Handoff and System Support for Indirect TCP/IP

Ajay Bakre and B.R. Badrinath
Department of Computer Science
Rutgers University, Piscataway, NJ 08855.
Email: {bakre, badri}@cs.rutgers.edu

Abstract

Over the past few years, Transmission Control Protocol (TCP) has become the most widely used transport layer protocol on the Internet. TCP performs poorly however, if one of the communicating hosts is a mobile wireless computer [6]. One way to address this performance problem is to modify TCP to make it aware of host mobility. Such an approach is infeasible because of the sheer number of hosts on the Internet using TCP. Another way is to build a separate transport layer protocol for mobile hosts but this raises interoperability problems with the existing stationary hosts.

An indirect transport layer protocol such as I-TCP [5] allows using a separate transport protocol for the wireless link between a mobile host and its Mobile Support Router (MSR) without causing interoperability problems with the fixed network. Experiments with I-TCP have shown significant improvement in throughput over regular TCP [5]. Use of indirect protocols requires additional functionality at the MSRs. In this paper we describe the implementation of such functionality for I-TCP. We also present measurements for the time needed for handoffs in case a mobile host with an active connection switches cells.

1 Motivation

Transmission Control Protocol (TCP) [21] has become the most popular transport layer protocol on the Internet. Evolution of TCP has kept pace with the advances in communication technology and current TCP implementations perform fairly well over wide ranging data links from low speed serial links to high speed LANs. Congestion control and flow control mechanisms of TCP are widely accepted as standards in the Internet community and have largely contributed to the growth of Internet on which millions of host end systems depend for communication.

Experiments with TCP have demonstrated poor performance however, if one of the communicating hosts happens to be a mobile wireless computer [6]. Host mobility causes temporary disruption in the network layer connectivity resulting in loss of TCP segments. Error prone wireless links used by mobile hosts to communicate with the fixed network also contribute to the increased loss of TCP segments sent to or from mobile hosts. This loss of TCP segments triggers congestion control at the transmitting host which seriously limits the end to end throughput. The congestion control mechanism used by TCP is clearly too conservative when faced with host mobility and wireless links.

One way to address the problem mentioned above is to modify the TCP specifications to take host mobility into account so that congestion control steps in only in case of a genuine network congestion. There are two problems with such an approach. First, it is infeasible to modify all the existing TCP implementations because of the sheer number of hosts on the Internet using TCP. Second, it is extremely difficult to determine the exact cause of a packet loss at the two end points of a TCP connection which may span multiple hops over the Internet. As an example, genuine congestion conditions may prevail on the fixed network when a mobile host switches cells. Attributing any packet loss in such a case purely to mobility can worsen the congestion on the fixed network.

Another alternative is to build a separate transport layer protocol for mobile hosts with specialized congestion and flow control mechanisms. Such a protocol may rely on information from the lower layers and from the base station or the mobile support router (MSR) to distinguish between packet loss due to congestion and that due to mobility or wireless errors. Using a separate protocol however, raises interoperability problems with the existing stationary hosts.

An *indirect* transport layer protocol such as I-TCP [5] allows the use of a separate protocol stack for mobile hosts without causing interoperability problems with the fixed network. I-TCP is built on top of Columbia Mobile IP [10] and provides TCP compatible access to the fixed network from the mobile hosts *indirectly* through the MSR. A *mobility and wireless aware* transport protocol is used for communication between the mobile host and its MSR while regular TCP

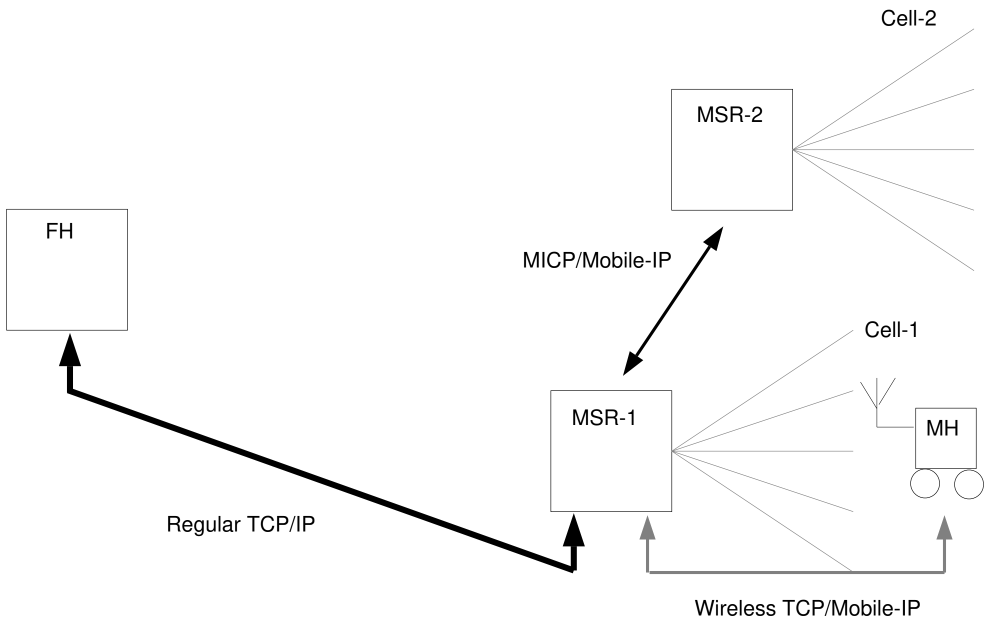

Figure 1. I-TCP connection between MH and FH via MSR-1

is used for communication between the MSR and the fixed hosts. No changes are thus needed to the TCP implementations on the fixed network. Experiments with I-TCP in the presence of host motion have shown improvements ranging from **1.5 to 4 times** in the end to end throughput compared to (regular) TCP [5].

Use of indirect protocols [4] requires specialized support for indirect connections and handoffs at the mobile hosts and the MSRs. In this paper we focus on the implementation of various mechanisms that we developed for the mobile hosts and the MSRs to support I-TCP connections and efficient handoffs in case a mobile host communicating over an open connection switches cells. Our scheme is not tied specifically with TCP and can be easily extended to support indirection for other transport layer protocols such as UDP [20] and RDP [27]. We present performance measurements for the time needed to achieve I-TCP handoffs when a mobile host moves from one wireless cell to another. We also analyze how the handoff time depends on the size of state information to be exchanged between two MSRs participating in the handoff.

The rest of this paper is organized as follows. The next section gives a brief overview of I-TCP. Section 3 describes the implementation of kernel and user level mechanisms needed to support I-TCP connections and handoffs. Section 4 deals with mobility management and the handoff procedure. Section 5 presents measurements on handoff performance. In section 6 we describe some applications that we modified to use I-TCP. Section 7 mentions some of the related research and section 8 contains concluding remarks.

2 Indirect TCP/IP Overview

We provide a brief overview of the I-TCP protocol in this section. More details on I-TCP can be found in [5]. As shown in figure 1, the central idea behind I-TCP is to break an MH to FH (end-to-end) TCP connection into two parts — one for the wireless link and the other for the wired network, the MSR serving the MH being the point where the break occurs. We can then easily modify the transport protocol on the wireless part (either TCP or some other lightweight protocol) without sacrificing interoperability with the TCP/IP hosts on the wired network. The following three features of I-TCP implemented over a suitable mobile IP scheme help to keep the indirection hidden from the fixed host (FH) communicating with the MH:

1. The MSR currently serving the MH (MSR-1 in the figure) *fakes* an image of the MH to the FH by using the IP address and port number of the MH for the wired part of the connection.
2. In case the MH switches cells, routing in mobile IP starts forwarding IP packets addressed to the MH to its new location i.e. the new MSR.
3. When an MH moves from one cell to another (e.g. from MSR-1 to MSR-2), the state of all its active I-TCP connections is handed over to the new MSR (MSR-2), which effectively restarts the connection

in the new cell *without* any participation from the communicating end points.

