M-UDP: UDP for Mobile Cellular Networks*

Kevin Brown and Suresh Singh
Department of Computer Science
University of South Carolina
Columbia, SC 29205
{*kbrown,singh*}*@cs.sc.edu*
Tel: 803-777-2596

September 4, 1996

Abstract

In this paper we present our implementation of a modified UDP protocol appropriate for the mobile networking environment. Our protocol, like UDP, does not guarantee reliable delivery of datagrams. However, unlike UDP, it does ensure that the number of lost datagrams is kept small. In this paper we discuss our implementation of M-UDP (in NetBSD) and compare its performance against that of UDP in an experimental mobile network that is currently under development at the University of South Carolina.

1 Introduction

Mobile networks refer to an emerging new networking technology that will allow users to maintain network connections even as they roam over large geographical areas. Users equipped with *personal digital assistants* (palm-top computers with wireless communications technology) will thus have continuous access to a wide variety of services that will be made available over national and international communication networks. They will be able to access their data and other services such as electronic mail, electronic news including special services such as stock market news, videotelephony, yellow pages, map services, electronic banking, etc., while on the move.

How can we maintain connections for users who are mobile? Clearly, we need to ensure that packets get *routed* to the current location of the user and we also need to ensure that the *throughput* of the connection is not adversely affected due to mobility. Mobile-IP [2] is an adequate protocol for routing packets in a mobile environment (section 2.1 discusses, briefly, how Mobile-IP fits into our system). However, Mobile-IP by itself cannot guarantee that the efficiency of transport connections is maintained. To see why this is the case, consider the behavior of a TCP connection where the receiver is mobile. In TCP, the sender begins retransmission of packets if they are not acknowledged within a short amount of time (hundreds of milliseconds). In a mobile environment, a user is subject to periods of fading (when the user is blocked

*This work was supported by the NSF under grant number NCR-9410357.

by some physical obstruction, for example), or high bit-error rate (resulting in lost datagrams). The fade periods can be as long as a few seconds causing the transmission of acknowledgements to be delayed. The TCP sender will timeout and retransmit the unacknowledged packets resulting in reduced efficiency for the TCP connection. A similar problem arises for UDP connections as well since datagrams transmitted when the user is in a fade will be lost.

How can we improve the efficiency of these transport connections? To improve TCP throughput, we need to ensure that the sender does not timeout and begin retransmission when the mobile user is in a fade. However, increasing the timeout interval is not an acceptable solution because, in certain situations, it will result in poor throughput. Consider the case where the mobile user is stationary but the wireless link is noisy (lots of lost packets). In this case, since the timeout interval is large, the sender will not retransmit lost packets quickly enough resulting in poor throughput because the channel will be underutilized. Bakre[1] has proposed a different implementation of TCP called I-TCP (for Indirect-TCP) that addresses this efficiency issue. I-TCP splits the TCP connection in two – one TCP connection from the sender to a fixed host 'close' to the current location of the mobile user and a second TCP connection from this fixed host to the mobile user. This fixed host effectively serves as the TCP connection end-point from the point of view of the sender and is responsible for forwarding all data reliably to the mobile user.

If UDP is used unmodified over a wireless channel a large percentage of packets will be lost because of two reasons. First, wireless links tend to be susceptible to bit errors and, second, packets transmitted to a mobile that is in a fade will be lost. The first problem may be alleviated to some degree by using some form of FEC (Forward Error Correction) encoding. The second problem, however, cannot be handled as easily at the data-link layer. In this paper we discuss the implementation and performance of our M-UDP (Mobile-UDP) protocol which follows traditional UDP semantics but provides improved efficiency (i.e., fewer lost packets). The protocol is based on an idea similar to the one used in I-TCP and M-TCP [3]. That is, the UDP connection is split in two at some host close to the mobile user. This host attempts to use any free bandwidth to retransmit packets lost during a fade thus ensuring that the number of lost packets is kept small. We study the performance of our protocol in terms of lost packets and buffer overhead and compare it against UDP. We show that unmodified UDP has a loss rate of up to 50% while M-UDP has a loss rate of less than 5% for a wide range of loads and fade intervals.

In the next section we describe our mobile network architecture briefly and describe the advantages of this architecture over others. We also summarize the overall design of our protocol stack for mobile networks. Section 3 presents three versions of M-UDP and section 4 discusses the implementation. Results of a performance study comparing UDP and M-UDP are presented in section 5. Section 6 concludes the paper.

2 Overview of our system architecture

The system model we consider is one where users equipped with PDAs (or similar wireless capable computing devices) roam in a region covered by a *cellular* network. Each cell contains a base station that provides wireless connectivity to the mobile. Note that we *do not* consider a model where a user unplugs a computer from one subnet, travels elsewhere and plugs it into a foreign subnet. Mobile-IP with no modifications at the transport level will be sufficient to handle this latter type of mobility.

Our mobile network architecture is presented in detail in [3]. We summarize the main points in this section. Our architecture may be viewed as a three-level hierarchy (see Figure 1) at the lowest level of which are the MHs (Mobile Hosts) who communicate with MSS (Mobile Support Stations) nodes in each cell. MSSs are responsible for providing a connection endpoint for MHs. Several MSSs are controlled by a machine called the Supervisor Host (SH) which is connected to the wired network and is responsible for handling most of the routing and other protocol details for the mobile users. We also assume that the MSSs are connected to the SH via a wired network. The SH maintains connections for mobile users, handles flow-control and is responsible for maintaining the negotiated quality of service. A single SH may thus control all MSS nodes within a small building. Our architecture, thus, separates the mobile network from the high-speed wired network and provides connectivity between the two via SHs who serve the function of a **gateway**.

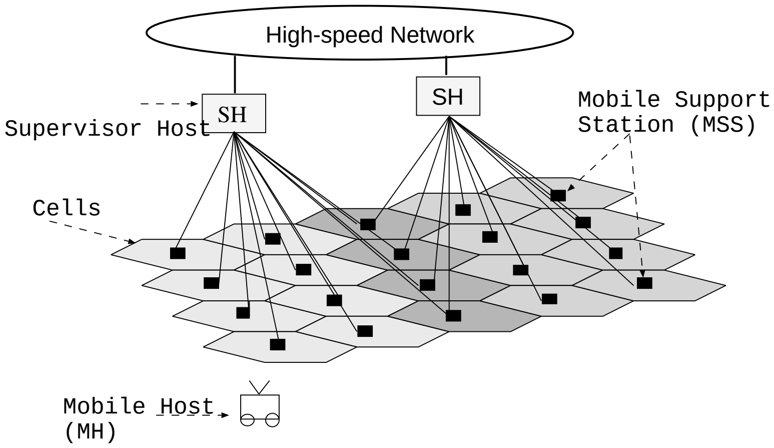


Figure 1: Our mobile network architecture.

What are some of the advantages of using our three-tier architecture?

1. The handoff process between MSS nodes is simplified. This is because, in our architecture, MSS nodes only operate at the *network* layer and as a result there is no transport layer state that needs to be transferred when a MH moves between two MSSs. The result is that the MSS nodes are cheap devices – an important consideration because every cell has one MSS node and any geographical area will be covered by many such cells.

