had moved to different machines, e.g., the captures, the ping series that let us measure queue sojourn time, and the large iperf transfer all moved to the new dedicated special purpose server.

Secondly, the scripts used in our initial setup did not incorporate a lot of error handling, meaning that batch experiments with faulty setups could continue to run for considerable amounts of time until the problem was picked up during quality control.

Now, almost all scripts return an exit code indicating whether they succeeded, many already test whether the system they are about to run on is in the right state (e.g., there isn't already a stale process from a previous interrupted experiment running), and most functionalities now use both a dedicated script on the command and control machine, which interacts with local helper scripts on the target machines. This allows us to keep the code in the main experiment script more compact and less cluttered with detail.

In total, we created or modified 59 new shell scripts for the command and control machine as part of the upgrade, eight scripts on each world server, another five on the special purpose Super Micro, as well as a couple on the special purpose Pis, and three scripts on each capture machine that allows us to start, stop, and interrupt captures there. We also modified a further 3 scripts used in experiment analysis. Among others, we

now compute iperf goodput directly based on the trace files as well, which gives us accurate results even for PEPs.

One issue that raised its head early in the upgrade was that the distribution of the functionalities across more machines meant that we usually had to have a larger number of terminal windows open to monitor and control what was happening on the important machines. Typically, this includes 2-3 terminals for the command and control machine (one to edit **`run-exp.sh`** or a supporting script, one to issue commands, and one to look up data), a window for each of the five machines on the satellite chain, one for each of the two capture machines, one on a world server, one on the special purpose server, one of the special purpose Pi on the world side, and one on a machine on the island side. Switching between these windows became increasingly cumbersome with the existing screen real estate both on the command and control machine as well as on the primary investigator's laptop. We therefore used leftover funds (with ISIF secretariat permission) to install additional screens, which has made working with the simulator a lot easier.

Terrestrial latencies

Last but not least, we also needed to revisit the question of terrestrial latencies. These are the delays that packets encounter between the end host and the satellite gateway. On most island ends, these are negligible owing to the small geographical size of most islands. However, on the world side, these can be quite considerable. In our original simulator setup, we had simply taken an educated guess as to what a terrestrial latency distribution might look like and had added corresponding **`netem`** ingress and egress latencies to our servers. After the upgrade, the new servers needed to be configured with appropriate latencies as well, and we took this as an opportunity to move away from pure guess work.

In order to find a realistic distribution, we pinged the off-island IP addresses seen in the Rarotonga TCP flow captures and collected their RTTs from Auckland. We then sorted this distribution, partitioned it by the number of world servers available to us (22 for regular experiments) and configured each server with the median RTT from the respective partition. Over the summer period, we have an intern lined up to transfer this technique to multiple satellite sites and to look into the possibility of implementing the delay on the OpenFlow switch that the servers connect to, rather than on the servers themselves.

UDP in traffic

Our experiments to date have used purely TCP in the bulk traffic. However, our **Objective 1** (*Develop a tool that allows for realistic simulation of UDP flows*) was to acquire the capability to run experiments in which a certain part of the bandwidth is taken up by UDP traffic which does not respond to packet loss. We are pleased to report that we now have developed such a tool, called **`udpsender`**. The tool is a C program that takes a text file with a flow size distribution as input, as well as a target byte rate. It then randomly selects as many UDP flows and transmits them as are required to reach the given target rate. The tool operates out of the special purpose Super Micro on the world side of the simulator.

MEO vs. GEO

The question as to which type of system is superior is often asked of us. Our baseline simulations to date on the upgraded simulator with the recommended queue capacities above show that 32 Mbps MEO links yield about 10% more goodput at a modest load of 80 channels than their GEO counterparts, and feature lower loss. They also complete almost twice the total number of flows during the experiment period. However, this benefit accrues almost exclusively to short flows. Our iperf transfers on the simulated MEO link typically achieved an average transfer rate of around 5 Mbps, whereas their GEO counterparts manage an average of 6.3 Mbps.

Work ongoing/planned

At this point, we are repeating a set of uncoded baselines in order to ensure that the upgraded simulator produces results that are comparable to that of the original version. Early indications are that the results are indeed somewhat different and that the new configuration gives us slightly lower overall goodput. This is not unexpected as the new empirical terrestrial latencies are on average a little larger than those we have previously

configured, which results in longer connection establishment times and overall slightly fewer flows with longer slow start phases.

These uncoded baselines will probably keep us busy until around mid-November. We run ten experiments for each configuration in order to get a reasonable sample of flow sizes. Our new scripting and the additional processing capacity of the new capture servers and the new, more powerful command & control machine have reduced the time for a single experiment from around 20 minutes to around 16 minutes, or just over 2 ½ hours for each combination of parameters. For each of our eight base scenarios, we run around a dozen parameter combinations just to cover the uncoded baselines. Additional time is spent on quality control and on repeating experiments for which there were errors, e.g., because of equipment outages.

At this point, we are then planning to proceed roughly in the following order:

1. Run coded MEO experiments, where the overhead timing is less critical.
2. Attempt to run PEPsal over a network-coded tunnel (we think this may be possible using a single machine at either end, but we will not know until we try).
3. Extend our coded/PEPsal/combined baselines to the remaining GEO and MEO cases (except for 160 and 320 Mbps MEO, which we would like to tackle once we have more hardware and air-conditioning in place)
4. Introduce UDP into the traffic mix (with the tool created under **Objective 1**) and compare results for different hypothetical sat gate locations.
5. Return to the 16 Mbps GEO coded case with (hopefully) upgraded coding software (we have applied to ISIF again for funding to finance this).
6. We also anticipate that another round of field trials in the Pacific might be useful after this has been completed.

Somewhere on this list will also be a downtime owing to building work: We hope that the simulator racks will soon be surrounded by walls and be given their own dedicated 24/7 air conditioning. This is likely to cause a few weeks' worth of disruption.

Relationships

Much of the first part of this project coincided with the principal investigator's research and study leave, which gave an opportunity to build and renew ties with the international research community. Highlights included:

- A three-week visit to the Massachusetts Institute of Technology hosted by Muriel Médard, whose work inspired our project. This provided opportunity to interact with her graduate students and postdocs, as well as a former postdoc working on commercial solutions in the network coding field. A paper we wrote on possible topologies for network coding over satellite links was presented at the Ninth International Conference on Advances in Satellite and Space Communications (SPACOMM 2017) in Venice in late April [18] and won a Best Paper Award there, see: https://www.iaria.org/conferences2017/AwardsSPACOMM17.html
- A three-week visit to Steinwurf ApS in Aalborg, Denmark. This gave us a chance to deepen the understanding of our project at their end and build a relationship that will hopefully further extend the enormous amount of goodwill that Steinwurf have shown.
- Two three-week visits at the University of Ulm, Germany, which deepened an existing relationship and resulted in the aforementioned ISIT paper on PUM codes.
- The primary investigator also gave talks on the project at the University of Victoria, BC, Notre Dame University, the University of Toronto and at the University of Tokyo.

