TCP's protocol radius: the distance where timers prevent communication

Lloyd Wood, Cathryn Peoples, Gerard Parr, Bryan Scotney and Adrian Moore

Abstract—**We examine how the design of the Transmission Control Protocol (TCP) implicitly presumes a limited range of path delays and distances between communicating endpoints. We show that TCP is less suited to larger delays due to the interaction of various timers present in TCP implementations that limit performance and, eventually, the ability to communicate at all as distances increase. The resulting *performance* and *protocol radius* metrics that we establish by simulation indicate how the TCP protocol performs with increasing distance radius between two communicating nodes, and show the boundaries where the protocol undergoes visible performance changes. This allows us to assess the suitability of TCP for long-delay communication, including for deep-space links.**

Index Terms—**Transmission Control Protocol, TCP, Delay Tolerant Networking, DTN**

I. Introduction

Successful electrical communication between two points requires that a signal be received and decoded. A link budget can be used to determine whether this can be achieved by adding up all the gains and losses accrued in the physical channel between sender and receiver. The receiver has a dynamic range in which signals are received, demodulated, and decoded; if the sender is too far away, its signal will not be received and decoded correctly, as the weak signal will lie below the receiver's noise floor, and be swamped by noise in the channel. Conversely, if the sender is much closer than its power output is designed for, its signal can exceed the receiver's dynamic range and oversaturate the receiver, as the expected free-space attenuation component of the link budget has decreased.

By analogy, for communications using networking protocols above the physical layer, we must also consider the delay budget between two communicating points. This delay budget is the sum of separate delays. These can include:

a. propagation delay between endpoints, governed by speed of light in the medium.
b. channelisation/serialisation delay; time between transmitting the first and last bit of a frame. This is significant at low data rates, but negligible at high data rates. As the link rate increases this tends towards zero.
c. medium access delays dealing with contention for a shared medium. Here, we assume point-to-point serial communications, and neglect complex Medium Access Control (MAC). Shared communication has not been used beyond geostationary orbit.
d. processing/queuing/endhost delays, which may be deliberately inserted to minimise resource use on the network or in the endhost. For TCP, mechanisms such as delayed acknowledgments [1] and Nagle's algorithm [2] can be significant in affecting two-way communications.
e. Codec delays. Minimising the other delay components permits more time for advanced high-complexity codecs to compress video or audio efficiently.

It is necessary to sum these time delays to see if the total delay is suitable for logical communication to take place, just as we sum decibels in the link budget to see if the physical signal can be heard by the endpoints.

Many medium access and transport-layer protocols are designed to perform within a certain delay range between the two communicating points. At increased distances and larger delays between communicating nodes, performance of the examined protocol can be expected to degrade, and protocol mechanisms can even cause communications using a protocol to fail at sufficiently large distances and time delays. Conversely, using a protocol between points with much smaller delays than expected for the protocol can result in exchanges being dominated by protocol transaction overheads that produce degraded performance when compared to alternative protocols more suited for the smaller delay times.

For example, consider TCP, the Internet's Transmission Control Protocol. TCP is widely recognised to perform poorly across geostationary satellite links of around half a second of path Round Trip Time (RTT) [3]. This is due to TCP's exponentially-increasing probing of path capacity in slow start, and its assumption about fair use of network resources – that every packet lost is due to network congestion, and that this can be addressed by slowing TCP further. Similarly, TCP has high delay overhead for fast parallel computers, where interprocess communications can be done using something that can use dedicated communications capacity without the need to be as cautious or slow to build to high speeds as TCP. These give us a very rough idea of the range of delays and distances where TCP communication performs sufficiently

Manuscript prepared for camera proof 15 June 2007.

L. Wood is with the Global Government Solutions Group, Cisco Systems, Bedfont Lakes, Feltham, Middlesex, England (email lwood@cisco.com).

C. Peoples is a PhD candidate in the School of Computing and Information Engineering, University of Ulster, Coleraine, Northern Ireland (email cathryn@infc.ulst.ac.uk).

