2009 First International Conference on Future Information Networks, October 14-17, Beijing, China

PETS: Persistent TCP using Simple Freeze*

Chakchai So-In[1], *Student Member, IEEE*, Raj Jain[1], *Fellow, IEEE*, Gopal Dommety[2], *Member, IEEE*

[1]Department of Computer Science & Engineering, Washington University in St. Louis, MO 63130 USA
[2]IP Mobility Group, Internet Technologies Division, Cisco Systems, CA 95134 USA

Mobile applications often get disconnected because TCP times out when a user moves from one location and reconnects at another location. This happens even with the use of Mobile IP since the Mobile IP hides the IP address change from TCP but does nothing to prevent it from timing out. TCP freeze is a technique that is already part of TCP implementations and allows a receiver to stop the transmitter from sending further data when the receiver's buffers are full. In our proposed PErsistent TCP using Simple freeze (PETS) framework, we combine the TCP freeze and Mobile IP to prevent TCP from disconnecting during mobile operations. No modifications of the TCP or IP protocol are needed. The required PETS module can be implemented in mobile agents which are part of the Mobile IP or in IP routers. This flexibility allows our technique to be deployed incrementally with PETS-aware hosts or PETS-aware routers intermixed with legacy hosts and legacy routers with or without the Mobile IP.

Index Terms—TCP Persistence, Persistent Connection, PETS, TCP Freeze, Mobile IP, Mobility, Connection Reliability.

I. Introduction

Mobile users lose their TCP connections when they move or their IP addresses change or any nodes along the routing path fails. Also, variability in the quality of the wireless medium may cause temporary loss of connectivity resulting in TCP connection terminations. This can be very inconvenient. For example, after downloading for 20 minutes, the users would prefer having a FTP client continue to download after a temporary disconnection rather than starting the download at the beginning of the file. Another example is that if the users are making a banking deposit or withdrawal, and the modem line is interrupted for a few seconds, or the users need to change rooms immediately and then reconnect in different locations, during the incomplete transactions, it would be preferable if the users continue the transaction or processes without the loss of. Thus, there is a need for TCP connections to be maintained in a simple and efficient way.

A. Mobility in Protocol Layers

Mobility can be classified according to the layers of an Open System Interconnection (OSI) reference model, that is, from layer 1 mobility to layer 7 mobility [1]. Layer 1 is the physical layer. An auto-dial to select the best connection or to connect to a protection path would be an example of layer 1 mobility. Forward Error Control (FEC) and Automatic Repeat reQuest (ARQ) are techniques for layer 2 or the link layer mobility. The network layer concerns the change of IP address. Mobile IP [2] is a very good example of layer 3 mobility. A similar concept is Virtual Private Network. Modifying Internet Key Exchange (IKE) Keep-alive timer or IP Security (IPsec) Re-keying are examples of VPN mobility techniques.

Layers 1 through 3 are generally connectionless and, therefore, mobility at these layers does not involve *connections*. The concept of a connection is first introduced in layer 4 (transport layer) and above. TCP, one of the transport layer protocols, is the main focus of this paper since most of the Internet traffic uses connection-oriented TCP [3].

Mobile IP allows a user to be reached even when she is in a foreign network. However, the Mobile IP does not solve the problem of TCP persistence. During a mobile handoff, TCP connections can get disconnected. Also, the Mobile IP does not solve the problem of TCP disconnection when the IP address does not change but there is a temporary disconnection. From here on, we will use the term *connection* and *TCP connection* interchangeably. Our goal is to maintain the TCP connection.

B. Why TCP Disconnects?

The first question about TCP persistence is what causes TCP disconnection? Basically, a TCP connection is broken if there is no TCP keep-alive response back from the other end within some threshold. An experimental study of various factors affecting different commercial TCP implementations is presented in [4]. The results show that Retransmission TimeOut (RTO) is the primary factor causing TCP disconnections. For example, under Windows 95, TCP retransmits the dropped segments up to five times. RTO is increased exponentially up to the upper bound of 260 seconds. After that, the connection is terminated.