We have also been able to raise our profile within our department and faculty, which has resulted in significant additional CAPEX support (around NZ$50,000 in new CAPEX).

We have also published blog contributions on the APNIC and ISIF blogs and kept stakeholders informed of progress via e-mail. Together with the graduation of 'Etuate Cocker (who worked with me on the previous project) in September 2017 as the (to the best of our knowledge) first Tongan with a PhD in Computer Science, the project has also raised our visibility in the Pacific Island community. As a result, we now have Fuli, a Samoan Master's student, in our group, and are still hopeful that a potential PhD candidate from Kiribati may be able to join us.

Lessons learned

A lot, both technical and conceptual. At a technical level, there were a large number of challenges to deal with, including:

- When developing the client and server software, we discovered that POSIX-based operating systems (i.e., practically all systems in common use today) cannot sustain solid data transfer on more than ~20 parallel sockets if each these gets renewed and used again for a new connection each time a TCP connection finishes. We have traced this to the policy by which socket file descriptors are allocated (lowest first) and to the priority with which read/write operations on file descriptors are being performed (also starting from lowest first in each time slice). This causes starvation for any additional connections. Note that this is not the same as saying that a machine cannot support more than 20 sockets – the problem only occurs when all of these sockets are being saturated with data.
- Making **`netem`** delays and token bucket filters work properly with each other to simulate the bandwidth constraints and latency of a satellite link is not straightforward. We found the all recipes for this that we could find on the Internet would work fine when bandwidth and latency were tested separately, but crumbled as soon as we tested latency under load. Solved with the help of intermediate function block devices.
- Network devices / NICs with IP fragmentation and TCP segmentation offload wreak havoc on packet streams – even when nothing gets dropped, packets still get split and recombined along the way, and capture software doesn't necessarily refrain from combining successive TCP packets either. Trying to ascertain packet loss in this way is an exercise in futility – this is why we now look at payload data loss instead.
- Coding is not merely a matter of selecting the right generation size and the number of coded overhead packets to send. The timing of the overhead is also important. The coded packets that are sent before the overhead only encapsulate incoming packets (these are called "systematically coded packets"). Their rate is in the same order of magnitude as the rate of the satellite link. E.g., for a 16 Mbps link, incoming packets will typically arrive at a rate of at most a few Mbps. The extra coded (overhead) packets can be generated and sent the moment the last systematically coded packet has left the encoder. However, the can leave the encoder at the full rate of the link between encoder and satellite gateway – typically 1 Gbps. As the overhead is only really needed at time of queue overflow, overhead sent immediately slams right into the queue as it is overflowing and gets dropped as well. Similarly, overhead that is sent at full rate can actually make the queue overflow, because it is a lot of data arriving in a very short time frame. On the other hand, if we send the overhead too late, TCP may lose patience, accept the losses and send a retransmissions – and the whole purpose of coding is to prevent TCP from doing just that.
- Making a nice graphics card work with your monitor isn't always a driver issue. Sometimes, it's the cable.
- Combining multiple monitors to form a single screen under Ubuntu requires an X extension called Xinerama. This is incompatible with the Composite extension, which could be disabled in earlier versions of Ubuntu. Now it's impossible to log into Ubuntu's Unity desktop if it is disabled, and when it is enabled, the machine will crash. Solution: Use another desktop that doesn't need Composite, e.g., xfce4 in our case.

- iperf reports on connection-breaking PEPs are grossly inaccurate. Once an iperf client has dumped its data onto its closest PEP, it declares the transfer to be over – even if a big part of the data has not made it across the satellite link yet.
- When deploying PEPsal, the MTU of the interface facing the satellite link must be set to the smallest MTU encountered along the satellite chain. The PEP will send packets marked DF (Don't Fragment) as if engaging in Path MTU Discovery, but if the satellite chain MTU is lower than its own MTU, will not capture any `ICMP 590 Destination unreachable (Fragmentation needed)` packets returned by a router with lower MTU along the chain. These end up at the original TCP sender, which can't really do anything about it!

Indicators

Indicators	Baseline	Progress assessment	Course of action
UDP simulation tool developed (Objective 1)	User Datagram Protocol (UDP) traffic is a non-negligible part of Internet traffic. We know how much arrives at the island end but because we know that some of the traffic is lost during queue overflows, we do not know how much we need to send in order to get a certain receive rate.	The naïve approach to rate control in this context would be to have a sender on the "world" side that receives feedback from the "island" side. Having assessed the total amount of data loss as a result of queue oscillation, we now regard hitting a precise rate as being of secondary importance. What was more important, however, was to get a UDP flow size mix supplied that was realistic – this determines queue behaviour. We developed a tool named **`udpsender`** that uses a configurable flow size distribution and a given target rate in order to deliver UDP from the world to the island side.	The tool has been developed and tested, however we have not had an opportunity to use it alongside the TCP traffic on the simulator just yet as the latter has just been recommissioned after its upgrade. We will start using this tool once our current TCP-only experiments are complete.
Experiment control scripts developed (Objectives 2 and 3).	A large number of individual components of the simulator needed to be started and stopped manually, log files needed to be copied manually, and routine analysis and quality assurance tasks needed to be triggered manually.	We have developed a central experiment control script in **`bash`** called **`run-exp.sh`** which runs on the command and control machine and takes care of the tasks common to all experiments. **`run-exp.sh`** uses a number of other scripts with more specialised tasks such as server start-up and takedown, capture start and stop, signalling, ping and iperf measurements, client load balancing, routine result analysis, system testing and quality assurance. We have also developed scripts for link configuration, network coding encoder/decoder and PEP configuration as well as terrestrial latency configuration. These are invoked as and when needed (i.e., not necessarily for every experiment run). Almost all scripts now check for and return exit codes, allowing anomalies to be caught quickly. Together with **`run-exp.sh`**, we invoke these scripts as parts of small, experiment batch-specific scripts, which typically take only a few minutes to write.	Objectives 2 and 3 have been achieved. As with any software, the scripts developed are subject to regular maintenance and extension as we add features.