2.1 Establishing I-TCP connections

When a mobile host (MH) requests for an I-TCP connection to be established with a fixed host (FH), the MSR under which the MH is currently registered performs the following steps. It first establishes a regular TCP connection with the FH named in the connection request on behalf of the MH *using the IP address and port number of the MH for local endpoint parameters* - this constitutes the wired part of the I-TCP connection. Subsequently another connection is established between the MSR and the MH using a transport protocol that is tailored for mobile hosts and wireless links - this forms the wireless part of the I-TCP connection. In our implementation of I-TCP, we use TCP itself for the wireless part of the connection with one important modification *which resets the retransmission timer for the wireless part after a handoff causing an immediate slow start*. For the wireless side TCP connection, the MSR uses its own IP address and port number to identify the local endpoint. Thus an I-TCP connection between the MH and the FH consists of two separate TCP connections; and is uniquely identified by the following 3–tuple: {*<mh-address, mh-portnumber>, <fh-address, fh-portnumber>, <msr-address, msr-portnumber>*}.

2.2 Handing off I-TCP connections

When the MH switches cells (e.g. it moves from MSR-1 to MSR-2), the state associated with the two sockets at MSR-1 belonging to the I-TCP connection, is handed over to MSR-2. MSR-2 recreates the two sockets corresponding to the I-TCP connection with the *same* endpoint parameters that the sockets at MSR-1 had associated with them. Since the connection endpoints for both wireless and fixed parts of the I-TCP connection do not change after a move, there is no need to *reestablish* the connection at the new MSR. This also ensures that the indirection in the transport layer connection is completely hidden from the FH. The port numbers used by the MSR for the I-TCP sockets on behalf of an MH cannot conflict with the same port numbers used by the MSR for its own TCP connections or by other MHs using I-TCP connections in the same cell because the IP address is different for each host.

2.3 End-to-end semantics

One consequence of using I-TCP instead of TCP is that the acknowledgments are not end-to-end but instead we have separate acknowledgments for the wireless and the wired parts of the connection. Many applications that use TCP such as ftp however, also have some kind of support built-in for application layer acknowledgment and error recovery. Such acknowledgments are necessary because TCP acknowledgments only indicate that the data was received by the TCP layer at the receiving host and not necessarily by the receiving application. For such applications, use of I-TCP does not give rise to *weaker* end-to-end semantics in comparison to regular TCP, *assuming that there are no MSR failures*. MSR failure can cause loss of state information resulting in loss of indirect connections. These drawbacks may be a small price to pay however, considering the performance benefits of using indirect protocols [5].

In particular, I-TCP is ideal for throughput intensive applications such as ftp and Mosaic which provide built-in error recovery mechanisms to retry a failed data transfer. Such applications are likely to become popular as mobile clients begin to access information services from the fixed network. Interactive applications can also benefit from using I-TCP if the MH makes frequent moves, in which case a TCP connection can back off into long pauses [6] but an I-TCP connection would attempt a *slow start* immediately after every handoff.

3 I-TCP Implementation

In this section we describe the implementation of I-TCP components at the mobile hosts and at the MSRs that provide the additional functionality needed to support indirect connections and handoffs. We have implemented I-TCP on 386/486 PC-ATs running Mach 3.0 micro kernel and Unix server (MK84/UX40). All the mobile hosts and the MSRs in our laboratory have similar hardware configuration and they all use 2 Mbps NCR WaveLAN technology for wireless communication. All the MSRs are also attached to a wired 10 Mbps Ethernet. The MSRs run a version of UX server modified for Columbia's Mobile IP and I-TCP. The mobile hosts run the regular UX server derived from 4.3 BSD Tahoe release. User lever mhmicp and msrmicp processes modified for I-TCP run on the mobile hosts and the MSRs respectively and execute the Mobile Internetworking Control Protocol (MICP) [10] which takes care of beaconing and registration of mobile hosts in a wireless cell. In addition, a user level I-TCP daemon running at each MSR manages I-TCP connections originating from all the MHs within its cell. The I-TCP daemon also participates in handoffs with other MSRs when an MH switches cells.

3.1 I-TCP interface at the MH

To establish an I-TCP connection instead of a regular TCP connection from an MH to a fixed host, we provide special I-TCP library calls which are similar to the socket interface provided in Unix 4.3 BSD. These library calls must be used by the MH applications instead of the regular socket system calls (*connect*, *listen*, *accept* and *close*) for opening and closing an I-TCP connection. To send or receive data on such an I-TCP connection however, the MH can use the regular send and receive primitives. The I-TCP calls only provide a wrapper around the regular socket system calls to perform the necessary handshake with the MSR, the parameters of these library calls being the same as the corresponding socket system calls. A TCP-based application thus needs minimal changes to run on an MH with I-TCP. On the other hand, *no modification is needed*[1] *to the applications running on a fixed host to communicate with an MH using I-TCP*.

The I-TCP library calls also inform the local mhmicp process about active I-TCP connections. This information is used in the registration protocol executed by an MH entering a new cell to assist in I-TCP handoffs. The details of I-TCP calls are given below:

1. **itcp_init** — Initializes the I-TCP library data structures which maintain information on open I-TCP connections on a per process basis.
2. **itcp_listen** — Similar to the *listen* system call except that an indirect listening socket is also created at the current MSR by its I-TCP daemon on behalf of the MH with the same address and port number which identify the listening socket at the MH. Any connection attempt made after this call by a remote host will be intercepted by the indirect socket at the MSR which will cause the I-TCP daemon to connect to the MH listening socket thus completing the two parts of the I-TCP connection. The local mhmicp process is also informed about the creation of a listening socket at the MSR which records the corresponding port number information.
3. **itcp_accept** — Similar to the *accept* system call but the connection request is received from the current MSR in response to a connection attempt made by some remote host which was intercepted by the listening socket at the MSR. The wrapper around the *accept* system call provided by the I-TCP library however, makes the indirection transparent to the calling process by returning the peer address and port number of the remote host that initiated the connection (and not of the MSR). The local mhmicp process is also informed about the newly established I-TCP connection which records the endpoint parameters of the connection.
4. **itcp_connect** — Similar to the *connect* system call except that the connection request is sent to a special thread at the I-TCP daemon which in turn makes the connection attempt to the remote host address specified in the call. The call returns successfully only after the remote connection attempt has succeeded which means that on a successful return from *itcp_connect* both parts of the I-TCP connection are established. The local mhmicp process is informed about the new I-TCP connection.
5. **itcp_close** — Similar to the *close* system call for a socket. We need to close the connection at both the remote host and the MSR I-TCP daemon. We also need to inform the mhmicp process to delete the connection parameters from its tables.
6. **itcp_rioctl** — This I-TCP call is expected to provide some functionality of a socket *ioctl* call to control the indirect (remote) socket at the MSR. In addition, this call can be used by a higher layer to send control messages to the indirect socket which are interpreted by a corresponding higher layer at the MSR. One possible use of this might be to install a filter in the data stream in either or both directions e.g. put data compression and decompression filters at the MSR so that the data transmitted on the wireless link is always compressed while the data transmitted on the fixed link is not. This particular I-TCP function has not yet been fully implemented.