2. Significant transfer of state information (i.e., transport state) between SHs takes place only when a MH moves out of a cell belonging to one SH and into a cell belonging to another SH (see [6] for a transport state handoff protocol that runs between SHs in this case). However, observe that users tend to remain for significant amounts of time within a building or within some other small geographical area the number of such handoffs is therefore reduced. Even in areas (such as downtown areas) with high user mobility the number of these handoffs is kept small because the number of users crossing the *perimeter* of an area covered by a SH is smaller than the total number of users moving between cells. In fact, if a SH controls n cells, the average number of handoffs per second that require

transfer of state information is of the order of n as opposed to n^2 which is the case if we do not use SHs (see Singh[10]).

3. From a *security* standpoint, centralizing complexity in the SH is a good idea because it is easier to protect SHs as opposed to protecting MSSs. There are two reasons for this: (1) MSSs are typically located in ceilings of buildings or in street light poles in downtown areas and are thus easily tampered with and, (2) protecting MSSs from software tampering is expensive (since the complexity of MSSs goes up). The SH, in contrast, can be physically protected because they can be located almost anywhere in the network. Software tampering is also easy to deal with since it is cheaper to build complex firewalls at the SH.

2.1 Overview of our protocol stack

The network layer at the SH uses Mobile-IP to route datagrams to the MH. Specifically, the SH acts as the Foreign Agent for new MHs who enter one of its cells. These MHs are assigned temporary IP addresses and *these address remain constant so long as the MH is in a cell controlled by the SH.* This ensures that the checksums in the TCP/UDP headers will not need to be modified if the MH roams locally. How are packets routed from the SH to the MH's current MSS? The SH maintains a cache containing a mapping between the MH's temporary IP address and the IP address of its current MSS. Packets are then routed to the MH via the MSS using *loose source routing* (see [8]). The MSS transmits all packets it receives from the SH over its wireless channel.

We have defined a set of transport layer protocols for mobile networks that are similar to protocols to be made available over high-speed networks. However, our protocols were designed while keeping in mind the constraints of the mobile environment such as arbitrary bandwidth fluctuations (as users move in and out of a cell) resulting in indeterminate bandwidth availability, and the unpredictable mobility of users. It is important to note that our protocols are only implemented in the mobile networks (i.e., SH, MSS and MH) and are not intended for high-speed networks. Thus, any connection that is established, where one end is mobile and another is fixed, will be split in two – one from the fixed host to the SH (using an appropriate high-speed network protocol) and another from the SH to MH (using one of our protocols) with all protocol translation being done at the SH. For a detailed explanation of our protocols and the functionality of the SH, please refer to Singh[3]. The *mobility management module* takes care of transferring transport connections when a mobile user moves into a cell controlled by another SH. The *bandwidth management module* allocates bandwidth to different connections in an effort to meet the negotiated QoS needs of the various MHs.

Figure 2 gives an overall picture of our transport layer. M-TCP and M-UDP refer to our versions of the TCP and UDP protocols respectively and are designed for providing efficient data connections to mobile users. The M-CM (Mobile-Continuous Media) protocol, on the other hand, is designed for applications requiring real-time transmission of data, for example, applications that use audio or video data. This type of data needs to be delivered to the mobile user within strict deadlines and, in addition, requires some guarantees on bandwidth. Unfortunately, because users are mobile, it is likely that the bandwidth

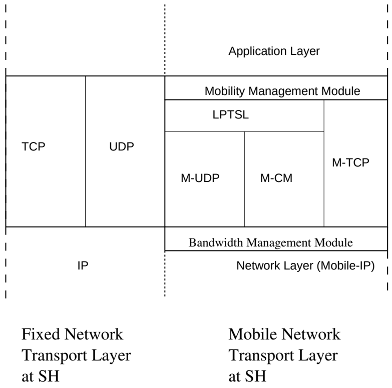


Figure 2: Transport layer stack for mobile networks.

available for a mobile user will vary over time[1]. M-CM attempts to provide a 'best-possible' service for these connections by starving data connections in favor of M-CM connections. LPTSL (Loss Profile Transport Sub-Layer) is a sublayer that may be used in conjunction with M-CM or M-UDP connections to provide the user with some control over dealing with situations where the bandwidth availability fluctuates. LPTSL *discards* data for real-time connections that are facing a bandwidth crunch. However, this data is discarded taking into consideration the structure of the data stream (e.g., compressed MPEG video or audio) and viewer perception. A discussion of LPTSL is beyond the scope of this paper and the interested reader is referred to Singh[10].

3 Design of M-UDP

Consider Figure 3. Packets arrive at the SH from some fixed host for delivery to the MH. As the MH moves about, it encounters periods of fade (as shown). If we use UDP to deliver packets to the MH, a significant fraction of these packets will be lost because they will be transmitted to the MH while the MH is in a fade. In fact, if r denotes the fraction of time the MH is in a fade, then it is easy to see that the fraction of data lost per second will be rD where D is the data rate of the UDP connection. How can this loss be avoided? The simplest solution would be to stop transmitting packets to the MH when it in a fade and resume transmission (including all packets that arrived during the fade interval) when the MH comes out of the fade (i.e., in the receive period). Unfortunately, in many cases the MH may simply be receiving data and it will not be possible for the SH to determine when the MH has gone into a fade (it is easy, however, to tell when the MH comes out of a fade by requiring the MH to send a message informing the SH of this fact). We have developed three versions of M-UDP that use information about the previous fade interval to retransmit packets that may have been lost.

Our goal in modifying UDP was to ensure that packets lost during a fade were retransmitted to the mobile once the mobile came out of the fade. The question is, **of the packets already transmitted, which packets need to be retransmitted?** Our first implementation is called *Statistical Fade Correction*

[1]If a user with open connections enters a crowded cell, the total requested bandwidth may well exceed the available bandwidth.

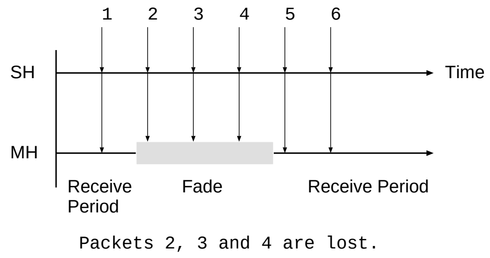


Figure 3: Loss due to fade.

(StFC) and works as follows.

Algorithm 1

- The SH maintains a data-structure that contains the mean *cell latency* of each cell controlled by the SH and the mean *fade interval* when moving from one cell to another. Consider, for example, an SH controlling cells covering a highway. Here, it is easy to determine the mean latency and fade intervals with some accuracy since, by and large, the mobile users have fairly predictable behavior.
- For a given M-UDP connection, the SH maintains a FIFO buffer that is large enough to contain the expected number of packets arriving for the mobile during an average fade interval.
- Whenever a mobile comes out of a fade, it informs the SH of this fact and the SH begins transmitting the buffered packets. Any new packets arriving during this period are placed at the back of this FIFO buffer.

