Link Layer–Based TCP Optimisation for Disconnecting Networks

James Scott [*]
Intel Research Cambridge
15 JJ Thomson Avenue
Cambridge CB3 0FD, UK
james.w.scott@intel.com

Glenford Mapp [†]
Middlesex University
Bounds Green Road
London N11 2NQ
g.mapp@mdx.ac.uk

ABSTRACT

This paper discusses a link layer approach to improving TCP performance in the face of periodic network disconnections. Network disconnections are encountered in many scenarios, including being out-of-range in a wireless network, during network handoff, and also in the case of Networked Surfaces, a novel LAN technology which provides the motivation for this work.

A "smart link layer" employing repetition of selected packets at reconnection time is shown to improve TCP's utilisation of a disconnecting network to nearly 100%. This solution is also demonstrated in the context of a Networked Surface prototype, improving TCP performance for both bulk transfers and interactive traffic.

The smart link layer solution is lightweight, requiring little processing and buffering only one packet per TCP connection. It is therefore easily retro-fitted to existing TCP-capable devices, without modifying the internal operation of those devices.

Keywords

TCP, Disconnection, Link Layer, Mobile Networking

1. INTRODUCTION

The Internet Protocol Suite, in particular TCP/IP, has been a runaway success. However, these protocols were conceived when all data communications was carried over wired links. Those days are long gone and new environments such as mobile telephony and Wireless LANs are now becoming ubiquitous.

In these new settings some of the original design assumptions of TCP no longer hold true with respect to the handling of errors such as lost packets. This unfortunately results in a large performance degradation in these environments. This is because TCP assumes all packet loss is due to network congestion, whereas in these settings it may be due to a number of factors including momentarily high link error rates and handoffs of mobile devices between adjacent base stations.

By assuming network congestion is the cause of these errors, TCP does the wrong thing: it drastically reduces its transmit window and deploys the slow start algorithm. This results in unutilised network bandwidth and applications experiencing increased network latency. This behaviour has also been observed in networks where devices can be disconnected, even when the disconnected interval is near the human threshold of noticeability (under 1 second), as well as longer durations (e.g. changing a network cable; 1 minute). Part of the motivation for this work was the development of a novel LAN technology called Networked Surfaces [21] which exhibits disconnections; details of this technology are discussed further in Section 3. Such disconnections may also be found when using wireless networking with signal fading (e.g. due to being on the limit of the range available), in wireless handoff scenarios, or in other situations where the network access path changes (e.g. when a device is removed from a wired docking station and starts to use a wireless network).

1.1 Related Work

Past attempts to address this problem can be neatly divided into two distinct camps.

The first group does not attempt to change or modify the TCP protocol, instead using methods such as injecting, removing or delaying TCP packets based on a superior understanding about what is happening at the link layer. Snoop [6] looks at this problem in the context of a wireless lossy link on the periphery of a wired network. It requires that base stations have large memory and processing power to store network packets while handoffs take place. The base station also gives local acks and suppresses duplicate acks. The "Delayed Dupacks" scheme [23] looks at the same problem but in the context of a reliable link layer protocol which acknowledges each packet and performs fast retransmission. The system allows the TCP receiver to delay acks by a set amount and does not send them at all if a new packet arrives prior to the timeout. TULIP (Transport Unaware Link Improvement Layer) [18] also attempts to recover from retransmission losses before TCP coarse-grain timeouts occur. AIRMAIL (Asymmetric Reliable Mobile Access In Link Layer) [2] uses similar ideas.

The second group focusses on modifying how TCP works. Caceres and Iftode [8] were one of the first to examine the problem

[*]Much of this work was done whilst at the Laboratory for Communication Engineering, which is part of the University of Cambridge.
[†]Much of this work was done whilst at AT&T Laboratories Cambridge.

of disconnected periods affecting TCP, which they found to occur during mobile handoff. They propose a system augmenting TCP so that, on reconnection, a mobile host would retransmit a number of duplicate acks, and so that a fast retransmit mode is automatically entered. Other research focuses on the use of proxies to try to isolate the effects of disconnection to a single link. I-TCP [3] does TCP proxying at such a gateway and modification must be made to the non-ideal segment (meaning the wireless side) to improve performance. The disadvantage is that end-to-end TCP semantics are not upheld, in that acks are sent for data which has not actually reached the final endpoint. M-TCP [7] also uses a proxy approach but maintains end-to-end semantics. It does so by using delayed acks and by placing TCP in persist mode to avoid losing packets during handoffs.

Other methods of modifying TCP behaviour use "flags" or control messages to trigger appropriate responses from TCP. Schemes based on this general theme including Explicit Loss Notification (ELN) [5], Explicit Bad State Notification (EBSN) [4], Explicit Link Failure Notification (ELFN) [13], Route Failure Notification (RFN) [9], as well as an ICMP-based solution [12].

It is also appropriate to summarise the recent efforts of standards bodies, in particular the IETF, with regard to this problem. The Performance Implications of Link Characteristics (PILC) Working Group is looking at how the IP Protocol Suite works with different types of link layers. The latest draft document from this group [16] attempts to characterise links and come up with best-practice suggestions for system administrators. The disconnection problem, described by that group as recovery from subnetwork outages, is addressed by recommending that packets are not discarded during an outage and an interface (such as those described above) be provided to allow IP and the higher layers to be notified once the link has been restored. If this is not feasible, it is recommended that the link layer retains one or more of the packets which could not be transmitted during the disconnected period, and retransmits these packets on reconnection. A similar approach was decided upon when the research presented in this paper was begun, in March 2000, and experimental results concerning the performance of this solution are presented below.

Other recent work in the IETF includes the development of the TRIGTRAN framework [10]. This proposal, like the PILC one, involves a mechanism to alert the transport layer about changes in individual links along the network path from source to destination. Under this scheme, hosts may request notification when trigger events such as Connectivity Interrupted, Connectivity Restored and Packets Discarded by Subnet occur. However, this work is very recent and is currently lacking in experimental support.

1.2 Paper Structure

This paper describes the implementation and experimental evaluation of a "smart link layer" to solve the problem of TCP performance degradation during disconnected periods, as motivated by the introduction of a new type of LAN named Networked Surfaces, which is prone to disconnections. Section 2 presents the TCP performance degradation problem in detail. Section 3 then describes Networked Surfaces and presents the motivation behind the work as a whole, as well as the impetus for making the specific design choices present in the smart link layer. Section 4 presents a discussion of the design space for the smart link layer approach, and identifies a number of candidate algorithms for experimental evaluation. Section 5 evaluates these algorithms on a testbed using a simulated channel and analyses the performance characteristics of the algorithms, and Section 6 goes on to determine the effectiveness of the best-performing algorithm on the Networked Surface platform, under both bulk transfer and interactive traffic patterns. Finally, Section 7 compares the techniques presented in this paper with those in the literature, and Section 8 concludes the paper, including a discussion of future work.