Keep-alive timeout is the second factor affecting TCP disconnections. Keep-alive timeout is an optional parameter [5]. Browsers, e.g., Internet Explorer, often set it to 1 second. If a response is not received within this interval, the connection is closed. The third factor is the connection timeout. Typically, this factor is defined by the vendor; there is no standard. This factor is based on the predefined time units or the number of retransmissions.

One way to resolve the problem of disconnection is to modify TCP stack directly. Making the TCP connection active indefinitely is good only in a very few cases. In fact, it is impractical and also breaks the design purpose of TCP timers, such as to distinguish network congestion from link failure. Therefore, we propose to use TCP Freeze to maintain the TCP connection during the short term disconnection on top of

Manuscript received April 25, 2009, revised July 30, 2009. *This work was partly done as a summer intern program at Cisco Systems, 2006.

Corresponding authors: C. So-In and R. Jain (e-mail: cs5 and jain@cse.wustl.edu), G. Dommetry (gdommety@cisco.com).

Mobile IP. Our goal is to come up with an easy algorithm with small modifications. Moreover, we do not consider the concept of socket migration or TCP stack modification.

The rest of this paper is organized as follows: Section II gives a brief survey of related work. TCP Freeze technique is described in Section III along with the details of the protocol design and architecture. The connection persistence schemes in mobile scenarios are discussed in Section IV. Experimental results are presented in Section V. Finally, conclusions are summarized in the last section, Section VI.

II. Related Work

The first work relating to the problem of persistent connection was presented by Yongguang Zhang and Son Dao [6]. They suggested mapping between logical and physical endpoints in terms of a virtual address. This address is composed of 3-tuples: socket descriptor, IP address, and TCP port number. However, this paper does not describe the implementation details. Similar to the virtual address concept, the Connection Identifier (CID) was introduced to uniquely identify the connection. The CID may be composed of many components such as random number, socket state information, IP address, and TCP port number. Another technique is to insert a new layer on top of the transport layer to hide the disconnection directly from application [7, 8]. Instead of inserting a new layer above the transport layer, a special layer called a Mobile Socket Layer (MSL) [9] can be inserted below the TCP socket layer as well.

The techniques described above, all modify either a client or a server or both in order to insert the extra layer that hides the disconnection from TCP or the application. However, it may be impractical to modify all nodes. In [10], the indirection concept, called MSOCK, was introduced to redirect all traffic through an extra proxy server. That proxy server maintains all TCP connection states for both ends. The proxy server can transparently splice the connection to alternate servers while ensuring the connection consistency.

TCP freeze modification, called *rack*, was first used in [11] to prevent the local TCP socket from aborting due to a connection failure. It is also used in wireless networks for the purpose of loss and congestion indication [12].

Another technique is to introduce an extra TCP state, called MIGRATE_WAIT state [13], which is basically the waiting state. When a link is disconnected, the TCP state is changed to the waiting state (there is no data sent or received). The general idea is similar to that in TCP Redirection (TCP-R) [8]. When a client changes an IP address or a link is broken, the extra layer hides the broken link from the transport layer. It stops data transmission (by going to MIGRATE_WAIT state) and freezes the application data or buffers the application data. Bytes sent and received numbers are stored. It also buffers the intermediate data received while the link is broken.

Our scheme, PETS, is very simple and easy to implement. There is no concept of CID as that in [6, 7, 9]. With the Mobile IP agent built in at the client, a PETS module can be added to the agent. There is no extra proxy server. Also, no TCP protocol modification is required. The PETS is similar to *rack* [11] with the use of TCP freeze, but we provide the details of the trade-off in order to retrieve the correct acknowledgement information. We also discuss the issues of link failure detection. In addition, we have specifically applied the use of TCP freeze in the Mobile IP environment.

III. TCP Freeze Technique

The TCP Freeze technique has already been implemented in standard TCP implementations in that when the receiving end host does not have enough buffer space left, it informs the sender to stop sending more packets. The basic idea is to freeze all TCP timers when the receiving window is full. Figs. 1 and 2 show an example of TCP Freeze operations.