This scheme works well (i.e., has a low loss) in systems where the fade intervals are chosen from a Normal distribution with a small standard deviation. Unfortunately, if the length of a fade is modelled as an exponential distribution, this scheme does not perform as well. This is primarily due to the fact that a buffer size proportional to the mean fade interval is not large enough to cover longer fades which occur with a high probability for the exponential case. Figure 4 illustrates the problem with this scheme. Here, the mean length of the fade interval yields a buffer size of 2 packets (assume all packets are of the same size). At the end of the first fade interval, packets 1 and 2 are in the buffer at the SH and they are retransmitted. Packets 3 and 4 are transmitted and are received before the next fade interval begins. The second fade is very long and packets 5 – 11 arrive during this interval. Since the buffer size is only 2, packets 10 and 11 get retransmitted after the end of the fade but packets 5 – 9 are lost.

A modification to this basic scheme is the *Smart Fade Correction* scheme (SmFC) described below:

Algorithm 2

- The SH maintains a FIFO buffer that is *three* times the size required to hold packets arriving during an average fade. A buffer of this size is large enough to buffer packets for 95% of fades, even when their length is exponentially distributed.

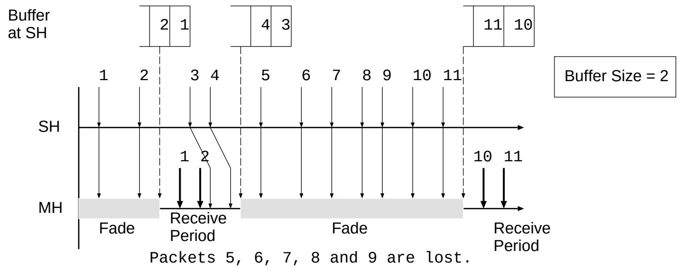


Figure 4: Retransmissions in StFC.

- When a mobile comes out of a fade it informs the SH of the *length* of the fade (note that the mobile can estimate the length of a fade by keeping track of the time during which it could not hear transmissions from the MSS). The SH then only retransmits those packets from the buffer that arrived during the actual fade. New packets that arrive during this retransmisson period are buffered.

Observe that SmFC does not blindly retransmit all packets stored in its buffer as StFC does. By using the length of fade information provided by the mobile, SmFC only retransmits the packets which were sent during the mobile's last fade period. As a result, the mobile sees very few duplicate packets, which saves power and extends battery life.

In our experiments we observed that this scheme did typically have lower losses as compared with the StFC scheme, however, it's performance for the exponential case was not as good as we had hoped. For high loads, we still observed a loss of up to 25%. The reason for this behavior is that there is a small but finite probability (about 5% in the exponential case and smaller in the normal case) of having very long fades and thus not all packets that arrive during this period can be buffered. Further, if a long fade is followed by a short period where the mobile is not in a fade, not all buffered packets can be retransmitted during the mobile's receive period. In this situation, some packets buffered during the previous fade may never be retransmitted.

Figure 5 illustrates the behavior of this scheme. Figure 5(a) presents a scenario that is identical to Figure 4. Here, however, because the buffer size is 6, packets 6–11 that arrive during the second fade get retransmitted. Packet 5 is still lost because the buffer is not large enough to hold all packets. Figure 5(b) illustrates a problem with the "memoryless" property of this scheme (i.e., retransmit packets that arrived during the previous fade only). Packets 3 and 4 arrived before the start of the second fade period but couldn't be transmitted because packets 1 and 2 were being retransmitted. After the end of the second fade, packets 5 onwards are retransmitted because these packets arrived at the SH *during* the second fade period. Thus packets 3 and 4 are lost even though the buffer is large enough to store packets 3 and 4.

The third, and best, scheme we implemented is called *Memory Fade Correction* (MFC) and is described below:

Algorithm 3

- We maintain a buffer that is *ten* times as large as that required to store packets arriving

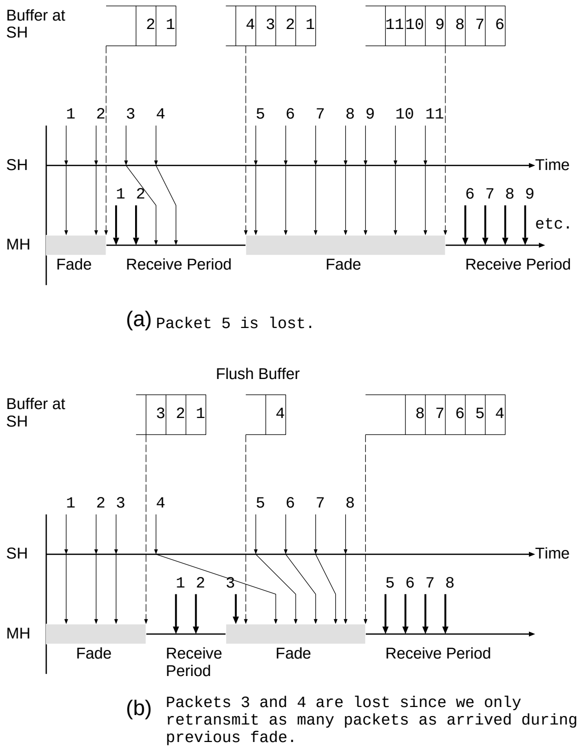


Figure 5: Retransmissions in SmFC.

during the mean fade interval. The reason for selecting such a large buffer is to ensure that losses are kept below 5%. A buffer of this size covers 99+% of the fades, even with exponential means.

- When a mobile comes out of a fade, it informs the SH of the length of the fade.
- The SH keeps track of the last time the mobile came out of the fade and, using these two numbers, can determine the length of the previous receive period during which the mobile was not in a fade. This information is used by the SH to determine from which packet to begin retransmissions.

Consider Figure 6(a). Here packets 1 and 2 get retransmitted as soon as the MH comes out of the fade. Packet 3 that arrives during this period is also transmitted. Packets 4–6 arrive during the second fade period. When the MH comes out of fade, the SH knows the length of the second fade interval and it knows that packets 1–3 were transmitted before the start of this fade period, so it only retransmits packets 4 onwards. In Figure 6(b), packets 1–6 arrived during the first fade period and only packets 1–4 could be retransmitted after the first fade ended. When the second fade ends, the SH knows the length of the first receive period and that of the second fade period. It uses this information to conclude that packets 5 and 6 were retransmitted during the second fade. Thus, after the end of the second fade, it begins retransmission starting at packet 5. It is important to observe that the two problems noted earlier for the SmFC scheme are thus eliminated. In addition, since MFC retransmits only the packets it knows were originally transmitted during a fade, it keeps the number of duplicates low.

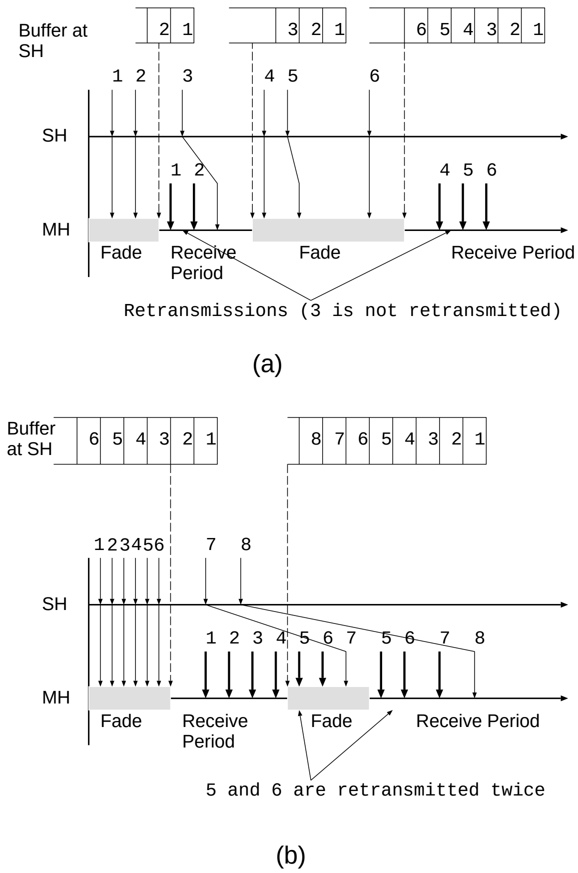


Figure 6: Retransmissions in MFC.