2. THE EFFECT OF DISCONNECTION ON TCP

TCP regards all packet losses as indications of congestion. While working well for wired infrastructure, this has caused many problems when combined with wireless access, in which channel errors causing dropped packets are more common. Distinguishing and coping with such losses in order to make TCP fully utilise a lossy channel has been the subject of much research, as described in the previous section.

However, there are many differences between disconnection and lossy channels, which means that the same solutions may not work well for both cases. Firstly, in the lossy channel case, it is obvious that discovering the loss and retransmitting quickly is desirable. However, with disconnection, retransmissions are not useful until the link is re-established. On the contrary, transmitting and retransmitting packets for a disconnected device is guaranteed to be a waste of network bandwidth.

Secondly, the timescales for channel losses and disconnections are very different, with the former operating on a packet-by-packet basis and in the microseconds range, while disconnections may last anywhere between milliseconds and minutes, depending on the cause. A short disconnection may be due to a Mobile IP handoff occurring, while a longer disconnection may be due to a modem connection failing and having to be reconnected. Disconnections typically last longer than a TCP timeout, while potentially being shorter than the lifetime of a TCP connection.

In order to illustrate the detailed effects of disconnection on a running TCP connection, a simple experiment was conducted in which a file was transferred over a disconnecting link. The TCP "trace" occurring in this experiment is shown in Figure 1. As the figure shows, the sender does not react to disconnection, and continues sending (pointlessly) until its window is full. It then waits for acks, but times out before any ack arrives and retransmits the first packet. Retransmission occurs two more times, with an increasing timeout period each time — this is because TCP assumes that the lack of response is due to the network being congested, and so tries to back off to let the network recover. When reconnection occurs, TCP does not immediately restart, instead continuing to wait until its next timeout. When this happens, the packet gets through, causing an ack to be received and further packet transmission to resume. Note that over 1.5s of connected time was wasted by TCP in this case.

3. NETWORKED SURFACES

The motivation behind this work is the introduction of a new type of network, known as Networked Surfaces [21]. This network is based on the use of physical surfaces such as desks to perform networking. Devices such as laptop computers and PDAs can acquire network connectivity by simply being placed on top of such a surface, in any position and at any orientation. Networked Surfaces can also provide power to devices such as mobile phones, they can

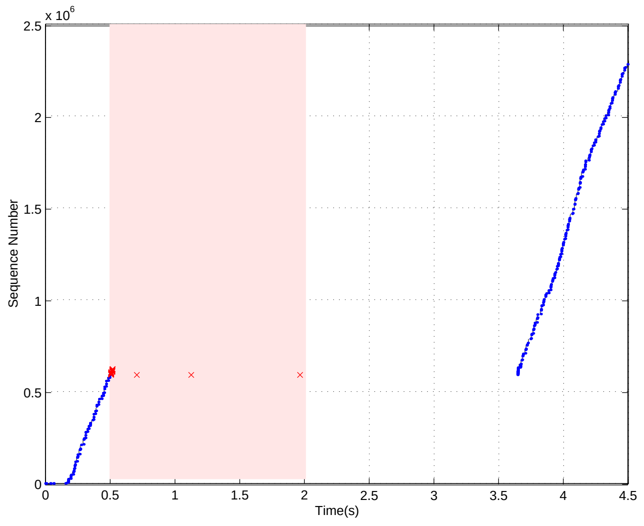

Figure 1: TCP File Transfer with Disconnection

TCP traces in this paper are presented as follows.

The vertical lines are transmitted segments, the dots are acks returning. In zoomed-out plots (such as the one above), these are hard to distinguish individually, and appear as sloped line.

The shaded portions are periods of disconnection. Segments dropped during these disconnected periods are shown with crosses.

Segments and acks inserted by the smart link layer (to be described) on reconnection are shown with circles and plus marks, respectively.

support low-speed devices such as keyboards and sensors, and they can locate devices to within a few centimetres and a few degrees.

The vision behind Networked Surfaces is that they provide the best of both worlds between wired and wireless paradigms. As with wired devices, networking is provided at a high bandwidth (5Mbit/s in the first prototype) and does not have to be shared with other devices in a physical space, and electrical power is provided. At the same time, the inconvenience and hassle of carrying and connecting cables is avoided, thus providing a very user-friendly environment for mobile computing users.

Networked Surfaces operate by using electrically conductive pads on the surface and the base of the device. When a device comes into contact with a surface, a handshaking procedure causes the various conducting paths formed to be assigned to functions such as ground, power, and networking buses. Disconnection is detected by the custom link layer protocol used, and when it occurs, the connected pads are returned to a disconnected state, ready for a new connection. It is important to note that Networked Surface NIC functionality is intended to be added to existing devices, including devices which are not reprogrammable, and which therefore have hardcoded TCP/IP stacks. Even when devices are reprogrammable, e.g. with notebook computers, it is not normally expected that installation of a NIC's software driver would entail modification of the TCP/IP stack of the device.

One key issue in the usability of Networked Surfaces is the fact that devices are susceptible to occasional disconnections, since any movement of a device may cause the connected pads to lose contact. When this happens, the pads must all undergo disconnection and then re-execute the handshaking protocol before data transfer can resume, a process typically taking between 200ms and 500ms. Movement may happen because a user is actively operating the device (e.g. typing); they may therefore be directly inconvenienced by the lack of connectivity (e.g. typing into a remote terminal). The optimisation of TCP performance in these circumstances is therefore important to the usefulness of Networked Surfaces. Note that Networked Surfaces do *not* suffer from high bit error rates; the 5Mbit/s prototype network offers a bit error rate of 10^{-10}. They also do not suffer from link-layer packet loss, as the link layer protocol used avoids losses due to collisions. For more information see [20].

In considering the implementation of a solution to this problem, a number of constraining factors are noted. Firstly, the solution must be applicable to a variety of devices using Networked Surfaces, which, as stated above, may not be internally modifiable. Even when programmable, the device may have limited CPU, memory and/or battery life, so minimal use of resources is important. The second factor to be considered is that the solution must also run on the device acting as IP-level gateway between the Networked Surface and other networks. A solution which demands high per-connection processing and memory requirements may therefore limit scalability of a Networked Surface to supporting many devices, and should be avoided. Finally, it is important to be able to communicate with unmodified corresponding hosts, as otherwise the solution would be impractical for reasons of deployability.

These factors (in particular the first two) dictate that a solution operating externally to TCP/IP is required. Such a solution, based in the link layer and known henceforth as the "smart link layer," is discussed in the next section, where the requirements above will have a strong role in guiding the design decisions made. Further discussion on advantages and disadvantages of a link layer approach when compared to solutions modifying TCP may be found in Section 7.

4. DESIGN OF THE SMART LINK LAYER

The "smart link layer" augments a traditional link layer design with limited awareness of transport-layer functionality, so that methods can be applied in order that TCP connections which have stalled during a disconnected period are promptly "kick-started" on reconnection, i.e. the flow of data is promptly resumed.