G. Parr, B. Scotney and A. Moore are PhD supervisors, also in the School at Ulster (email gp.parr, bw.scotney, aa.moore@ulster.ac.uk).

well to be useful.

This paper examines the delays and distances more precisely, in order to characterise TCP and its performance limits more accurately.

II. Performance and Protocol Radii

We call the limit within which a protocol works at the high performance for which it was designed its *high performance radius*. An endpoint using the protocol will be able to communicate well by direct line of sight with another endpoint within its performance radius.

The distance beyond which a protocol can no longer work or be used to successfully communicate information, because set timers within the protocol cause it to fail, is the protocol's limiting *protocol radius*. This limit forms a larger 'bubble' around the endpoint that encompasses the high performance radius, and other performance radii where protocol timers have caused changes in the performance of the protocol, rather like the outer skin and concentric inner layers of an onion.

For wireless communication, this protocol radius can be thought of as analogous to the *Schwarzchild radius* – the distance beyond which information cannot escape from a black hole. This bubble around an endpoint indicates the range within which it can usefully exchange information with another endpoint using the protocol: the protocol's event horizon, or a very different form of 'networking black hole' for outgoing traffic. These bubbles will fall well within the boundaries of the ever-increasing Minkowski light cone that indicates the radiation of signal from an endpoint. For paths via a relay point, the limiting shape of the bubble is an ellipsoid. These radii are shown [Fig. 1].

For convenience, we will define a distance between communicating nodes in seconds of delay needed to travel that distance. This simplifies calculations mapping the protocol timers and their delay limits to path distances. Thus, for wireless communications between two points in the vacuum of space communicating at light speed *c*, we can translate the seconds of path delay directly into light-seconds of distance, assuming that the link bitrate is high enough that the serialisation delay of the packet can be neglected, and ignoring MAC timers. For other media where light travels more slowly, we can compute the distance by dividing the delay by the refractive index of the medium.

III. Examination of TCP's Radii

A. Dependence on IP

TCP segments are carried in IP packets, so it is worth examining IP itself for limiting factors. IP's Time-To-Live (TTL) counter was originally specified to measure time in seconds or hops [4]. TTL later became just a hop count, as decrementing the TTL counter by one at each hop was easier to compute. As TTL is stored in an octet, an IP packet can traverse a maximum of 255 hops before the counter hits zero and demands that the packet be deleted without forwarding. However, the initial value is rarely set this high by the sender even for multicast thresholds, and is more likely to be 32.