4 Design Issues

Before discussing our implementation, it is important to note that M-UDP is only implemented at the SH – no changes are needed at the MSS, MH or hosts in the fixed networks. In operation, the IP layer at the SH identifies incoming datagrams intended for mobile hosts which are currently in one of the cells controlled by the SH, and passes these datagrams up to M-UDP (via UDP). M-UDP buffers these datagrams and transmits them to the MH. At the MH, datagrams are received by UDP and processed in the normal way. If the MH is sending datagrams, it again uses UDP. We chose not to implement M-UDP at the MH (for transmitting datagrams) because, we believe that, a large percentage of the wireless bandwidth will be used on the *downlink* for transmitting data to the MH rather than on the *uplink*. In any event, if our assumption is false, it is trivial to incorporate M-UDP at the MH for transmitting datagrams as well. In the remainder of this section we focus on M-UDP implementation at the SH.

At the top level, the choice was whether to implement M-UDP as a part of UDP or as a protocol in its own right, with the attendant entry in the *proto_sw* data structure, a protocol number for ip, etc. The choice was made to implement it as a separate protocol giving a cleaner separation of functionality and allowing for possible future enhancement of features. IP uses the *proto_sw* struct to demultiplex packets to the appropriate transport layer protocol. Currently, the M-UDP protocol emulates UDP on the link *to* *the mobile host* (the IP protocol field is set to IPPROTO_UDP).

When a datagram first arrives at the SH, IP needs to decide if the datagram is to be passed up to M-UDP or routed on normally. To make this decision, IP examines a network-layer list of MHs that are currently in a cell managed by the SH. In our current implementation M-UDP also maintains a similar transport-layer version of this list as a linked-list of mpcbs (mobile process control blocks). Each mpcb contains information such as the MH address, MSS address and a pointer to a list of M-UDP datagrams. It is noteworthy that the mpcbs are shared between all transport layer protocols. The *mobility management module* module at the SH will maintain this mpcb list in the future but, currently, a *sysctl()* call to M-UDP is used to add or delete MH entries from these lists.

Newly arriving datagrams need to be buffered by M-UDP. We had a choice here in terms of dealing with the IP header of the buffered datagram – we could either store the IP header along with the datagram or recreate it each time the datagram is retransmitted. We chose the first option even though it is slightly more expensive in terms of using mbufs[2]. This is because the number of mbuf operations would be greatly increased if we were to recreate the IP header for each retransmission.

4.1 Implementation Details

In order to implement M-UDP we had to make changes to existing UDP/IP code (of NetBSD) and, in addition, add several routines of our own. To see what changes were made, it is best to use, as a guide, the sequence of steps followed to handle newly arriving datagrams or retransmission of buffered datagrams.

When a datagram first arrives at a SH, the sequence of calls made to process it are:

- *ip_intr()* (the IP input routine)
- *ip_dooptions()* (if packet was loose source routed from the sender to the MH via the SH)
- *udp_input()*
- *mudp_input()*
- *mudp_output()*
- *ip_output()*

For retransmissions, the sequence of calls is:

- *mudp_timeout()*
- *mudp_output()*
- *ip_output()*

The *changes* made to existing code are:

1. In the file *in.h* we added a protocol number IPPROTO_MUDP for M-UDP and an entry to the

[2] In most cases the *number* of mbufs used will be the same for both cases because the header is only 20 bytes and mbufs can hold 128 bytes.

CTL_IPPROTO_NAMES array which is used for *sysctl* calls to locate protocols by name.

2. In the file *ip_var.h* we added the *mhaddr* struct and a kernel declaration of the *mhaddr_list*, which is the network layer list of local MH addresses. Note that the MH addresses are stored in network byte order for efficiency. We also added a function prototype for *ip_localmobile()*, the function ip calls to see if an incoming packet is destined for a local MH.

3. In *in_proto.c* we added an entry in the *proto_sw* array for M-UDP and includes for the M-UDP header files.

4. The routines *ip_intr()* and *ip_dooptions()* were modified to take care of a curious problem that arises depending on whether the destination MH is present in a cell controlled by its home SH or is located in a cell controlled by a foreign SH. In the latter case datagrams are routed through the SH to the MH using the *loose source routing option* (LSRR). If the MH is in its home SH's domain, the packet may be sent normally using the MH's address as the destination. If the MH is away from its home SH, the packet must be loose source routed through the foreign SH first. We use a variable *mh_found*, which is global to this file, to bring the two cases together. One of the first things IP does when it receives a datagram is to call *ip_dooptions()* to see if there are any ip options attached (like LSRR). Normally a LSRR packet would just be forwarded along, however, *ip_dooptions()* has been changed to call *ip_localmobile()* and pass the packet back to *ip_intr()* if the packet is for a local mobile rather than sending it on to *ip_forward()*. It also sets *mh_found* so that *ip_intr()* need not check the destination address again. In the non-LSRR case, IP first checks to see if the packet is for a local address (local to the SH), and if it fails, calls *ip_localmobile()*. In both cases, if *ip_localmobile()* succeeds, the packet is demultiplexed up to UDP (*udp_input()*).

5. *udp_input()* has been modified to work as follows. If UDP finds that a datagram is not for a local port, it calls *tp_localmobile()*, the transport layer function that searches the mpcb list to see if this packet is for a local MH. If the address is found, the packet is passed to M-UDP, via a call to *mudp_input()*. A pointer to the mpcb is included so that M-UDP need not look it up again. The routine *tp_localmobile()* has been added to this file as well.

In addition to the above changes made to existing TCP/IP code, we needed to add several routines for M-UDP. We defined two new header files – *mobile.h* and *mudp_var.h*. mobile.h includes definitions that are to be shared amongst all transport layer protocols which deal with the consequences of mobility. The definitions of mpcbs as well as an M-UDP packet header are here as is the declaration of the *mpcb_list*. mudp_var.h contains declarations specific to M-UDP. A number of M-UDP statistics are declared here, as are the definitions needed for the many *sysctl* options that M-UDP supports. Function prototypes for the M-UDP functions are also declared here. All M-UDP routines are contained in the file *mudp_ussreq.c*. These routines are:

1. *mudp_init():* called from the *proto_sw* entry on system bootup, it simply starts up the M-UDP transmit timer that is used to emulate the wireless link.