In Fig. 1, the sender transmits a packet. The receiver sends back an acknowledgement of the packet along with an updated window of four packets (TCP window is in bytes but here we use packets for simplicity). The sender transmits four more packets as allowed by the window size. When the receiver has received all four packets, but for some reasons, it does not want the sender to transmit any more packets, it sends the acknowledgement of the last received packet with a window size of zero. Such an acknowledgement is called Zero Window Advertisement (ZWA). After receiving the ZWA, the sender knows it cannot send any more packets.

Instead of terminating the connection, the sender enters a "persistence" mode and periodically sends a probe packet to the receiver. This packet is called Zero Window Probe (ZWP). After receiving the ZWP, the receiver sends the ZWA to the sender with the same acknowledgement number and still window size of zero. As shown in Fig. 2, this process keeps on repeating and the connection is not terminated. When the receiver wants to continue the transmission, it sends the acknowledgement with a non-zero window. After that, the normal operation continues.

A. Protocol Design

To make use of TCP Freeze technique, we make use of "PETS modules" of the two sides of TCP connections. These modules snoop on the TCP connection and observe the acknowledgements and windows. The modules consist of a software code inserted between layer 3 and 4 (IP and TCP). This service can be provided by the routers, and the PETS modules can be in the routers adjacent to the end nodes as shown in Fig 3. It is also possible to have the PETS modules in the router on one side and in the end-host on the other side. This flexibility allows our scheme to work with unmodified end-hosts with modified routers, or modified end-hosts with legacy routers.