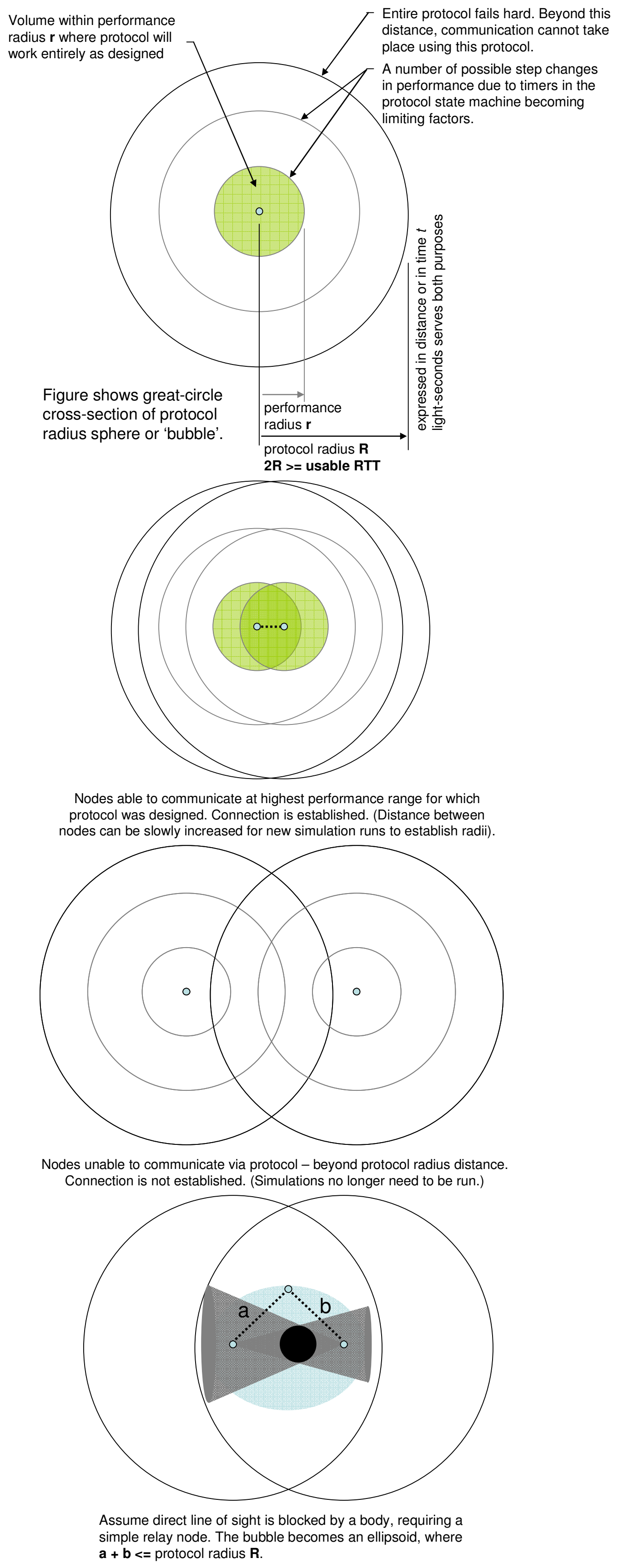
Volume within performance radius **r** where protocol will work entirely as designed
Entire protocol fails hard. Beyond this distance, communication cannot take place using this protocol.
A number of possible step changes in performance due to timers in the protocol state machine becoming limiting factors.
Figure shows great-circle cross-section of protocol radius sphere or 'bubble'.
performance radius **r**
protocol radius **R**
2R >= usable RTT
expressed in distance or in time *t* light-seconds serves both purposes
Nodes able to communicate at highest performance range for which protocol was designed. Connection is established. (Distance between nodes can be slowly increased for new simulation runs to establish radii).
Nodes unable to communicate via protocol – beyond protocol radius distance. Connection is not established. (Simulations no longer need to be run.)
a
b
Assume direct line of sight is blocked by a body, requiring a simple relay node. The bubble becomes an ellipsoid, where **a + b** <= protocol radius **R**.

Fig. 1. Depictions of protocol and performance radii for free-space communication

TTL places an upper limit on the number of links that can be concatenated for end-to-end communication, but this limit is rarely encountered, so we can ignore it while focusing on total path delay. Although there are many timers involved in underlying MAC layers setting up an IP link, there are no timers in IP as such. IP does have other limitations – lack of support for mobility in a design intended for fixed, wired links being one – but these are outside the scope of this paper. We can move on to examine TCP with a clear conscience.

B. Test scenarios: the simulation environments

We examined the performance of TCP by increasing path delay between nodes in the network simulator *ns* 2.30 [5] and in *Opnet* 11.5. We examined TCP Reno, TCP SACK and timestamps. We relied to a large extent on given simulator defaults, as we recognise that implementation defaults can vary widely, and seeing common behaviour across a wide range of conditions (including differing simulation environments) leads to insights. In both simulators, a single reliable serial link was used to remove the effects of errors and of MAC protocol timers. Unidirectional HDLC/Frame Relay serial links carrying IP are in use for wireless point-to-point space communications, so a serial link is not unrealistic [6, 7].This link was set at a high enough rate to remove serialisation artefacts discussed earlier. Channel-induced errors were eliminated to allow us to focus solely on TCP's own performance.

Using the default 16-bit pointer to its fullest with a 64K window makes sense with long delays, so that was tested alongside default buffer sizes. In the interests of keeping simulations tractable, we did not examine large windows extensions to TCP. TCP link utilization is ultimately limited by its buffer sizes; a large link rate ensures that that rate does not affect simulations, and that TCP is limited by its windows. The initial retransmission timeout (RTO) value was 3 seconds, minimum RTO 1s and maximum RTO 64s. Timer granularity was the default in the simulators: a fairly coarse-grained 0.5s in *Opnet*, and 0.1s in *ns*. FTP file transfers were used.