2. *mudp_input():* Called from *udp_input()* when a new udp packet is received. A copy of the packet is made and placed on the mudp packet queue in the mpcb for the destination MH. The original packet is sent on to *mudp_output()*. A per-mpcb memory limit is enforced here. If the memory limit is exceeded, or there are no mbufs available, the original packet is simply passed on to *mudp_output()*. No resends will be possible, but the packet is always sent at least once. In this way the behavior of M-UDP is always at least as good as UDP in terms of packet delivery.

3. *mudp_output():* Fills in the UDP header fields, and calculates the checksum[3]. An ip option structure is allocated in an mbuf and the LSRR option is built. In the next version of our software, this will be done by the network layer at the SH). In order to emulate the low speed of the wireless link, the packet and option are queued, awaiting a timeout to be sent. In the StFC and SmFC algorithms (but not MFC), the packet is then removed from the mudp packet queue for this MH.

4. *mudp_timeout():* Called by the *timeout()* routine at a frequency determined by the kernel variable *mudptimeout*. *mudp_timeout()* emulates the speed of the wireless link. On timeout, the packet at the head of the queue is removed and passed, (with its ip options), to *ip_output()*. *mudp_timeout()* then reschedules itself.

5. *mudp_drain():* Called by IP via the *proto_sw* entry when an mbuf shortage occurs. The mudp packet queues of each mpcb are searched and all mudp packets are removed and freed. Note that all packets are sent once before being placed in these queues so all packets will still be sent at least once. This behavior is analogous to the drain functions of other TCP/IP protocols, for example, UDP throws away all fragments in its reassembly queue when *udp_drain()* is called.

6. *mudp_sysctl():* Called by the user level *sysctl* function, this routine allows kernel-level variables and structures relating to M-UDP to be changed while the kernel is running. Variables used to determine M-UDP functionality are:

   mudpcksum - enable or disable checksum on outgoing M-UDP packets

   mudptimeout - how long to wait between sends on the wireless link

   mudpmemlimit - the per-mpcb memory limit

7. *mudp_ctlinput():* Used for upcalls from the mobility management module, this routine handles MH movement messages by flushing all queues associated with the mobile and resetting fields in the appropriate mpcb. *mudp_ctlinput()* would also be called from the *proto_sw* entry if a control message was received by ICMP relating to IPPROTO_MUDP. Since M-UDP marks the packets it sends as UDP for now, this will not happen. If the MH is made aware of M-UDP in the future, *mudp_ctlinput()* would handle M-UDP errors reported by the MSS and MH.

In addition to all of the above changes, *netstat* and *vmstat* were modified to allow M-UDP statistics to be shown.

[3] We require checksum on M-UDP packets since, at this point, we know that they are going across an unreliable wireless link. The original source may not have used a checksum assuming delivery on a reliable fixed network.

5 Performance analysis

To test the performance of M-UDP we used the experimental set up shown in Figure 7. The sender is a DecStation 5000 that transmits UDP packets to the SH which is a Pentium PC running NetBSD. The SH sends these (M-UDP) packets to the MH (another PC) via the MSS (which is another DecStation 5000 that serves as a router) using *loose source routing* (see Johnson[8]).

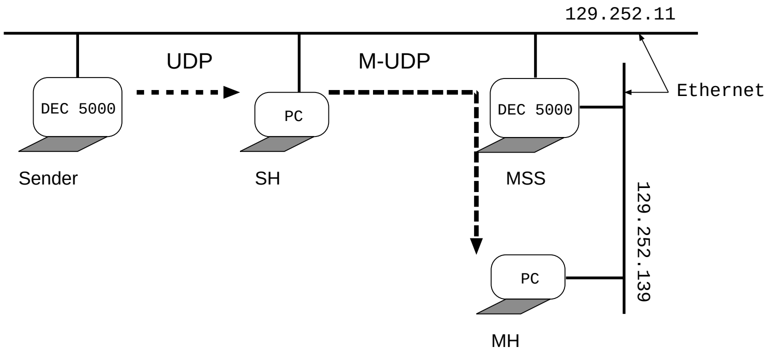


Figure 7: Experimental set up.

In these experiments all the machines are connected via ethernets. Thus, to emulate the *transmission speed* over the wireless link (from the DEC 5000 to the MH) and the *fades* we made the following modification at the SH. All calls to ip_output() are intercepted and then, based on the outgoing link speed (32 kbps) and the packet size (including UDP and IP headers, but not link layer headers), a timer is set to go off each time a packet can be sent on the wireless link. At each timeout, 1 packet is removed from the queue and ip_output() is called normally. In order to emulate a fading wireless channel, packets are discarded at the MH as follows. A MH may be viewed either as being reachable or as being in a fade. Each fade period is followed by a period where the MH is reachable and each period of reachability is followed by a fade. Thus, we divide time into alternating periods of fade and reachability. Packets transmitted to the MH during a fade are discarded. In addition to packet loss during fades, packets are also lost due to noise on the channel. This is simulated by requiring the MH to discard a packet with a probability of 0.008. This is because the packet size used is 80 bytes of data plus 20 bytes of IP header. Thus, a BER of 10^{-5} yields a packet loss rate of 0.8%.

The cell latency and length of the fade interval are modeled as random variables chosen either from an exponential distribution or from a normal distribution. The mean cell latency is fixed at 5 seconds while the mean fade interval varies between 0.5 and 4.5 seconds. The standard deviation (normal case) is either 0.05, 0.15 or 0.25 of the mean for both, the cell latency and the length of the fade. The link speed used is 32kbps. However, since the MH is frequently in a fade, the actual data rate seen by the MH is much smaller. Specifically, if the MH is in a fade x% of the time, the *maximum acheiveable* data rate is no more than $0.32x$ kbps. A *load* of 0.9 thus means that data is generated by the sender at a rate of $0.9 \times 0.32x$ kbps or $0.288x$ kbps. In these experiments we generate data at a *constant rate.* Each experiment is run for 4.5 minutes and the first 30 seconds of data is discarded (to counter transient effects). For each data point, we compute 95% confidence intervals and the interval half-widths kept to less than 5% of the point values in all cases.

Figures 8 and 9 illustrate the performance of UDP and the three versions of M-UDP for a load of 0.9. The X-axis plots the length of the fade interval and the Y-axis plots the *percentage of packets* received at the MH. Three graphs display the performance of these schemes for the case when the fade interval and cell latency are chosen from a normal distribution. Here, the loss rate for the three versions of M-UDP, is always less than 5% while the loss rate for UDP is between 10% (for short fades where $\frac{\text{fade}}{\text{fade+latency}} = 0.09$) to about 47% for long fades (where this ratio is 0.47) as expected. The performance of MFC is better than StFC and SmFC for the case when the cell latency and fade periods are exponentially distributed because the SH maintains information about the lengths of previous fade intervals and uses this information to decide which packets to retransmit to the MH. StFC, on the other hand, relies on the *mean* fade length to make this decision. Also, StFC and SmFC only retransmit packets from the most recent fade interval and, as a result, frequently retransmit fewer packets than MFC, which may reach back to previous intervals. Figure 9 shows that StFC has a lower loss rate than MFC and SmFC for the case when the standard deviation is very small (0.05*Mean). This happens because StFC retransmits packets that MFC and SmFC may not because StFC is a more pessimistic scheme. In the presence of bit errors (1% in our system), some packets transmitted to the MH will be considered lost even though they are received. StFC sometimes retransmits these packets resulting in a lower percentage of total loss.

