Protocol Enhancements for Intermittently Connected Hosts

Simon Schütz, Lars Eggert, Stefan Schmid and Marcus Brunner
NEC Europe Ltd, Network Laboratories, Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
{schuetz,eggert,schmid,brunner}@netlab.nec.de

ABSTRACT

Internet users are increasingly mobile. Their hosts are often only intermittently connected to the Internet, due to using multiple access networks, gaps in wireless coverage or explicit user choice. When such hosts communicate using the current Internet protocols, intermittent connectivity can significantly decrease performance and even cause connections to fail altogether. This paper experimentally measures the behavior of Internet communication across a dynamically changing, intermittently connected path. An analysis of the experimental results finds that address changes together with transport-layer timeout and retransmission behaviors are the main limiting factors. Based on these experimental results, this paper proposes a solution that combines the Host Identity Protocol (HIP) with two new protocol enhancements, the TCP User Timeout Option and the TCP Retransmission Trigger. Detailed experiments with HIP and a prototype implementation of these protocol enhancements show that they tolerate address changes and arbitrary-length disconnections while significantly increasing performance under intermittent connectivity to within 86-96% of a scenario with constant connectivity.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols – *protocol architecture, protocol verification*

General Terms

Measurement, Performance, Design, Experimentation

Keywords

Communication system performance, Internet, mobile communication, intermittent connectivity, disruption tolerance, transport protocols.

1. INTRODUCTION

Within the last decade, nomadic networking has become common. Users frequently connect to the Internet with mobile devices such as notebooks, handhelds or mobile phones, using a wide variety of wireless and wired access technologies, including cellular networks (GSM, UMTS), wired or wireless local area networks, or other broadband connections (DSL, cable networks.) In between connected periods, these devices frequently experience disconnected periods – either voluntarily, *e.g.*, based on user preference, or involuntary, *e.g.*, due to bad coverage. This *intermittent connectivity* is a key characteristic of nomadic networking.

Figure 1 shows an example of networking under intermittent connectivity: a mobile user traveling by car [6]. At the beginning of the trip, the user connects to the Internet via wireless access network *A* (on the left.). After leaving the range of access network *A* at t_1, a period of disconnection begins. After a while, wireless access network *B* (on the right) becomes available at t_2 and Internet connectivity is restored. After leaving the range of *B* at t_3, another period of disconnection begins (not shown in Figure 1) that will end when the user reaches the next access opportunity.

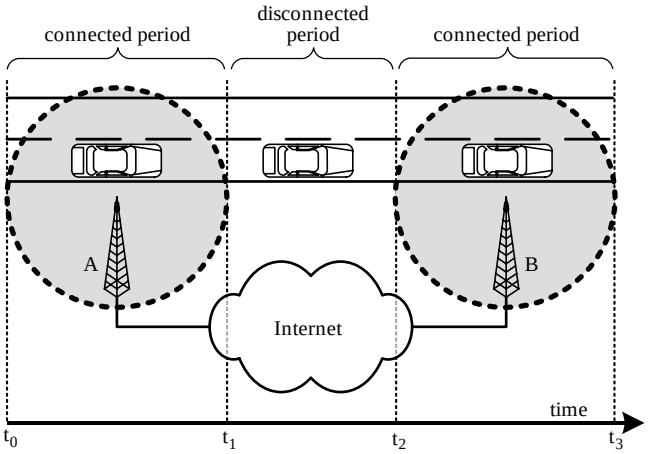

Figure 1. Example scenario with intermittent connectivity.

Many Internet protocols, including the Internet Protocol (IP) [1] and the Transmission Control Protocol (TCP) [2], do not operate well under intermittent connectivity and in the presence of host mobility. When they were designed, most hosts were stationary and interconnected through permanent, wired links. Support for highly mobile hosts that frequently switch access technologies was not a design objective. These limitations of the Internet protocols are also apparent in two related scenarios: *interplanetary* and *ad hoc* networks.

Communication across *interplanetary* networks [11] must also tolerate connectivity disruptions, *e.g.*, due to the movement of celestial bodies. Additionally, interplanetary end-to-end delays are orders of magnitude higher than for earthbound communication, causing traditional protocols based on an end-to-end feedback loop to fail. This second characteristic does not apply to earthbound communication across intermittently connected paths discussed in this paper – when connectivity exists, end-to-end delays are comparable to the current Internet.

Ad hoc networks [12] are a different kind of disconnection-prone networking environment. In an *ad hoc* network, both hosts and routers can be mobile, making routing significantly more difficult. Furthermore, communication in *ad hoc* networks is usually limited by other constraints, such as energy consumption or node processing power. Compared to the intermittently connected scenarios that are the focus of this paper, *ad hoc* networks require significantly different mechanisms to address their specific constraints. The result is often a protocol suite that significantly differs from the current Internet. However, scenarios

that integrate *ad hoc* networking with the Internet may use mechanisms similar to the ones described in this paper.

This paper focuses on the arguably more common mobile networking scenario of mobile hosts that intermittently connect to the Internet through different access networks. Unlike *interplanetary* and *ad hoc* networks, which demand radically new approaches to address their specific characteristics, this paper shows that evolutionary enhancements to existing Internet protocols can efficiently support networking under intermittent connectivity. Extending existing protocols aids deployment of the solution and improves the operation of today's applications and protocols without requiring modifications.

Section 2 analyzes the shortcomings of the current Internet protocols in intermittently connected scenarios. It identifies three main reasons that significantly decrease the performance of standard TCP connections across intermittently connected paths and can even lead to complete connection failures: address changes, disconnection duration and retransmission behavior.

This paper addresses all three limitations and proposes TCP enhancements that enable efficient operation across intermittently connected paths. The improved TCP does not abort connections during disconnected periods and efficiently transmits data during any connected periods. The proposed solution combines two new TCP extensions, the TCP User Timeout Option [8] and the TCP Retransmission Trigger [9], with the Host Identity Protocol (HIP) [3]:

The TCP User Timeout Option [8] prevents connection aborts due to user timeout expiration and allows hosts to tolerate extended periods of disconnection. The new option allows mobile hosts to agree on individual user timeouts for specific connections.

The TCP Retransmission Trigger [9] improves the overall performance of TCP across an intermittently connected path. System- and environment-specific retransmission triggers that detect network reconnections allow TCP to better utilize periods of connectivity.

The Host Identity Protocol (HIP) [3] prevents TCP connections from aborting due to changes of IP addresses. HIP decouples the two functions that IP addresses serve in the current Internet: they both identify hosts and describe their topological location. The present overloading of these two functions significantly limits the evolution of the Internet in areas as mobility, multi-homing and security. HIP introduces a shim layer between the traditional transport and network layers. Transport-layer connections bind to logical *host identities* instead of IP addresses and HIP transparently maps host identities to IP addresses.

Section 2 analyzes the shortcomings of the current Internet protocols across intermittently connected paths. Based on these findings, Section 3 describes how a combination of the two new TCP enhancements and HIP address these limitations. Section 4 experimentally evaluates the performance of the enhanced protocols in multiple scenarios and finds that they increase performance under intermittent connectivity to within 86-96% of a scenario with constant connectivity. Section 5 compares the proposed solution against other related work, and Section 6 discusses future work in this area and concludes this paper.

2. PROBLEM ANALYSIS

In the Internet, most communication occurs "end-to-end" – *i.e.*, directly between two end hosts, without involving explicit intermediaries. The network path between two immobile hosts does not frequently change. With static IP addressing, the first and last hops of a path are fixed and any changes are due to routing events inside the network. Furthermore, connectivity disruptions are infrequent.

When hosts become mobile, this is no longer the case. When mobile hosts switch between access networks, their topological location in the network usually changes as well. In the extreme case, no part of the path between two nodes will remain the same after one node changes access networks. Additionally, disruptions in connectivity are frequent, either due to link-layer effects (*e.g.*, wireless coverage) or by user choice.

This section analyzes the behavior of two fundamental Internet protocols – IP and TCP – across intermittently connected paths and discusses their shortcomings. Three different aspects of intermittently connected paths cause problems for the current protocols: IP address changes, disconnection duration and retransmission behavior. The first two cause established connections to abort prematurely, *i.e.*, before all data has been exchanged, whereas the latter reduces performance.

2.1 IP Address Changes

Switching between access networks – due to physical mobility or not – usually also changes a host's topological location in the network. Because it is no longer reachable at its old address, a host must start to use a different IP address that identifies its new network attachment point.

Many Internet transport protocols, including TCP, use IP addresses (often together with other identifiers such as port and protocol numbers) to uniquely and permanently identify the endpoints of transport-layer connections during their lifetime. When a host changes its IP address, the local endpoints of its established TCP connections thus change as well. Whereas some newer transport protocols such as SCTP [4] support changing the endpoints of established connections and can continue to operate across IP address changes, TCP does not.

Consequently, TCP implementations must abort established connections when IP addresses change. This error condition is visible to applications and may cause them in turn to abort or behave undesirably. At the very least, applications must roll back to a defined state, establish new connections and retransmit. This decreases performance and increases application complexity.

