Network Working Group
Internet-Draft
Expires: August 5, 2006

Y. Swami
K. Le
Nokia Research Center, Dallas
W. Eddy
NASA GRC/Verizon FNS
Feb 2006

Lightweight Mobility Detection and Response (LMDR) Algorithm for TCP
draft-swami-tcp-lmdr-07

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 5, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

TCP congestion control is based on the assumption that the end-to-end path of a connection changes only insignificantly after connection establishment. Network layer mobility protocols that change a connection's point of attachment transparently to the transport layer may violate this assumption and cause TCP to make congestion control decisions based on invalid information. This document describes a

TCP option that allows a connection endpoint to inform a peer when it changes location. This document also outlines the proper congestion control behavior that should take place in the face of such network layer mobility.

1. Introduction

TCP congestion control [1] assumes that once a connection is established, the end-to-end path it traverses is relatively stable. Several network layer mobility protocols may be in use underneath TCP, which are capable of quickly changing a node's point of Internet attachment (and thus the end-to-end connection path) without notifying TCP. This can easily lead to invalid TCP congestion control decisions. Examples of such network layer mobility protocols include Mobile IPv4 [2], Mobile IPv6 [3] and HIP-based mobility [4].

When a TCP sender or receiver changes its point of attachment to the Internet (henceforth referred as "changes subnets"), the entire end-to-end path between the sender and receiver can change. In many host mobility scenarios, it is expected that only a small portion of the path nearest to the mobile node changes, and the links exchanged are assumed to be heterogenous. This makes the path change have little effect on TCP congestion control state, and so in-progress connections are mostly oblivious to the change. However, it is easy to envision mobility scenarios where large portions of the end-to-end path change. For instance, a mobile device may transition between wireless service providers, and thus have its packets routed over distinct backbone networks. A host may also have multiple interfaces (perhaps of widely varying media type) and change its IP connectivity over from one to the other as signal levels change. If the interfaces are of different speeds or the networks at different loads, the paths can change significantly.

There are several problems with allowing topologically-significant path changes to occur transparently to TCP. There is no guarantee that the congestion control state associated with the old path has any meaning for the new path, and the congestion control and RTTM state should be reinitialized. ACKs received for packets sent on the old path do not indicate the congestion state of the new path, and should not be used in the computation of TCP's congestion window. Slow-start based probing from the initial window should take place in the new path, with the slow-start threshold (SS_THRESH) reset to its initial value. These actions should take place in both directions of a TCP connection. It is relatively easy for the mobile node to perform these congestion control modifications when it moves, but the host on the other side of the connection has no means of inferring the path change.

In this document, we describe a network-layer-independent mechanism by which mobile hosts can propagate path-change notifications to their peers, based on which both sides can react to correct their performance. We assume that a mobile host always knows about its own subnet information (for example, by looking at its neighbor cache,

destination cache, default router, or a combination of these [5], but it is not able to inform its peer of subnet changes without implementing the TCP option described in this document.

While some network layer mobility management techniques may be used to indirectly derive a remote peer's mobility information (e.g. by looking into the binding cache when using Mobile IPv6 with route optimization), such techniques are not available with other network layer mobility protocols such as Mobile IPv6 with reverse tunneling, Mobile IPv4, or traditional cellular networking. This motivates the need for a means of signalling such mobility information.

Other modern transport protocols have features similar to LMDR. For example, in spirit, LMDR is similar to the "Reset Congestion State" option in DCCP [11], which is part of the base specification. DCCP has been designed more recently than TCP, with mobility as a possible consideration from the beginning. Originally, TCP was not designed with mobility in mind, and so, understandably, lacks mobility-support features. Adding LMDR to TCP brings the protocol more up-to-date with regards to mobility support, and extends the range of viable environments where TCP can be effectively used.

This document does not describe a response to link-up/link-down events. Link-up/link-down events are triggered by link layer state changes which may or may not indicate subnet change. For example, unplugging and replugging an Ethernet cable constitutes a link-up/ link-down event, even though the host might remain in the same subnet after replugging the cable. This document does not specify the processing of such events, instead the protocol described in this document acts only after detection of attachment to *new* subnets.

2. Terminology

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," "OPTIONAL," and "silently ignore" in this document are to be interpreted as described in RFC 2119.