What is the overhead associated with these three versions of M-UDP? As we stated in section 3, the *buffer size* used for MFC is large enough to hold all packets arriving during a period whose length is ten times as long as an average fade period (this ensures that losses are kept less than 5%). The buffer for SmFC holds three times as many packets as are received during an average fade and the buffer used for StFC only holds packets that arrive during one average fade period.

While buffer usage at the SH is an important measure of overhead, we believe that more important measures are *the number of times a packet is retransmitted over the wireless link* and *the number of times the packet is received at the mobile.* Both, the bandwidth on the wireless link and power at the mobile station, are scarce resources and must not be wasted by extraneous sends or receives. Figure 10 shows the average number of times a packet is transmitted over the wireless link and the average number of times it is received at the MH (i.e., average number of duplicates). The X-axis plots the length of the fade interval used and the Y-axis plots the copies per packet. The fade interval and the cell latency are normally distributed random variables. The top two graphs represent the case when the standard deviation is 0.25*Mean while the bottom two represent the case when the standard deviation is 0.05*Mean. The number of *duplicate sends* is between 1.1 (for short fades) and 1.6 (for long fades) for all three versions of M-UDP. The number of *duplicate receives* is, however, very close to 1 (i.e., very few duplicates are received by the MH) for SmFc and MFC. Figure 11 illustrates the behavior of these schemes for the most difficult case, exponential means on the latency and fade intervals. The number of duplicate sends here is almost 1.9 (i.e., on the average each packet is transmitted twice) while the number of duplicate receives is still close to 1. If we consider battery power consumption at the MH, the number of receives is clearly an important criteria and we would like to keep the number of *duplicate receives small.* As we see, all three schemes ensure that the percentage of duplicate receives is very small.

For exponentially distributed means, MFC is clearly the best version of M-UDP because it has a low

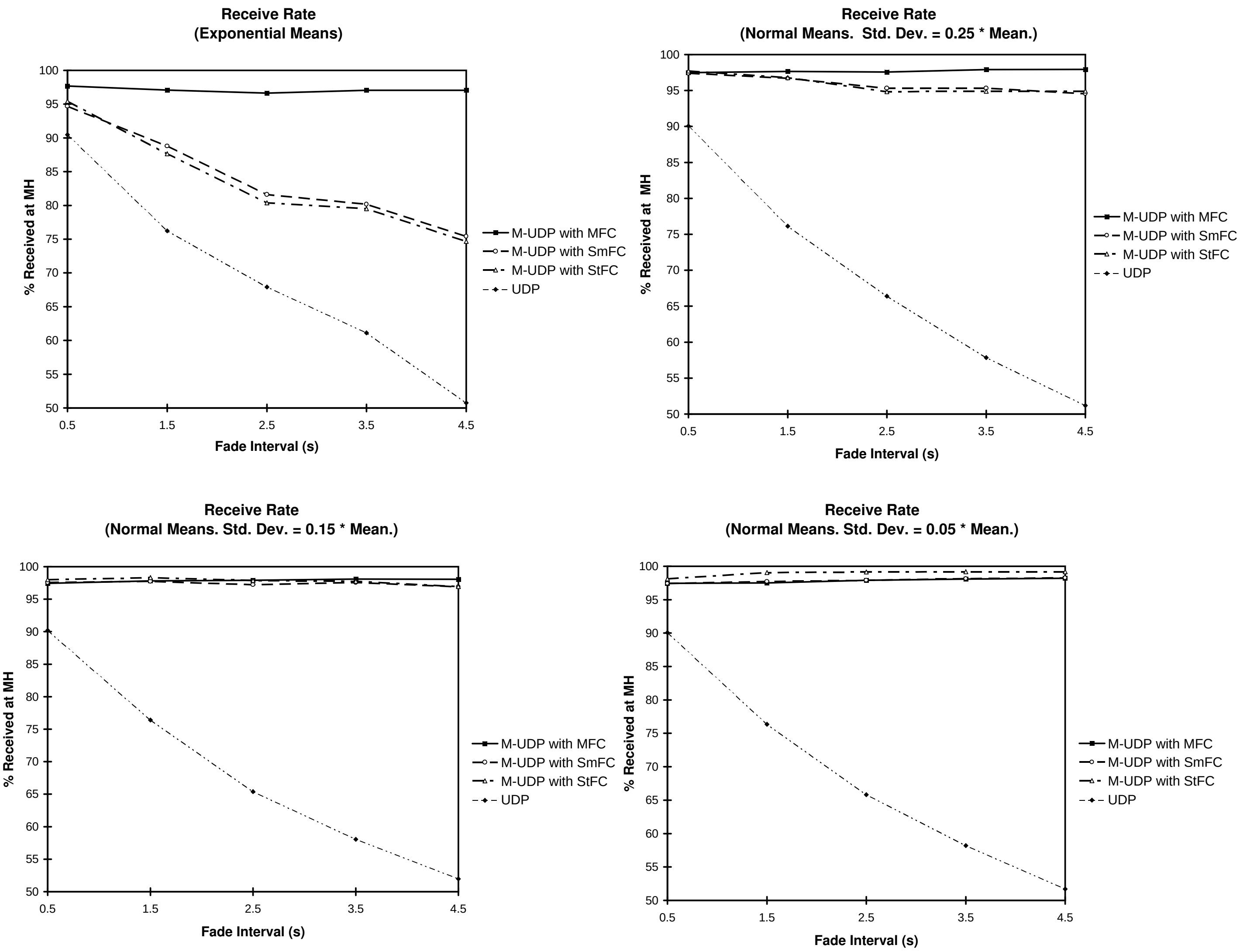


Figure 8: Mean Latency = 5s. Load = 90%.

loss (see Figure 8) and low percentage of duplicates at the receiver. Figure 12 illustrates MFC's overhead for a wide range of loads. Here, the X-axis plots the load and the Y-axis plots the copies per packet transmitted to the MH. The mean fade period chosen is 2.5 seconds. We see that for the exponential case, each packet is transmitted, on an average, between 1.4 and 1.6 times. For the normal case (the standard deviation is 0.25*Mean), this number is constant at about 1.35.

5.1 Summary

Of the three schemes, which scheme is most appropriate? If we know that the mean latency and fade intervals are normally distributed, it appears the SmFC or StFC are appropriate. This is because the percentage of loss is small and, in addition, the buffer space used at the SH is small. If, on the other