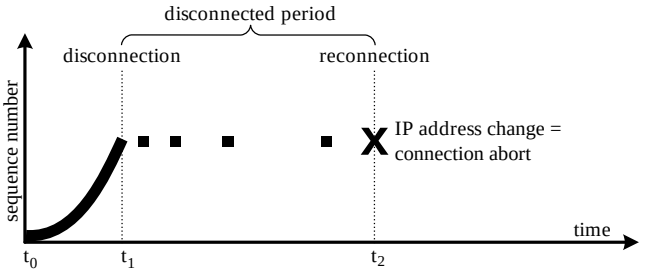

Figure 2. Connection abort caused by IP address change.

Figure 2 shows an illustration of this issue. First, the host transmits normally during the initial connected period between t_0 and t_1. When the disconnection occurs at t_1, TCP attempts periodic retransmissions at exponentially increasing intervals. At t_2, the host reconnects to a different access network, its IP address changes and the TCP connection aborts.

The protocol enhancements presented in this paper use HIP together with its proposed *mobility and multi-homing extensions* [27] to shield established TCP connections from changes in IP addressing (see Section 3.1.) *MobileIP* [21] would be another option for addressing this part of the problem.

2.2 Disconnection Duration

Besides changes in IP addressing, the duration of a disconnection is another factor that can cause established TCP connections to abort.

TCP defines a *user timeout* that specifies the maximum time that data may remain unacknowledged by the peer. The intent behind the *user timeout* is to periodically reclaim system resources allocated to stale connections, where the peer has gone away. Typically, TCP implementations use a system-wide user timeout of a few minutes [5]. Peers do not need to use the same user timeout. After one peer has aborted its half of a connection, a communication attempt by the other peer will result in a reset. Note that the user timeout is different from the socket timeout provided by socket API. The socket timeout defines how long a send or receive call to the socket waits until data can be written to the send buffer or read from the receive buffer, whereas the user timeout applies to data already sent to the receiver but remains in the output buffer until it is acknowledged.

During a disconnection, no acknowledgments from peers can reach a node. To TCP, this appears as if the connections have gone stale. It will consequently abort them to reclaim associated system resources when the user timeout expires. As described in the previous section, connection aborts are error conditions for applications and can cause undesirable effects.

A simple attempt to prevent such connection aborts would disable the user timeout during a disconnection. However, only the disconnected host may have sufficient local information for this decision. The remote host may thus still abort the connection when its user timeout expires. Furthermore, some other timeout must still exist, to reclaim resources in the case where peers disappear during disconnections.

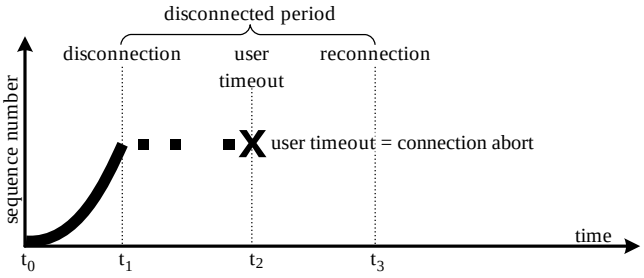

Figure 3. Connection abort caused by user timeout.

Figure 3 illustrates this issue. In the initial connected period between t_0 and t_1, the host transmits normally. At t_1, the host looses Internet connectivity and attempts retransmissions in exponentially increasing intervals. Finally, at t_2, the host's user timeout expires and TCP closes the affected connection due to inactivity.

This paper chooses a different solution and allows hosts to establish custom, per-connection user timeouts in order to tolerate long disconnections. Section 3.2 discusses the specifics of this protocol enhancement.

2.3 Retransmission Behavior

The first two characteristics of intermittent connectivity may cause established connections to abort. This section describes a third issue that does not cause connection aborts, but significantly decreases TCP performance.

When a disconnection occurs along the path between a host and its peer while the host is transmitting data, it stops to receive acknowledgments. After the *retransmission timeout* (RTO) expires, the host attempts to retransmit the first unacknowledged segment. TCP's recommended RTO management procedure [7] doubles the RTO after each failed retransmission attempt until the RTO exceeds 60 seconds.

This retransmission behavior is inefficient. When a disconnection ends, many TCP implementations still wait until the RTO expires before attempting retransmission. Depending on when connectivity becomes available again relative to the next scheduled RTO, this behavior can waste up to a minute of connection time for TCP implementations that follow the recommended RTO management and even more for others.

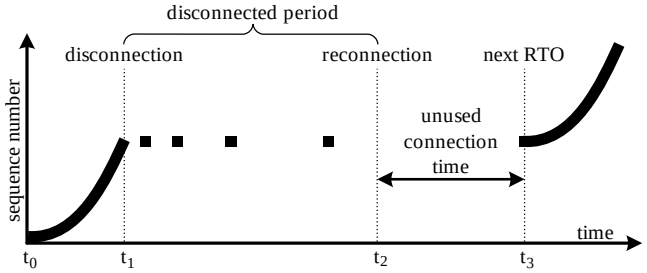

Figure 4. Unused connection time after reconnection.

Figure 4 illustrates this inefficiency. As before, the host transmits normally during the initial connected period between t_0 and t_1. When connectivity is lost at t_1, TCP begins to attempt periodic retransmissions at exponentially increasing intervals. For this example, assume that the user timeout is longer than the disconnected period and no change of IP addresses occurs when the disconnection ends at t_2. Although the means for communication are reestablished at t_2, TCP waits until the next scheduled RTO at t_3 before resuming transmission. The part of the second connected period between t_2 and t_3, which could have been used for transmission, remains unused.

This paper eliminates this inefficiency with the new TCP Retransmission Trigger. The retransmission trigger causes TCP to attempt an additional, speculative retransmission when connectivity is restored. The next section describes the TCP Retransmission Trigger, the TCP User Timeout Option and their combination with HIP in detail.

3. PROTOCOL ENHANCEMENTS

The previous section has discussed three cases where the characteristics of intermittently connected paths can either disrupt or slow down TCP transmissions. This section describes how a combination of existing protocols with new protocol enhancements can solve these problems. The improved protocols closely approximate the ideal transmission behavior in the presence of intermittent connectivity. Figure 5 illustrates the ideal behavior. TCP fully utilizes periods of connectivity for transmission (from t_0 to t_1 and after t_2) and does not abort the connection during the disconnected period between t_1 and t_2.

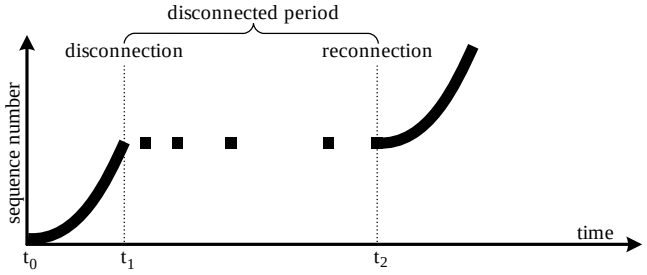

Figure 5. Ideal transmission behavior in the presence of a disconnection.

The reminder of this section discusses the different protocols and protocol enhancements used to address the problems described in Section 2. This proposed solution requires only minimal changes to existing end system protocols (new TCP options, simple modification of HIP and TCP's retransmission scheme.) It requires no further changes to TCP, HIP, other parts of the network infrastructure or applications.

3.1 IP Address Changes

The Host Identity Protocol (HIP) [3] and its mobility and multi-homing extensions [27] provide the necessary mobility support for hosts that roam across different access networks. The HIP layer, a new shim layer between the network and transport layers, enables host mobility that is transparent to applications and services. Mobility support for established connections is provided end-to-end. Each host informs its current peers when IP address changes occur.

HIP decouples transport-layer protocols from IP address changes. With HIP, connections transparently bind to static host identities instead of IP addresses. Consequently, the IP addresses of communicating hosts can change without affecting established connections. HIP even enables hosts to switch between different network protocols (*e.g.*, from IPv4 to IPv6) while maintaining established transport-layer connections.

The current mapping from host identities to IP addresses has to be updated each time a host changes its IP address. For established connections, the communicating hosts themselves store the current mapping between host identities and IP addresses. If a mobile host changes its IP address, it distributes its new address to all current peers so they can update their local mappings. This involves a three-way HIP readdressing handshake, which helps to prevent distributed denial-of-service attacks.

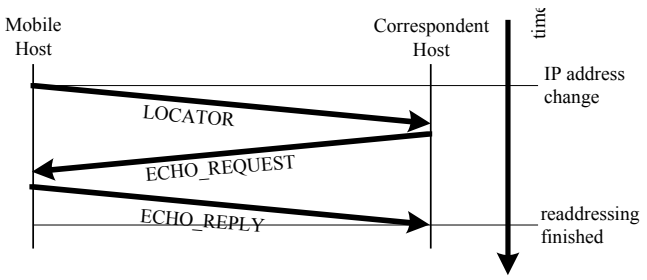

Figure 6. HIP readdressing handshake.