C. TCP's protocol radius - the SYN/ACK handshake

TCP was initially said to have a maximum segment lifetime of two minutes, with a global timeout of five minutes to abort the connection if no data was delivered [8], although this was not widely adopted in implementations. A two-minute segment and packet lifetime, mapped to a four-minute RTT, would mean that nodes attempting to communicate across a path longer than two minutes in length would fail to talk. There is no formal upper bound on RTT, but an RTT of greater than one minute is claimed to be unlikely [9].

TCP's tolerance to delay is governed by its initial retransmission timeout (RTO) value, and by the timers used to open a connection in the three-way SYN/ACK exchange between sender and receiver. If this exchange is not successfully completed, data transfer cannot even begin.

When opening a TCP connection, the sender will send a SYN packet for the receiver to acknowledge with a SYN/ACK, before the sender sends its own SYN/ACK in a three-way handshake.

The sender waits for a reply to its initial SYN, but does not wait indefinitely. After the initial RTO time of three seconds has passed, another SYN will be sent – and the sender will then double the time it waits to six seconds. If this is not replied to, another SYN is sent after six seconds, and the sender doubles its waiting time again, leading to SYNs at 0, 3, 6, 12, 24, ... second intervals, or 0, 3, 9, 21, 45, ... s after the first SYN. So, for long-delay paths, more than one initial SYN is sent, and a reply to the first SYN is finally received by the sender in the window of a subsequent SYN that is still in transit to the receiver. This is shown in Figure 2. This patient doubling and waiting for an initial response could go on indefinitely, but TCP implementations give up and report an error to the application eventually. The inefficiencies of TCP's timer mechanisms have long been noted [10].

In *Opnet* simulations, a TCP sender and receiver were unable to establish a connection when the distance between the two was greater than 22.5 seconds, or a path RTT of 45 seconds. 45 seconds is the sum of the intervals 0+3+6+12+24 seconds, resulting from the *Opnet* TCP implementation attempting to open the connection five times before giving up because no response has been received. If we consider this for line of sight, 22.5 light seconds is enough to include all the Earth/Moon Lagrange points, but is a smaller upper bound than the minute-sized estimates we read of earlier.

In *ns*, this exponential doubling sequence continues indefinitely as path delay increases, until a reply is received. This is a useful reminder that simulation does not accurately reflect implementations. *Opnet* is more realistic here.

We looked at Microsoft Windows as a common example implementation. `TcpMaxConnectRetransmissions` in the registry settings defaults to 2 retransmissions in a range of 0-255, giving a maximum RTT of 9 seconds using the default `TCPInitialRTT` value of 3 seconds to double on [11].

These default settings and the radius of 4.5 light-seconds that results are enough to encompass the Earth and Moon; the maximum RTT supported here using a value of 255 SYN resends (1.7×10^{77} seconds) translates into a propagation distance that far exceeds the width of the visible universe.