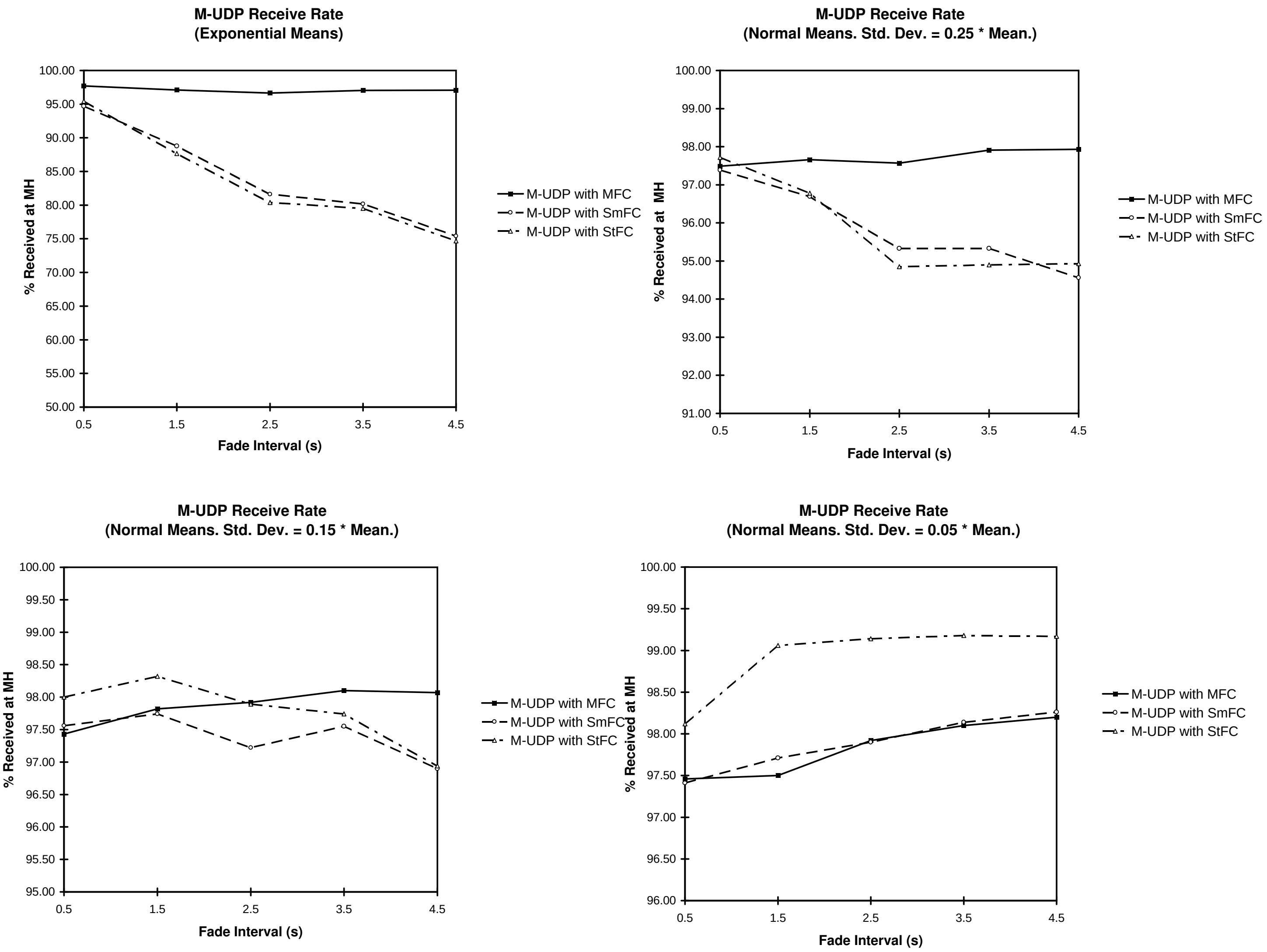


Figure 9: Mean Latency = 5s. Load = 90%.

hand, we know that the means are exponentially distributed, then it is clear the MFC ensures the smallest percentage of loss even though its buffer requirements at the SH are high.

6 Conclusions and future work

In this paper we have presented an implementation of a mobile version of the UDP protocol called M-UDP that attempts to keep the number of lost datagrams small in the event that the receiver is a mobile user. It does so by retransmitting datagrams lost during a fade.more than once. We have implemented M-UDP and tested its performance. The previous section shows that M-UDP performs very well in a mobile environment for a wide range of system parameters. M-UDP forms a part of our larger transport layer design that is discussed in Singh[3].

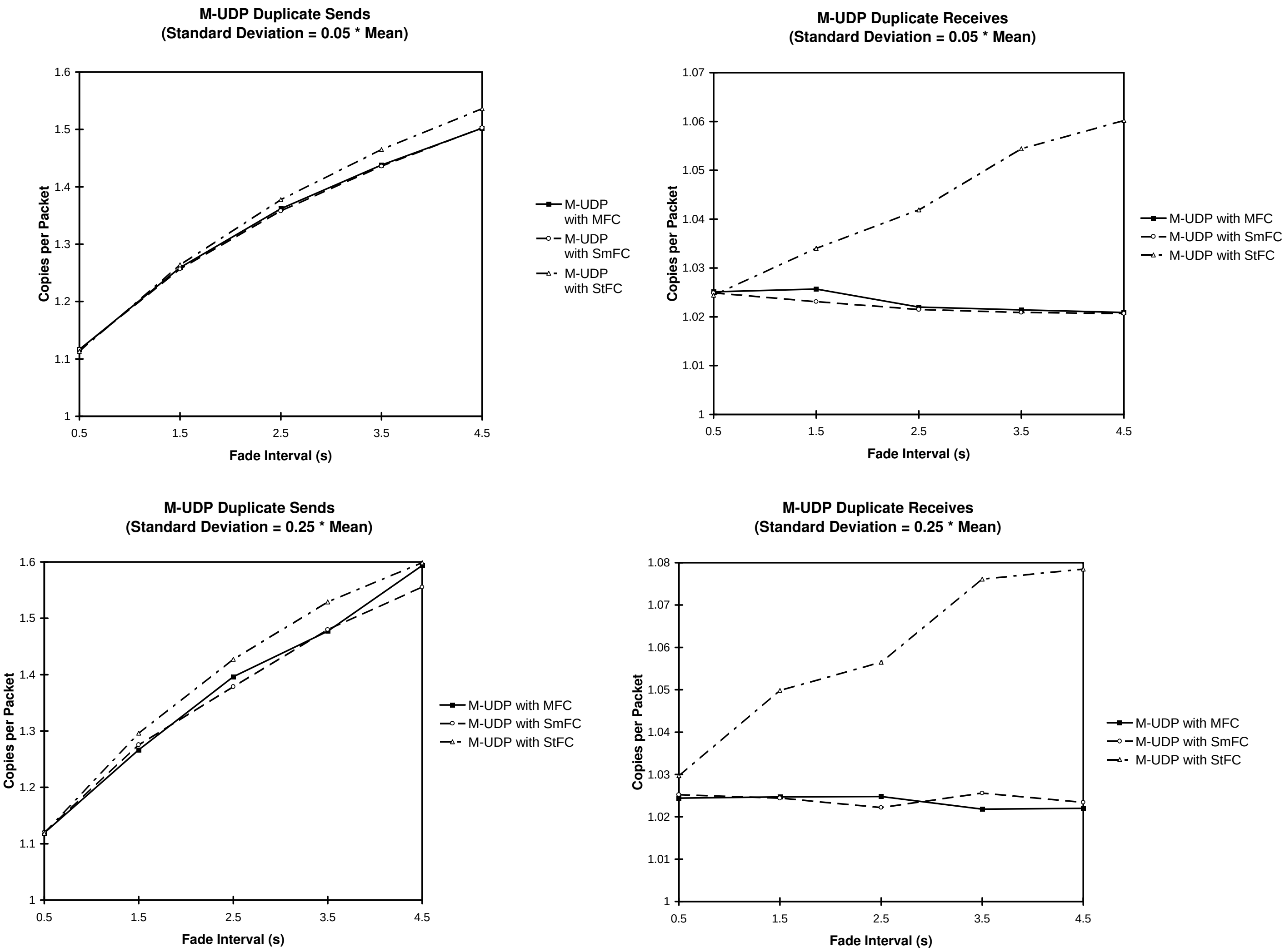


Figure 10: Normal Means. 90% Load. Mean Latency = 5s.