Indicators	Baseline	Progress assessment	Course of action
Capability to simulate realistic "world Internet" latency distributions between the "world" satellite gateway and the "world servers" in the simulator (Objective 4).	World gateway and world servers are connected directly via a dumb switch with negligible latency.	We now have two OpenFlow capable switches interfacing with the world servers (a single switch would have been too small for the upgraded simulator). We have also empirically determined a latency distribution for an Auckland-based satellite gateway. At present, we are still simulating the delay on the world servers themselves, however we are planning to transfer this functionality to the switch in due course.	Objective 4 has been achieved in terms of capability but not in the envisaged form yet.
Ability to run automated experiments using coded configurations using different codes (Objective 5, nice to have)	At the start of the project, only one sort of code was available: A systematic random linear network code whose generation size and overhead amount could be configured. As we were mostly interested in its ability to correct packet erasures, the question arose as to whether other error-correcting codes could work as well or better. We also wanted to look at constraints imposed by the implementation of the existing code.	Progress has been made in two respects: Firstly, the existing software by Steinwurf has been extensively modified such that there are now three more controllable parameters: an interleaver size that gets us around the generation size limit, a packet size cutoff threshold that allows us to skip small packets that we cannot code efficiently, and a switch that lets us defer overhead by a short amount of time. However, the code here is still a random linear network code. We are also looking for funding to enable us to implement a longer delay here and make other changes to the coding. Secondly, we have used uncoded data from the simulator to look at the feasibility of using another family of codes, the Partial Unit Memory (PUM) codes. At this point, we have found that they hold some promise. A paper to this effect was presented at IEEE ISIT2017 in Aachen, Germany, at the end of June.	Objective 5 has been partially achieved and we intend to perform further research in this direction.

Project implementation

Project activities	Input	Outputs	Timeline	Status
Simulator hardware assembly	PhD student, 17 Super Micro servers, 10 Intel NUCs, and 86 Raspberry Pis, two inherited 19" 7ft equipment racks, small parts and cabling.	Simulator hardware assembly	Pre-project in 2015 and 2016	Completed.
Simulator equipment setup	PhD student, simulator hardware assembly	Machines in the simulator configured with operating systems and any required open source helper applications and tools.	Pre-project in 2015 and 2016	Completed.
Development of simulator client and server software	Principal Investigator with PhD student, a selection of Super Micro servers, Intel NUCs, and Raspberry Pis from the simulator.	Simulator client and server software capable of providing calibrated traffic levels with TCP flow sizes following configurable distributions.	1/3/2016 (pre-project) - 31/10/2016: software development, testing, troubleshooting	Completed (except for follow-up with Linux kernel team and completion of a draft paper on an issue discovered)
Configuration and troubleshooting of satellite emulation (bandwidth constraint, latency, input queue size)	Principal Investigator with PhD student, full simulator hardware	A script for automated setup of the satellite emulation.	1/3/2016 (pre-project) - 30/11/2016: configuration and (a lot of) troubleshooting and testing	Completed.
Development of a client load balancer	Principal Investigator with PhD student, full simulator hardware, client software	A script that distributes the requested number of client channels across all physical clients.	1/6/2016 (pre-project) - 31/10/2016	Completed.
Development of core experiment script **`run-exp.sh`**	PhD student with Principal Investigator, full simulator hardware, client load balancer, client and server software, analysis and quality assurance scripts at a later stage.	A script that automatically runs an experiment and starts and stops all processes involved on the various machines across the simulator.	1/9/2016 (pre-project) - 30/12/2016 Various versions were used progressively from early September.	Completed.
Uncoded baseline experiments	PhD student with Principal Investigator, full simulator hardware, client load balancer, client and server software, progressive versions of **`run-exp.sh`**, analysis and quality assurance scripts at a later stage.	Huge amounts of data, which gave us a coarse idea of which queue capacities to use on which sort of satellite link.	1/9/2016 (pre-project) - present	Completed, except for a reference run with the upgraded simulator and the new terrestrial latency distribution.

isif asia

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Project activities	Input	Outputs	Timeline	Status
Development of a main analysis and quality control script.	Principal Investigator with PhD student, full simulator hardware and software, other scripts	An extensive analysis and quality control script. used by `run-exp.sh`. This script produces information we want to see for every run: throughput, goodput, loss, download time/rate, flow information, quality control information (did all clients and servers participate for the whole experiment and carry their fair share of the load?)	1/10/2016 - ongoing	Various versions of the script have been in use since last October, but we continue to extend and improve functionality.
Development of a script that sets up configurable delays and jitters on the world servers.	Principal Investigator, full simulator hardware	A script that sets up configurable delays and jitters on the world servers.	1/11/2016 - 30/11/2016	Completed. Transition to an OpenFlow version is being planned.
Development of a script that checks whether all network interfaces in the simulator are reachable and respond in the right time frame.	Principal Investigator, full simulator hardware	A script that pings all machines via the various networks in the simulator and checks whether their response times are within the expected range. Used by `run-exp.sh`.	1/11/2016 - 30/11/2016	Completed.
Development of scripts to assist with the coding	Principal Investigator with PhD student, full simulator hardware and software.	A script running on the encoder and decoder machines to configure and start the coded tunnel, another script to take it down, and a script on the main command and control machine that orchestrates these scripts on both encoders and decoders.	1/11/2016 - 31/1/2017	Completed in various stages reflecting coding software updates from Steinwurf.
Coded baseline experiments	PhD student with principal investigator, full simulator hardware and software, existing results.	More data, yielding insight into which code parameter combinations work and which ones do not, insights that informed changes to the software.	1/11/2016 - ongoing	Ongoing
Development of offline analysis scripts	PhD student with principal investigator, full simulator hardware and software, existing results.	We have developed a variety of analysis scripts that allow us to look deeper into the data we capture, analyse and compute additional observables and visualise them. These scripts are mostly intended to allow offline processing of measurement data for types of analysis that are time-consuming and that we do not need to perform for every measurement. E.g., we can compute instantaneous throughput or goodput or visualise queue sojourn time.	1/11/2016 - ongoing	Ongoing

isif asia

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Project activities	Input	Outputs	Timeline	Status
Configuration of PEPsal and development of a configuration script for the simulator	PhD student with principal investigator, full simulator hardware and software, existing scripts packaged with PEPsal.	Scripts to configure and run PEPsal at both ends of the link, and to take it down after an experiment. Scripts for the central command and control machine to allow remote activation / deactivation of PEPsal.	February 2017	Completed
PEP baseline experiments	PhD student with principal investigator, full simulator hardware and software.	Even more data, yielding insights into the comparative performance of PEPs in our scenarios.	16/2/2017 - ongoing	Ongoing (queued for repeat in new simulator setup)
Simulator hardware upgrade	PhD student with principal investigator and secondary student intern. From department CAPEX: a 19"/7ft rack, 15 additional SuperMicro servers, 12 additional Raspberry Pis, 2 HP ProCurve switches, 4 copper taps, lots of cables and small parts. From the grant: six additional screens, two docking stations for laptops, and an additional 2 HP Pro Curve switches. Also: 1 copper tap inherited from another project.	Upgraded simulator with the capability to simulate over 4000 island clients. The simulator is now also easier to use thanks to additional screen real estate.	1/7/2017 – 6/10/2017	Completed
Simulator script upgrade	PhD student with principal investigator, full simulator hardware and base software (OS and applications) after upgrade.	A large number of new and upgraded scripts to control the running of experiments and the analysis of results, with improved error handling. Includes upgraded configuration of PEPs and coded tunnel endpoints	1/7/2017 – 12/10/2017	Completed to the point where we can run batch experiments again
Uncoded baselines on upgraded simulator	PhD student with principal investigator, full simulator hardware and software	Data on throughput, goodput, queue sojourn time, data loss, "large TCP transfer" rates, number of parallel TCP flows as well as satellite chain traces for further investigation. Ten such experiments are performed for each combination of link type and load level. We investigate around 10-12 load levels for each link type.	13/10/2017-approx. end of 10/2017	Ongoing
Coded baselines on upgraded simulator	PhD student with principal investigator, full simulator hardware and software	We intend to run a small number of baselines here in order to get a better understanding of the timing problems seen on the old simulator and how serious they are in the context of various link types (GEO/MEO/bandwidth). More coded baselines will follow once we have a software version available that allows us to delay coded overhead packets.	November 2017	Scheduled