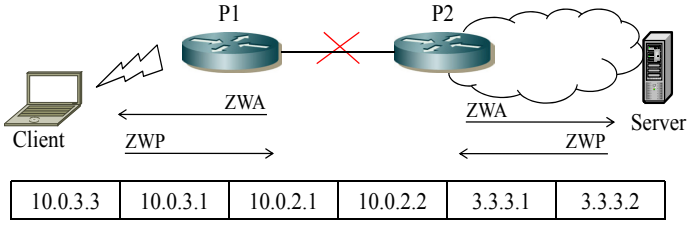

Fig. 3. Setup with PETS implementation in Routers

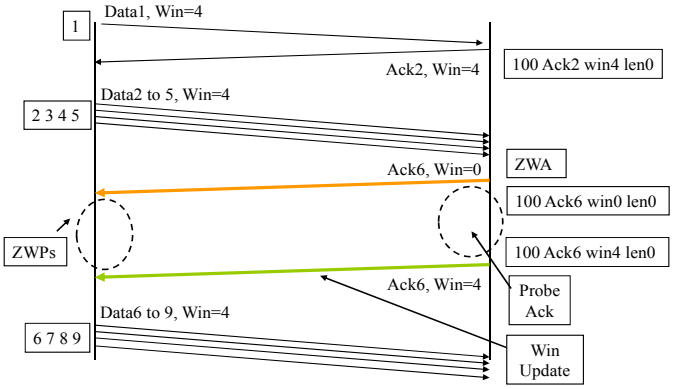

Fig. 1. TCP Freeze Operation

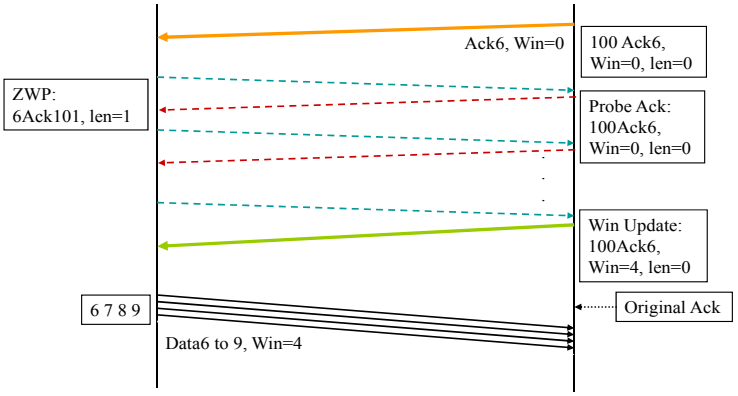

Fig. 2. TCP Freeze diagram

The PETS modules act as proxies for the TCP hosts when any links on the path between the two PETS modules fail. During the failed state, the PETS modules send ZWA or ZWP to the local host TCP as required to keep the TCP connection alive. In the following discussion we use the term "link" to mean any links on the path between the two PETS modules. If the PETS modules reside in the routers, failure of the link between the end-host and the router is not covered by our scheme.

The main issue for the PETS modules is to find *the correct acknowledgement number*. To retrieve that number, there are three techniques: stateful approach, iterative approach and best-guess approach [14]. In the stateful approach, all states are stored in the PETS module. Although we can acquire the real acknowledgement number, we have to keep many states during the normal operation (when the link is not broken). It also increases the delay and latency during normal operation.

In the iteration approach, the real acknowledgement number can be obtained from three duplication acknowledgement packets, when a link is disconnected. In this approach, number of states kept is less than that in the stateful approach; however, we may spend too much time in getting the real acknowledgement number. In some cases, the application idle timer may expire and consequently the connection may be terminated by the application.

In the best-guess approach, the PETS modules track the retransmission packets for each connection. Whenever the second retransmission packet is received, it is assumed that a link is disconnected. The local PETS module immediately sends ZWA back to the TCP layer in the local host based on the least sequence number seen (if there are many in-flight packets). When the link is disconnected and retransmission timeout (RTO) expires, TCP congestion window (CWND) is normally reduced to one (which means that the TCP can send out at most one packet). The sender can retransmit only the first unacknowledged packet. That number is used as an acknowledgement number for ZWA.

To make use of TCP Freeze, we also need a link state detection mechanism. For this, we chose a heartbeat mechanism in our design. The flow chart of PETS algorithm is illustrated in Fig. 4. In the figure, there are two parts: link state detection (above dashed line) and connection persistence mechanism (below dashed line). The algorithm uses three timer parameters: keep-alive timer, inactivity timer, and fault indication timer.

For link state detection, the PETS modules keep watching all TCP flows. If there is any data flowing through the module, it indicates that the link is not disconnected. Then, the PETS modules just forward the packets out (normal state). If there is no data for some pre-defined time and the inactivity timer expires, the PETS modules send a keep-alive packet consisting of an Internet Control Message Protocol (ICMP) message to check if the link is disconnected or it is just idle. The time interval and the number of ICMP messages are set as a function of Round Trip Time (RTT). If the keep-alive time is too small, it will slow down the overall operation. However, if it is too large, it may take too long to detect the disconnection and it may cause termination of TCP connections.

The ICMP messages are replied by the PETS modules on the other side. If the PETS modules do not receive the ICMP reply within some threshold, the assumption is that the link is disconnected and the state operation is moved to a disconnection state. A flag *Wf* is used to indicate the state of the link. The flag *Wf* is zero during normal operation and is set to one if the link is disconnected. On sensing disconnection, the PETS modules set *Wf* flag to one and start a fault indication timer. After that, each connection' socket state is hashed. The socket states are defined by the 4-tuples: source IP address, destination IP address, source port number, and destination port number. The sequence number is stored in the hash entry although it is not used to compute the hash index. The hash table design is a dynamic hash with a standard hashing algorithm. The size of the table can be updated dynamically based on the number of connections.

After hashing, a socket state is stored in the table if it is not a duplicated socket state. This indicates that this is a new packet since the disconnection occurred. However, if it is a duplicated socket state, the algorithm compares the sequence numbers and updates that number in the table. This is done until the fault indication timer expires. If the fault indication timer expires or the stored sequence number is same as that in the table, a ZWA packet is sent to the local TCP. The first condition ensures that the ZWA packet is sent in a timely manner. The second condition implies that the received packet is a "second retransmission" and, therefore, the ZWA packet needs to be sent.