The I-TCP socket calls were implemented as a library rather than using a kernel based mechanism for the following reasons. First, we wanted minimal changes to be made in MH applications to switch from TCP to I-TCP and therefore the I-TCP calls export the same interface as the regular socket calls. Second, we wanted to allow the use of TCP and I-TCP connections in the same application depending on the kind of traffic expected on those connections. One example of such a *hybrid* application is described in section 6. Third, since we used a version of TCP itself as the transport protocol over the wireless link, we did not have to provide a separate socket family or a separate protocol number for I-TCP. Such a scheme would be appropriate however, in a mobile wireless environment where a different (possibly lightweight) transport protocol is used over the wireless link. Lastly, a library implementation is much easier to develop and debug than a kernel based mechanism.

3.2 MSR I-TCP daemon

The I-TCP daemon running on every MSR is responsible for managing I-TCP connections through that MSR for all the MHs that are currently local to the MSR. The daemon maintains two open sockets for each

[1] Other than a mechanism to recover from MSR failures, if necessary.

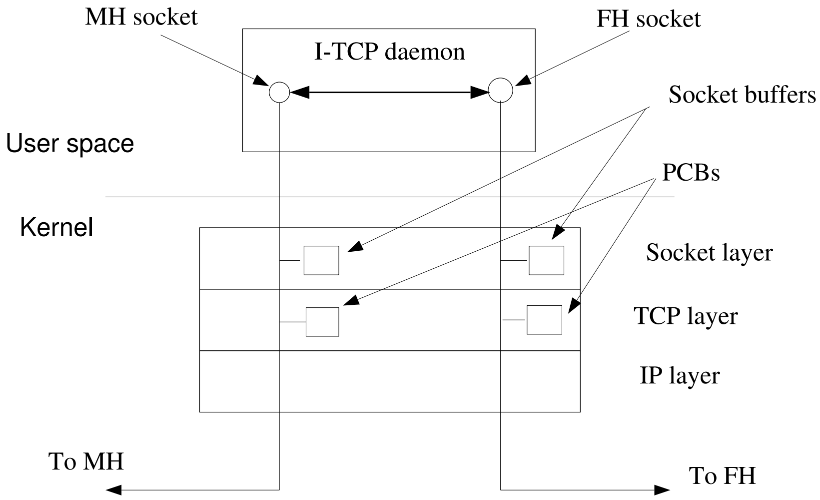

Figure 2. I-TCP connection state at the MSR

active I-TCP connection between a locally registered mobile host (MH) and a non-MSR fixed host (FH) as shown in figure 2. One of these sockets is used for communicating with the FH on the wired network and the other for communicating with the MH on the wireless link. The I-TCP daemon binds its FH side socket to the address and port number of the MH using an extended *bind* system call. Such a binding allows the MSR to fake an image of the MH to the FH which is unaware of the indirection. Binding to the MH address and port number also provides a mechanism at the IP layer in the MSR kernel to grab the TCP segments sent by the FH which are destined for the MH address and port number.

The I-TCP daemon is a threaded process [7] with different modules to communicate with the local MHs, the msrmicp process on the MSR and with the I-TCP daemons on other MSRs. These modules are described below:

1. **MH service module** — This module services I-TCP requests from the MHs local to the MSR. Such requests include those to open an I-TCP connection (either active or passive), to close a connection and remote *ioctl*s on the indirect sockets. Servicing an I-TCP request from the MHs involves handshake with the requesting process and notification of success or failure of the requested operation. This module also starts up appropriate worker threads to operate a fully established connection between an MH and a fixed host (FH) or a listening thread for a listening indirect socket established in the name of an MH.
2. **MICP module** — This module handles the communication with the msrmicp process on the same machine (MSR). The msrmicp process executes the MICP protocol, which is part of the Mobile-IP software, to communicate with the local MHs and with the msrmicp processes at other MSRs. The I-TCP daemon depends on the msrmicp process for the information on the MHs entering and leaving the MSR's cell. In particular, an MH's entry in the cell causes the MICP module to establish all the indirect sockets that were active for this MH at its previous MSR. The information about all such connections is maintained by the MH itself and is passed to the msrmicp process of the new MSR as part of the registration protocol. The exit of an MH from the MSR's cell causes the initiation of handoff by the handoff module as described below.
3. **Handoff module** — This module manages cell handoffs for MHs entering and leaving the MSR's cell in cooperation with the handoff modules of the I-TCP daemons at other MSRs. A designated thread in this module listens for handoff requests from other MSRs. Handoffs are always initiated by the I-TCP daemon of the old MSR after a forward pointer is received by its msrmicp process from the new MSR. The I-TCP daemon at the new MSR must have already established the skeletons for the indirect sockets on behalf of the migrating MH for the handoff to be successful. On receipt of a handoff request from the old MSR, a handoff thread receives the state of each indirect socket and populates the previously established socket skeleton. After installing the state of each I-TCP connection, the handoff thread starts up worker threads (or a listening thread if needed), for the connection. The details of a cell handoff are described in a later section.
4. **Work module** — This consists simply of the worker

threads that copy the TCP data from the MH side of the connection to the fixed side and vice versa. Every I-TCP connection has two worker threads assigned to it - one for each direction. The MH worker thread receives data from the MH and sends it over to the fixed host, the FH worker thread does the opposite. The MH worker thread is also responsible for tearing down the connection and freeing any resources after it is shutdown by either the MH or by the FH. Both the worker threads check a handoff flag which is set by the MICP module when it discovers that an MH that was local has moved to another cell. The state of the two sockets (one for the MH side and the other for the fixed side) of the I-TCP connection can be safely handed over to the new MSR after the worker threads exit on finding the handoff flag set. It is possible to install a filter in either direction of the fully duplex stream of the I-TCP connection which is applied to the data sent in that direction. Such a filter can be installed during the connection setup time or even after that using a remote *ioctl* from the MH process. Such a filter may even send the data up to a higher layer for further processing before it is sent out on the outgoing link.

Managing the I-TCP connections at the MSR from a process in user space involves additional copying overhead on each half of a duplex connection. At the MSR data received from the wireless side (MH) on an I-TCP connection has to go up through the TCP and socket layers in the kernel and to the user space to the I-TCP daemon; and down again on the fixed side of the connection through the socket and TCP layers of the kernel to the IP output routine. On the other hand, a TCP/IP packet received from the MH over a direct connection would be forwarded by the IP layer in the kernel to the fixed network with nominal processing overhead.

We chose to build a user level daemon to manage I-TCP connections for two reasons. First, it is easier to build an indirect higher layer protocol such as RPC with such a scheme. Second, a user level daemon is easier to modify and debug than a full kernel implementation. The data copying overhead can be reduced by providing a mechanism in Unix that allows connecting two sockets in a way that the data received on one of them is directly copied to the send buffers of the other by the kernel. The end points of the two parts of the connections could still be maintained by the MSR I-TCP daemon thus retaining the flexibility afforded by a user level implementation[2]. The additional data copying overhead is not so critical for throughput intensive applications which are the primary focus of our implementation. This overhead will cause increased latency as seen by MH applications, however.