Communication and dissemination

From the list of outputs above, please specify what dissemination efforts were made with special attention to those intending to reach target groups by gender, age, ethnic and socio-economic profiles to impact marginalized and disadvantaged groups. If your organization has a communication strategy for this project, please describe how it is implemented, funded and measured.

The start of the project coincided with the principal investigator's research and study leave. I used this opportunity to give talks about the project at:

- 21/7/2016: Simon Fraser University, Burnaby, B.C, Canada
- 4/8/2016: University of Victoria, B.C., Canada
- 23/8/2016: Notre Dame University, South Bend, Illinois
- 31/8/2016: University of Toronto
- 23/9/2016: Massachusetts Institute of Technology, Cambridge, USA
- 28/1/2017: University of Tokyo

I also gave personal introductions to the project to individual researchers at the University of Hawaii at Manoa, at the Cooperative Association for Internet Data Analysis (CAIDA at the San Diego Supercomputer Center at UCSD), at the University of Ulm and the University of Duisburg-Essen in Germany, and at Steinwurf ApS in Aalborg, Denmark.

Since, I have presented on the project at ICCNA2017 in Kota Kinabalu, Sabah, and at APNIC44 in Taichung. A further presentation at NetHui in Auckland in November is planned.

We have also had two papers accepted into international conferences already:

- Speidel, U., Cocker, E'., Médard, M., Vingelmann, P., Heide, J., *Topologies for the Provision of Network-Coded Services via Shared Satellite Channels*, to be presented at the Ninth International Conference on Advances in Satellite and Space Communications (SPACOMM 2017), April 23 - 27, 2017 - Venice, Italy. [18] This paper was subsequently announced as the winner of one of two best paper awards at the conference (see https://www.iaria.org/conferences2017/AwardsSPACOMM17.html).
- Speidel, U., Puchinger, S., Bossert, M., *Constraints for coded tunnels across long latency bottlenecks with ARQ-based congestion control*, was presented at the IEEE International Symposium on Information Theory (ISIT2017), June 25-30, 2017, Aachen, Germany, and is available on IEEExplore (http://ieeexplore.ieee.org/document/8006532/). [17]

We also have two journal articles in the pipeline:

- An extended version of our SPACOMM paper was submitted to an invitation-only edition of the International Journal on Advances in Telecommunications (v 10 n 3&4 2017, to appear in December 2017).
- We also submitted a further article to the IEEE Journal on Selected Areas in Communication in July, for a special issue on satellite communication.

We have also contributed to the ISIF Asia Discover blog [20], with a condensed version appearing on the APNIC technical blog [19]. We also use our own blog (http://sde.blogs.auckland.ac.nz/) and keep other key stakeholders personally informed via e-mail.

Project Management and Sustainability

As a university research team, we were very fortunate to have much of the financial side of the project handled for us by our Faculty of Science accounting staff, Staff Service Centre and the Research Office. This includes oversight on how much we spent and what we spent it on - all our project expenditure is subject to approval by our Head of Department, for example. On this side, the issue of sustainability does not arise. Having interfaced with APNIC/ISIF before has left a footprint in the institutional knowledge, however, which makes it easier for us to make this project work on the financial side (and we have gained the impression that the same applies at the APNIC/ISIF end).

The support by APNIC/ISIF Asia has raised our internal visibility. The project has over the years grown from a student desk based one to a facility that now has its own laboratory with equipment racks and a number of student desks, plus (very important but often overlooked) plenty of whiteboard space. At present, the racks and the equipment still share the same airspace (and the same air), but we are looking forward to having the equipment racks partitioned off and given their own air-conditioning, hopefully over the summer.

We have also been able to benefit from significant additional faculty CAPEX (around NZ$50,000) in 2017, which allowed us to upgrade the simulator and turn it into a research facility that is sustainable over a longer term. Some of the existing CAPEX equipment in the laboratory is also up for renewal in 2018.

Brian Carpenter, former IETF chair and erstwhile professor in our department, has shown close interest in our activities and to his end has made the balance of a research account available for us to pay small top-up scholarships to PhD students over the next ~3 years. We are also noticing more interest in postgraduate opportunities at both Masters and PhD level - the simulator proves to be a significant drawcard here.

We have also sought further funding from ISIF to support us in the development of the coding software, which is guided by what the project has taught us to date.

As such, the project will definitely continue well beyond the end of the grant.

Project Outcomes and Impact

Outcomes: At this point, we have achieved all of the four immediate technical objectives and made good progress towards the nice-to-have fifth objective. We have run many thousands of experiments on the original simulator build and have retained the data for well over two thousand. Each experiment runs a complex sequence of processes. With the new version of the simulator, we can now run an experiment in under 16 minutes in most cases. We would not have been able to do this without the scripts developed under this project.

We have produced two research papers informed by the project [17][18], as well as contributions to the ISIF Discover and APNIC technical blogs, and have given a number of presentations in international venues.

On the scientific side, the project has already yielded significant insight into the dynamic behaviour of TCP over satellite connections, as well as helped us build capability in terms of technology and methodology. Not only have we been able to confirm the effects seen in the field with the simulator, we now also have a much better understanding of the impact of small TCP flows on the dynamics. We have also learned a lot about the necessity to consider overhead timing when coding.

Impact: The most significant immediate impact to date has been our success in negotiating a dedicated network laboratory, and the outlook of being able to grow this with further support from our department. We have also been able to recruit a Samoan Masters student, and now have a facility that is clearly proving to be attractive. We are looking forward to being able to publish more results as the simulator produces them.

Our simulator data has already led to several aspects of the coding software to be re-thought and has led to several revisions of the software. This includes the need to minimise coding-related overhead (coding and encapsulation headers) as well as the timing of the overhead, and the implementation of an interleaver to be able to cope with the longer burst loss sequences seen in all-of-island coding.