Figure 6 illustrates the packet trace of a HIP readdressing exchange. First, the mobile host sends an *UPDATE* packet with a *LOCATOR* parameter to its peer host to announce its new address. In the next step, the peer attempts to verify whether the mobile host is indeed reachable at received new address by sending an *UPDATE* packet with an *ECHO_REQUEST* parameter back to the mobile host. In the last step, the mobile host confirms its reachability by responding with an *UPDATE* packet carrying an *ECHO_REPLY* parameter. The peer host will not send any data traffic to the apparent new IP address of the mobile host before the handshake is complete.

HIP's mobility extensions [3] do not currently support the case where two communicating mobile nodes move at the same time. The proposed HIP *rendezvous service* will support reestablishment of communication in this case, but is still under design [44].

Although HIP successfully prevents transport-layer connections from aborting due to changes in IP addressing, it cannot prevent aborts caused by extended periods of disconnection. The next section describes the TCP User Timeout Option, which addresses this second issue.

3.2 Disconnection Duration

Section 2.2 described how periods of disconnection that exceed the user timeout of one of the communicating peers cause established connections to abort. Today's TCP implementations use a system-wide, constant user timeout for all connections [5], independent of a host's characteristics, networking environment or active applications. Typically, this user timeout is only few minutes long.

Increasing the system-wide user timeout allows established TCP connections to tolerate correspondingly longer disconnections. However, this simple approach can significantly increase system resource utilization, because TCP will maintain all connection state across longer disconnected periods. Note, that the user timeout has to be extended on both communicating hosts to be effective, which incurs that all hosts in the internet have to change their default. This also includes highly loaded stationary servers, which are rather likely to use shorter user timeout values to free system resources earlier. The user timeout should therefore be adjusted according to the context of the local host as well as the context of the peer host.

The proposed TCP User Timeout Option [8] allows conforming hosts to exchange per-connection abort timeout requests. This allows mobile hosts to maintain TCP connections across disconnected periods that are longer than their system's default user timeout. A second use of the TCP User Timeout Option is exchange of shorter-than-default user timeouts. This can allow busy servers to explicitly notify their clients that they will maintain the state associated with established connections only across short periods of disconnection.

TCP User Timeout Options allow hosts to both request specific user timeouts for new connections and to request changes to the effective user timeouts of established connections. The latter allows connections to start with short timeouts and only request longer timeouts when external information suggests that disconnection was imminent, and only for connections considered important. The ability to request changes to user timeouts of established connections is also useful to raise the user timeout after in-band authentication has occurred. For example, peers could request longer user timeouts for the TCP connections underlying two-way authenticated TLS connections [28] after their authentication handshakes.

This paper does not discuss policies to choose appropriate user timeout values for specific connections, because such policies are highly specific to the particular combination of applications and network environments. However, in general mobile hosts will

tend to ask for longer user timeouts as they are more likely to encounter connectivity disruptions. On the contrary, stationary servers will tend to use shorter user timeouts to keep less state information. It will only extend it when requested by its peer and given enough resources are available. Other research activities investigate mechanisms to predict time and length of connectivity disruptions.

Figure 7 shows the format of the TCP User Timeout. Besides the standard *kind* and *length* fields that all TCP options share [2], the TCP User Timeout Option includes a granularity bit and the user timeout value. The *granularity* bit (G) indicates whether the following *user timeout* value is specified in seconds or minutes. The 15-bit user timeout value can express durations from zero seconds to over 9 hours at a granularity of seconds and from zero minutes to over 22 days at a granularity of minutes.

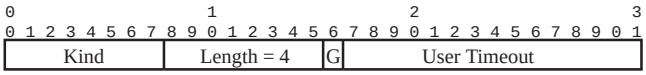

Figure 7. Format of the TCP User Timeout Option.

Sending a TCP User Timeout Option signals to the receiving peer that the sender will start to use the indicated user timeout value locally for the connection and is requesting the receiving peer to do the same. The receiving peer may then decide whether to change the connection's local user timeout accordingly. Hosts can exchange TCP User Timeout Options during the initial handshake or after the connection is established.

Generally, hosts that implement the TCP User Timeout Option should honor incoming requests for changes to the user timeout. When they do, they should signal acceptance by including a corresponding TCP User Timeout Option in their next segment to the peer. Sometimes, security concerns or external policies may disallow a host to change a connection's user timeout above or below specific limits. If so, hosts may propose a lower user timeout for the connection or even ignore the peer's TCP User Timeout Option altogether.

It is important to note that TCP User Timeout Options do not change the semantics of the TCP protocol. Hosts remain free to abort connections at any time for any reason, whether or not they use custom user timeouts or have requested the peer to use them. Furthermore, the TCP specification mandates that implementations ignore unsupported TCP options and experimental results indicate that almost all implementations do so [45]. This behavior enables graceful backwards compatibility with hosts that do not implement the proposed option.

A different approach to tolerate longer disconnected periods would be to simply increase the system-wide user timeout on both peers. This approach has the benefit of not requiring a new TCP option. However, it can also significantly increase the amount of connection state a host must maintain, because the longer global timeout value will apply to all its connections. The proposed timeout option, on the other hand, allows hosts to selectively manage the user timeouts of individual connections. They must then only maintain the state associated with selected connections across disconnected periods.

3.3 Retransmission Behavior

The previous two sections described mechanisms to prevent TCP connections across intermittently connected, dynamically changing paths from aborting. This section describes a modification of TCP's retransmission scheme to improve performance over a path with frequent disconnections. The basic idea behind this TCP Retransmission Trigger [9] is to attempt additional, speculative retransmissions when a TCP implementation receives an indication that connectivity to a previously disconnected peer node may have been restored.

The TCP Retransmission Trigger does not modify TCP's basic congestion, fairness properties or slow-start algorithms. The only difference in TCP behavior is the timing of retransmission events and, in some cases, a minor, fixed increase in the number of initially retransmitted segments. The TCP Retransmission Trigger increases performance through better utilization of connected periods, not through sending traffic at a faster rate or modifying TCP's congestion control mechanisms.

The next two sections describe which events at hosts and in the network may cause retransmission triggers to be generated and how TCP should react in response to one.

3.3.1 Connectivity Indicators

Connectivity indicators, which signal the retransmission trigger, depend on the specifics of a node and its environment, such as the link-layer technologies attaches to or the presence of network-layer mechanisms such as DHCP [31], MobileIP [21] or HIP [3]. The IETF's Detection of Network Attachment (DNA) working group currently investigates the specifics of providing such indicators (triggers) [32].

One example of a connectivity indicator is *next hop reachable*. This indicator could occur if a combination of the following conditions is true, depending on host specifics:

- Network-layer connectivity along the path to the destination is restored, *e.g.*, the outbound interface has an IP address and a next-hop router is known, maybe due to DHCP [31] or IPv6 router advertisements [33].
- Link-layer connectivity of the link to the next-hop router along the path to the destination is restored, *e.g.*, link-layer "up."
- Other local conditions that affect reachability of the destination are satisfied, *e.g.*, IKE key exchanges [34], MobileIP binding updates [21] or HIP readdressing [27] have completed.

The next hop reachable retransmission indicator only depends on locally determinable information (*e.g.*, state of directly-connected links, etc.) and does not require network cooperation. It can signal TCP to restart active connections across intermittently connected links where disruptions occur on the first or last hop. This simple indicator has the potential to improve TCP performance in many cases, because connection disruptions at the first or last hop are arguably the most common cause of disconnections in today's Internet.

A second, more general example of a connectivity indicator would be *end-to-end connectivity restored*. If hosts have the ability to detect or be notified of connectivity changes inside the network (*i.e.*, not only at the first or last hop), a more general trigger could act on those pieces of information. This can improve TCP performance across intermittently connected paths where disruptions occur at arbitrary links along the path, even inside the network. However, providing this more general trigger is problematic due to its dependence on remote information.

Connectivity indicators are generally *asymmetric*, *i.e.*, one may occur on one peer host but not the other. As discussed above, a local event at one host may signal the retransmission trigger, while the other host is unable to detect this event across the network. *Symmetric* connectivity indicators are a special case and always occur concurrently at both communicating hosts. They are

usually based on handshake events such as IKE exchanges or HIP readdressing. Symmetric connectivity indicators are an important special case, because the TCP retransmission procedure required in response to a symmetric connectivity indicator is simpler than that for an asymmetric one. The next section will describe this in detail.

3.3.2 TCP Retransmission Enhancement

The TCP Retransmission Trigger augments TCP's standard retransmission scheme. When receiving a symmetric or asymmetric connectivity indicator, conforming TCP implementations immediately initiate the standard retransmission procedure, as if the RTO had just expired. If the connectivity indicator is symmetric, *i.e.*, both peers receive it simultaneously; this simple change is sufficient to kick-start a TCP connection.

If the connectivity indicator is asymmetric, a simple retransmission by one peer is not sufficient. Asymmetric connectivity indicators only occur at one peer but not the other. If the host receiving the trigger has no (or too little) unacknowledged data awaiting retransmission, it will not emit enough segments to cause the peer node, which may have unacknowledged data, to attempt a retransmission itself. The retransmission trigger would thus only function in one direction, which is ineffective for asymmetric communication.