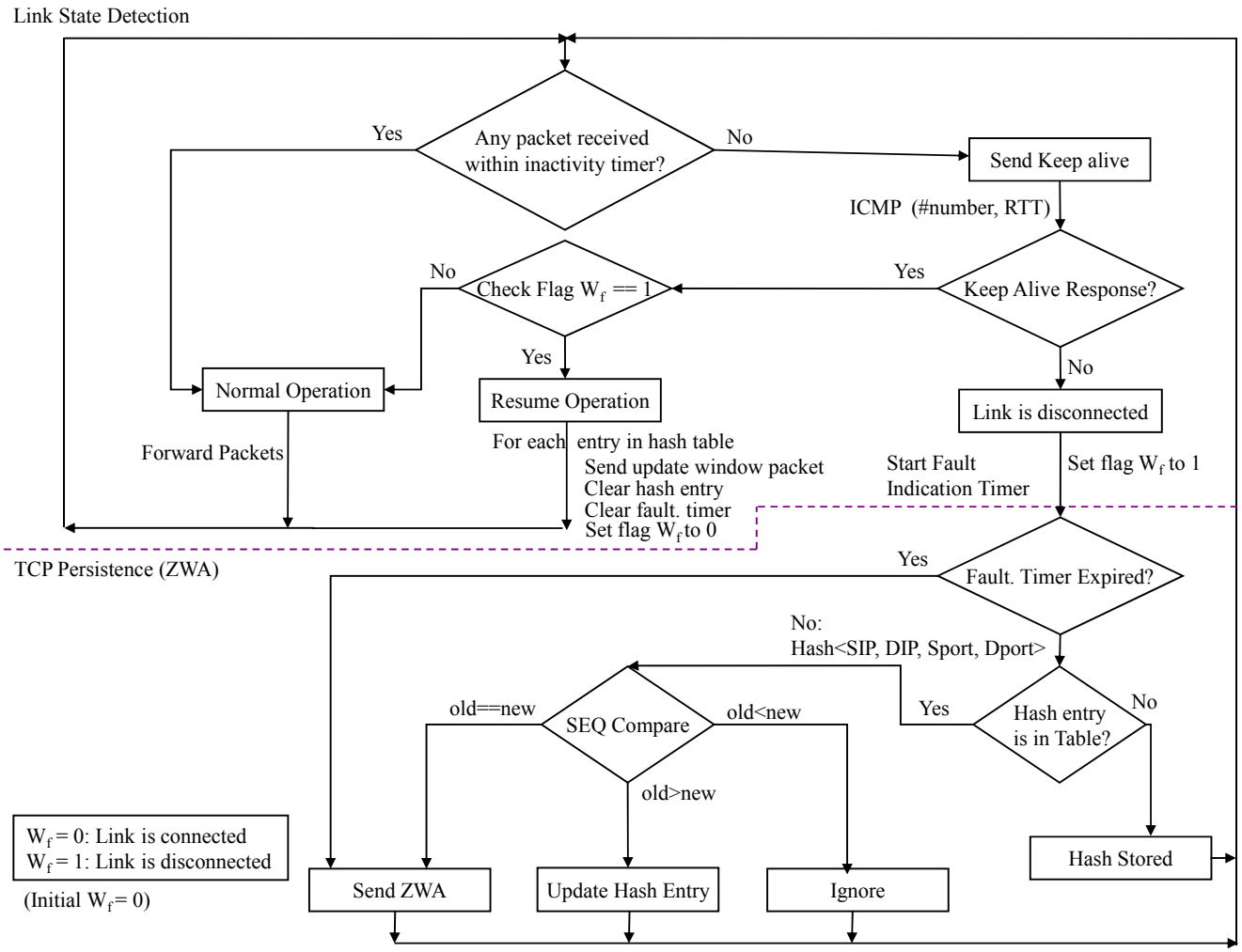

Fig. 4. Link detection and PETS algorithms

On the other hand, if the PETS modules receive the ICMP reply, it is obvious that the link is in operation -either normal operation or the link has just resumed after a failure. We can determine this based on the flag *Wf*. If the flag *Wf* is zero, the link continues to be in a normal state and the packet is forwarded as usual.