We have been able to give recommended queue capacities for a range of uncoded scenarios.

Overall Assessment

We consider the project to have achieved its objectives. We are still not quite as far down the track with our experiments as we had hoped to be, however this delay was caused mainly by the opportunity to improve our facility substantially over the design we envisaged when we were proposed the project to ISIF.

The objectives that are complete have helped us complete thousands of experiments to date, with more being added on a daily basis. The scripting has given us the scalability we needed, but its implementation was more complex than we initially thought, largely due to quirks in the other software that the scripts leverage. On the positive side, the scripts allow us to run large numbers of experiments unattended. This has somewhat compensated us for the fact that individual experiments take a long time to complete, and that multiple experiments with the same configuration are needed to gain reliable statistics about its performance.

At the time of writing, we are producing more results and will publish these as we go.

Recommendations and Use of Findings

At this stage, our recommendations to satellite operators and ISPs (our primary end user group) are:

- TCP causes queue oscillation on satellite links. As load on links increases, they move from a regime without significant packet loss on an underutilised link to a regime with queue oscillation (burst losses and underutilisation) to a congested regime (near-continuous losses, no underutilisation) where only short TCP transfers succeed. Our results show that TCP queue oscillation adequately explains the losses seen at Pacific sites where packet loss is observed along link underutilisation.
- TCP queue oscillation should always be considered alongside rain fading or other physical performance problems as a possible cause of packet loss. Rain fading packet loss correlates with the weather, TCP queue oscillation packet loss correlates with peak use hours – so the difference is easy to tell as long as one looks at statistics over the shorter term.
- Input queue capacity matters. Approximate recommended capacities for GEO and MEO byte queues are given in the narrative section of this report. These are below the conventional recommendation (bandwidth-delay product) but above the more general router buffer sizing recommendation of Appenzeller.
- Queue dimensioning represents a trade-off between TCP queue oscillation risk (queue capacity too small, affecting large downloads) and the risk of standing queues (queue capacity too large, affecting the timing of short flows). Depending on the intended traffic mix, the recommended values should be adjusted upwards or downwards. E.g., a link predominantly used for large downloads should use higher queue capacities.
- In contexts where the queue is a packet queue rather than a byte queue, we recommend at this point that the packet capacity be computed by dividing the recommended byte capacity by the average packet size observed on the link.
- Performance-enhancing proxies assist in maintaining large flows (with the obvious drawbacks of a connection-breaking setup).
- Network coding remains a promising alternative, but we will first need to find an appropriate timing scheme.
- Location of the satellite operator's off-island gateway matters. Our experiments with the upgraded simulator show that its slightly larger terrestrial latencies to the hosts that island users communicate with causes a significant drop in goodput at comparable load levels. ISPs should monitor where most of their traffic originates from and choose their satellite partners carefully to ensure their off-island teleport is close to the source of the bulk of the traffic.
- Our experiments to date show that overall MEO goodput is slightly higher than GEO goodput for the same bandwidth and offered load, and that the shorter RTT allows almost twice the number of flows to complete in the same experiment time. However, this benefit accrues mostly to short flows – the large iperf transfers observed on MEO are actually slower than their GEO counterparts, indicating that the MEO links may be somewhat more susceptible to TCP queue oscillation. We note however that this drawback may be outweighed by the lower cost of bandwidth in MEO links.
- When using PEPsal, note that MTUs along the satellite chain between PEPs should be at least as large as the smallest MTU on the path between hosts either side of the link. This is required to ensure that path MTU discovery works correctly, as the PEPs do not process `ICMP 590 Destination unreachable (Fragmentation needed)` packets correctly.

We are also in a position to make recommendations to others planning to simulate satellite networks:

- Real-life Internet traffic contains a mix of TCP flow sizes, with most flows being small while a large part of the data transferred is part of large flows. On satellite links, small flows tend to complete before TCP flow control has had an opportunity to adjust the congestion window size. These flows thus do not respond to developing congestion. The portion of such flows on a link determines how much capacity remains for large flows with congestion control, and they may crowd large flows out. In simulations, this

needs to be taken into account: Simulations that are exclusively based on multiple parallel large flows that all respond to congestion are not realistic in such scenarios.
- When simulating satellite links with latency and rate constraints, it is important to test these constraints both separately *and jointly*. That is, run an iperf test to confirm capacity while pinging to confirm RTT.
- Ensure that clients and servers are capable of handling the load - not just in terms of total data rates, but also in terms of the number of parallel sockets that can sustain a fair rate.
- Ensure that your data capture method has sufficient buffer space to capture all packets, and be mindful that capture may not happen exactly in correct temporal order and that packets may be "repackaged" by routers along the way, as well as by capture tools.
- Ensure that you also configure realistic terrestrial latencies – the distribution has an impact on results!

Our findings will of course also be used in the continuation of our own experiments with the simulator.

Bibliography

[1] Mauldin, A., "Frequently Asked Questions: Submarine Cables 101", APNIC Blog Guest Post, 22 March 2017, https://blog.apnic.net/2017/03/22/frequently-asked-questions-submarine-cables-101/ (last accessed 13/4/2017)

[2] Cerf, V., Dalal, Y., and Sunshine, C., "Specification of Internet Transmission Control Program", RFC 675, 1974, http://tools.ietf.org/html/rfc675 (last accessed 13/4/2017)

[3] Postel, J. (ed.), "Transmission Control Protocol", RFC 793, 1981, http://tools.ietf.org/html/rfc793 (last accessed 13/4/2017)

[4] O3b Networks home page, http://www.o3bnetworks.com/ (last accessed 13/4/2017)

[5] Allman, M., Paxson, V., Stevens, W., "TCP Congestion Control", RFC 2581, 1999, http://tools.ietf.org/html/rfc2581 (last accessed 13/4/2017)

[6] Allman, M., Paxson, V., Blanton, E., "TCP Congestion Control", RFC 5681, 2009, http://tools.ietf.org/html/rfc5681 (last accessed 13/4/2017)

[7] Jouanigot, J. M. 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, 20-22 April 1993, Sevilla, Spain, CERN CN/93/6.

[8] Floyd, S. and Jacobson, V., "Random Early Detection (RED) gateways for Congestion Avoidance". IEEE/ACM Transactions on Networking. 1 (4): 397–413, August 1993.

[9] Nichols, K. and Jacobson, V.,"Controlling Queue Delay". ACM Queue. ACM Publishing, 6 May 2012.