To avoid this issue, conforming TCP implementation thus perform a different retransmission procedure in response to an asymmetric connectivity indicator. This paper proposes two possible solutions.

The first solution uses a new TCP Immediate Retransmission Option that relays a locally received asymmetric connectivity indicator to the remote peer, effectively turning it into a symmetric indicator. When a host receives an asymmetric connectivity trigger, it must ensure that at least the next segment sent to its peer includes this new TCP option. The segment can either be an outstanding retransmission or an additional pure ACK if the retransmission queue is empty. When receiving a TCP Immediate Retransmission Option, conforming TCP implementations react just if they had received a symmetric connectivity indicator, *i.e.*, they immediately initiate the standard retransmission procedure, as if the RTO had expired. Consequently, data flow will resume in reverse direction. However, this solution requires both communicating hosts to implement the TCP Retransmission Trigger as well as the TCP Immediate Retransmission Option. (Note that this new option will be described in a forthcoming update to [9].)

The second solution for responding to asymmetric connectivity indicators is easier to deploy, because it does not require changes to both communicating peers. As proposed by Cáceres and Iftode [42], generating gratuitous *triple-duplicate acknowledgments* (ACKs) can restart the reverse data flow by activating the peer's fast retransmission mechanism. Therefore, TCP implementations conforming to this second approach must ensure to send at least four segments that all acknowledge the last segment received from the peer. If the TCP implementation that receives the connectivity indicator has four or more unacknowledged data segments in its retransmission queue, it can piggyback the triple-duplicate ACK to the regular retransmissions of those data segments. In this case, the TCP Retransmission Trigger does not require additional messages, compared to standard TCP.

If the retransmission queue contains three or less unacknowledged data segments, TCP implementations supporting retransmission triggers will send additional pure ACKs until a complete triple-duplicate ACK has been sent. In the worst case, when the retransmission queue is empty, this scheme requires four additional ACKs, compared to standard TCP.

After the peer's fast retransmit algorithm sends the assumed missing segment, standard TCP performs either *fast recovery* or a *slow-start* [30], depending on the length of the disconnection. If the retransmission trigger occurs before the RTO, *i.e.*, for very short disconnections, TCP has not yet lost its ACK clock and can thus perform fast recovery. After longer disconnections, TCP falls back to slow-start to restart the ACK clock, just as it does at the beginning of a connection.

Mobile hosts will frequently attach to different networks after moving, which may have different speeds and levels of load. This means that the path information and congestion state in a connection's protocol control block will likely be stale. For disconnections that exceed the RTO, this is not a problem, because TCP will automatically perform slow-start after a retransmission trigger occurs. For disconnections shorter than the RTO – arguably a less frequent case – an additional mechanism is required to *force* TCP to slow-start, to avoid overloading the network. The prototype implementation does not currently implement this forced slow-start.

The result of this modification is twofold. First, the TCP implementation receiving the retransmission trigger attempts retransmission of its unacknowledged segments before its next scheduled RTO. This increases utilization of the connected period. Second, the TCP implementation receiving the retransmission trigger uses either a new TCP Immediate Retransmission Option or an existing TCP mechanism (triple-duplicate ACK) to signal its peer – that may not have received a retransmission trigger itself – to attempt faster retransmission as well.

As mentioned above, this retransmission scheme can generate up to four additional segments, compared to standard TCP. All additional segments are pure ACKs and hence small, resulting in a minor total overhead. Furthermore, measurements have shown that increasing TCP's initial window is not problematic [29]; this may indicate that a minor increase in traffic at retransmission time may be tolerable as well.

Finally, to protect the TCP retransmission trigger from abuse, *e.g.*, launching denial-of-service attacks by flooding TCP with triggers, a control mechanism that "rate-limits" connectivity indications may be effective. Furthermore, some connectivity indicators may trigger retransmissions for a large number of TCP connections, causing a burst of retransmissions that may negatively affect other traffic. This paper does not discuss these security aspects of retransmission triggers further.

3.3.3 HIP Retransmission Trigger

When TCP runs on top of IP, retransmission triggers can occur based on network-layer events such as router advertisements [33] or DHCP leases [31], which in turn execute in response to link-layer events. This paper, however, uses TCP on top of HIP, to prevent against connections aborts when IP addresses change. Triggering TCP connections based on IP events is thus not possible.

Before transport-layer communication can resume after a disconnection, HIP first has to update the mappings between host identities and network addresses, as described in Section 3.1. If TCP attempts a retransmission before this HIP readdressing is complete, it will fail and the benefit of the retransmission trigger is lost.

To avoid that the TCP Retransmission Trigger is signaled too early, *e.g.*, before lower-layer protocols are ready to communicate, the TCP Retransmission Trigger should be signaled from the protocol layer directly below the transport layer. Because TCP runs on top of HIP in this paper, the retransmission trigger is based called from the HIP readdressing procedure. When the HIP readdressing exchange completes, *i.e.*, when nodes send or receive the last *ECHO_REPLY* parameter, they trigger TCP to resume transport-layer communication. This procedure is also effective if two communicating mobile hosts move at the same time. In that case, the readdressing occurs via HIP rendezvous servers, but the local trigger mechanism still operates as described.

Although this mechanism is generally effective (see the measurements in Section 4), a race condition exists that can render it ineffective in some scenarios. This can occur if the network reorders the *UPDATE* packet carrying the *ECHO_REPLY* and the first retransmitted TCP segments, such that the retransmissions arrive before the *ECHO_REPLY*. In that case, the receiver will simply drop the retransmissions. This can also occur if the packets arrive in-order, but the receiver's processing of the *ECHO_REPLY* is delayed. In these cases, although the benefit of the retransmission trigger is lost, TCP communication still resumes successfully once the next scheduled RTO expires. At most four extra segments will have been generated.

3.4 Summary

This section has described a solution that improves the operation of TCP across intermittently connected paths. It consists of three components: HIP, to protect against connection aborts due to IP address changes, the TCP User Timeout Option, to protect against connection aborts due to prolonged disconnections, and the TCP Retransmission Trigger, to improve TCP performance through better utilization of connected periods.

The proposed protocol enhancements are end-to-end mechanisms – like TCP itself – that both peers must implement to see any benefit. To aid deployment of these enhancements, they may initially be implemented in split-connection proxies. This approach improves the communication performance of unmodified end systems in disruptive networking environments, similar to other performance-enhancing proxies [43].

4. EXPERIMENTAL EVALUATION

This section experimentally analyzes the protocol enhancements described in Section 3 in two different scenarios: a bulk transfer in the presence of a single disconnection and a bulk transfer in the presence of periodic disconnections.

4.1 Bulk Transfer with a Single Disconnection

The first experiment analyzes the performance of a bulk data transfer in the presence of a single disconnection. Figure 8 illustrates the scenario for this experiment. A *mobile host* (*M*) transmits a fixed amount of data to its immobile *correspondent host* (*C*) over a single TCP connection. During the duration of this transmission, *M* moves between two access networks. At the beginning of the transmission at t_0, *M* connects to the Internet and thus its peer *C* through access network *A*. At t_1, while the transmission is ongoing, *M* moves out of range of *A* and a period of disconnection begins. *M* continues to move and at t_2 reestablishes Internet connectivity through access network *B*. For the remainder of the bulk transfer, *M* stays in range of *B*.

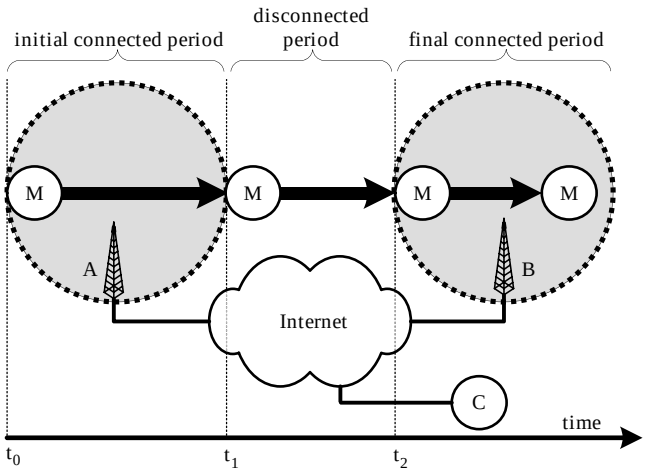

Figure 8. Disconnection scenario modeled by the experimental setup.

The first phase of the experiment between t_0 and t_1, when *M* is connected to *A*, is the *initial connected period*. Its length is the first parameter of the experiment. The second phase of the experiment between t_1 and t_2 is the *disconnected period*. Its length is the second parameter of the experiment. The third phase of the experiment starts at t_2 when *M* enters range of *B*. It is the *final connected period* and has an infinite length, as M stays in range of B until the bulk transfer finishes.

Each experiment measures the *net transmission time* of the bulk data transfer for different lengths of initial connected periods and disconnected periods. The net transmission time of a bulk transfer is its total transmission time minus the length of the disconnected period. It allows comparing the efficiency at which TCP utilizes both connected periods for transmission. Obviously, lower net transmission times indicate better TCP performance.