Mobile Node (MN): An end-host capable of changing its point of attachment to the Internet without breaking transport layer connectivity. Hosts that change their point of attachment to the Internet but use DHCP or other mechanism to get a new IP address are not considered in this document.

Corresponding Node (CN): An end-host that has active TCP connections with a Mobile Node. The corresponding node may itself be mobile without influencing the applicability of the protocol described in this document.

Old Subnet: MN's point of attachment to the Internet prior to movement. The Old Subnet is a component of the "Old Path".

New Subnet: MN's point of attachment after movement. "New Subnet" is a component of "New Path".

Initial Window: The initial congestion window size at the start of a connection as described in [6].

Stale ACK: When a New Path is in use, acknowledgements corresponding to data sent on the Old Path are termed "stale". These stale ACKs don't contain meaningful information about the new path and should be ignored for congestion window calculations on the new path.

3. Congestion Control Issues with Subnet Change

For concreteness, the description below assumes network mobility based on Mobile IP, but the same concepts are readily applicable to other network layer mobility protocols.

To illustrate the problems that transparent network layer mobility may cause for TCP congestion control, consider Figure 1. At time=T, MN is reachable on the Old Subnet through access router AR-1 and has the care-of address <Old Subnet, MN>. A TCP connection is established between MN and CN. While MN is attached to AR-1, packets between CN and <Old Subnet, MN> are routed using PATH-1 (through Cloud-1 and AR-1). Assume that at some time, T+1, MN moves and becomes reattached New Subnet, which is reachable through AR-2 with the care-of address <New Subnet, MN>. While MN is attached to AR-2, all packets between CN and <New Subnet, MN> are routed using PATH-2 (through Cloud-2 and AR-2).

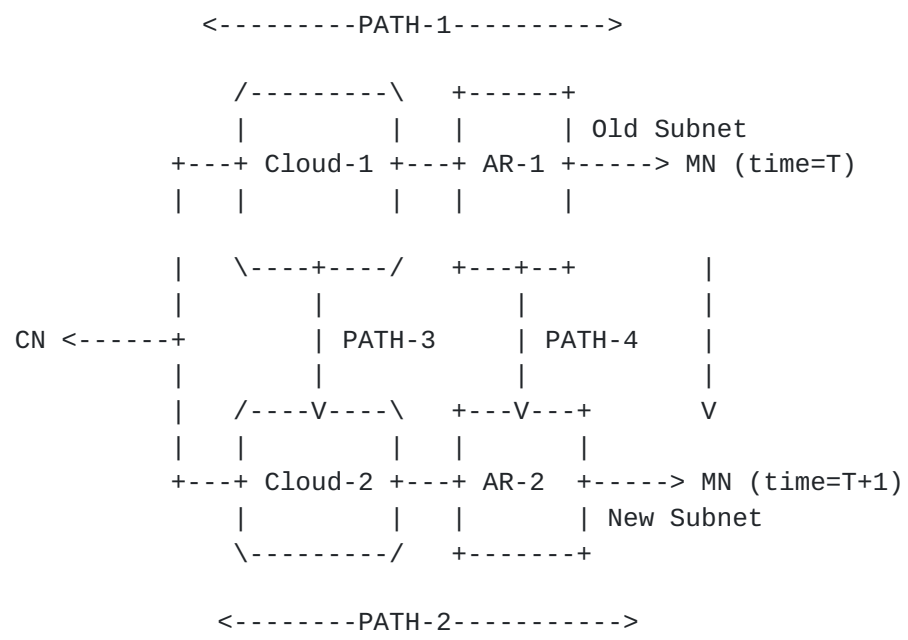

Figure 1

During the transitional period, when MN moves from Old to New Subnet, AR-1 might not be able to deliver packets it receives which are addressed for MN. This could result in a large burst of packet loss. To address this, there are several suggested means of doing "fast" or "seamless" handovers, which involve adding machinery in the ARs to buffer and redirect packets originally sent to the Old Subnet, to the New Subnet (e.g. [7]). These redirected packets may travel through either PATH-3 or PATH-4. The distinction between PATH-3 and PATH-4 is that PATH-4 may belong to a well-provisioned network specially

designed to accommodate extremely bursty traffic. On the other hand, in the absence of such a PATH-4, PATH-3 will be used, and may consist of more arbitrary routers without special provisioning.