3.3 MSR kernel support

In the following subsections, we describe the modifications needed in the kernel networking code for I-TCP connections and handoffs.

3.3.1 IP layer support At any MSR, we allow binding sockets to the addresses and port numbers of MHs that are currently local to the MSR. This is essential to grab, on a per connection basis, the TCP packets which originate from fixed hosts and are addressed to a local MH. It also provides a mechanism using which the MSR can act as a *proxy* for the local MHs. The *in_pcbbind* function in the kernel verifies from the *mobinfo* data structure maintained by mobile IP, that the address named in such a bind request belongs to a locally registered MH. It is possible to modify the TCP code on the wireless side so that the connection between the MH and the MSR is *reconfigured* for the IP address and the port number at the new MSR after a move. Currently however, we keep the connection endpoints on the wireless side fixed even after a move for the sake of simplicity. For this purpose, at any MSR we also allow binding sockets to addresses and port numbers of other MSRs. The validity of an MSR address can again be checked by consulting the *msrinfo* structure maintained by mobile IP.

A small change is needed to the IP input routine at the MSR which should send the IP packets that are addressed to MH address and I-TCP port numbers up to the TCP layer at the MSR instead of forwarding them to the MH using mobile IP routing. A list of such I-TCP port numbers (which correspond to active connections) is maintained on a per MH basis by the MSR along with the *mhinfo* entry of the MH used for routing in mobile IP. The search cost to decide whether a TCP segment addressed to a local MH should be forwarded by the IP layer at the MSR or sent up to the TCP layer is thus proportional only to the number of I-TCP ports registered on behalf of the MH. We did not allocate a range of port numbers for I-TCP as it would have reduced the available port numbers for TCP and would have made *wild card* port allocation and matching more difficult.

3.3.2 Primitives for I-TCP handoff If a mobile host with an open I-TCP connection switches cells, handoff in mobile IP causes rerouting of IP packets addressed

[2] This is somewhat similar to DVMRP multicast routing with mrouted where the user level daemon dictates the multicast forwarding policy while the actual forwarding is done by the kernel.

Ioctl	Parameters	Description
SIOCCREATE	target socket, connection 4-tuple	Establish the connection endpoints of the target socket as if *bind* and *connect* have been called on the socket but without any communication with the peer host.
SIOCDELETE	target socket	Delete the target socket without any communication with the peer host.
SIOCGETSTATE	target socket, *struct socreq*	Capture the state of the socket listed in structure *socreq* and place it in the send buffer of the target socket.
SIOCSETSTATE	target socket, *struct socreq*	Retrieve and establish the state of the socket listed in the structure *socreq* from the receive buffer of the target socket.

Table 1. New socket *ioctl*s

to the MH to the new MSR. To maintain the I-TCP connection therefore, the I-TCP daemon at the new MSR must reconstruct the two sockets for the wired and the wireless parts of the connection from the state information received from the I-TCP daemon at the previous MSR. This movement of socket should be completely transparent to the fixed host at the other end of the wired side connection. The socket state handoff needs to be fast enough so as not to become a performance bottleneck even with frequent moves. In addition, the data segments which are in transit at the time of handoff must be buffered at the new MSR because they cannot be processed until the full state information is available. This is necessary to avoid congestion on both sides (wired and wireless) of the I-TCP connection. Complete details of handoff are described in a later section. Here we restrict ourselves to describing the primitives needed to achieve a socket handoff.

We provided special *ioctl* calls in the MSR kernels to perform efficient socket handoffs. Two of these *ioctl*s, namely SIOCGETSTATE and SIOCSETSTATE are used by the I-TCP daemons to reliably transfer the state information corresponding to a connected socket from the kernel of one MSR to that of another with minimum copying overhead. We also provided two other *ioctl*s called SIOCCREATE and SIOCDELETE to create a *connected* skeleton socket which is waiting for its state to be installed and to silently delete a socket whose state has been transferred to another MSR. Table 1 summarizes the four *ioctl*s. The *ioctl* mechanism was preferred over system calls primarily because it is much easier to add an *ioctl* than it is to add a system call to Unix. SIOCCREATE and SIOCDELETE operate on the I-TCP sockets whereas SIOCGETSTATE and SIOCSETSTATE operate on a handoff socket which is used for MSR-to-MSR state transfer. The latter two take *struct socreq* as an argument which lists the I-TCP socket descriptor whose state is to be sent or received. For the following discussion we will assume that the state of an I-TCP socket is being handed over from MSR-1 to MSR-2.

1. SIOCCREATE — This *ioctl* is used to establish the connection 4–tuple for a newly created I-TCP socket at MSR-2. The connection 4–tuple consists of local and foreign addresses and port numbers for the TCP connection. After this *ioctl*, the socket is marked as *moving* which means that it can accept incoming TCP segments and buffer them but these segments cannot be processed until the complete socket state becomes available at MSR-2.
2. SIOCDELETE — This *ioctl* is used to delete an I-TCP socket at MSR-1 after its state has been successfully handed over to MSR-2. It differs from the *close* system call in that it only deletes the socket data structures and does not attempt to close the connection with the remote peer.
3. SIOCGETSTATE — MSR-1 makes this *ioctl* call on a handoff socket with an established connection with MSR-2 to capture and send the state of an I-TCP socket after making sure that MSR-2 is prepared to accept the state of the I-TCP socket. The socket descriptor of the I-TCP socket is passed in the *struct socreq* argument to the *ioctl*. The TCP portion of this *ioctl* packs the TCP control block and any pending segments in the reassembly queue for the I-TCP socket. After this call, the I-TCP socket is marked as *frozen* which means it cannot send or receive any more data and it stays in this state until it is deleted by a SIOCDELETE call.
4. SIOCSETSTATE — MSR-2 makes this *ioctl* call on a handoff socket to receive and install the state of an I-TCP socket which is sent by MSR-1 with a corresponding SIOCGETSTATE call. The I-TCP socket must have been prepared by SIOCCREATE and marked *moving* prior to making this call, and

its descriptor is passed in the *struct socreq* argument to the *ioctl*. As part of this *ioctl*, the TCP control block of the I-TCP socket is re-initialized from the transferred state and the reassembly queue is also initialized. Also, if there are any pending TCP segments buffered for this connection which were received since the SIOCCREATE call, these segments are processed by calling the TCP input routine. The *moving* flag is cleared and the socket is ready for communication in both directions.

The SIOCGETSTATE *ioctl* avoids unnecessary copying of socket buffers and other state between kernel and user spaces by packing the send/receive buffers and state of the I-TCP socket inside the kernel and directly queuing it in the send buffer of the handoff socket. Similarly, SIOCSETSTATE avoids copying by retrieving the state and send/receive buffers of the I-TCP socket directly from the receive buffer of the handoff socket.

In addition to the above mentioned *ioctl*s, some minor changes were needed to the timer handling routines in TCP to inhibit retransmit timer handling on a socket which is marked as *frozen* or *moving*. Also, the TCP input routine needs to save segments received on a socket marked as *moving* in the reassembly queue for deferred processing. Any segments received on a *frozen* socket are silently dropped.[3]

4 Mobility Management — Interaction with Mobile IP

In Columbia mobile IP protocol a user level msrmicp daemon at the MSR is responsible for sending periodic beacons and for registration and expiration of mobile hosts. Similarly a user level mhmicp daemon running at each mobile host is responsible for listening to MSR beacons, sending greeting messages and informing the MSR about its previous location so that the MSR can send a forwarding pointer to the old MSR which was earlier routing IP packets for the MH. We integrated I-TCP handoff with the handoff in mobile IP since both need the information about MHs entering a cell and their earlier locations.