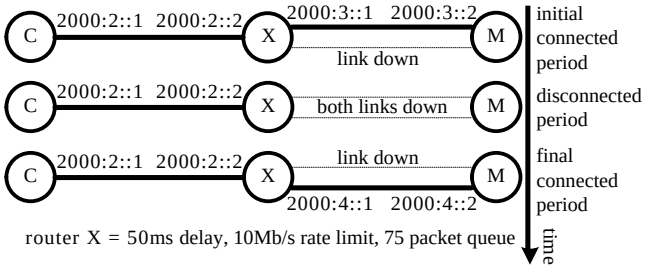

Figure 9. Emulation of the scenario during the experiment.

Figure 9 illustrates the experimental setup chosen to emulate the scenario. Instead of wireless access networks, the experiments use wired links. This eliminates potential wireless link-layer effects that could influence TCP behavior from the experiment. This was a deliberate decision, as this paper focuses on protocol enhancements for intermittent connectivity – protocol enhancements for wireless links are a separate, well-researched area. In a sense, wired links with connectivity disruptions are a best-case scenario for TCP; performance across wireless links with similar connectivity disruptions would be poorer still.

Because the prototype HIP implementation does not support IPv6 auto-configuration, mobile host *M* connects to access router

X across two separate, preconfigured links that emulate different access networks. As shown in Figure 9, during the initial connected period, the first link is active while the second link is disabled. To simulate the start of the disconnected period, the first link is disabled as well. Finally, to simulate a reconnection to a different access network, the second link is activated.

As shown in Figure 9, three identical workstations (Intel Pentium IV 2.8 GHz, 256 MB DDR memory, Linux kernel 2.4.20, prototype HIP implementation [15]) connect through Intel PRO/1000 MT 1Gb/s Ethernet cards. Two workstations act as mobile host *M* and correspondent host *C*. A third workstation *X* emulates a 10Mb/s wide-area link with a capacity of 10Mb/s and a queue limit of 75 packets using the *Click* modular router [13].

Based on empirical measurements on this test network, 25MB was chosen as the size for each bulk transfer. Under ideal conditions (no connectivity disruptions), the test network can transmit 25MB of data in approximately 21 seconds without HIP and 22 seconds with TCP over HIP. The *iperf* benchmark [14] executes and measures the performance of each bulk transfer.

Based on this best-case transmission time of approximately 21 seconds, the two parameters of the experiment – length of the initial connected period and length of the disconnected period – were chosen accordingly. The length of the initial connected period increases from 2 to 26 seconds in increments of 2 seconds. This allows investigation of whether the timing of the disconnection relative to the connection – early during slow-start or late during the lifetime – influences performance. Additionally, with initial connected periods of 24-26 seconds, the complete bulk transfer finishes *before* the disconnection, illustrating the best-case performance.

The length of the disconnection increases from 0-208 seconds in increments of 13 seconds, to determine whether the length of the disconnection influences TCP performance. For these experiments, the Linux TCP implementation is configured to abort connections after nine failed retransmission attempts, which occurs after approximately 150 seconds of disconnection. Decreasing the default Linux timeout of 15 retransmissions reduces the time required for each experiment significantly. This does not invalidate the results, because TCP has already reached its maximum retransmission interval of 120 seconds after this period. (Whereas some TCP implementations use 60 seconds as their maximum retransmission interval [7], the Linux TCP stack is even more conservative.)

For each combination of initial connected period length and disconnected period length, 10 bulk transfers of 25MB each are run to compute medians and quartiles. A complete experiment involves 13*17*10 = 2210 transfers of 25MB and takes approximately five days to finish. Three such experiments measure the performance of TCP over HIP (TCP/HIP), TCP with the TCP User Timeout Option over HIP (TCP+UTO/HIP) and TCP with both the TCP User Timeout Option as well as the TCP Retransmission Trigger over HIP (TCP+UTO+RT/HIP). (Traditional TCP over IP was omitted from the experiments, because all connections break after the reconnection, as discussed in Section 2.) A surface plot visualizes the result of each experiment.

4.1.1 Bulk Transfer: TCP/HIP

The first experiment investigates the basic case of standard TCP over HIP. Figure 10 shows the results as a series of plots. The larger surface plot at the top of Figure 10 shows median net transmission time for different combinations of initial connected period length and disconnected period length. The smaller plot on the bottom left of Figure 10 shows the same information, but as a density plot. The smaller density plot on the bottom right shows the interquartile range of the measurement series for the corresponding surface point. In all three plots, darker shades of gray indicate worse performance.

Section 2.2 discussed how disconnections longer than TCP's user timeout cause connection aborts. The measurements shown in Figure 10 validate this prediction. During the experiment, TCP connections start aborting when disconnections exceed 130 seconds and all connections abort for disconnections that exceed 143 seconds. (The only connections that do not abort occur during runs with initial connected periods of 24-26 seconds, *i.e.*, when the disconnection occurs after the connections finish.) This result motivates the need for a mechanism to manage user timeouts during disconnected periods, such as the TCP User Timeout Option measured in the next section.

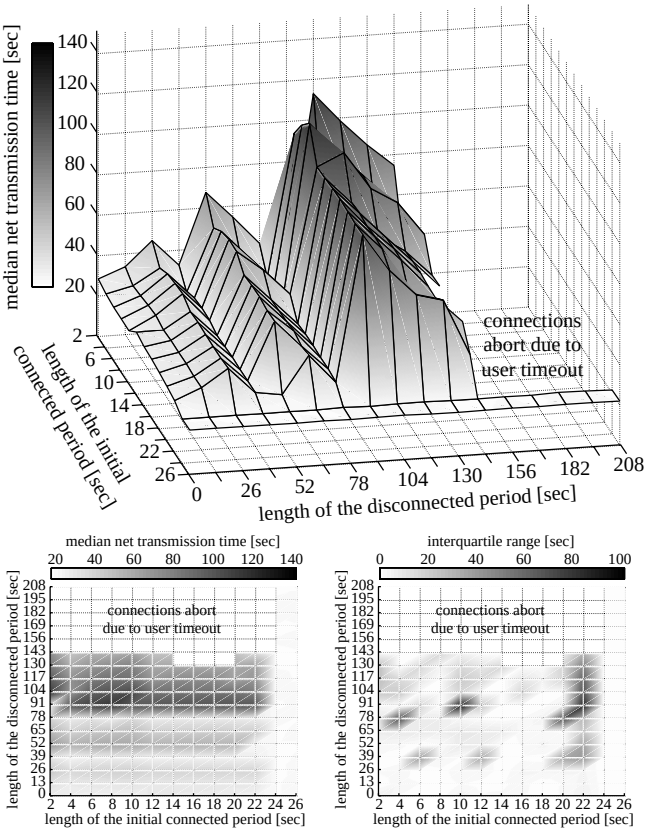

Figure 10. Surface and density plots showing median net transmission time and interquartile ranges of standard TCP over HIP under intermittent connectivity.

TCP performance – for cases where the bulk transfer does not abort – is generally poor. In the worst case, net transmission time for the 25MB bulk transfer exceeds 100 seconds, which corresponds to an average bandwidth of 2Mb/s – only approximately 20% of the available path capacity.

Another observation about Figure 10 concerns the timing of disconnections relative to the beginning of the bulk transfers, *i.e.*, the length of the initial connected period. Whether the

disconnection occurs after 2 seconds (while TCP is still within the slow-start phase) or later (*i.e.*, during steady state) has very little impact on the net transmission time. This is an interesting result, because it can significantly reduce the parameter space for future experiments.

Finally, net transmission times increase in a seesaw pattern as disconnected periods grow longer, *i.e.*, a longer disconnection may in fact increase performance of a TCP connection. The surface plot in Figure 10 has distinct peaks at disconnection lengths of 26, 52 and 91 seconds as well as distinct dips at disconnection lengths of 39 and 78 seconds. This behavior occurs, because longer disconnections may lead to a *reduction* of unused connection time before the next RTO (see Figure 4) and because the metric used (*net* transmission time) does not reflect the length of the disconnection. With disconnected periods of 39 and 78 seconds, the disconnection ends shortly *before* the next RTO, leaving very little connection time unused for transmission. The other extreme occurs at disconnection lengths of 26, 52 and 91 seconds. Here, the disconnection ends shortly *after* the last RTO, causing a large fraction of the connected period to remain unutilized.

This first experiment illustrates the need for management of TCP user timeouts to prevent connection failures during intermittent connectivity. The next section will repeat the same experiment using the TCP User Timeout Option, *i.e.*, with longer user timeouts.

4.1.2 Bulk Transfer: TCP+UTO/HIP

The previous experiment has shown that a bulk transfer using standard TCP over HIP aborts when the disconnected period exceeds the user timeout. The TCP User Timeout Option allows conforming TCP implementations to use custom, per-connection user timeouts, as discussed in Section 3.2. While a prototype implementation of the TCP User Timeout Option is being prepared, it was not ready in time for this experiment. For these measurements, increasing the system-wide user timeout to 15 minutes simulates the effect of the TCP User Timeout Option. The setup for this experiment is identical to the previous section, except for the increase in user timeout.