Congestion control on PATH-1 is governed by basic slow-start and congestion avoidance mechanisms [1]. As long as MN remains in Subnet-1, standard congestion control algorithms is sufficient. But once it moves from Subnet-1 to Subnet-2, two different scenarios are possible depending on the network topology. Access routers may either buffer and forward packets via a PATH-4 (or PATH-3), or not.

In a typical Mobile IPv4 scenario, all packets destined to <Subnet-1, MN> are dropped by AR-1 once the mobile node has moved. Since the latency involved in establishing a new tunnel to the HA is of the order of RTT (2*RTT in case of Mobile IPv6), roughly an entire window worth of data and ACKs will be dropped by AR-1. Because of this window loss, the CN and MN are likely to take expensive retransmission timeouts.

In the alternative scenario, all packets destined to <Subnet-1, MN> are forwarded to <Subnet-2, MN> by AR-1 by some means [7]. AR-1 might forward packets to <Subnet-2, MN> using PATH-3 or PATH-4. These two cases are considered separately. If AR-1 forwards packets to AR-2 using PATH-3, PATH-3 may experience a sudden burst of packets. If multiple MNs move between AR-2 and AR-1, PATH-3 may easily become congested. The exact means of buffering and forwarding segments between the ARs is not guaranteed to occur in a manner relative to the congestion level of PATH-3, nor to conform to TCP's clocking expectations. This may be risky behavior. If PATH-4 is available, and used to redirect packets to MN, the resulting burst of packets may still be an issue with regards to clocking, even though congestion control on PATH-4 itself is not an issue.

Whether PATH-3 or PATH-4 is used, receiving stale ACKs (for data sent on PATH-1) will cause MN to wrongly inflate its congestion window. Stale ACKs do not provide any indication of the congestion state on the New Path, and should not be used for this computation. MN will also generate stale ACKs for any redirected data segments. This will similarly cause CN to improperly adjust its congestion window. If the congestion windows from the Old Path are already too big for the New Path, this may be a problem. Consider, for example, the case where a train moves across a subnet boundary due to wireless radio coverage limitations, and hundreds of mobile users on that train handoff to a new subnet. In this case, the New Subnet and Path will see a burst of segments that can cause unnecessary packet loss and timeouts.

Conversely, if PATH-2 is of greater capacity or more lightly loaded

than PATH-1, and if the sender is in congestion avoidance, it will spend multiple RTTs before reaching a reasonable throughput. This is due to the slowness of additive increase in probing available capacity, and caused by a value of SS_THRESH that has become irrelevant. Consider the case where the Old Path's available capacity was 10 segments, while the New Path can handle 1000 segments. With a normal timeout based loss recovery algorithm, the sender's SS_THRESH will be set to 10 segments, and reaching a reasonable window of around 500 segments (half of the available capacity) will require (log_2(10/2) + (500-5)) RTTs (recall that congestion window increase during congestion avoidance is just one packet per RTT). Contrast this with a scheme where the sender resets the SS_THRESH to a large value after subnet change and only spends log_2(500/2) RTT to reach a reasonable throughput.

4. Subnet Change Detection and Notification

For proper congestion control behavior in the face of mobility, a mobile node first needs to know if it has moved from one subnet to another, then it needs a means to propagate this information to its peer. Detecting when a mobile node has changed subnets can be performed using neighbor discovery [5]. In this document we assume that mobile hosts can determine their own subnet changes either from lower layers or through other out of band mechanisms. The remaining problem is relaying this change of path information to the other connection endpoint. A TCP option can be employed for this purpose.

The LMDR option holds a counter that represents the number of times a side has changed attachment points. At the start of the connection, both endpoints use this option in the SYN and SYN-ACK segments, with an initial counter value of 7, to advertise support for the option. A host MUST NOT place the LMDR option on a SYN-ACK unless it was present on the generating SYN. After the SYN exchange is completed, hosts SHOULD NOT send this option until there is a subnet change. After connection setup, the LMDR option is only generated by a host's detection of its own mobility, or in response to a received LMDR option. A host MUST NOT send the LMDR option during the course of a connection unless it was advertised by both sides at startup.