4.1 MICP i-entries

We extended the MICP protocol to carry information about I-TCP connections in the initial greeting that an MH entering a cell sends to the MSR managing the cell. At every MH, the mhmicp daemon maintains the endpoint information for each active I-TCP connection

[3] In normal circumstances, no TCP segments should be received on a socket which is *frozen* because routing in mobile IP will carry all such packets at the new MSR (MSR-2).

Figure 3. Handoff algorithms for MSRs

```
New MSR:

wait for handoff request from old MSR
receive the MH address
lookup MH entry in the tables
for every I-TCP connection do {
  receive i-entry for connection
  lookup the connection entry
  receive and install state of
      MH side socket
  restart the wireless part of
      connection
  receive and install state of
      FH side socket
  restart the wired part of
      connection
  receive pending user level
      data buffers
  install MH and FH side user
      level buffers
  fork worker threads connecting the
      two parts of I-TCP connection
}

Old MSR:

locate the entry for MH that moved out
for every I-TCP connection do {
  signal worker threads to terminate
}
make a handoff connection to the new MSR
send MH address
for every I-TCP connection do {
  send i-entry for the connection
  freeze and send state of
      MH side socket
  freeze and send state of
      FH side socket
  send MH and FH side user level
      data buffers
  delete MH and FH side sockets
  free resources and delete MH entry
}
```

at the MH. Such information is maintained in MICP *i-entries* and it consists of the unique 3–tuple identifying the connection and some flags. The *i-entries* are updated from the messages sent by the I-TCP library linked with an MH application, when the application makes an I-TCP call. Unix domain sockets are used for communication between I-TCP library and mhmicp daemon. When an MH enters a new cell, i.e. when mhmicp daemon hears a beacon in the new cell after losing contact with the previous MSR, a greeting message (called MICP_GREET packet in mobile IP) is sent to the

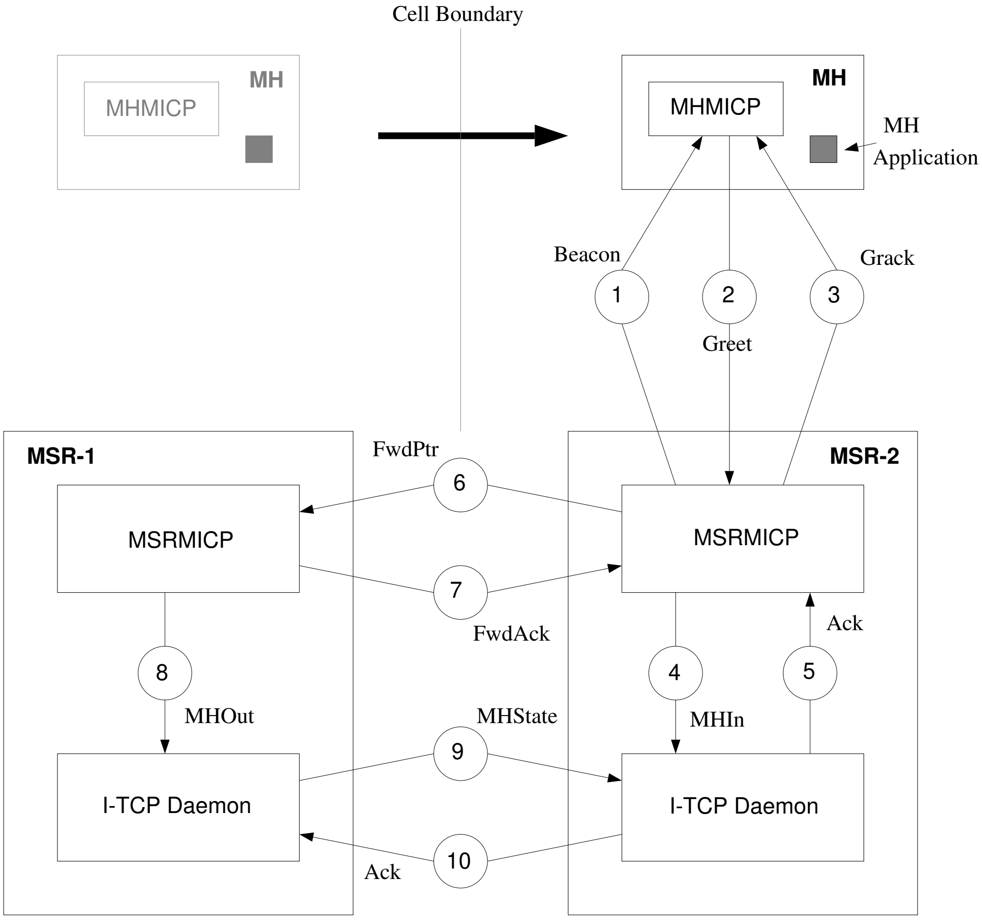

Figure 4. I-TCP / mobile-IP handoff sequence

new MSR. The mhmicp daemon sends its list of MICP *i-entries* in the initial greeting message in addition to the list of previously visited MSRs that may still believe that the MH is local to them.

4.2 Handoff sequence at MSRs

On the MSR side, when the msrmicp daemon receives an initial greeting from a new MH entering its cell which contains MICP *i-entries*, it sends a copy of the greeting message to the local I-TCP daemon. The I-TCP daemon establishes skeleton sockets for each I-TCP connection from the *i-entries* using SIOCCREATE calls. It also sends an ACK to the msrmicp daemon after which a forwarding pointer (MICP_FWDPTR) is sent to the previous MSR named in the greeting message.

When the msrmicp daemon at an MSR, where the MH had established I-TCP connections earlier, receives a forwarding pointer from another MSR, it updates its data structures to reflect the new location of the MH. It also sends a copy of the forwarding pointer to its local I-TCP daemon which then transfers the state of each I-TCP connection to the I-TCP daemon at the new MSR using a reliable protocol (TCP) and SIOCGETSTATE calls. The handoff algorithms executed by the new and the old MSRs are listed in figure 3. The I-TCP handoff sequence is shown in figure 4 and figure 5 shows the intermediate stages of an I-TCP connection during handoff.