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

[11] Delannoy, G., "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 (last accessed 13/4/2017)

[12] Border, J. et al., "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC3135, https://tools.ietf.org/html/rfc3135 (last accessed 13/4/2017)

[13] Sundararajan, J. et al., "Network Coding Meets TCP: Theory and Implementation", Proc. IEEE, 99(3), March 2011, pp. 490-512, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5688180 (last accessed 13/4/2017)

[14] Speidel, U., Cocker, 'E., Vingelmann, P., Heide, J. , Médard, M., "Can network coding bridge the digital divide in the Pacfic?", 2015 International Symposium on Network Coding (NetCod 2015), Sydney, 22-24 June, 2015, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7176795 (last accessed 13/4/2017)

[15] Speidel, U., Cocker, 'E., Vingelmann, P., Heide, J. , Médard, M., "Can network coding bridge the digital divide in the Pacfic?", lightning presentation, Asia Pacific regional Internet Governance Forum (APrIGF) 2015, Macau, 30 June - 3 July, 2015

[16] Speidel, U. et al., "Can network coding mitigate TCP-induced queue oscillation on narrowband satellite links?", invited paper, 7th EAI International Conference on Wireless and Satellite Systems (WiSATS 2015), 6-7 July, Bradford, UK

[17] Speidel, U., Puchinger, S., and Bossert, M., "Constraints for coded tunnels across long latency bottlenecks with ARQ-based congestion control", IEEE International Symposium on Information Theory, Aachen, Germany, June 25-30, 2017. http://ieeexplore.ieee.org/document/8006532/

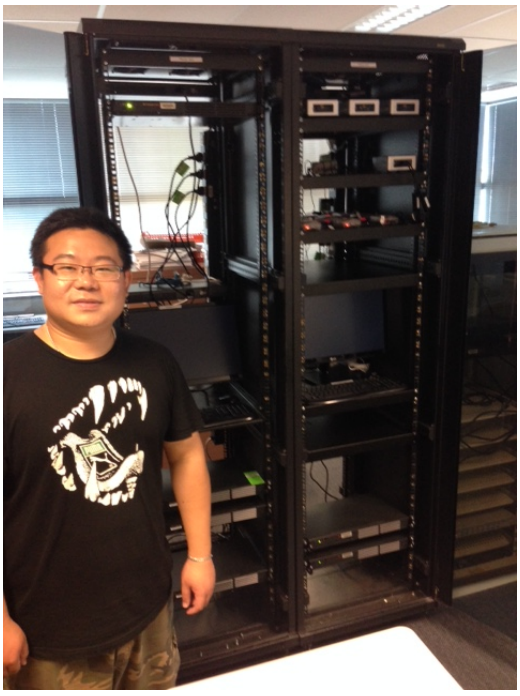
[18] Speidel, U., Cocker, 'E., Médard, M., Vingelmann, P., Heide, J., "Topologies for the Provision of Network-Coded Services via Shared Satellite Channels", Ninth International Conference on Advances in Satellite and Space Communications (SPACOMM 2017), April 23 - 27, 2017 - Venice, Italy. Best Paper Award Winner.

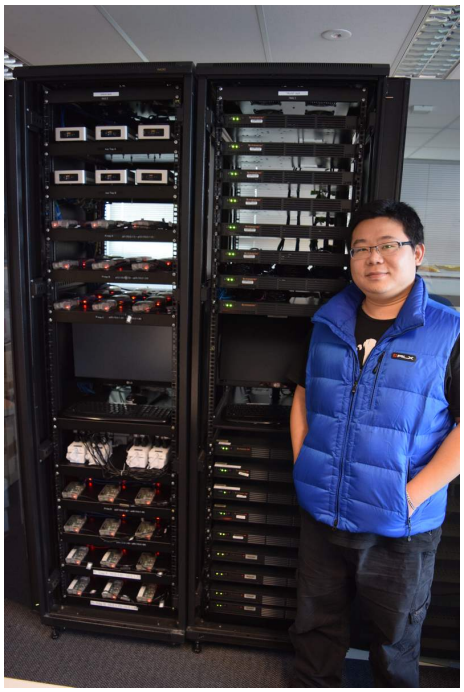
[19] U. Speidel, "Those who measure blindly, tend to produce rubbish results", APNIC Blog, https://blog.apnic.net/author/ulrich-speidel/, February 10, 2017

[20] U. Speidel, "Simulating satellite Internet traffic to a small island Internet provider", https://isif.asia/simulating-satellite-internet-traffic-to-a-small-island-internet-provider/, January 18, 2017

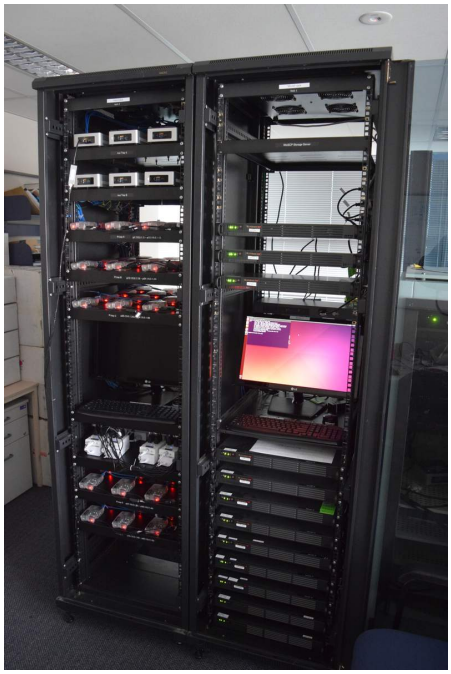
This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Additional material





The simulator during initial construction (left top) and in its present configuration (right top) just before the start of the project. Lei Qian cabled it all up and configured most of the machines. Two terminals allow local work with the machines - although most of the time, we interact with the simulator remotely from our command and control machine, Ulrich Speidel's office, or even from home.



Above: Nothing like a tray of freshly configured Raspberry Pis! We have eight of these now, accommodating a total of 96 Pis.

Left: The first version of the simulator taking shape.