Figure 2 depicts the LMDR TCP Option format:

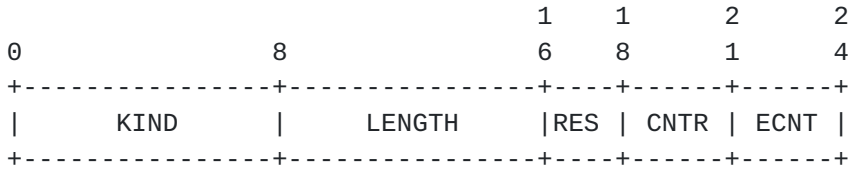

Figure 2

TYPE: (8 Bits) TCP Option Type. Value set to 25 for experimental purposes.

LENGTH: (8 Bits) TCP Option Length. Value = 3.

RES: (2 Bits) Reserved bits. Sender should set the value to zero. Receiver should ignore these fields.

CNTR: (3 Bits) The subnet counter value of the host sending this option. This value is decremented once for ever subnet change (i.e., if the mobile host moves from Subnet-A to Subnet-B, and the counter value in Subnet-A was C1, then the counter value in Subnet-B will be C1-1, wrapping back to 7 after 0).

ECNT: (3 Bits) The echoed value of CNTR. On reception of an LMDR option, a host copies the received CNTR value to the ECNT field of its response. The CNTR field is filled in with the host's own subnet counter value.

When there is a subnet change, the Initiator (the host that wants to inform its peer, the Responder, about the subnet change) decrements its counter and sends an LMDR option in every subsequent ACK or data segment, until it sees its new counter value echoed back. When the Responder sees an LMDR option, it echoes back the Initiator's counter. The Responder keeps echoing back the value until the Initiator stops sending the option. In the case of simultaneous movement by both sides of a connection, the side who sent the highest initial sequence number assumes itself to be the Initiator, and the other host assumes itself to be the Responder.

As an example, assume MN-A has a subnet counter CNTR-A = 5 and MN-B has CNTR-B = 3. If at some point MN-B moves to a new subnet, Figure 3 shows the LMDR options exchange.

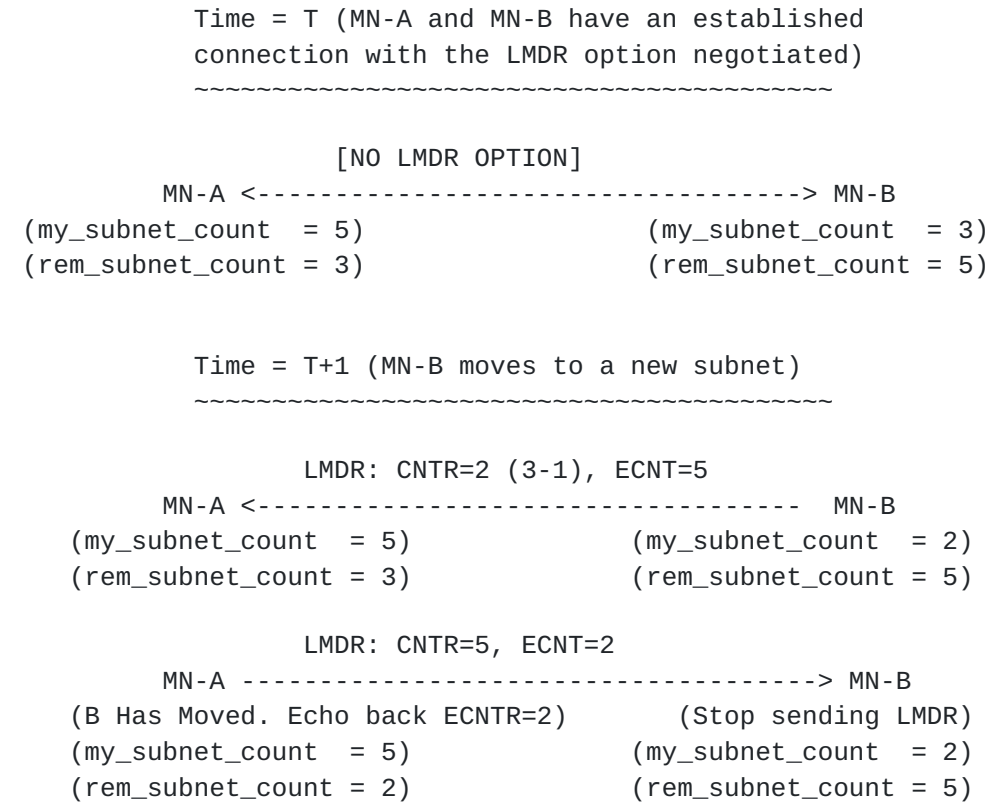

Figure 3

Each TCP implementation should keep three new variables: my_subnet_count, remote_subnet_count, and in_transition. The