5 Handoff Performance

We present measurements of handoff time for an I-TCP connection between a mobile host (MH) and a fixed host (FH) using different socket buffer sizes. We also analyze how the handoff time depends on the size of the state information to be transferred from one MSR to another. Fast handoffs are essential for the throughput improvement that I-TCP affords over regular TCP. Comparative throughput figures for I-TCP and TCP under a variety of wireless cell configurations can

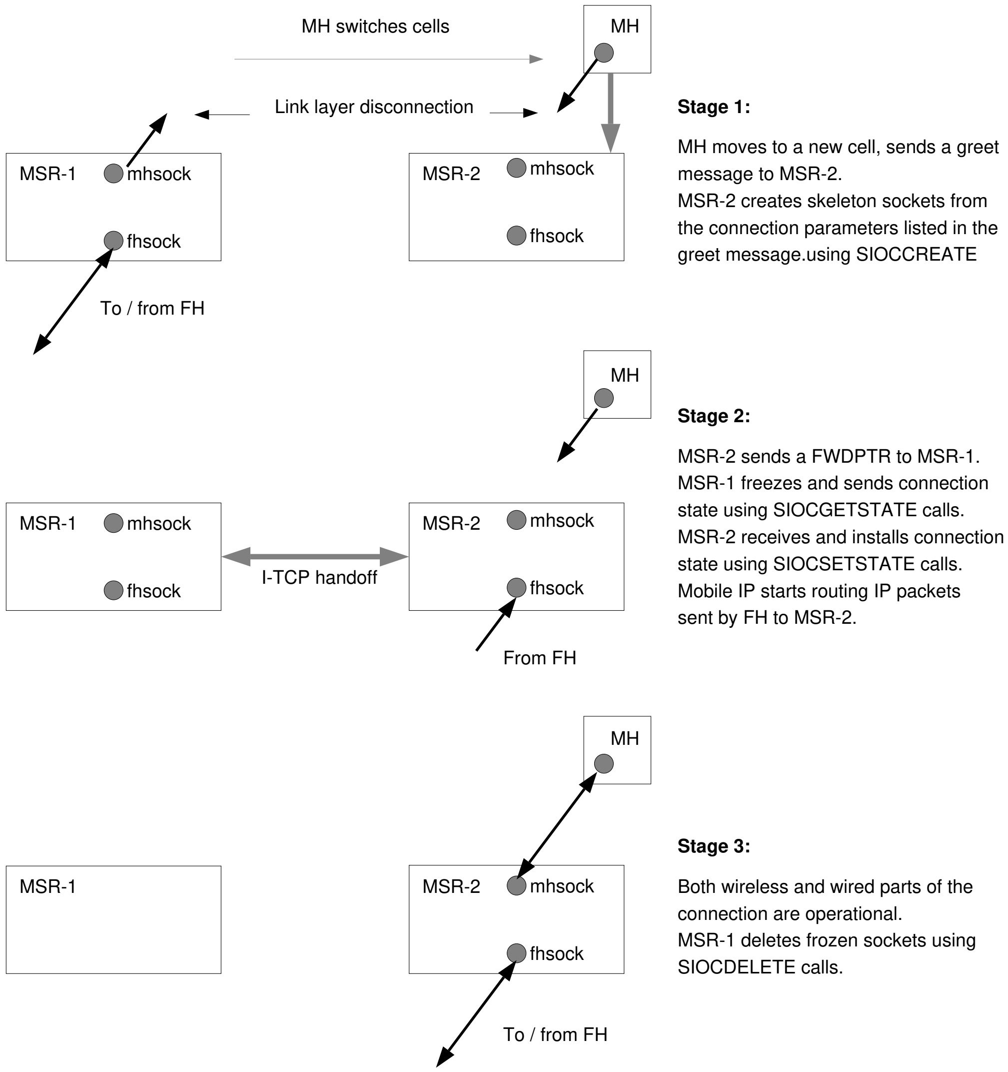

Figure 5. Intermediate stages in I-TCP connection handoff

be found in [5][4]. The focus of this paper is on the handoff and system support we developed for I-TCP and therefore we will restrict ourselves to analyzing handoff performance.

We used the implementation platform described in section 3 for our handoff measurements. We measured the handoff time for an MH that had one I-TCP connection established with a fixed host 2 Ethernet hops away from the MSRs. The MH kept switching back and forth between two wireless cells managed by two MSRs at a constant rate (once every 10 seconds). The two wireless cells used in our experiments in fact overlapped, but we simulated non-overlapped cells using different MAC layer network IDs for the WaveLAN radio in the two cells. Cell switching was accomplished by a user process periodically changing the MAC layer network ID on the WaveLAN card at the MH which causes a loss of link layer connectivity with the current MSR and the MH starts listening to the new MSR. This user process also sends a signal to the mhmicp process immediately after changing the network ID, which forces mhmicp to look for a beacon from the new MSR. The beacon period used by the MSRs was 1 second which was the disconnection interval for the MH in the worst case. The handoff time was measured at the new MSR as the difference between the time when a greeting message arrives from the MH entering the cell and the time when I-TCP handoff completes.

Figure 6 shows the time taken by I-TCP connection handoff using different socket buffer sizes in a non-overlapped cell configuration where cell boundaries are

[4] Also available as Rutgers DCS technical report DCS-TR-314 (ftp://paul.rutgers.edu/pub/badri/itcp-tr314.ps.Z).

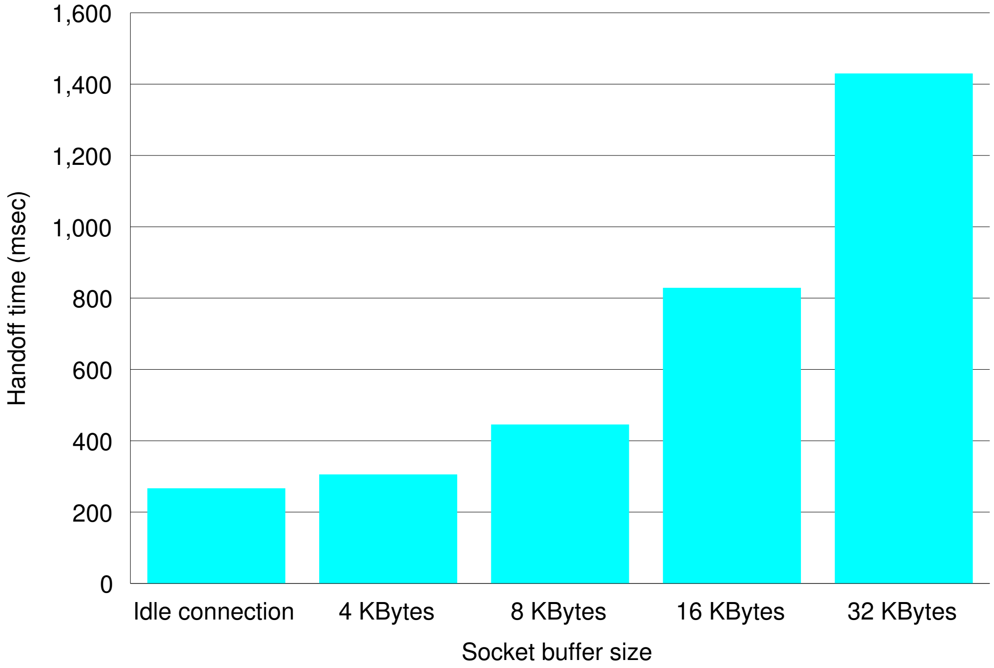

Figure 6. Handoff times for different socket buffer sizes

sharply defined and cell switching is instantaneous. The case marked *idle connection* denotes the handoff time for a connection on which no data is being sent. The graph shows that it takes about 265 milliseconds just to transfer the connection state with no (empty) socket buffers. The size of such an idle connection state is about 500 bytes which includes the socket data structure and the TCP control block for the two (MH side and FH side) sockets plus some control information. In all the other cases, the handoff times were measured with the FH pumping data as fast as possible, given that the maximum window size for both parts of the I-TCP connection was equal to the socket buffer size. The handoff time increases with the size of socket buffers. This is because of the large difference in the bandwidths of the wireless and the wired media (2 Mbps WaveLAN versus 10 Mbps Ethernet) which causes the socket buffers at the MSR *to be full most of the time*. Thus when the MH switches cells, I-TCP handoff involves transferring a full send buffer for the MH side socket and a full receive buffer for the FH side socket from one MSR to another. The I-TCP handoff in the case with 32 Kbyte socket buffers can add up to 64 Kbytes (and possibly some more due to any pending user level buffers) to the basic connection state of about 500 bytes.