Figure 11 shows the results with increased user timeouts. For disconnected periods less then 130 seconds, median net transmission times differ very little from the previous case, shown in Figure 10. This is not surprising, because longer user timeouts do not affect those transfers.

When disconnected periods exceed 130-143 seconds in Figure 11, the difference to the previous case shown in Figure 10 becomes apparent. Unlike before, no connections abort with longer user timeouts, even in the presence of long disconnections. This illustrates the effectiveness of the TCP User Timeout Option.

However, Figure 11 also shows that net transmission times in the presence of long disconnections are high. Most connections require 105-140 seconds to transfer 25MB, resulting in an average data rate of only 1.4Mb/s or 14% of the available bandwidth.

As discussed in Section 2.3, this poor performance is due to TCP's retransmission behavior. As disconnections grow longer, TCP will also attempt retransmissions less frequently, which results in a higher likelihood of not utilizing available connectivity for transmission. The TCP Retransmission Trigger described in Section 3.3 attempts to improve performance through smarter utilization of connected periods. The next section analyzes the performance of the bulk transfer when TCP uses retransmission triggers.

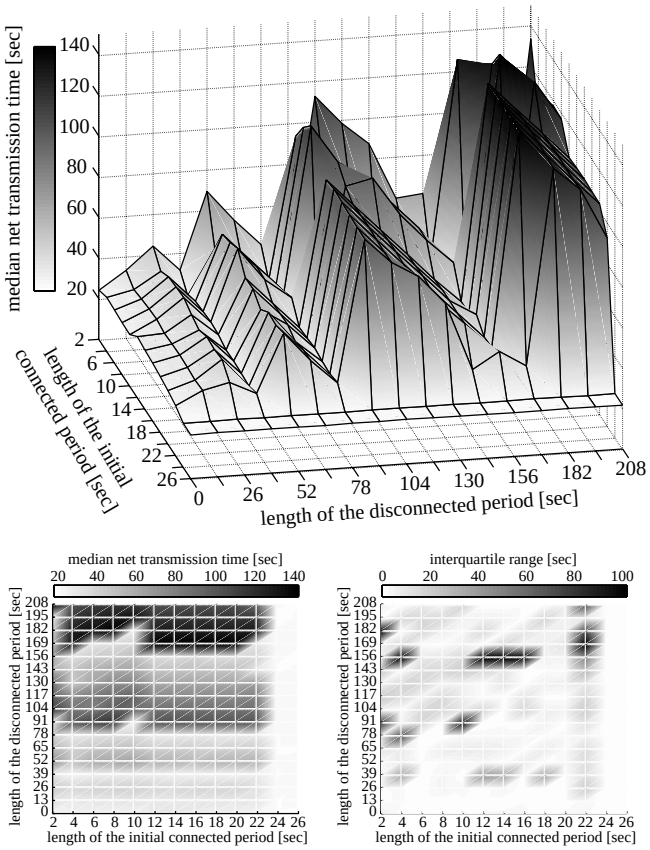

Figure 11. Surface and density plots showing median net transmission time and interquartile ranges of TCP with the TCP User Timeout Option over HIP under intermittent connectivity.

4.1.3 Bulk Transfer: TCP+UTO+RT/HIP

This section evaluates the performance of the full solution described in this paper. Here, the 25MB bulk transfers use TCP with both the User Timeout Option (simulated as before) and a prototype implementation of the TCP Retransmission Trigger. Except for this change, the experimental setup is identical to the previous cases.

Figure 12 shows the results with the TCP Retransmission Trigger. The difference to the previous case is apparent. Unlike before, net transmission time for the 25MB transfer lies between 24 and 27 seconds, independent of the length of the disconnected period or the initial connected period. Even in the presence of intermittent connectivity, the bulk transfers reach average throughputs of 7.4-8.1Mb/s on a 10Mb/s path. The result is a flat surface plot and an evenly shaded density plot shown in Figure 12.

This result is already close to the best-case net transmission time of 21 seconds. An analysis of a packet trace shows that a large fraction of the time difference between the best case (21 seconds) and these results (24-27 seconds) is caused by HIP readdressing exchanges. When a disconnected period ends, HIP must first complete a readdressing procedure before TCP can resume transmission. This readdressing should typically complete in 2-3 round-trip times, approximately 200-300ms on the test network. However, with the prototype HIP implementation used for these measurements [15], readdressing took 2-3 seconds.

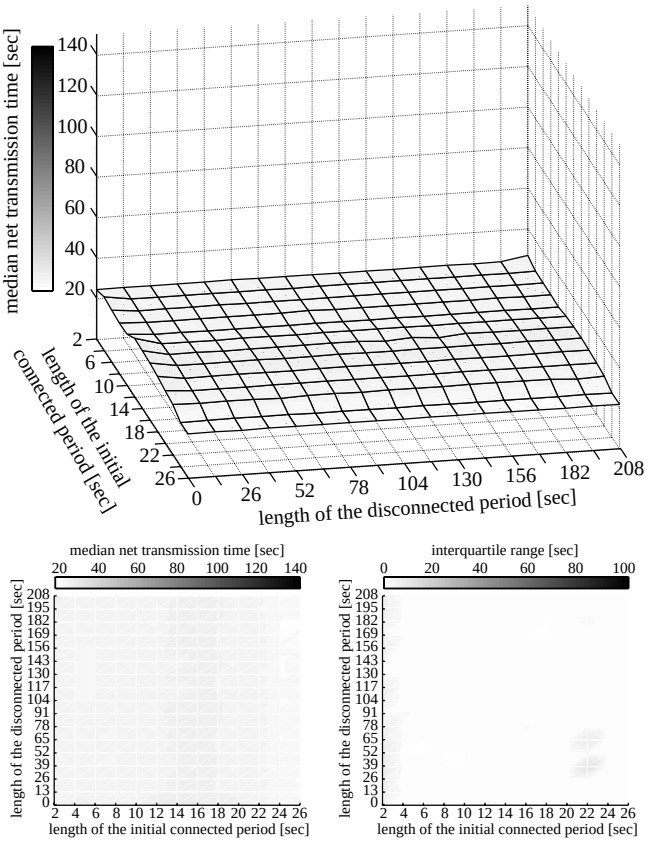

Figure 12. Surface and density plots showing median net transmission time and interquartile ranges of TCP with the TCP User Timeout Option and the TCP Retransmission Trigger over HIP under intermittent connectivity.

Consequently, net transmission time is expected to further approach best-case performance once HIP implementations improve.

4.1.4 Summary

This section experimentally investigated the performance of a bulk TCP transfer across an intermittently connected path. Three different experiments analyzed the performance of different TCP variants over HIP: standard TCP, TCP with a simulated TCP User Timeout Option and TCP with the TCP User Timeout Option and the TCP Retransmission Trigger.

The measurement results indicate that all three proposed mechanisms improve different aspects of TCP behavior. The combined mechanisms allow TCP to tolerate mobility and long periods of disconnection. Compared to the basic case, they reduce median net transmission time up to a factor of five (24-27 seconds vs. 120 seconds.) This performance is within 72-86% of the best case without a disconnection. With more mature HIP implementations, performance is expected to increase to within 86-96% of the best case.

4.2 Bulk Transfer with Periodic Disconnections

The previous section investigated the performance of a bulk transmission across an intermittently connected path with a single disconnected period. This section will focus on a similar transfer, but across an intermittently connected path with periodic disconnections. Compared to the previous scenario shown in Figure 8, the mobile host will not remain in range of access network B until the transmission finishes, but leave the range of B again and move back towards A, starting another period of disconnection.

To reduce the number of parameters, all connected periods have the same length; this is the first parameter of this experiment. Three distinct lengths of connected periods were measured, 17, 37 and 67 seconds. Similarly, all disconnected periods have the same length. For this second parameter of the experiment, two distinct values were measured, 37 and 67 seconds. (Prime numbers were chosen for all parameters to reduce the likelihood of synchronization effects.) This results in six different experimental runs to measure all possible combinations of the parameters.

Each experimental run transmits 600MB of data across the test network (10Mb/s, 50ms delay, as in the previous section), once using standard TCP and once using TCP with the TCP Retransmission Trigger. Median net transmission times and interquartile ranges are computed based on the results of ten separate measurements. Figure 13 shows the results in a series of bar graphs.

The top row of Figure 13 shows the results when the connected period is 17 seconds and the disconnected periods are 37 (on the left) and 67 seconds (on the right), respectively. In these cases, standard TCP is again unable to transmit the full amount of data – the standard user timeout aborts the connection. TCP with the TCP Retransmission Trigger is successful and transmits the data in approximately 640-650 seconds in both cases at an average rate of 7.4Mb/s across the 10Mb/s link.