To illustrate the placement of the smart link layer in the network, a network connection involving a disconnecting link is depicted in Figure 2. This diagram shows the general case in which the disconnection is occurring on an unspecified link somewhere in the network path between the end-to-end TCP connection. In practice, it is expected that most disconnections will occur in the edge links, e.g. on the access network for a mobile device, such as a Networked Surface–enabled PDA. It is also possible that disconnections may be present on more than one link of the network, for example during peer-to-peer transmissions between two Networked Surface devices.

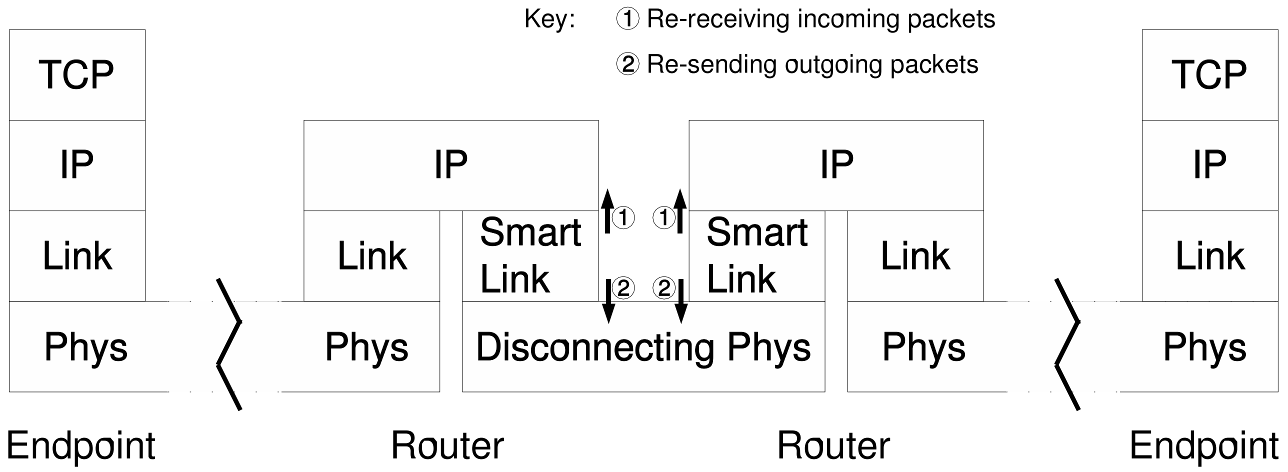

Figure 2: Network Connection including Disconnecting Link

For the purposes of the smart link layer solution, the key locations in the network path are at either end of the disconnecting link; these are represented as "Smart Link" in Figure 2. The smart link layer operating at these points may modify the network traffic in a number of ways. Network packets in transit for a given connection may be recorded, dropped or modified (the latter being more computationally expensive than the other two as checksums must be recalculated). Packets may also be inserted at these points in one of two ways; as shown in the diagram, inserted packets may either be placed in the outgoing queue for the disconnecting link (denoted hereon as "re-sending" packets), or in the incoming queue as if they had just been received on that link (denoted hereon as "re-receiving" packets).

One key advantage of a smart link layer solution is that disconnection and reconnection of the link may be automatically detected at these points and may be used to trigger events. This is not true, in general, for an end-to-end protocol, which may not be able to easily determine link states for individual links on the network path used. In order for the smart link layer to force a reaction from the end-to-end TCP engines at reconnection time, the obvious method is to insert one or more packets into the network at the moment when reconnection is detected.

The design space for solutions involving link layer insertion of packets on reconnection is discussed below. The priority motivating the particular designs discussed is the minimal use of resources such as processing and memory, for the reasons described in the previous section.

4.1 Parameters for Packet Insertion

Irrespective of whether inserted packets are re-sent or re-received, there are a number of other issues to be solved. In particular, there is the issue of how the inserted packets are constructed, and the issue of how many packets are inserted.

For the issue of the construction of inserted packets, there are two possibilities. These packets may either be copies of packets that have already passed through the network (and were recorded by the link layer), or they may be constructed afresh by the link layer, presumably using information gathered from monitoring previous traffic. While the latter approach gives the maximum flexibility to the smart link layer, allowing it to choose precisely the content of its inserted packets to potentially force the quickest TCP recovery, it is also resource-intensive in that the inserted packets must be constructed and checksummed by the link layer itself.

In contrast, although the use of pre-transmitted packets provides less flexibility, it is also much simpler to implement. Relatively little "knowledge" of TCP/IP is required to be duplicated at the link layer, and the computation required to construct valid TCP/IP packets is avoided. In addition, the use of copying assumes less about the particular TCP implementation being used, in that any unknown options or parameters present in packets are simply passed on without modification. For these reasons, this research focuses on the use of copied packets retransmitted at reconnection time.

The next consideration is the number of packets that should be repeated on reconnection. While it is possible to have a policy such as link layer retransmission of all unacknowledged data on reconnection, this would require a large amount of buffer space, may result in a large waste of bandwidth (as packets may be retransmitted needlessly), and may interfere with TCP's own retransmissions. For these reasons, only one packet per TCP connection is buffered. Under this policy, a quick "back of the envelope" calculation shows that the overhead is not burdensome; an access router attached to a disconnecting link would only need 3kb of storage per TCP connection; with 100 edge nodes each using 10 active TCP connections, this would result in a requirement of 3Mb of extra RAM to implement smart link layer functionality.

The one-packet policy still leaves the possibility for duplicates of the buffered packet to be inserted on reconnection. Duplicate packets may be useful as they can cause the receiving TCP state machine to be forced into a fast retransmit mode [15], since the packets will cause duplicate acks to be received. The benefit of repeating packets is therefore examined experimentally in this research.

4.2 Re-receiving Packets

Re-receiving means that the packets are inserted into the incoming queues of the hosts on either side of the disconnecting link. While such packets will not traverse the disconnecting link, they may be transmitted across other parts of the end-to-end network. The advantage of this approach is that, in the likely case that the disconnecting link is at the periphery of the network, one of the TCP correspondents will receive its kick-start very quickly, as no network latency will be incurred. A key disadvantage of re-reception is that such packets are by definition never going to provide the receiving TCP engines with any data they have not seen before. These packets are therefore confined to repeating old data, old acks, and/or old window advertisements.

The choice of data to re-receive is determined by that data which is most likely to cause TCP to immediately send out more data, and initiate recovery mechanisms to re-establish data flow as quickly as possible. Since these mechanisms are governed largely by the reception of acks, the best information to re-receive is obviously the highest ack already received. To achieve this, it suffices to compare each packet passing through the link layer with the packet in the buffer, and replace the buffered packet if the new packet has a higher ack number.

Finally, in order to ensure that idle connections are not needlessly kick-started, it is sensible to only re-receive on reconnection if there was a send attempt on that connection during the disconnected period. (Other methods of monitoring connection activity, such as idle timers, could also be used.)

4.3 Re-sending Packets

The smart link layer is also capable of re-sending packets at reconnection time, by inserting them in the outgoing queue for the disconnecting link. Re-sending has the disadvantage that some network latency is bound to be incurred before either TCP engine receives its kick-start. However, the advantage is that the re-sent packets may include new data, new acks, and/or new window updates.