However, if the flag *Wf* is one, the link was disconnected and has just resumed. Therefore, updated window packets are sent back to the local TCP of each connection whose state is stored in hash table. Then, the hash table is destroyed. Also, the flag *Wf* is cleared to zero. The hash table is initialized again when the PETS modules sense a link disconnection.

The fault indication timer helps prevent the situation in which the PETS modules keep receiving sequence numbers different from the stored sequence number. The equality of stored and received sequence numbers indicates a "second retransmission" and provides a good sequence number for ZWA. Without a sequence match, the ZWA will not be sent. This may happen for a long time and TCP may disconnect.

This situation is avoided by the fault indication timer. When the fault indication timer expires, the ZWA is sent regardless of the sequence numbers. This is also shown in Fig. 4.

Intuitively the best-guess approach outperforms the stateful and iteration techniques since there are fewer states, especially since no state information is kept during the normal operation. However, there are a few issues with this algorithm in some cases. The probability of these cases is extremely low. For example, if RTO expires for some reason other than a real disconnection; the TCP congestion window (CWND) is increased from one to two. If this happens just after a disconnection, TCP sends out two more packets instead of just one. Then, the PETS modules may send the ZWA with a wrong acknowledgement number. Consequently, the application loses data. One way to avoid this is to use three retransmissions as the indication of the disconnection. However, since the retransmission timer is increased exponentially, the longer the algorithm is delayed the chances of application idle timer expiring also increases.

IV. PETS IN MOBILE SCENARIOS

Mobile IP helps ensure that the mobile host can still be reached at the old IP address (e.g., home address) regardless of the actual location of the host, but it does not ensure the connectivity if any links get disconnected. By combining PETS with Mobile IP, we can handle both disconnections and mobility.