We repeated the above experiments with two other cell configurations i.e. - *i)* overlapped cells and *ii)* non-overlapped cells which are not adjacent. The second configuration was simulated by the MH staying disconnected for 2 seconds before it switched to a new cell. We did not find any significant difference in the handoff time in both these configurations compared to our test configuration of adjacent non-overlapped cells mentioned above.

We further analyzed the I-TCP handoffs to determine how much time is spent in each step. The results of our analysis are summarized in figure 7 for two representative cases - *a)* idle connection and *b)* active connection with 32 Kbyte socket buffers. All timing data shown is with reference to the instant when a greeting is received by the MSR from an MH entering its cell. In the case with idle connection, the new MSR took about 60 msecs to establish the socket skeletons from the information contained in the MH greeting message after which it sends a forward pointer (MICP_FWDPTR) to the old MSR. This included the time for two context switches — from the msrmicp process which receives the MH greeting to I-TCP daemon which creates the skeleton sockets and back to msrmicp which then sends the forwarding pointer. It took additional 150 msecs before the handoff request arrives at the new MSR. The actual state transfer took only 55 msecs which included two control messages and the corresponding ACKs (one to identify the MH, and another to identify the connection) in addition to two SIOCSETSTATE calls by the new MSR.

In the second case, with 32 Kbyte socket buffers, we see that after the handoff request is received by the new MSR, another 1230 msecs elapsed before the handoff was completed. Out of these, the state of MH

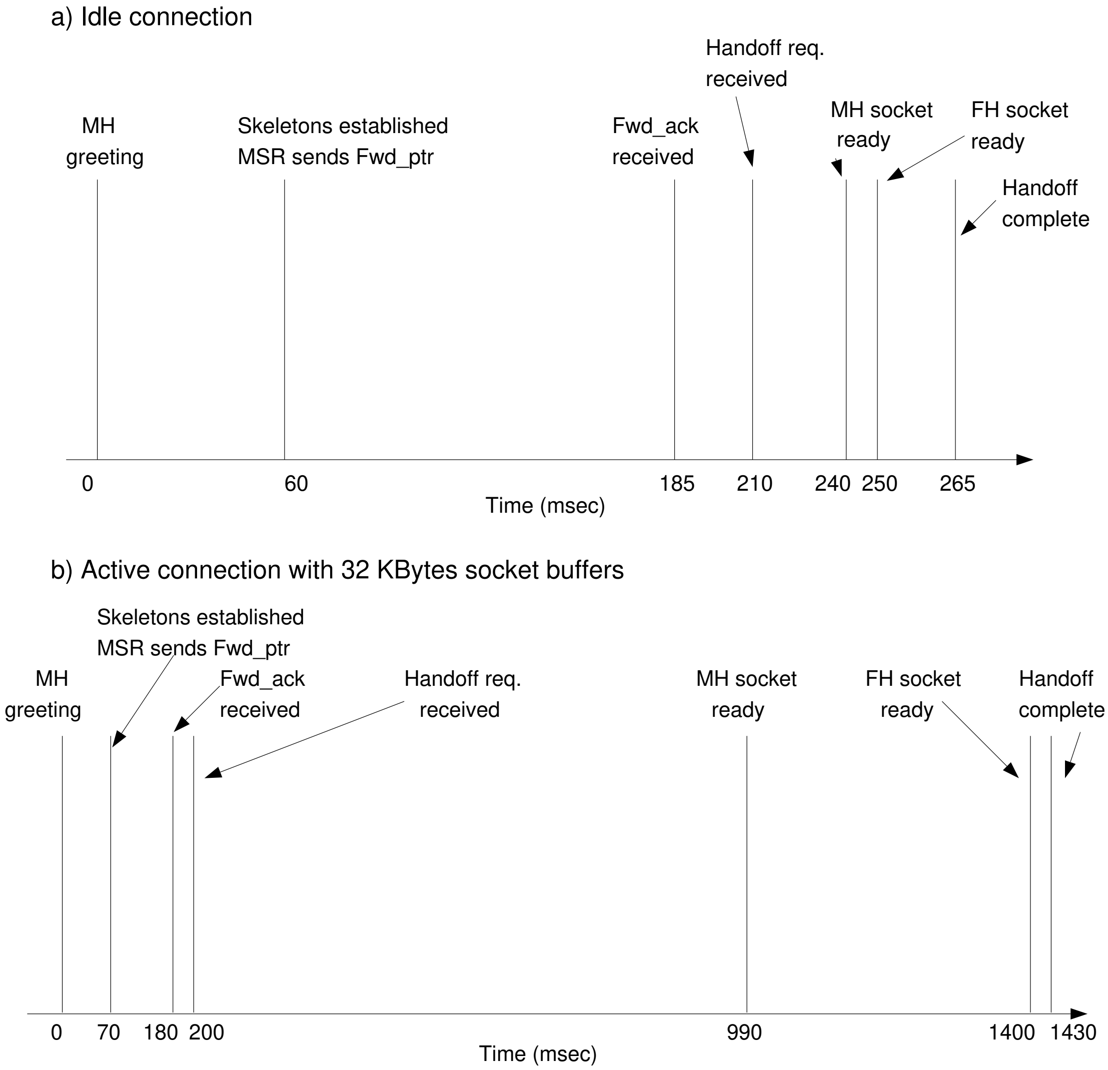

Figure 7. Analysis of I-TCP handoff events

side socket was transferred in 790 msecs whereas the state of FH side socket took 410 msecs. Thus with large socket buffer sizes, a major part of handoff time is spent in transferring the buffered data. The time to transfer MH side socket is larger than the corresponding time for the FH side socket mainly because the TCP handoff connection between the two MSRs is still in a *slow start* phase while transferring the MH side socket state.

From the above figure, state transfer of each idle I-TCP connection needs approximately 50 msecs, after a handoff connection is established between the MSRs; whereas transferring the state of a connection that was active takes 1 to 1.5 seconds for large socket buffer sizes. It should be possible to reduce the handoff time with pre-established handoff connections between MSRs and faster context switches between the MSR daemon processes.

6 Example Applications

We have used I-TCP to improve the performance of TCP based applications running on mobile wireless machines in our indoor wireless LAN environment. Some of these applications are listed below along with a description of the changes needed for I-TCP:

1. **ttcp benchmark —** We used the ttcp benchmark to perform the handoff experiments reported in the previous section. The regular socket calls for *connect, listen* and *accept* were replaced by their I-TCP equivalents. Throughput experiments conducted using the ttcp benchmark comparing the performance of I-TCP with that of regular TCP have been reported in [5].
2. **ftp —** ftp uses two kinds of TCP connections - one for control and another for data. The control

connection is used to exchange application layer control messages between the ftp client and server while a separate data connection is opened for every file transfer. We developed a *hybrid* ftp which we call i-ftp to use I-TCP for data connections but regular TCP for control connections. Using I-TCP for the data connections affords us with throughput advantages over regular TCP while the end-to-end nature of the control connection which uses regular TCP gives us adequate error detection for MSR failures which may cause loss of I-TCP (data) connections.