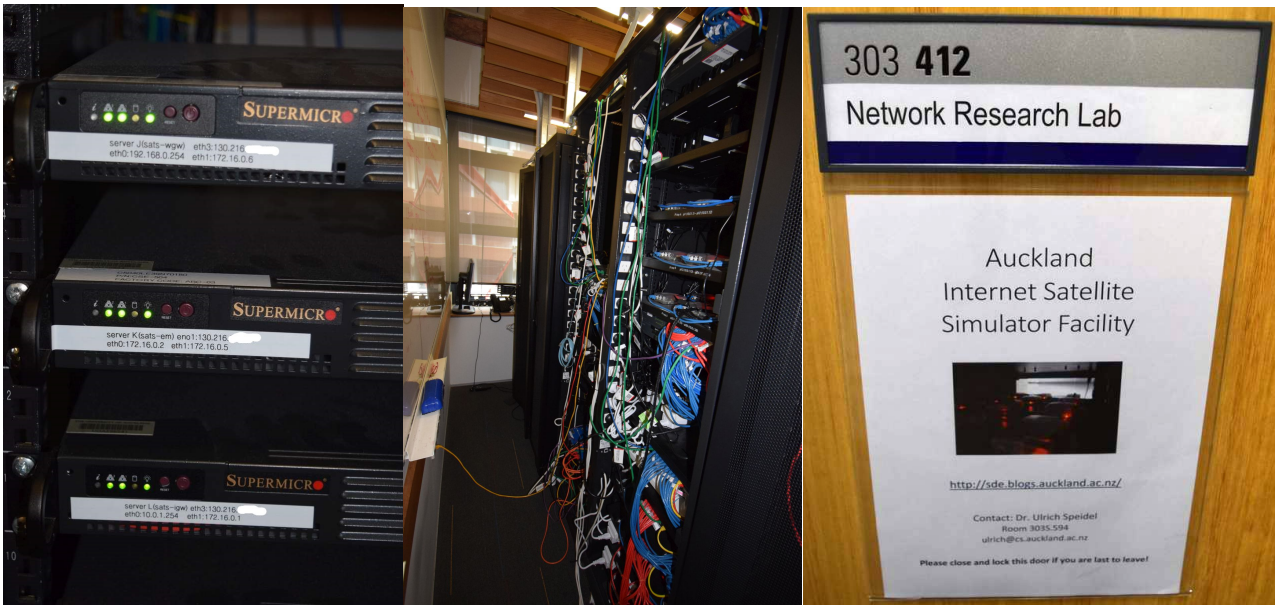
This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Top left: Dropping in on Emani and Jopoy in Niue during a private visit before the start of the project - the sort of Internet pioneers at the far end of a narrowband satellite link that we are aiming the project at. Top right: The primary investigator (right) taking sage advice from Martin Bossert (left), one of the world's top coding experts, in his natural habitat at the University of Ulm, Germany, in late 2016 during the primary investigator's sabbatical. During the visit, we completed a paper on the potential use of PUM codes that we will present at IEEE ISIT in Aachen in June, using data obtained from the simulator as part of this project.

Left below: The Steinwurf team during the principal investigator's visit in November 2016 - from left to right: Lars, Jeppe, Janus, Jeppe, Morten and Vanja. As Steinwurf's CEO, Janus has backed us all the way & given heaps of critical feedback and great suggestions. Sadly, the lead developer on the Steinwurf software, Péter Vingelmann, is not in the picture - he works permanently off-site from Hungary.

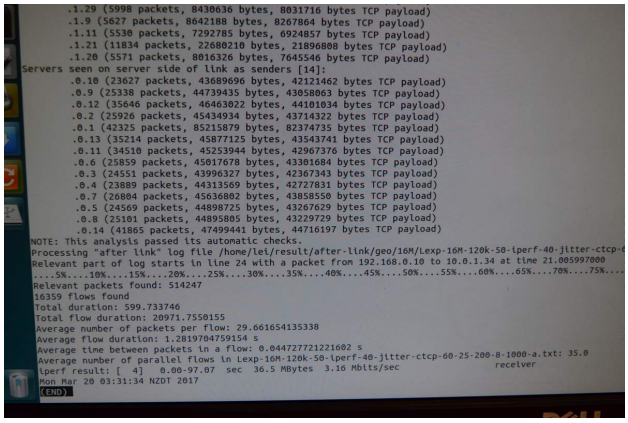
Above right: The simulator in its new home, our new dedicated network lab. The yellow rack on the left is due to be replaced by another 7ft rack of the same type as the two simulator racks in the middle. The racks on the far right are deeper and are intended for a different project. We now have direct fibre connectivity straight into the lab, with up to 10 Gbps possible. The view onto the beautiful equipment will disappear later in the year as we are awaiting partition walls and possibly a false ceiling, as well as dedicated 24/7 air-conditioning. These dedicated premises are an early outcome of the project, which will also probably see us upgrade the simulator with additional servers.

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

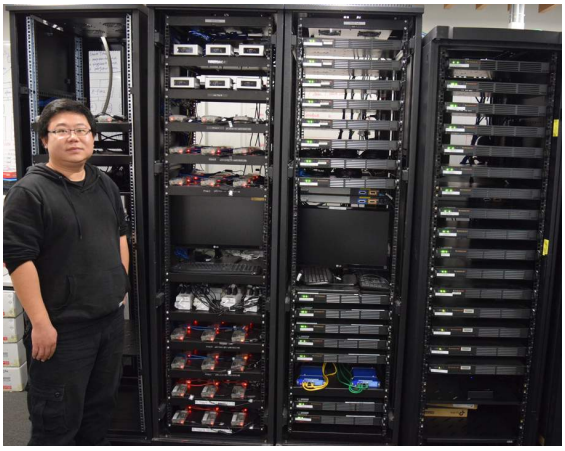


Top left: The heart of the simulator: The three Super Micro Servers that simulate (from top) world gateway (with encoder/decoder/PEP as required), satellite link, and island gateway (encoder/decoder/PEP). Middle: The “ugly” side of the racks gives an idea of just how much cabling is required to connect about 140 devices and supply power to them. Right: Our new door sign!

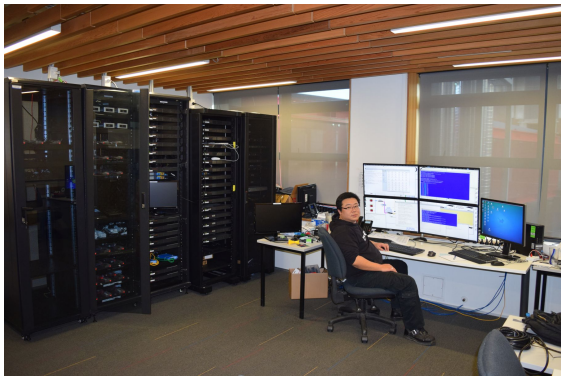
Below left: The lab has a bit of street appeal, but we are still hiding the boxes. Right: Just what we want to see - experiments that pass their automatic checks! This one used a network-coded tunnel on a simulated 16 Mbps GEO link with 120 kB of input queue.



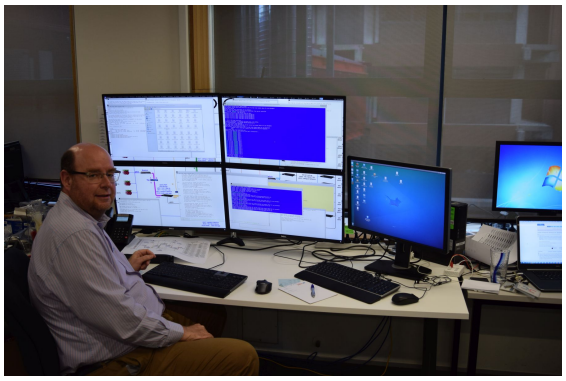
This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.