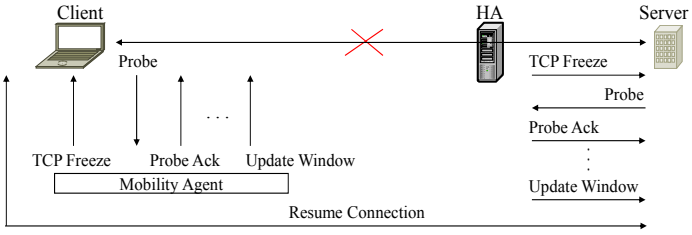

Fig. 5. Connection disconnection with HA

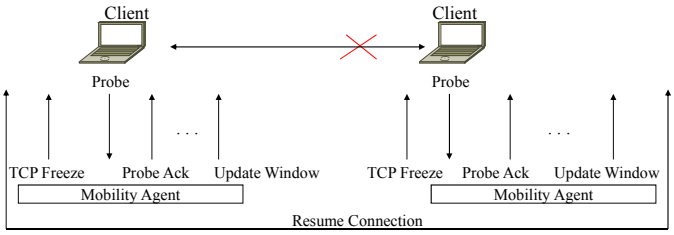

Fig. 6. Connection disconnection without HA

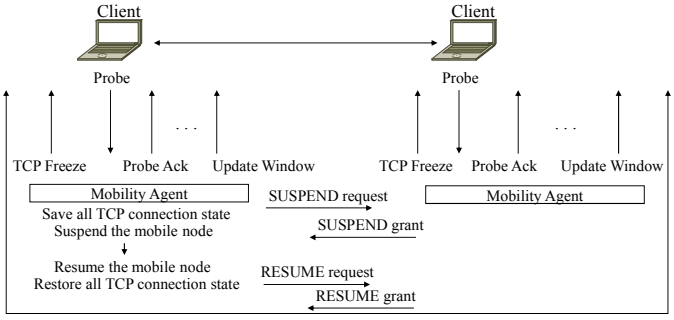

Fig. 7. Mobile node suspension with HA

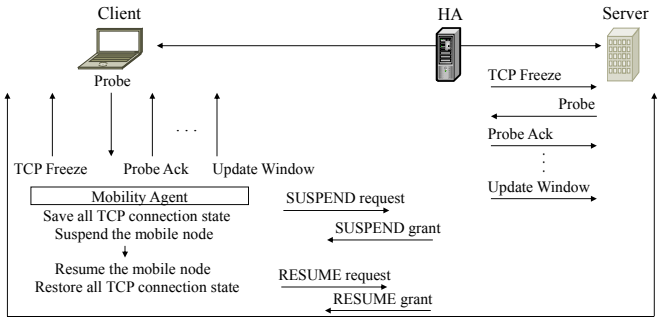

Fig. 8. Mobile node suspension without HA

With Mobile IP, PETS can be optionally implemented in Home Agent (HA). This helps provide TCP persistence service for PETS-aware hosts. Fig. 5 shows how the TCP persistence is achieved with the HA that has an internal PETS implementation. The server in Fig. 5 is PETS unaware. The client has a local PETS module. When the client's PETS module and HA detect the disconnection, they each send a ZWA packet to their TCP hosts.

Fig. 6 shows TCP persistence between two PETS-aware Mobile IP hosts. In this case, we do not need to implement PETS in the HA.

A related issue is that when the mobile host goes into a suspended state ("sleep" or "hibernation" mode). The PETS technique can still be applied. However, the PETS module has to store the TCP connection states which basically are IP address, port number, bytes-received/sent, SYN, and acknowledgement number before going to the suspended state. Figs. 7 and 8 show the use of the PETS technique with mobile host suspension. In Fig. 7, the HA has an internal PETS implementation. The process is almost the same as that in Fig. 5. However, before going to a suspension mode, the PETS module in the client sends a "SUSPENSION request" to the HA to inform about the suspension. After the request is granted, the PETS module saves all TCP connection states and the client goes into the suspension mode. After the HA grants the suspension, it keeps the other end host connection open by the request. It also sends the window update packet to the other end host.

Fig. 8 shows mobile host operations if both end hosts are PETS aware. In this case, HA does not need to have an internal PETS implementation. It should be mentioned that the link between the HA and PETS-unaware host is not protected by this technique.

V. Experimental Results

In this section, we describe the experiments with the PETS technique for two different commercial applications: FTP and Telnet. We used a testbed setup similar to that shown earlier in Fig. 3. We have not yet developed the host-based PETS modules. Instead, we used router-based PETS modules. Traffic generator node (TGN) and Packet Count and Capture (PKTS) [15] are the router software packages that were used to capture all traffic and generate the TCP packets in these experiments.

Basically, a TCP connection is setup between a client and a server through routers P1 and P2. The disconnection occurs at a link between the two routers. FileZilla FTP server and BFTelnet server were installed at Windows 2000 Server. These two server programs, unlike Window's FTP and Telnet servers, allow us to configure the connection idle timeout and a few other parameters. The clients are standard Windows Telnet client, standard Windows FTP client, and FileZilla FTP client running on Windows XP Professional.

The TCP connection is started with either Telnet or FTP from the client to the server. At both routers, their PETS modules snoop the last acknowledgement packet for the connection. With a preconfigured application idle timeout, the link is disconnected between routers P1 and P2. Then, using the TGN and PKTS tools, the routers send ZWA packets. After the end hosts receive the ZWA, they basically do not send any more packets, but instead send the ZWP packets. Once the routers receive these packets, they send the ZWA packets back to the end hosts. These operations continue until the link resumes. When the link resumes and the updated window packets are sent back to the end hosts, the connection continues normal operation.

With Telnet experiments, we found that without PETS, the

connection terminates in about twenty seconds. On the other hand, if with PETS, the connection can be kept active until the application idle timeout timer expires (about 1 minute in our experiment). Of course, if the application timeout timer can be changed, the persistence for longer interval can be achieved.