Unlike in the re-receiving case, where the choice of which packet to buffer is simple (as there is no possibility of providing new data, merely repeating old data), care must be taken in the choice of a buffered packet for re-sending. In order to facilitate the quick restarting of TCP traffic, there are two obvious criteria for buffering, and two more subtle criteria. The obvious criteria are that the packet should have the highest acknowledgement number sent thus far, for the same reasons as for re-reception. The packet should also have the lowest unacknowledged sequence number, as this is the next “in-order” data that the remote TCP engine is expecting, and is therefore most likely to promote quick recovery of the TCP traffic flow. It must be noted that this criterion relies on the ability to monitor the current acknowledgement number, which must be found by scanning packets going in the opposite direction. This would not work if the acknowledgements take a different network path to the data, however, this is unlikely to be the case, as the disconnecting link would most often be on the access network for one of the endpoints.

More subtly, the buffered packet should be chosen as the longest length packet, and the one advertising the largest receive window. These criteria are relevant when considering packets retransmitted during the disconnected period. The former criterion relates to Nagle’s algorithm [17], which states that only one packet which is smaller than the MTU should be unacknowledged at any time; other data should be queued at source rather than sent as further small packets. A corollary of using this algorithm is that, when timeout and retransmission occurs on small packets, the retransmitted packet may be longer than the original packet, as the retransmitting TCP will put as much data as possible in this packet, up to the MTU. A larger packet is obviously a better choice for buffering at the smart link layer, as most or all of the outstanding data can be sent immediately at reconnection, rather than incurring multiple round trip times for the data to be transferred.

The criterion of having the largest receive window advertisement is also related to retransmitted packets. During a disconnection period, it is likely that a host which has received but not processed some number of packets will be able to conduct some or all of this processing and therefore be able to send larger receive window advertisements in subsequent retransmissions.

In summary, for the re-sending algorithm, a packet is placed in the per-connection single packet buffer, if it:

1. Has a larger acknowledgement number than the current buffered packet, or

2. Has the same acknowledgement number but an older unacknowledged sequence number, or

3. Has the same acknowledgement and sequence numbers, but a longer length, or

4. Has the same acknowledgement number, sequence number and length, but advertises a larger receive window.

Finally, in order to avoid disturbing idle connections with this policy, re-sending on reconnection only occurs if there is unacknowledged data outstanding.

5. TESTBED EXPERIMENTS

To analyse the effect of the various link layer methods described above, an experimental setup using a simulated network channel was constructed. This allowed tests to be run using real traffic, but with precise control over the network connectivity, and also provided a platform for implementation and bug-fixing of the smart link layer algorithms.

5.1 Experimental Setup

To simulate a disconnecting channel, the Linux “ethertap” driver was used. This allows network packets to be routed to a user-level program, which simulates the lossy channel, and implements the send and receive components of the smart link layers. This setup is illustrated in Figure 3.

To implement the simulated channel, a two-state Markov model was used, with one state having 100% reliability and the other state having 0% reliability. The mean time spent in each state was configurable to allow different channel characteristics to be simulated. This is illustrated in Figure 4. The period spent in each state was modelled by uniform random distributions, between half and one-and-a-half of the desired means; this ensured that the results were not subject to interaction between TCP timers and the channel timing.

In addition to implementing this model, the simulation program was also made to reverse the IP addresses and TCP port numbers of all packets, thereby “mirroring” packets so that the local TCP/IP

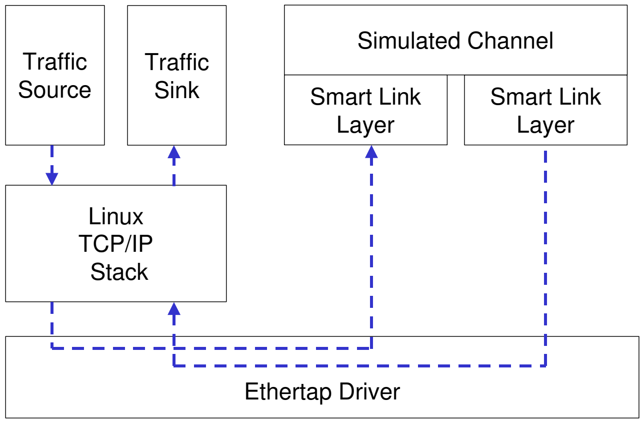

Figure 3: Ethertap Experiment Setup

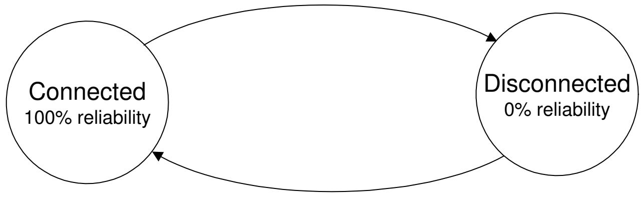

Figure 4: Simulated Channel Markov Model

stack treats the packets as incoming rather than outgoing. This allows networking tests to be performed on a single machine. Using this arrangement, networks with various connection and disconnection patterns can be simulated, and various smart link layer algorithms can be tested. One potential disadvantage of this arrangement is that the sender, channel model and receiver are all competing for CPU and memory bandwidth, however, this is not a factor in the tests below, since the behaviour being monitored is the time taken to recover from disconnected periods, during which no traffic is being sent (apart from retransmissions), and all elements of the testing machine are idle, with the sender stalled while waiting on TCP timers.

5.2 Optimisations Tested

From the design space discussed in the previous section, five potential optimisations were identified for experimental evaluation. These optimisations, known as "smartlvls," are outlined below:

Smartlvl 0 This is the control case, and represents unaided TCP.

Smartlvl 1 Re-receiving is used as defined in Section 4.2, and the buffered packet is re-received once upon reconnection.

Smartlvl 2 As for smartlvl 1, but the packet is re-received five times on reconnection.

Smartlvl 3 Re-sending is used as defined in Section 4.3, and the buffered packet is re-sent once upon reconnection.

Smartlvl 4 As for smartlvl 3, but re-sending five times on reconnection.

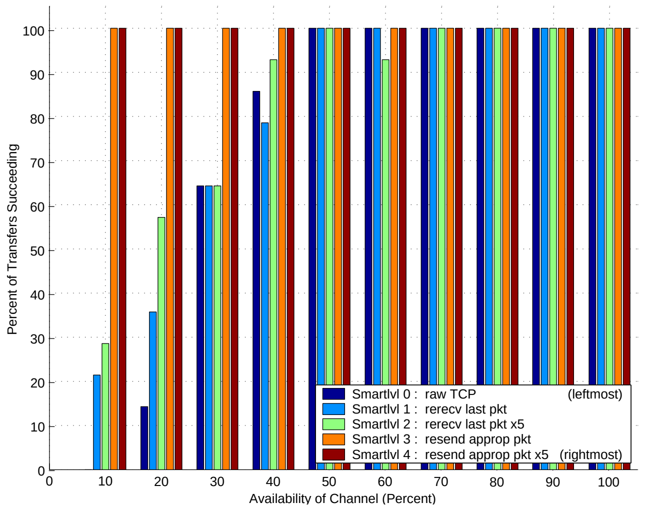

Figure 5: TCP File Transfer Tests over Simulated Disconnecting Channel

5.3 Results

In order to determine the relative performance of the various optimisations outlined above, timed bulk transfers were sent using the disconnecting channel described above. These were conducted with the following parameters.