variables my_subnet_count, and rem_subnet_count store the mobility counters for this host and the remote host respectively. The variable in_transition is set to one when the Responder receives the first LMDR option. The value is reset to zero when the Responder receives a packet without the LMDR option set.

For each packet sent, a host should determine if it has moved to a new subnet. If a host determines that it has moved, it SHOULD update the value of my_subnet_count as follows: my_subnet_count = (my_subnet_count - 1); in_transition = 1; The node that updates this value is referred to as Initiator. The Initiator SHOULD send an LMDR option for every packet as long as in_transition == 1. The Initiator MUST follow the congestion response algorithm described in Section 5. The Initiator MUST keep the in_transition value unaltered until it receives a packet with ECNT == my_subnet_counter; (i.e., until the recent most CNTR value is echoed back by the Responder).

When a host receives a valid TCP segment (one that meets the sequence number and ACK sequence number criteria of RFC 793 [8]), it should compare the value of 'CNTR' with the value of 'rem_subnet_counter.' If the two values match, the Responder should conclude that the Initiator has not moved and MUST NOT update its in_transition variable. (Although it MUST echo back the LMDR option. Note that in case of simultaneous move it will result in sending the option for every subsequent packet. To break this infinite loop, the host with largest Initial TCP sequence number should assume the role of Initiator.)

If the two values of remote_subnet_counter and the CNTR in a received LMDR option differ, a host can conclude that the other side has moved. The host MUST update the variables as follows: rem_subnet_count = CNTR; in_transition = 1; After making these changes, the host MUST follow the congestion response algorithm as described in Section 5. The value of in_transition SHOULD be reset to zero when the Responder receives a segment from the Initiator without the LMDR option.

NOTE: In certain network architectures it's possible that a mobile (and the associated link technology) have sufficient congestion information about the new path. In these cases, a node MAY choose not to indicate subnet change information to the sender since there is no need to reduce the data rate. However, the mobility information MUST be indicated if no such information is available, or if the congestion information is not for the entire path (i.e., if the congestion information is only for a part of the new path, then the Initiator MUST inform about subnet change). It might be possible to use the reserved bits in the LMDR option for some advantage in such situations, however, this document does not discuss the matter

further.

5. Congestion Response After Subnet Change

The motivations behind a congestion response after subnet change are to prevent a large burst of congestion on PATH-2 and to quickly probe the path's available capacity. In principle, the congestion control state for PATH-2 has the same requirements as that of a new connection: The sender should transmit no more than a default initial window of data outstanding on the New Path, in order to prevent over-congesting it, and the slow-start threshold (SS_THRESH) should be set to a large value, to allow for rapid probing of available capacity. What makes this slightly more complex is that connections after subnet change may have segments in flight from before the subnet change. Therefore, after subnet change, congestion control MUST ignore any stale ACKs and MUST update the congestion window based solely on ACKs for data sent on new path.

The LMDR congestion response to subnet changes can be described in two steps:

1. When a TCP end-host concludes that there has been a remote subnet change, its value of in_transition is set to one (as described previously). At this time, it may send INIT_WINDOW worth of data on the new path and MUST reset the congestion control state, RTTM state, and RTO timer as if this were a new connection [1][9]. This applies whether the host detects its own subnet change, or infers a movement by the other side of the connection via a received LMDR option.

2. For each stale ACK received, a host MUST NOT adjust the congestion window and MUST NOT send any new data into the network. This behavior should continue until in_transition is zero again or there is a timeout. Once in_transition is set to zero, the sender should consider any unsacked segments below the highest received ACK or SACK as lost, and discount them from the segments in flight. The sender MUST use slow-start based loss recovery for these segments.