With FTP experiments, by tuning the application idle timeout option called "*how long the client will disconnect regarding the received zero window packet*", which is applicable only for Filezilla FTP client, with PETS, the connection remains active until the application idle timeout timer expires (more than five minutes). Without PETS the TCP connection terminates in about two minutes.

With Windows FTP client, we cannot find a way to modify the application idle timeout timer. As a result, the connection is terminated at around one minute with or without PETS.

VI. CONCLUSIONS

PETS allows preserving the TCP connections during temporary link disconnections. It does not handle the change of IP address. Thus, it complements the Mobile IP very well, which handles IP address change but does not handle link disconnections. By combining the PETS and Mobile IP, we can support both disconnection operation and mobility.

PETS uses the standard TCP Freeze technique to keep the connections alive. Since TCP Freeze is already implemented in standard TCP stacks, the one change required is to implement "PETS modules" that act as proxies for the other side. These agents can be implemented either in the hosts (as part of the Mobile IP's mobile agents) or in the routers. Thus, our technique works with legacy hosts (and PETS-aware routers) or legacy routers (and PETS-aware hosts). This flexibility allows our scheme to be incrementally implemented.

PETS is a very simple technique. There is no concept of process checkpoint or migration [16] and no need to change TCP protocol or its implementation. In addition, PETS allows the TCP connection to be preserved indefinitely or until the application timeouts due to inactivity.

REFERENCES

[1] W-E. Eddy and J. Ishac, "Location Management in a Transport Layer Mobility Architecture," *NASA/TM-2005-213844.*, Aug. 2005.

[2] C-E. Perkins, "Mobile IP," *IEEE Commun. Mag.*, vol. 40, no. 5, pp. 66-82, May 2002.

[3] K. Thompson, G. Miller, and R. Wilder, "Wide Area Internet Traffic Patterns and Characteristics," *IEEE Network*s., vol. 11, no. 6, pp. 10-23, Dec. 1997.

[4] S. Dawson, F. Jahanian, and T. Mitton, "Experiments on Six Commercial TCP Implementations Using a Software Fault Injection Tool," *Software-Practice and Experience.*, vol. 27, no. 12, pp. 1385-1410, Oct. 1997.

[5] J. Postel, "Transmission control protocol," Request for Comments: 793, Internet Engineering Task Force (IETF), 1981.

[6] Y. Zhang and S. Dao, "A Persistent Connection Model for Mobile and Distributed Systems," in *Proc. Int. Conf. on Computer Communications and Networks.*, 1995, p. 300.

[7] R. Ekwall, P. Urban, and A. Schiper, "Robust TCP Connections for Fault Tolerant Computing," in *Proc. Int. Conf. on Parallel and Distributed Systems.*, 2003, pp. 501-508.

[8] D. Funato, K. Yasuda, and H. Tokuda, "TCP-R: TCP mobility support for continuous operation," in *Proc. IEEE Int. Conf. on Network Protocols.,* 1997, pp. 229-236.

[9] X. Qu, J-X. Yu, and R-P. Brent, "A mobile TCP socket," *Technical report TR-CS-97-08.*, Computer Sciences Laboratory, RSISE, The Australian National University, Canberra, ACT 0200, Australia, Apr. 1997.

[10] D-A. Maltz and P. Bhagwat, "MSOCKS: An Architecture for Transport Layer Mobility," in *Proc. IEEE Conf. on Computer Communications.*, 1998, vol. 3, pp. 1037-1045.

[11] V-C. Zandy and B-P. Miller, "Reliable network connections," in *Proc. Mobile Computing and Networking.*, 2002, pp.95-106.

[12] T. Goff, J. Moronski, and D-S. Phatak, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in *Proc. IEEE Conf. on Computer Communication.*, 2000, vol. 3, pp. 1537-1545.

[13] A-C. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proc. Mobile Computing and Networking.*, 2000, pp. 155-166.

[14] Cisco Systems, "TGN: traffic generator node and PKTS: Packet Count and Capture".

[15] C. So-In, "Session Persistence," Technical report (Cisco Systems), 2006 (unpublished).