The simulator racks in their final configuration. The “world side” takes up the rightmost rack and the top of the right middle rack. Below to terminal in the right middle rack sit the machines of the satellite chain: world-side PEP, world-side encoder/decoder, satellite emulator, island-side encoder-decoder, and island-side PEP. The blue bits below them are the copper taps that capture the traffic heading to the island (currently 5 taps, but only two are visible here). The two Super Micros at the bottom are the capture servers – one for the island side of the link, and one for the world side. Each captures from up to three copper taps.



The photo we hope will soon be history: The simulator in its current home with the command and control seat on the right. We are currently talking to the university’s property services to have walls installed that will separate the racks from the rest of the room and provide them with dedicated air conditioning. The room currently only has business hours cooling, meaning the temperatures can get tropical during weekends and after hours.

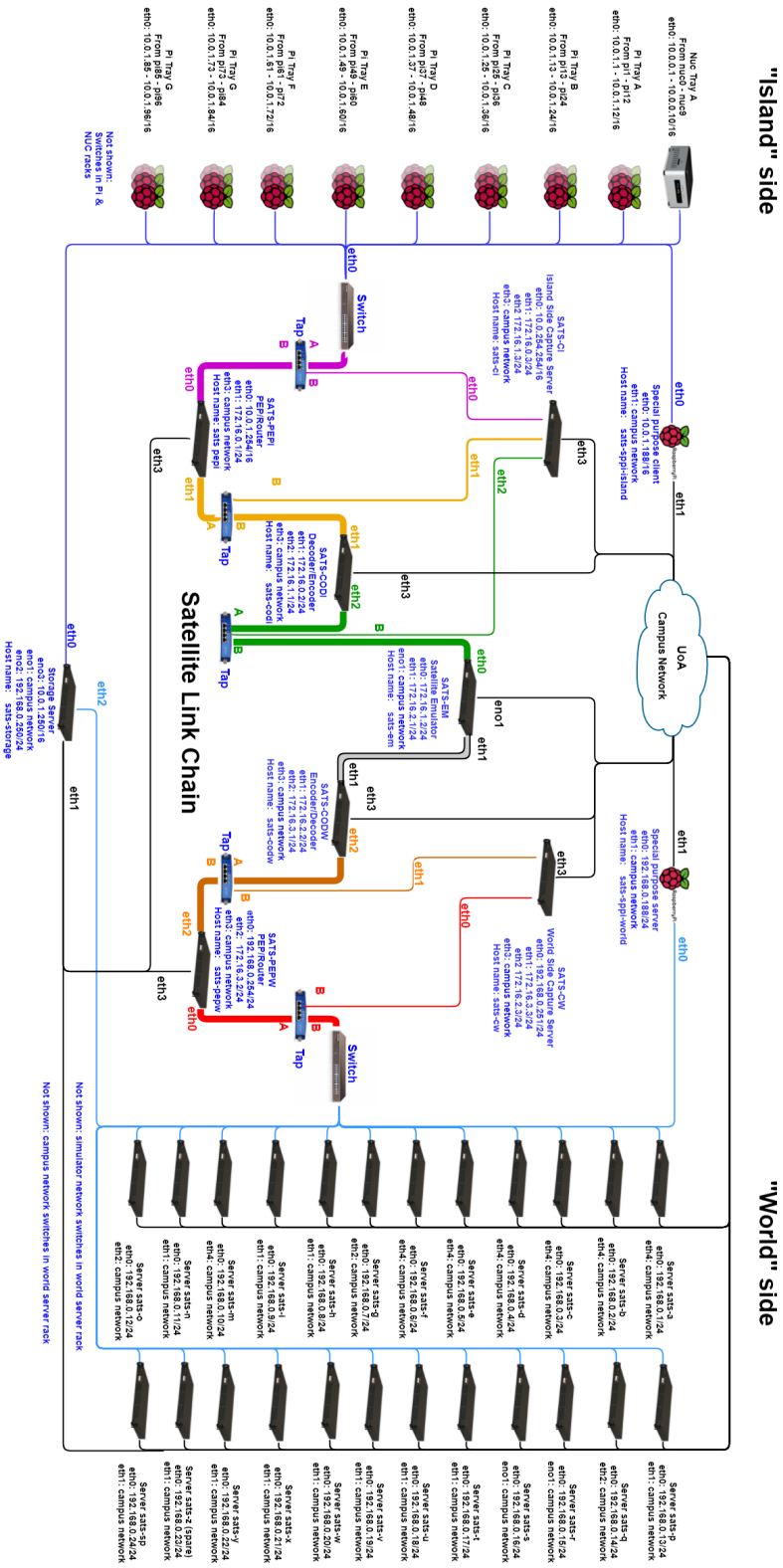


Our new command and control seat is now set up with a large display, which lets us keep a large number of windows in view simultaneously – interacting with the simulator manually often requires talking to up to 11 machines at the same time.

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.

Our four new HP ProCurve switches (two of which were procured from the grant) are OpenFlow compatible and give us the option of reshaping the behaviour of the wold-side network of the simulator.

Fuli works on non-connection-breaking PEP software that we will hopefully be able to deploy on the simulator next year. He uses two Raspberry Pis and a PC to emulate a small network.



Left: Our current simulator topology. The cyan “world” network connects the Super Micro “world servers” and a special purpose Super Micro, which act as the TCP senders, to the satellite chain. The world-side special purpose Raspberry Pi also injects its “whistle” pings into this network to signal the start and end of the official data collection in each experiment.

The thin blue network on the other end of the topology is the “island” network that connects the Raspberry Pis and Intel NUCs representing the “island client” TCP receivers to the satellite chain.

The satellite chain (thick coloured lines) centres around the Super Micro that acts as the satellite emulator (SATS-EM). It is flanked on each side by a Super Micro each that acts as the network coding encoder and decoder (SATS-CODW and SATS-CODI). In uncoded experiments, these machines simply act as forwarding routers.

The outer part of the satellite chain consists of the two performance-enhancing proxy machines (PEPs), one on each side, which connect to the island and world networks, respectively. In experiments where we do not make use of PEP functionality, these act as simple forwarding routers.

The two capture Super Micros, SATS-CW and SATS-CI, listen to island-bound traffic via one of the five copper taps.

During an experiment, most control interaction is with the machines on the satellite chain. In order to keep control traffic and experiment traffic separate, we send the former via the black campus network. This also allows us to drive the simulator remotely. Selected machines on the island side are also accessible in this way via Ethernet dongles connected to the campus network.

The command-and-control machine (called “Storage Server” as it also holds the disks where the results accrue) connects to island, world, and campus networks.

This work has been developed with the support provided by the Information Society Innovation Fund (ISIF Asia) – 2015, licensed under the Creative Commons Attribution-ShareAlike 4.0 International.