There are several ways that a host may detect stale ACKs. In the simplest case, when the SACK Option is enabled, the host stores the highest sequence number of data in its retransmission queue before the movement, and uses the sequence number as a marker to determine whether the ACKs (and SACK in case of loss) are stale or not. Only after the host receives an ACK or SACK greater than the marker value, does it increase it's congestion window. A host could use other techniques, either independently or in conjunction with different TCP options (such as time stamps) to achieve similar results.

In case no reasonable means is available to disambiguate stale ACKs

from new ACKs, a host could use the LMDR option indirectly. For example, an LMDR Responder can assume that as long as it continues seeing LMDR options on incoming segments, the LMDR Initiator has not received an echo of its new counter value. Since the Initiator will stop sending the LMDR option after receiving the echoed value, the first segment where the Responder stops receiving the LMDR option is a good indication that the Responder's packets have completed one round trip. This strategy works well if the Initiator and Responder don't move simultaneously (i.e., the two sides don't move within an RTT duration.)

When a mobile node moves from a low-speed link to a high-speed link, there is a possibility that the packets sent on the new path reach the other side before the low-speed queue is cleared. This could cause severe packet reordering. Since the LMDR response algorithm assumes that out of order packets are lost, in some cases the response algorithm might unnecessarily resend data on the new path. We believe that this behavior will occur infrequently, given that the delay in establishing the new route often takes greater than a round-trip time. If a host wants to reduce the possibility of such unnecessary retransmission, it MAY wait for one RTT (measured on the old path), from the time of movement before increasing its congestion window.

6. Architectural Considerations

The LMDR technique described in this document does not add any new requirements to the network. The LMDR modifications are strictly to end-hosts, making them perform properly with regards to congestion control, in a way more friendly to the network. LMDR addresses a problem created when transparent network layer mobility protocols modify lower layers of the protocol stack without considering the possible ill-effects for higher layers [10].

7. Security Considerations

Use of the LMDR option does not open up a TCP connection to any form of abuse not already present in TCP. If an attacker possesses the ability to generate segments that would normally appear valid and acceptable to a TCP stack, then the attacker might produce a stream of LMDR options that could keep a connection in slow-start at the initial window. This is probably less serious than other attacks such an adversary could perform, however, like resetting the connection or injecting data, and a similar effect could be achieved without the LMDR option by forging duplicate ACKs that would keep a sender in loss recovery. If both sets of IP addresses, port numbers, and sequence numbers are guessable for a connection, then the connection should use IPSec for protection against spoofed segments.

8. Acknowledgments

We would like to thank Lars Eggert, Ilkka Oksanen, Shashikant Maheshwari and Mark Allman for their comments and suggestions.

9. References

[1] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[2] Perkins, C., "Mobility Support for IPv4", RFC 3344, August 2002.

[3] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

[4] Nikander, P., Arkko, J., and T. Henderson, "End-Host Mobility and Multi-Homing with Host Identity Protocol", Internet Draft (work in progress), July 2004.

[5] Narten, T., Nordmark, V., and W. Simpson, "Neighbor Discover for IP Version 6 (IPv6)", RFC 2461, December 1998.

[6] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[7] Koodli, R. and C. Perkins, "Fast Handovers and Context Transfers in Mobile Networks", ACM Computer Communication Review (31) 5, October 2001.

[8] Postel, J., "Transmission Control Protocol", RFC 793, September 1981.

[9] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[10] Eddy, W., "At What Layer Does Mobility Belong?", to appear in IEEE Communications, 2004.

[11] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", Internet Draft (work in progress), July 2004.

Authors' Addresses

Yogesh Prem Swami
Nokia Research Center, Dallas
6000 Connection Drive
Irving, TX 75603
USA

Phone: +1 972 374 0669
Email: yogesh.swami@nokia.com

Khiem Le
Nokia Research Center, Dallas
6000 Connection Drive
Irving, TX 75603
USA

Phone: +1 972 894 4882
Email: khiem.le@nokia.com

Wesley M. Eddy
NASA GRC/Verizon FNS
NASA Glenn Research Center
21000 Brookpark Road, MS 54-5
Cleveland, OH 44135
USA

Email: weddy@grc.nasa.gov

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.