- The transfer size was 50Mbyte.
- Mean "uptime" was set to 0.5s, while mean "downtime" was varied from to 0.0s to 4.5s to simulate channel availabilities from 10% to 100%.
- Fourteen trials were conducted at each of the ten availability levels and five smartlvls.
- The success or failure of each trial, and the time it took if successful, was noted.

As Figure 5 shows, the trials all succeed when availability is high. For availabilities below 50%, smartlvls 0, 1 and 2 begin to fail. Smartlvl 0 (raw TCP) fails most quickly, and at 10% availability experiences no successful transfers. Smartlvls 1 and 2, which use re-reception of packets, show some improvement, but smartlvls 3 and 4 are the definite "winners," with 100% of transfers completed successfully, even when the channel is only available 10% of the time.

Figure 6 shows the mean transfer time for the successful transfers, with an unmarked line indicating the minimum transfer time, which would occur if the channel were used optimally. As expected, unaided TCP degrades most quickly, and smartlvls 1 and 2 exhibit some improvement. Smartlvls 3 and 4, however, stay very close to the optimal line, providing good performance even on a channel with 10% availability.

This is further illustrated in Figure 7, which shows the low standard deviation of the trials, as compared to those of unaided TCP. This plot also allows the observation that, even at 80% or 90% availability, unaided TCP has already diverged from optimal performance,

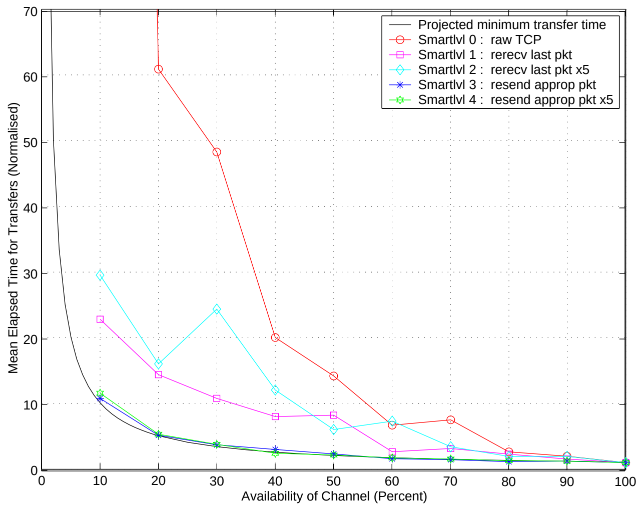

Figure 6: Mean Duration of Successful TCP File Transfer Tests over Simulated Disconnecting Channel

with degradations of about 100% and 50% respectively from the optimal case.

5.4 Analysis using TCP Traces

To further explore the behaviour above, traces of file transfers were taken with various smartlvls. These are shown in Figures 8 to 11. (Figure 1 showed a trace for unaided TCP, and includes a key useful for interpreting the traces.)

The traces for smartlvls 1 and 2[1] show that TCP does not respond immediately when receiving repeated acks. Although repeated acks are used as a signal to TCP to start fast-retransmit of a packet, this is not successfully invoked in either case. This can be explained by noting that the fast-retransmit mechanism is designed to be used *before* retransmission takes place due to timeout. If such a timeout has already occurred, then the congestion window has been reset to one packet. Hence, re-receptions of an old ack do not cause any response, as the window is not advanced by this event.

The traces for smartlvls 3 and 4,[2] on the other hand, show that TCP is immediately restarted after reconnection. This is because the packet re-sent on reconnection is chosen so that it sends unacknowledged data. When this packet is acknowledged, the congestion window is opened and slow-start proceeds as normal. Resending five packets at a time with smartlvl 4 does not seem to have any added effect; although they may cause multiple acks of that packet, the fast-retransmit mechanism is not useful in this case, due to the congestion window being reset as described previously.

The conclusion of these experiments is that a “smart link layer” employing re-sending of a well-chosen packet on reconnection can improve the performance of TCP on disconnecting channels. Whereas unaided TCP experiences bad performance even when the channel

[1]The five individual plus marks for the five re-receptions used in smartlvl 2 are not distinguishable at this scale, and look like a single plus mark.

[2]The five individual circles for the five re-sent packets used in smartlvl 4 are not distinguishable at this scale, and look like a single circle.

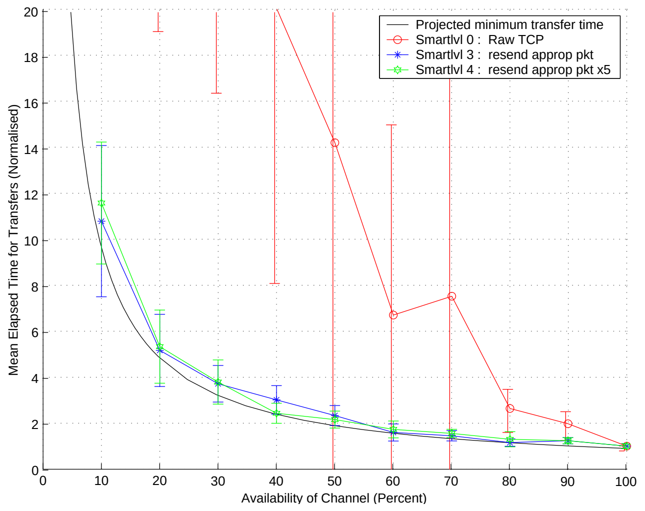

Figure 7: Detail of Figure 6 for Selected Smartlvls, with Standard Deviations

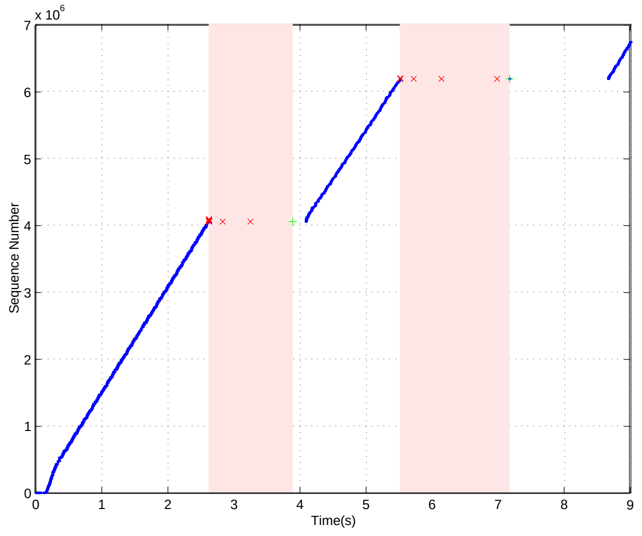

Figure 8: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 1)