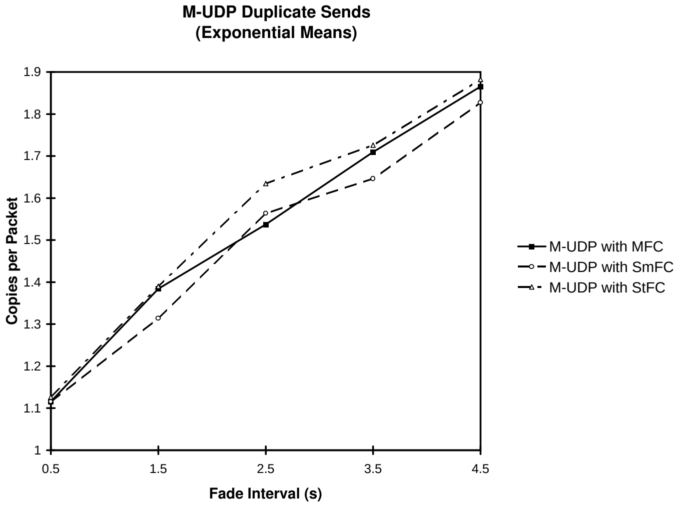
References

[1] A. Bakre and B. R. Badrinath,"I-TCP: Indirect TCP for Mobile Hosts", *Technical Report* DCS-TR-314, Rutgers University, Piscataway, NJ 08855.

[2] P. Bhagwat, S. Tripathi and C. Perkins, "Network Layer Mobility: an Architecture and Survey", *Technical Report*, CS-TR-3570, University of Maryland, (September 13, 1995).

[3] K. Brown and S. Singh,"A Network Architecture for Mobile Computing", *IEEE INFOCOM'96*, (April 1996), pp. 1388-1396.

[4] R. Ghai and S. Singh,"An Architecture and Communication Protocol for Picocellular Networks",*IEEE Personal Communications Magazine*, Vol. 1(3), 1994, pp. 36-46.



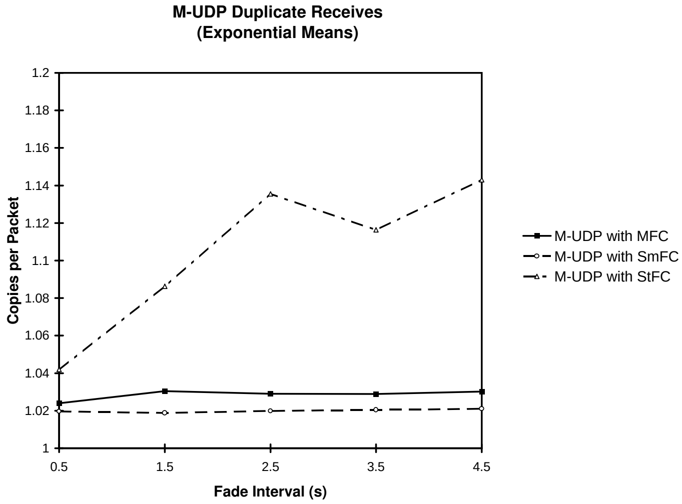


Figure 11: Exponential Means. 90% Load. Mean Latency = 5s.

[5] David J. Goodman, "Cellular Packet Communications", *IEEE Trans. on Comm.*, vol. 38, no. 8, pp 1272-1280, August 1990.

[6] R. Gopalakrishnan, K. Brown and S. Singh, "Transport State Handoff in Mobile Networks", *ACM/IEEE Mobicom'96*, (submitted).

[7] J. Ionaidis, D. Duchamp and G. Q. Maguire, "IP-based protocols for mobile internetworking" *Proc. of ACM SIGCOMM'91*, pp 235-245, September 1991.

[8] D. B. Johnson,"Mobile Host Internetworking Using IP Loose Source Routing", Technical Report CMU-CS-93-128, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.

[9] M. Moran and B. Wolfinger,"Design of a Continuous Media Data Transport Service and Protocol", Technical Report TR-92-019, Computer Science Division, University of California Berkeley, April 1992.

[10] K. Seal and S. Singh,"Loss Profiles: A Quality of Service Measure in Mobile Computing", *J. Wireless Networks*, Vol. 2, (1996), pp. 45-61.

[11] S. Singh,"Quality of Service Guarantees in Mobile Computing", *J. Computer Communications*, (to appear in mid-1996).

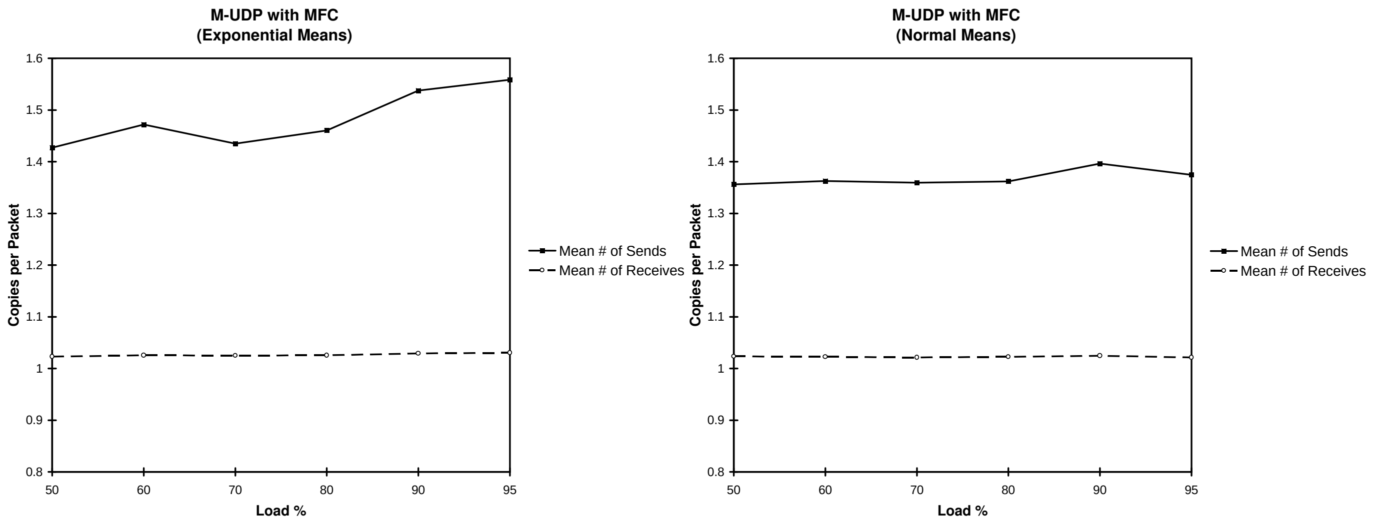


Figure 12: MFC with Mean Latency = 5s, Mean Fade = 2.5s.