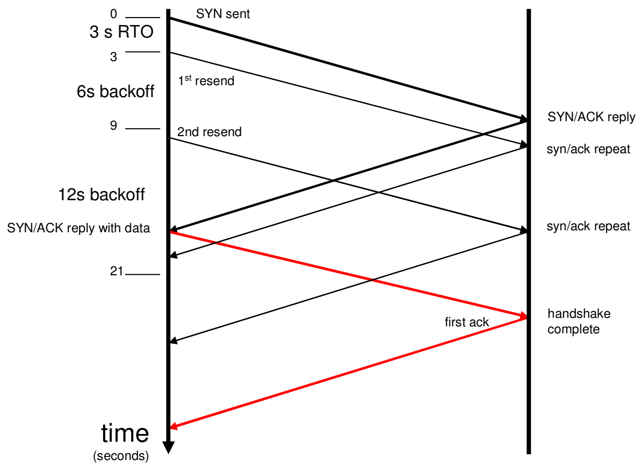

Fig. 2. TCP working with path of 16s delay, with resends before connection opens

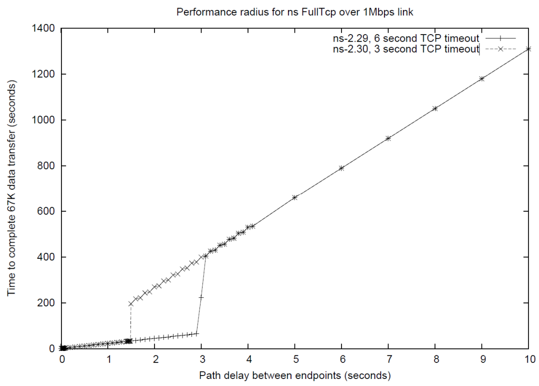

Fig. 3. Clear RTO-related step in TCP transfer time as path delay increases

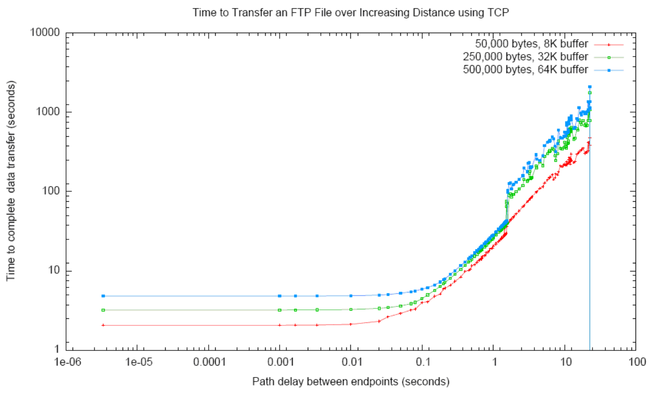

Fig. 4. Time to transfer a file via FTP

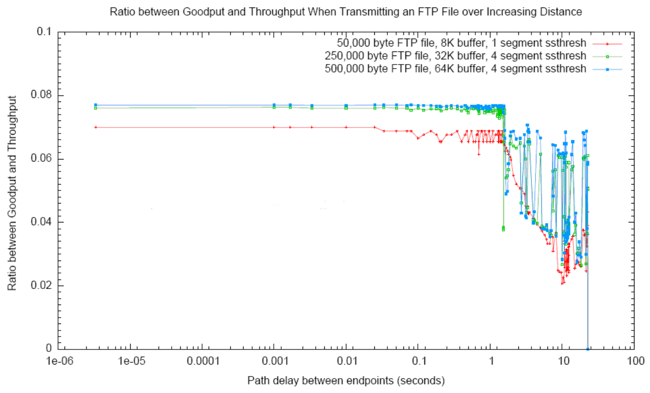

Fig. 5. Efficiency ratio for FTP file transfer

The upper limit on times the initial SYN is sent out, and the initial RTO time that determines the intervals between times, can be easily altered in a communicating stack to other values to change the limits to which TCP will communicate, as can other defaults controlling TCP timers. But how well does TCP perform in communicating within these established bounds, once a connection has been opened at a long distance?

D. Identified performance radii in TCP

One performance radius in TCP is defined by its initial retransmission timeout – when the sender stops waiting for a segment to be acknowledged by the receiver, and retransmits it. This timeout value is initially set to 3 seconds [12]. Thus, a path of up to 1.5 seconds end-to-end can use this initial timer value; longer paths will fall outside this initial value, so some retransmissions will be seen as the timeout value is increased and TCP's congestion window will be reduced as a result. We can show this by comparing the total times to transfer a file by ftp over TCP, for a range of path delays between end nodes. We used a single, reliable, error-free serial link between two nodes to avoid introducing the effects of MAC protocols and their timers into our results.

In *ns*, the initial RTO value has traditionally been 6 seconds; this was brought into line with RFC2988 by changing it to 3 seconds for the *ns* 2.30 release [13]. By comparing the amount of time taken to transfer a file by ftp for *ns* 2.29 and 2.30 as path delay between the endhosts is increased by a minimum of 10ms across a number of transfers in separate simulation runs, we can see the effect on TCP SACK performance of this altered retransmission timer [Fig. 3].