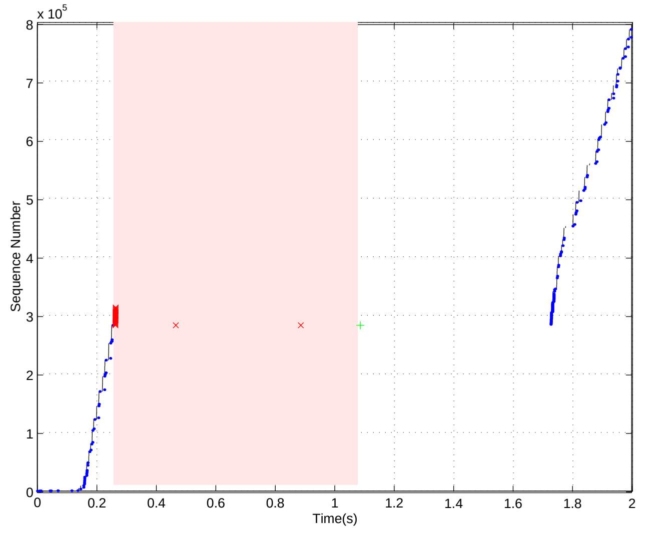

Figure 9: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 2)

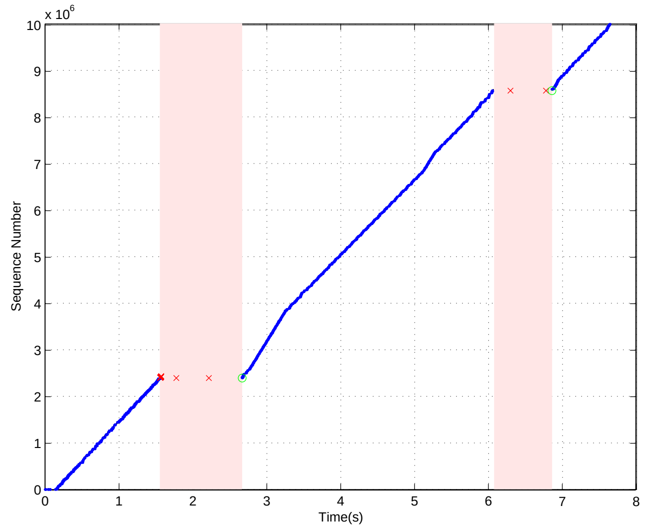

Figure 10: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 3)

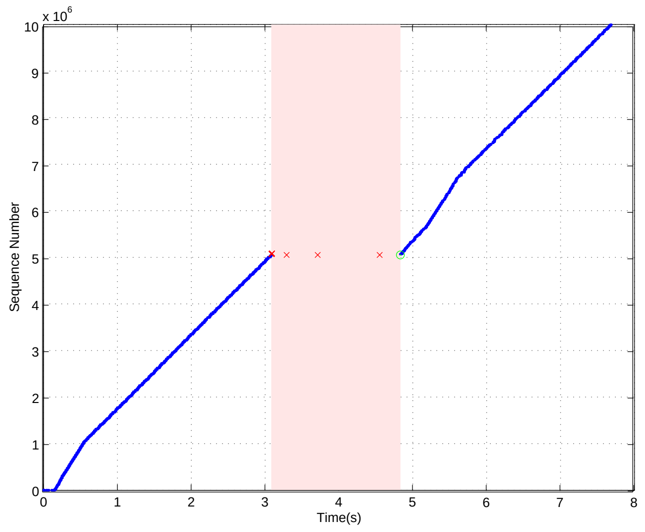

Figure 11: TCP File Transfer with Disconnection and Smart Link Layer (Smartlvl 4)

is 90% available, the use of the smart link layer gives nearly 100% channel utilisation at availabilities down to 10%.

6. EVALUATION OF THE SMART LINK LAYER ON THE PROTOTYPE NETWORKED SURFACE

This section describes the effects of the smart link layer described above, when deployed in the context of the prototype Networked Surface. The smart link layer was deployed at both the device and the node acting as the Networked Surface IP gateway, i.e. on both sides of the disconnecting link. Two types of test were then conducted. The first was similar to the experiments described in the previous section, and evaluated the bulk transfer performance of TCP. The second characterises the interactive response of TCP over a disconnecting Surface network, by using the Virtual Network Computing (VNC) remote desktop tool.

In order to conduct these tests, the prototype Networked Surface hardware was augmented with a testing mode to allow a device to disconnect at random intervals, with an adjustable mean connected period. The "uptime" and "downtime" of the network was also recorded, so that its availability could be calculated.

6.1 Performance for Bulk Transfers

In order to examine TCP performance for bulk transfers over a disconnecting Networked Surface, experiments were conducted according to the following parameters:

- The Networked Surface prototype was configured to provide a 1Mbit/s network.
- Disconnections were caused at various rates, producing various channel availabilities.
- Smartlvls 0 and 3 were used; smartlvl 0 is the unaided TCP case, and smartlvl 3 is the best-performing smart link layer optimisation, as shown previously.

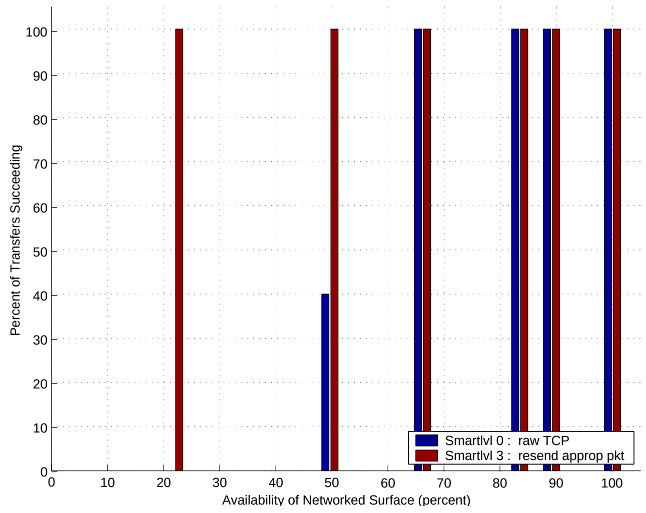

Figure 12: TCP File Transfer Tests over Disconnecting Networked Surface

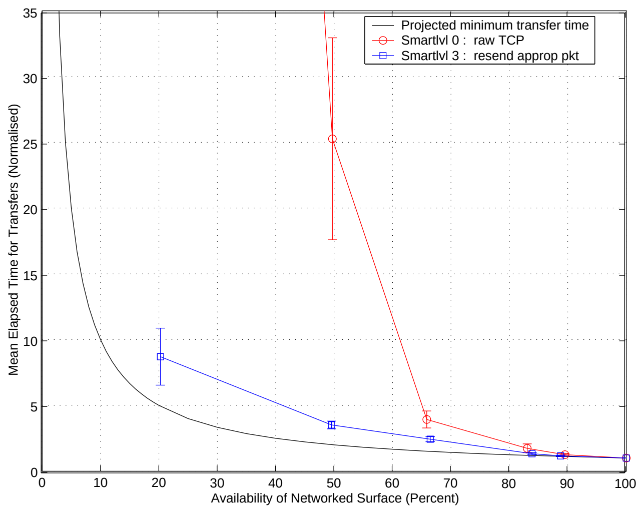

Figure 13: Mean Duration of Successful TCP File Transfer Tests over Disconnecting Networked Surface

- Ten transfers of 5Mbyte were conducted at each availability and smartlvl, and the elapsed times were recorded.