3. **Chimera WWW browser —** Chimera[5] uses HTTP to access WWW services over the Internet which is based on TCP. We modified the Chimera sources to use I-TCP instead of TCP which considerably improved the time needed to download large files at a mobile host.

7 Related Work

Performance improvements in the end-to-end transport layer throughput with the indirect model have also been reported elsewhere using a separate transport protocol for the wireless link [29]. Termination of the wireless protocol stack at an intermediate base station has also been explored in [15] for a digital cellular telephone network. A scheme for installing filters at the MSRs under MH control is described in [30]. In addition, the indirect approach where the intermediary (or a set of intermediaries) exists at higher protocol layers has been investigated by [2, 3].

Link layer retransmissions can be used on error-prone wireless links to bring their error rate on par with that on the wired networks but such an approach interferes with the end-to-end retransmissions of TCP and thus may not result in improved performance [8]. *Fast retransmissions* coupled with modifications to the TCP software on the mobile hosts [6] solve only part of the problem since most TCP implementations on the fixed network do not support fast retransmissions. I-TCP is based on an *indirect protocol model* proposed in [4] which takes a more general approach towards host mobility and wireless links.

8 Conclusion

In this paper we have described the implementation of system support for handing off active transport layer connection state in mobile environments. Such support is essential for indirect protocols such as I-TCP which show improved performance over conventional end-to-end protocols for wireless mobile hosts. The system support consists of daemon processes at the MSRs managing indirect connections and handoffs and also the coordination with mobile IP handoff between the old and the new MSRs when a mobile host switches cells.

Our measurements with I-TCP handoffs show that operating system support in the form of suitable handoff mechanisms can help in achieving efficient transport layer handoffs. We also analyzed the handoff data to determine the time consuming activities in I-TCP handoffs. A kernel resident implementation of I-TCP, though less flexible than a user level implementation such as ours, should further cut down on the copying overhead incurred by I-TCP connections.

We also described TCP based applications that we ported to I-TCP resulting in improved performance. We are planning to build indirect presentation layer protocols over I-TCP. Partial support for building such protocols already exists in our I-TCP implementation in the form of remote *ioctl*. Using the remote *ioctl* on the indirect socket at the MSR, makes it possible for a mobile host to install a filter in either direction of the fully duplex stream of the I-TCP connection. Such a filter can also be used to send the data received from a fixed host to a higher layer at the MSR for further processing before it is sent to the mobile host over the wireless link.

Acknowledgments

We thank John Ioannidis for providing us with the source code of Columbia's Mobile-IP implementation. The WaveLAN driver for Mach used in our experiments was originally written by Anders Klemets and adapted for our network configuration by Girish Welling. We also thank the anonymous reviewers of the extended abstract for their valuable comments.

Availability

We plan to make the sources of our I-TCP implementation publicly available sometime during summer 1995.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young. Mach: a new kernel foundation for UNIX development. In *Proc. of the USENIX 1986 Summer Conference*, July 1986.

[2] A. Athan and D. Duchamp. Agent-mediated message passing for constrained environments. In

[5] Chimera was developed by John Kilburg.

USENIX Symposium on Mobile and Location-Independent Computing, August 1993.

[3] B.R. Badrinath, A. Acharya, and T. Imielinski. Structuring distributed algorithms for mobile hosts. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, pages 21–28, June 1994.

[4] B.R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *4th Workshop on Workstation Operating Systems*, October 1993.

[5] A. Bakre and B.R. Badrinath. I-TCP: Indirect TCP for mobile hosts. To appear in the *Proc. of the 15th Intl. Conf. on Distributed Computing Systems*, May 1995.

[6] R. Caceres and L. Iftode. The effects of mobility on reliable transport protocols. In *Proc. of the 14th Intl. Conf. on Distributed Computing Systems*, pages 12–20, June 1994.

[7] E.C. Cooper and R.P. Draves. C Threads. Technical Report CMU-CS-88–154, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1988.

[8] A. DeSimone, M.C. Chuah, and O.C. Yue. Throughput performance of transport-layer protocols over wireless LANs. In *Proc. of Globecom '93*, December 1993.

[9] D.J. Farber, G.S. Delp, and T.M. Conte. A thinwire protocol for connecting personal computers to the Internet. *RFC 914*, September 1984.

[10] J. Ioannidis, D. Duchamp, and G.Q. Maguire. IP-based protocols for mobile internetworking. In *Proc. of ACM SIGCOMM*, pages 235–245, September 1991.

[11] J. Ioannidis and G.Q. Maguire. The design and implementation of a mobile internetworking architecture. In *Proc. of USENIX Winter Technical Conference*, January 1993.

[12] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM*, pages 314–329, August 1988.

[13] V. Jacobson. Compressing TCP/IP headers for low-speed serial links. *RFC 1144*, February 1990.

[14] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. *RFC 1323*, May 1992.

[15] M. Kojo, K. Raatikainen, and T. Alanko. Connecting mobile workstations to the Internet over a digital cellular telephone network. In *Mobidata Workshop on Mobile and Wireless Information Systems*, November 1994.

[16] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman. *The design and implementation of the 4.3BSD UNIX Operating System*. Addison Wesley, 1989.

[17] P. Manzoni, D. Ghosal, and G. Serazzi. Impact of mobility on TCP/IP: an integrated performance study. To appear in *the IEEE Journal on Selected Areas in Communications*, 1995.

[18] A. Myles and D. Skellern. Comparison of mobile host protocols for IP. *Journal of Internetworking Research and Experience*, 4(4), December 1993.

[19] J. Nagle. Congestion control in IP/TCP Internetworks. *RFC 896*, January 1984.

[20] J. Postel. User datagram protocol. *RFC 768*, August 1980.

[21] J. Postel. Transmission control protocol. *RFC 793*, September 1981.

[22] Y. Rekhter and C. Perkins. Optimal routing for mobile hosts using IP's loose source route option. *Internet Draft*, October 1992.

[23] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), November 1984.

[24] W.T. Strayer, M.J. Lewis, and R.E. Cline Jr. XTP as a transport protocol for distributed parallel processing. In *Proc. of the USENIX Symposium on High-Speed Networking, Oakland, CA*, August 1994.

[25] W.T. Strayer, M.J. Lewis, and Jr. R.E. Cline. An object-oriented implementation of the Xpress Transfer Protocol. In *Proc. of the Second Intl. Workshop on Advanced Communications and Applications for High-Speed Networks (IWACA), Heidelberg, Germany*, September 1994.

[26] C. A. Thekkath, T.D. Nguyen, E. Moy, and E.D. Lazowska. Implementing network protocols at user level. In *Proc. of ACM SIGCOMM*, pages 64–73, September 1993.

[27] D. Velten, R. Hinden, and J. Sax. Reliable data protocol. *RFC 908*, July 1984.

[28] H. Wada, T. Yozawa, T. Ohnishi, and Y. Tanaka. Mobile computing environment based on internet packet forwarding. In *Proc. of the USENIX Winter Technical Conference*, January 1993.

[29] R. Yavatkar and N. Bhagwat. Improving end-to-end performance of TCP over mobile internetworks. In *IEEE Workshop on Mobile Computing*, December 1994.

[30] B. Zenel and D. Duchamp. Intelligent communication filtering for limited bandwidth environments. *To appear in the Fifth Workshop on Hot Topics in Operating Systems (HoTOS-V)*, May 1995.