Paths on the terrestrial Internet lie at the very left of this performance curve, with rapid response times well below the initial RTO limit. Transfer time increases with path delay either side of this step, but at different rates. TCP continues to deliver the file on paths whose end-to-end delay exceeds the timeout value (but whose length is bounded by the SYN/ACK exchange discussed earlier), but transfer performance suffers more than just the increase in path delay would suggest.

We used *Opnet* to examine overall averaged TCP Reno goodput and throughput (the average fixed equivalent rate at which the file would have been transferred, and the average equivalent rate for all traffic sent, respectively) in bits per second as distance varies [Figs. 4 and 5]. These clearly show degradation of performance with increasing distance and path delay, with step changes in performance and phase changes in behaviour visible at half of known sum-of-SYN-timer-values (at 1.5s, 12s, and the limiting 22.5s). Examining the goodput/throughput ratio gives a measure of TCP's efficiency for the traffic that it does send. This measure falls off rapidly once round-trip time exceeds the initial 3s RTO at 1.5s path delay, as the proportion of useful packets sent diminishes and the number of retransmissions increases. TCP's throughput is still ultimately limited by its buffer sizes, and the lone TCP flow utilizes only a small fraction of available link capacity.

Protocol radius behaviour is clearly independent of TCP buffer sizes and filesizes. As these sizes increase, performance changes remain at the known performance radii, caused by initial RTO timer values. Jitter occurs above the first 1.5s radius due to unavoidable timeouts and the effect of the coarse timer granularity, and become increasingly pronounced as the buffers and files increase in size, because more traffic is affected and resent when a timeout occurs.

Our simulation results from *ns* and *Opnet* suggest that the SACK and timestamp mechanisms, which improve TCP performance in terrestrial networks, do not significantly benefit a single loss-free flow over extremely long distances, although their use can compensate for a coarser system clock.

Performance slopes in *ns* were generally smoother due to the finer default timer granularity (0.1s in *ns*, 0.5s in *Opnet*) leading to fewer unnecessary timeouts.

E. TCP and Delay-Tolerant Networking

With these results, we can assess the suitability of using

TCP as a convergence layer or transport layer for Delay Tolerant Networking (DTN) [14]. TCP's link utilization is poor, therefore, when two peer nodes are communicating across a dedicated temporary or intermittent link with a single flow from peer to peer, TCP would clearly be the last choice of protocol for efficiency reasons.

However, where DTN bundles [15] need to be transported across relatively small path distances over the terrestrial fixed Internet, TCP's understanding of fairness when multiplexing multiple competing flows and its ability to effectively support file transfers as low-priority background traffic in the face of internetwork congestion come to the fore.

TCP is very much a product of the prevailing conditions on the terrestrial Internet – and helped create those conditions. TCP's design is unsuited to the long-distance paths of deep space, or to short ad-hoc communications in sensor networks where efficient use of a link by a single flow is unlike the shared, cooperative yet competing, Internet. TCP's performance here could be improved, but it is an edge case as far as TCP's design and the majority of its use is concerned. TCP would be unsuitable to communicate with a deep space probe around Mars, even if the initial SYN/ACK exchange timers were tweaked; data transfer would not use available link capacity effectively, and performance would be very poor. Where TCP breaks, much of the Internet infrastructure that depends on TCP also breaks.

UDP-based transport protocols that can avoid sharing TCP's assumptions have been developed, some for use with DTN [16]. However, these protocols have not seen large implementation deployment or a consistent feature set across the implementations that are in use. We feel there remains an opportunity for the development of a new approach to communication in DTN. While TCP has shown to be unsuitable for DTN store-and-forward scenarios, we are involved in the development of different approaches to approaching the long-distance communication and link utilization challenges in other, related, work [17] [18].

IV. Further work

We began with TCP to examine how protocol performance degrades with distance and is affected by internal protocol timers, simply because TCP is well-understood and straightforward to simulate. By deliberately ignoring errors we have found the outer bounds to the limits of communication; learning how errors bring these bounds inward is further work.