As Figure 12 shows, the smart link layer completed 100% of transfers, down to an availability of 23%, while unaided TCP did not reliably transfer the data at 50% availability or less. Figure 13 illustrates that the smart link layer stays relatively close to the "ideal" transfer time, even down to 20% availability. Unaided TCP has twice the overhead of the smart link layer at 65% availability, and very bad performance at lower availabilities.

6.2 Interactive Performance

While bulk transfer performance is important for some applications, the user of a networked device may also wish to communicate interactively. Examples of interactive applications are remote terminal programs, web interfaces, and real-time multimedia applications such as streaming audio or video. Such applications may not stress the bandwidth of the network available, so the tests in Section 6.1 are not necessarily applicable in this case. What is applicable, however, is *synchronisation*, i.e. the need for a local interface and the remote application to be representing the same state as much as possible. An example of bad synchronisation is when a user clicks on a webpage link, but only after a number of seconds does the page change to reflect this action. Another example might be a remote desktop mouse icon not following the local mouse icon closely while it is being moved.

In order to test the smart link layer's benefits for interactive applications, a quantitative metric must be found for synchronisation performance. As described below, the "frame rate" of a remote desktop application is one such metric.

6.2.1 Testing Method

The VNC [19] remote desktop application allows a user of one computer to interact with a remote computer, by "forwarding" the remote display across a network, and similarly relaying keyboard and mouse input. The VNC protocol operates as follows. When the client connects to the server, it issues an "update request" to that server. The server responds by waiting until its display differs from its record of the client's display, and then sending a "framebuffer update" containing changes to the client's display. On receiving this update, the client applies it and immediately sends another "update request."

Since the server only sends updates in response to requests from the client, the protocol is self-clocking. The requests and updates are sent over TCP, which retransmits the data if it arrives corrupted or is lost in transit, providing the guarantee that all messages eventually get delivered correctly (if channel conditions permit).

Due to the protocol outlined above, only one update is sent at any time. This implies that, for small updates, the frame rate achieved is determined by the latency of the TCP connection used and not by its bandwidth. The frame rate is therefore a good measure of interactive performance.

6.2.2 Experiments and Results

In order to gather frame rate data, tests were conducted using to the following parameters.

- A VNC desktop session was set up on a machine on the network.
- A program was run on the remote desktop, which caused a small dot to appear and disappear at regular intervals. The effect of this program was to cause the display to require a small update every 200ms.
- The Networked Surface was configured to have various availabilities, as described previously.
- Smartlvls 0 and 3 were used.
- For each test, the VNC viewer program was run on a computer using the Networked Surface for 100s, and record was made of the number of updates received over this time period.

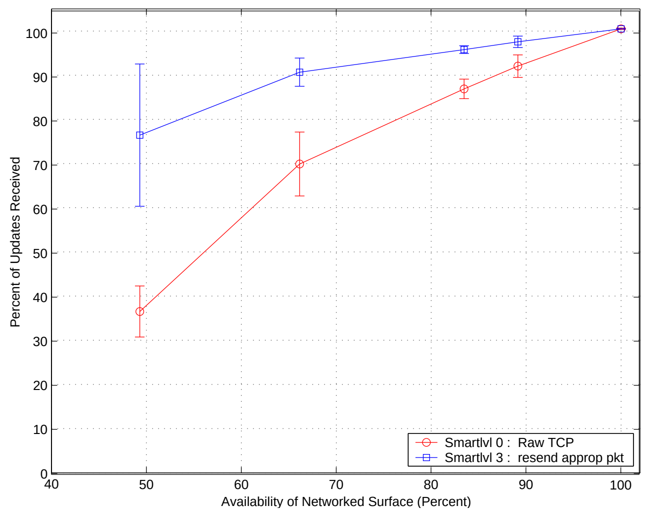

Figure 14: VNC Interactivity Tests over Disconnecting Networked Surface

- Ten tests were conducted for each smartlvl and availability.

Figure 14 shows the results of these tests. The smart link layer is shown to perform well, providing 80% of the frame updates even when the channel availability is halved, as opposed to 40% for the unaided TCP case. These results show that interactive performance of TCP over the Networked Surface channel is significantly improved when a smart link layer is used.

7. COMPARISON OF THE SMART LINK LAYER WITH OTHER SOLUTIONS

The link layer–based nature of the smart link layer allows it to enjoy several advantages when compared with solutions involving the modification of IP, ICMP and/or TCP that were described in Section 1.1. Link layer solutions can be independent of the TCP implementation used, and can therefore be deployed without concern for the devices' particular TCP implementations. In contrast, a TCP-based solution would have to be added to many different TCP implementations that are in use, some of which may be hard or impossible to modify (e.g. hardware-based TCP implementations). Also, link layer solutions may only affect nodes local to the disconnecting link. Deployment of such solutions is therefore easily carried out concurrently with deployment of the disconnecting network type.

In addition, the smart link layer's particular attributes allow it to enjoy some further advantages. As it is very lightweight, it can be deployed in modest hardware as part of a NIC for mobile computing devices, thus allowing it to avoid use of the scarce CPU and memory resources of those devices. Also, since it uses only a small amount of network bandwidth, it is relatively easy to secure it against malicious use, e.g. for denial of service attacks. Furthermore, because the smart link layer operates unilaterally, it does not require the use of authentication methods that might be necessary in solutions which communicate state information, e.g. for signalling a disconnected state to the TCP endpoints. Lastly, since the smart link layer only resends traffic that has already been sent, it is easy to see that no data security issues are created when using this solution.

However, there are also disadvantages to using link layer solutions, some of which are relevant to the smart link layer. To begin with, such solutions may experience bad interactions between TCP's retransmissions and retransmissions at the link layer [11]. The smart link layer, however, does not try to usurp TCP's role of ensuring data delivery; the retransmitted packets are solely used to kick-start TCP's own recovery mechanisms, and the fact that the data contained in this packet is delivered is a (welcome) side-effect. Bad interactions between retransmissions are therefore avoided.

Another disadvantage of link layer solutions is that they do not prevent TCP transmitting during the disconnected period. This may be seen in the first diagram, Figure 2, which shows that a full window of packets is transmitted fruitlessly during disconnection, and a periodic retransmission of the first packet in this window also occurs. In contrast, a TCP implementation which is "aware" of disconnection would halt transmission whilst in a disconnected state, thus saving network bandwidth.

The next disadvantage is that, not being integrated with TCP, it is not certain that a link layer solution will perform well with every version of TCP currently deployed, or with future versions of TCP that may become available. The smart link layer solution may incur this problem, however, as it only retransmits a single packet on reconnection, the network overhead imposed is low. Therefore, if the smart link layer solution fails to kick-start a given TCP connection, the network will not be significantly burdened, and TCP will simply assume its normal, if suboptimal, behaviour.