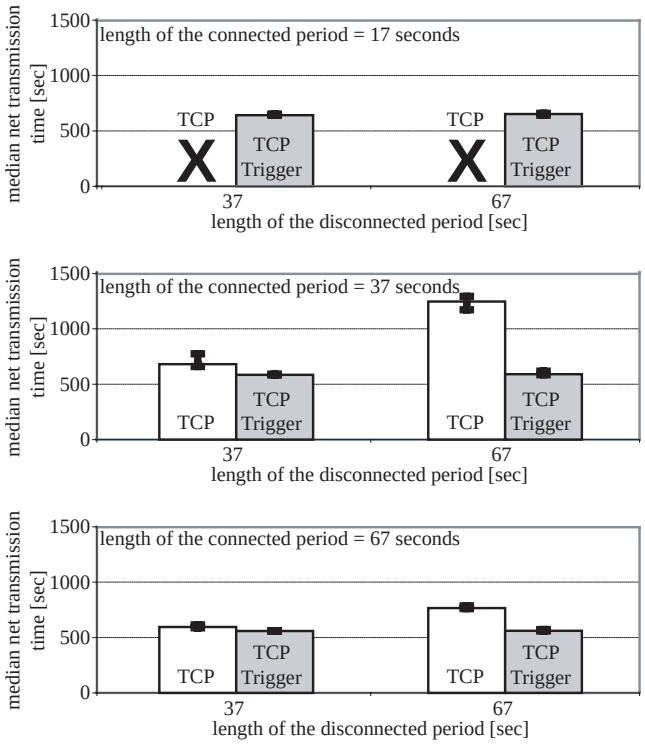

Figure 13. Bar graphs showing median net transmission time and interquartile ranges of standard TCP ("TCP") and TCP with TCP Retransmission Trigger ("TCP Trigger") over HIP under intermittent periodic connectivity of different lengths.

The middle row of Figure 13 shows the results with periodic connected periods of 37 seconds (disconnected periods are again 37 and 67 seconds.) In both cases, standard TCP now finishes the transmission without aborting. With disconnections of 37 seconds, standard TCP is even relatively efficient and transmits 600MB in 680 seconds at an average rate of 7.05Mb/s. TCP with the TCP Retransmission Trigger is only approximately 10% more efficient, reaching 8.21Mb/s. With a disconnected period of 67 seconds, this changes. Standard TCP now needs over 1200 seconds (= 3.85Mb/s) whereas TCP with the Retransmission Trigger still requires only 590 seconds – an improvement of over 50%.

Empirical analysis of a packet trace shows that this difference in the performance of standard TCP is due to the timing of the RTO [10]. With disconnections of 37 seconds, the next RTO expires 3-8 seconds after the disconnection ends. With disconnections of 67 seconds, the next RTO expires 15-25 seconds after the disconnection ends. In the latter case, less connection time is utilized for transmission.

Finally, the bottom row of Figure 13 shows the results with periodic connected periods of 67 seconds. With disconnected periods of 37 seconds (on the left), the performance improvement of the retransmission trigger is modest: standard TCP transmits in 595 seconds, the enhanced TCP requires 558 seconds. The improvement here is only 6%. This is again due to the relative timing of the next RTO after the disconnection. With disconnected periods of 67 seconds (on the right), standard TCP requires 765 seconds for the transmission, while TCP with the Retransmission Trigger only requires 560. The retransmission trigger improves performance by over 25% (6.27Mb/s vs. 8.57Mb/s.)

This section experimentally investigated the performance of a bulk TCP transfer across an intermittently connected path with repeated disconnections. Different experiments analyzed the performance of standard TCP and TCP with the TCP Retransmission Trigger. The measurement results indicate that the TCP Retransmission Trigger improves performance from 6% to over 50% in all measured scenarios.

5. RELATED WORK

This section discusses related work in the area of supporting intermittent connectivity. Related work falls into three major areas: mobility solutions, mobility solutions that include disconnection support and TCP performance enhancements.

5.1 Mobility Solutions

With *MobileIP* [21], a mobile host is always reachable at a static home address when connected to the network. A home agent in the mobile host's home network is responsible for transparently redirecting some packets to the mobile host's current network location. *MobileIP* allows legacy hosts to transparently communicate with *MobileIP*-enabled hosts. Home agents are new network infrastructure that must be deployed before *MobileIP* can operate.

Virtual IP [16] introduces a new *virtual IP* layer in between the network and transport layers. Similar to HIP, this new layer decouples transport-layer connections from network-layer IP. Transport-layer connections bind to *virtual IP* addresses that are negotiated between communicating hosts at connection setup. These *virtual IP* addresses remain valid throughout the lifetime of a connection. The *virtual IP* layer on the end-hosts maps *virtual IP* addresses into the current network-layer IP addresses of the communication peers. Unlike MobileIP, *Virtual IP* does not rely on new network infrastructure. However, all communicating hosts must implement *Virtual IP*. Compared to HIP, *Virtual IP* lacks a global naming scheme. *Virtual IP* addresses are negotiated between host pairs and are only locally valid.

5.2 Mobility Solutions with Disconnection Support

Several approaches, such as *Mobile TCP Sockets* [17], the *Persistent Connection Model* [18], *Reliable Sockets* and *Reliable Packets* [19] as well as *TCP Migrate* [20] combine mobility support with mechanisms to operate under intermittent connectivity. Most proposals introduce new functionality in between the socket and application layers. Applications open "socket-layer" connections to peers through new APIs that transparently reestablish underlying TCP connections when IP address changes or temporary disconnections abort them.

Such "middleware" approaches to tolerate intermittent connectivity are more complex than the TCP User Timeout Option. They must carefully synchronize communication state when disconnections occur, to prevent violations of TCP semantics due to corrupted delivery. Furthermore, existing applications must be modified to support the new APIs.

Similar to one aspect of the solution proposed in this paper, *Reliable Packets* [19] prevent TCP connections from aborting during disconnected periods. *Reliable Packets* use packet filters on the end hosts to modify TCP segments on the fly. During disconnections, this approach creates spoofed TCP segments that close the receive window to stop further data transmission. Modifying segments has a high risk of violating TCP semantics.

5.3 Transport Layer Performance Enhancements

The *Implicit Link-Up Notification* [22] and the *Smart Link Layer* [23] attempt to kick-start TCP transmissions in response to link-layer reconnection events. They maintain a buffer of recent TCP segments and retransmit duplicates when connectivity is restored, attempting to trigger a faster recovery. Both approaches operate at the network layer but interpret TCP header information to capture and buffer appropriate segments. While this is a layering violation, this approach also increases per-packet processing overhead and requires per-connection storage. Furthermore, network-layer security mechanisms such as *IPsec* [26] disable these techniques by encrypting TCP header information. Finally, retransmitting duplicates of buffered segments after the maximum segment lifetime of 2 minutes potentially violates TCP semantics. The TCP Retransmission Trigger described in this paper does not suffer from these limitations, because it operates at the transport layer at the end systems.

Other transport layer approaches such as *Explicit Link Failure Notification* [24] or *TCP-F* [25] use specific messages generated by intermediate routers to inform TCP senders about disrupted paths. The former extends the TCP state machine with a new "stand by" state during which the standard retransmission timers are disabled. Instead, TCP periodically probes the network to detect connectivity reestablishment. Depending on the frequency of the probes and the network environment, this can cause significant amounts of extra traffic. *TCP-F* completely suspends ongoing connections until receiving route reestablishment notifications that indicate peer reachability. Both

proposals are primarily designed for *ad hoc* networks and rely on changes to intermediate routers, whereas the TCP Retransmission Trigger only requires end system support. *ATCP* [41] uses a similar approach as the *Explicit Link Failure Notification*, but discovers link failures through *ICMP Destination Unreachable* messages. Cáceres and Iftode [42] propose and evaluate a solution similar to the TCP Retransmission Trigger that improves performance during MobileIP handoffs. Unlike the solution proposed in this paper, the handoff mechanism is targeted at disconnections of a few seconds and does not include mechanisms to tolerate extended periods of disconnection.

Finally, the current solution increases TCP performance through better utilization of connected periods. Other transport-layer mechanisms could further increase this utilization and thus further increase performance. TCP enhancements such as *rate-based pacing* [35], *fast start* [37], *control block sharing* [38][39] or *quick-start* [40] may all help to further speed up the slow-start after reconnection. Another transport-layer approach for optimizing the use of connected periods would exchange TCP for a completely different transport protocol, such as *XCP* [36].

6. CONCLUSIONS AND FUTURE WORK

This paper described problems of communication in intermittently connected scenarios. It proposed and experimentally evaluated a solution that combines the Host Identity Protocol with two new TCP enhancements, the TCP User Timeout Option and the TCP Retransmission Triggers. These enhancements allow unmodified applications and services to operate transparently in cases where the current Internet protocols fail to support their communication. In other cases, it significantly improves their communication performance.

The proposed protocol enhancements are a pure end-to-end solution. They do not require new or modified network infrastructure and are transparent to applications and services, because they maintain the semantics of the existing transport protocols. Although the proposed solutions are simple enhancements of the standard protocols, they greatly improve operation in common scenarios where the unmodified protocols fail or only operate inefficiently. Although designed as an end-to-end solution, the proposed protocol enhancements could initially be deployed in split-connection proxies. This approach allows unmodified end systems to benefit from the proposed protocol enhancements, but suffers from the usual drawbacks of splitting end-to-end connections.