The differences in simulation between *ns* and *Opnet* have shown limits to accuracy in simulation of TCP, as well as the importance of default values.

It would be helpful to examine real implementations of TCP with a delay emulator that stores and releases packets.

We have also analysed timers in other Internet protocols and the limits that they lead to, but lack of space prevents further discussion of those protocols here.

Further work would also examine MAC protocols and their timers. Accurate and complete MAC simulation is difficult, so it would be desirable to examine MAC implementations in detail using a delay emulator.

V. Conclusions

Just as we use a link budget in radio communications to ensure that physical communications can take place successfully across a given distance, we need to design protocol communications with distance in mind, and check that the delay budget is adequate for the delays encountered.

The design of many existing communication protocols assumes a limited distance or path delay between communicating endpoints. When this delay is exceeded, the performance of the protocol will be degraded.

We have examined this behaviour for the popular Transmission Control Protocol, and have shown how TCP's performance degrades with increasing distance to the point where communications finally fail at its protocol radius. Although TCP's relatively weak performance across geostationary satellite links of around 0.5s round-trip time is well-known, how TCP's performance degrades with distance beyond that was not. TCP could be used for direct Earth/Moon communication, as that range falls within the first 1.5s performance radius of TCP. However, a TCP transfer would not make effective use of available link capacity in that scenario; a more effective transport protocol would be needed to replace TCP to fill the link.

References

[1] M. Allman, “On the Generation and Use of TCP Acknowledgments,” ACM *Computer Communications Review*, 1998.

[2] J. Nagle, “Congestion control in IP/TCP internetworks,” Internet Engineering Task Force RFC 896, January 1984.

[3] C. Partridge and T. Shepard, “TCP Performance Over Satellite Links,” *IEEE Network*, vol. 11 no. 5, September/October 1997.

[4] J. Postel, “Internet Protocol specification,” Internet Engineering Task Force RFC 791, September 1981.

[5] K. Fall and K. Varadhan, *The ns manual*, 2007.

[6] K. Hogie *et al.*, “Using standard Internet Protocols and applications in space,” *Computer Networks*, vol. 47 no. 5, pp. 603-650, April 2005.

[7] L. Wood, W. Ivancic *et al.*, “Using Internet nodes and routers onboard satellites,” *International Journal of Satellite Communications and Networking*, volume 25 issue 2, pp. 195-216, March/April 2007.

[8] J. Postel, “Transmission Control Protocol specification,” Internet Engineering Task Force RFC 793, September 1981.

[9] V. Jacobson, R. Braden and D. Borman, “TCP Extensions for High Performance,” Internet Engineering Task Force RFC 1323, May 1992.

[10] L. Zhang, “Why TCP timers don't work well,” ACM SIGCOMM, pp. 397-405, 1986.

[11] Microsoft Windows 2003 TCP/IP Implementation, TechNet, Microsoft Corporation, June 2006.

[12] V. Paxson and M. Allman, “Computing TCP's retransmission timer,” Internet Engineering Task Force RFC 2988, November 2000.

[13] S. Floyd, *ns* changelog entry, 23 January 2006.

[14] V. Cerf, S. Burleigh *et al.*, “Delay Tolerant Network Architecture,” IETF RFC 4838, April 2007.

[15] K. Scott and S. Burleigh, “Bundle Protocol Specification,” work in progress as an IETF internet draft, December 2006.

[16] M. Ramadas *et al.*, “Licklider Transmission Protocol – Specification,” work in progress as an IETF internet draft, September 2006.

[17] C. Peoples, G. Parr, B. Scotney, A. Moore, “A Reconfigurable Context-Aware Protocol Stack for Interplanetary Communication,” Third International Workshop on Satellite and Space Communications (IWSSC '07), September 2007.

[18] L. Wood, W. Eddy, W. Ivancic, J. McKim and C. Jackson, “Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization,” Third International Workshop on Satellite and Space Communications (IWSSC '07), September 2007.