The final disadvantage to be highlighted is that the use of end-to-end encryption [1] may hinder or even disable link layer solutions which rely on being able to "sniff" packets. To handle encrypted TCP connections, the smart link layer could simply buffer the most recent packet per source and destination IP addresses, and retransmit this on reconnection. However, this technique would not cope with multiple TCP connections between two IP addresses (which it would be unable to distinguish), and may not send result in the best kick-start packet being buffered.

It must be noted that the smart link layer does *not* attempt to bypass the slow-start procedure of TCP [14], nor does it try to induce a raising of the congestion window. This policy is in contrast to many of the link layer solutions presented for TCP problems (and described in Section 1.1), which attempt to keep the congestion window wide despite bad channel characteristics. This difference is largely due to the timescales for which the solutions are designed; single packet losses happen on a microsecond scale, while disconnections may last a number of seconds. Also, during disconnection, devices may be moved to a different network; this may cause the congestion characteristics to change, so a slow-start is appropriate to discover the correct new value of the congestion window.

Finally, the relevance of the smart link layer to the current IETF proposals is now discussed. The smart link layer technique is in accord with the PILC working group's recommendation [16] for disconnecting networks to buffer and resend one or more of the packets arriving during the disconnected period. The validity of that technique is confirmed experimentally here, and different buffering criteria and retransmission techniques are examined.

Both PILC and the TRIGTRAN [10] proposal indicate that a preferred solution for disconnection handling is the end-to-end notification of disconnections (and other network events) so that TCP (or another transport layer protocol) can react appropriately. These proposals would avoid the disadvantages highlighted above, however, they require much more intrusive modifications of both the nodes attached to the disconnecting network and the edge nodes, as well as cooperation between these nodes, in order to solve this problem. This paper has shown that it is possible to work around the disconnecting network problem, at least in some circumstances. The techniques used in this research may prove a useful "stop-gap" solution for use while more thorough solutions involving modification of many entities in the network can be standardised and deployed.

8. CONCLUSIONS AND FUTURE WORK

This paper has presented the "smart link layer" solution for the problem of TCP's bad performance in the face of disconnecting links. Various algorithms repeating packets on reconnection were explored using an experimental testbed. Re-sending packets on reconnection proved to be more effective than re-receiving packets, and repetition of packets was shown to offer no additional improvement.

The best-performing solution was shown to achieve nearly 100% utilisation of a disconnecting channel, at availabilities down to 10%. Unaided TCP, on the other hand, shows a 50% degradation at 90% availability, and fails to complete some of the tests below 50% availability. The effect of this solution was also tested on the prototype Networked Surface, with good results shown for both bulk transfers and interactive traffic. In summary, this paper has shown the effectiveness of a simple and lightweight link layer–based solution for enhancing TCP performance in the face of disconnecting networks.

Future work involving the smart link layer may include testing of the link layer solution in a broader variety of circumstances. This may include tests involving connections over high-bandwidth high-latency networks, and tests where disconnection occurs in the middle of the network, or at multiple places in the network. Next, the current solution relies on the "kick-start" packet not being lost; this is valid given the assumption that the network is reliable when connected. In order to avoid this assumption, the smart link layer could be made to monitor the round-trip time of the connection, and then automatically retransmit the kick-start packet if it does not cause a reply within a suitable time. Thirdly, this solution might be ported to other situations where disconnected periods occur. For example, Mobile IP handoffs may take a significant time, during which the network appears to be disconnected at the TCP level. Fourthly, the use of this solution in the presence of encryption might be explored. Lastly, the viability of this solution with other transport-level protocols, e.g. SCTP [22], may be examined.

9. ACKNOWLEDGEMENTS

The authors are grateful to Frank Hoffmann, Andy Hopper and Mike Addlesee of the Networked Surfaces project at the LCE for indirect contribution to this work. The authors would also like to thank James Weatherall for helpful discussions concerning the operation of VNC, Mike Hazas and Tom Kelly for proofreading this paper, and John Wroclawski and the anonymous reviewers for their helpful comments.

This research was funded by the Schiff Foundation of Cambridge and by AT&T Laboratories Cambridge.

10. REFERENCES

[1] R. Atkinson. IP encapsulating security payload (ESP). RFC 1827, August 1995.

[2] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani, and R. D. Gitlin. AIRMAIL: A link-layer protocol for wireless networks. *ACM Wireless Networks*, 1(1):47–60, 1995.

[3] A. V. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 1995 International Conference on Distributed Computing Systems*, pages 136–143, 1995.

[4] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan. Improving performance of TCP over wireless networks. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, 1997.

[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[6] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4):469–481, 1995.

[7] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communication Review*, 27(5):19–43, 1997.

[8] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE Journal of Selected Areas in Communications*, 13(5):850–857, 1995.

[9] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback-based scheme for improving TCP performance in ad-hoc wireless networks. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, pages 472–479, 1997.

[10] S. Dawkins, C. E. Williams, and A. E. Yegin. Framework and requirements for trigtran. Internet-Draft `draft-dawkins-trigtran-framework-00.txt`, February 2003.

[11] A. DeSimone, M. Chuah, and O. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proceedings of GLOBECOM '93*. IEEE, 1993.

[12] S. Goel and D. Sanghi. Improving TCP performance over wireless links. In *Proceedings of TENCON '98*, pages 332–335. IEEE, December 1998.

[13] G. Holland and N. H. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking*, pages 219–230. ACM/IEEE, August 1999.

[14] V. Jacobsen. Congestion avoidance and control. In *Proceedings of SIGCOMM '98*. ACM, 1988.

[15] V. Jacobsen. Modified TCP congestion avoidance algorithm. Email on `end2end-interest` mailing list (`ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt`), April 1990.

[16] P. Karn, Editor. Advice for internet subnetwork developers. Performance Implications of Link Characteristics (PILC) Working Group: Internet-Draft `draft-ietf-pilc-link-design-13.txt`, February 2003.

[17] J. Nagle. Congestion control in IP/TCP internetworks. RFC 896, January 1984.

[18] C. Parsa and J. J. Garcia-Luna-Aceves. Improving TCP performance over wireless networks at the link layer. *ACM Mobile Networks and Applications*, 5(1):57–71, March 2000.

[19] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, February 1998.

[20] J. Scott. *Networked Surfaces: A Novel LAN Technology*. PhD thesis, University of Cambridge, 2002.

[21] J. Scott, F. Hoffmann, G. Mapp, M. D. Addlesee, and A. Hopper. Networked Surfaces: A new concept in mobile networking. *ACM Mobile Networks and Applications*, 7(5), October 2002.

[22] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960, October 2000.

[23] N. Vaidya, M. Mehta, C. Perkins, and G. Montenegro. Delayed duplicate acknowledgements: A TCP-unaware approach to improve performance of TCP over wireless. Technical report, Computer Science Department, Texas A&M University, 1999.