The initial experimental investigation has demonstrated the effectiveness of the protocol enhancements in some usage scenarios. One area of future work is further experiments with the prototype implementation. Section 4 evaluated the proposed protocol enhancements under a single bulk transfer. Although this workload allows studying the effects of the enhancements in detail, it is arguably synthetic. Further evaluations will validate the performance improvements for more realistic workloads, including bursty traffic, multiple, concurrent connections and in the presence of cross traffic. It would also be interesting to investigate the effectiveness of the proposed TCP enhancements with different mobility solutions, *e.g.*, MobileIP.

The TCP User Timeout Option exchanges individual user timeouts for specific connections. The experimental evaluation in Section 4 assumed that the hosts choose user timeouts that are large enough to tolerate all disconnections. Typically, communicating hosts need to pick specific user timeouts based on local policies or heuristics, balancing resource utilization against the likelihood of premature connection termination. Future research will investigate possible mechanisms based on geographical or topological location and movement, resource utilization or connectivity history.

7. ACKNOWLEDGMENTS

We would like to thank the members and chairs of the IETF's TCPM working group for feedback on the different TCP extensions presented and evaluated in this paper. Additionally, Ralf Schmitz, Jürgen Quittek and a number of anonymous reviewers have provided valuable feedback on an earlier revision of this paper.

Part of this work is a product of the *Ambient Networks* project supported in part by the European Commission under its *Sixth Framework Program*. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the *Ambient Networks* project or the European Commission.

8. References

[1] Jon Postel. Internet Protocol. STD 5, RFC 791, September 1981.

[2] Jon Postel. Transmission Control Protocol. STD 7, RFC 793, September 1981.

[3] Robert Moskowitz and Pekka Nikander. Host Identity Protocol Architecture. Work in Progress (draft-ietf-hip-arch-02), January 2005.

[4] Randall R. Stewart, Qiaobing Xie, Ken Morneault, Chip Sharp, Hanns Juergen Schwarzbauer, Tom Taylor, Ian Rytina, Malleswar Kalla, Lixia Zhang and Vern Paxson. Stream Control Transmission Protocol, RFC 2960, October 2000.

[5] W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley, 1994.

[6] Jörg Ott and Dirk Kutscher. Drive-Thru Internet: IEEE 802.11b for "Automobile" Users. Proc. *IEEE INFOCOM 2004*, Hong Kong, March 7-11, 2004.

[7] Vern Paxson and Mark Allman. Computing TCP's Retransmission Timer. RFC 2988, November 2000.

[8] Lars Eggert and Fernando Gont. TCP User Timeout Option. Work in Progress (draft-ietf-tcpm-tcp-uto-00), May 2005.

[9] Lars Eggert, Simon Schütz and Stefan Schmid. TCP Extensions for Immediate Retransmissions. Work in Progress (draft-eggert-tcpm-tcp-retransmit-now-01), September 2004.

[10] Simon Schütz. Network Support for Intermittently Connected Mobile Nodes. *Diploma Thesis*, University of Mannheim, Germany, June 2004.

[11] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott and Howard Weiss. Delay-Tolerant Networking – An Approach to Interplanetary Internet. *IEEE Communications Magazine*, June 2003, pp. 128-136.

[12] M. Scott Corson, Joseph P. Macker and Gregory H. Cirincione. Internet-Based Mobile Ad Hoc Networking. *IEEE Internet Computing*, Vol. 3, No. 4, July/August 1999, pp. 63-70.

[13] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti and M. Frans Kaashoek. The Click modular router. *ACM*

Transactions on Computer Systems, Vol. 18, No. 3, August 2000, pp. 263-297.

[14] Mark Gates, Ajay Tirumala, Jon Dugan and Kevin Gibbs. Iperf User Docs. White Paper, National Laboratory for Applied Network Research (NLANR), March 2003.

[15] Miika Komu, Mika Kousa, Janne Lundberg and Catharina Candolin. An Implementation of HIP for Linux. Proc. *Linux Symposium 2003*, Ottawa, Ontario, Canada, July 23-26, 2003, pp. 97-105.

[16] Praveen Yalagandula, Amit Garg, Mike Dahlin, Lorenzo Alvisi and Harrick Vin. Transparent Mobility with Minimal Infrastructure. Technical Report TR01-30, University of Texas, Austin, TX, USA, July 2001.

[17] Xun Qu, Jeffrey Xu Yu and Richard P. Brent. A Mobile TCP Socket. Technical Report TR-CS-97-08, Department of Computer Science, Australian National University, Canberra, Australia, April 1997.

[18] Yongguang Zhang and Son Dao. A "Persistent Connection" Model for Mobile and Distributed Systems. Proc. *4th International Conference on Computer Communications and Networks*, Las Vegas, NV, USA, September 1995, pages 300–305.

[19] Victor C. Zandy and Barton P. Miller. Reliable Network Connections. Proc. *8th Annual International Conference on Mobile Computing and Networking*, Atlanta, GA, USA, 2002, pp. 95-106.

[20] Alex C. Snoeren and Hari Balakrishnan. An End-to-End Approach to Host Mobility. Proc. *6th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '00)*, Boston, MA, USA, August 2000, pp. 155-166.

[21] David Johnson, Charles Perkins and Jari Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[22] Spencer Dawkins and Carl Williams. End-to-End, Implicit "Link-Up" Notification. Work in Progress (draft-dawkins-trigtran-linkup-01), October 2003.

[23] James Scott and Glenford Mapp. Link Layer-Based TCP Optimisation for Disconnecting Networks. *ACM SIGCOMM Computer Communications Review*, Vol. 33, No. 5, October 2003, pp. 31-42.

[24] Gavin Holland and Nitin Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. Proc. *5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, WA, USA, 1999, pp. 219-230.

[25] Kartik Chandran, Sudarshan Raghunathan, S. Venkatesan and Ravi Prakash. A Feedback Based Scheme For Improving TCP Performance In Ad-Hoc Wireless Networks. *IEEE Personal Communication Systems (PCS) Magazine: Special Issue on Ad Hoc Networks*, Vol. 8, No. 1, February 2001, pp. 34-39.

[26] Stephen Kent and Randall Atkinson. Security Architecture for the Internet Protocol. RFC 2401, November 1998.

[27] Pekka Nikander, Jari Arkko and Tom Henderson. End-Host Mobility and Multi-Homing with Host Identity Protocol. Work in Progress (draft-ietf-hip-mm-01), February 2005.

[28] Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[29] Mark Allman, Chris Hayes and Shawn Ostermann. An Evaluation of TCP with Larger Initial Windows. *ACM Computer Communication Review*, Vol. 28, No. 3, July 1998, pp. 41-52.

[30] Mark Allman, Vern Paxson and W. Richard Stevens. TCP Congestion Control. RFC 2581, April 1999.

[31] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[32] JinHyeock Choi and Greg Daley. Detecting Network Attachment in IPv6 Goals. Work in Progress (draft-ietf-dna-goals-04), December 2004.

[33] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[34] Dan Harkins and Dave Carrel. The Internet Key Exchange (IKE). RFC 2409, November 1998.

[35] Vikram Visweswaraiah and John Heidemann. Improving Restart of Idle TCP Connections. Technical Report 97-661, University of Southern California, November 1997.

[36] Dina Katabi, Mark Handley and Charlie Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. Proc. *ACM SIGCOMM*, Pittsburgh, PA, USA, August 19-23, 2002, pp. 89-102.

[37] Venkata N. Padmanabhan and Randy H. Katz. TCP Fast Start: A Technique for Speeding Up Web Transfers. Proc. *IEEE Globecom '98 Internet Mini-Conference*, Sydney, Australia, November 1998, pp. 41-46.

[38] Joe Touch. TCP Control Block Interdependence. RFC 2140, April 1997.

[39] Lars Eggert, John Heidemann and Joe Touch. Effects of Ensemble-TCP. *ACM Computer Communication Review*, Vol. 30, No. 1, January 2000, pp. 15-29.

[40] Amit Jain, Sally Floyd, Mark Allman, Pasi Sarolahti. Quick-Start for TCP and IP. Work in Progress (draft-amit-quick-start-04), February 2005.

[41] Jian Liu and Suresh Singh. ATCP: TCP for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas in Communication*, Vol. 19, No. 7, July 2001, pp. 1300-1315.

[42] Ramón Cáceres and Liviu Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 5, 1995, pp. 850-857.

[43] John Border, Markku Kojo, Jim Griner, Gabriel Montenegro and Zach Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, June 2001.

[44] Julien Laganier and Lars Eggert. Host Identity Protocol (HIP) Rendezvous Extension. Work in Progress (draft-ietf-hip-rvs-01), February 2005.

[45] Alberto Medina, Mark Allman and Sally Floyd. Measuring Interactions Between Transport Protocols and Middleboxes. Proc. *ACM SIGCOMM/USENIX Internet Measurement Conference*, Taormina, Sicily, Italy, October 2004, pp. 336-341