Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2006, Article ID 95149, Pages 1–16
DOI 10.1155/WCN/2006/95149

TCP-M: Multiflow Transmission Control Protocol for Ad Hoc Networks

Nagaraja Thanthry, Anand Kalamkar, and Ravi Pendse

Department of Electrical and Computer Engineering, Wichita State University, 1845 N Fairmount, Wichita, KS 67260, USA

Received 2 August 2005; Revised 18 February 2006; Accepted 13 March 2006

Recent research has indicated that transmission control protocol (TCP) in its base form does not perform well in an ad hoc environment. The main reason identified for this behavior involves the ad hoc network dynamics. By nature, an ad hoc network does not support any form of quality of service. The reduction in congestion window size during packet drops, a property of the TCP used to ensure guaranteed delivery, further deteriorates the overall performance. While other researchers have proposed modifying congestion window properties to improve TCP performance in an ad hoc environment, the authors of this paper propose using multiple TCP flows per connection. The proposed protocol reduces the influence of packet drops that occurred in any single path on the overall system performance. The analysis carried out by the authors indicates a significant improvement in overall performance.

1. INTRODUCTION

The transmission control protocol (TCP) is the most widely used transport layer protocol in the networking world. Many of the features supported by the TCP were developed based on a wired network environment, although now they are also being used in wireless networks. It has been observed that the TCP does not perform well in a wireless network environment due to such issues as link failures, collision, interference, and fading. Link failures are considered one of the major causes of performance degradation. The TCP works on the principle of guaranteed delivery. When a sender does not receive an acknowledgment for a packet being transmitted, the TCP assumes that the packet is dropped due to congestion and therefore attempts to retransmit it. In a wired network environment, packet drops generally result from network congestion, but in a wireless network, packet drops could also result from link failures. However, the TCP, by design, attributes packet drops to network congestion and attempts to avoid this by reducing the transmission rate. This further degrades network performance in a wireless network environment. Apart from the reduction in transmission rate, TCP behavior also results in an increase in the retransmission timeout period (RTO), which further delays packet delivery.

This situation further deteriorates when the TCP is used in its base form in an ad hoc network environment. An ad hoc network is formed on the fly and can be characterized by the absence of a centralized authority, random network topology, high mobility, and a high degree of link failures. This absence of a centralized authority makes it difficult to deploy any form of quality of service improvement techniques in the ad hoc network environment. With mobility in place, nodes often need to rediscover the path to a destination. During the route discovery process, the path to the destination will be unavailable, thereby resulting in packet drops.

Many researchers have considered the issue of improving TCP performance in the wireless network environment, and some of them have suggested using a feedback mechanism. TCP implementations like TCP-ELFN [1] and TCP-F [2] make use of an immediate neighbor to provide notification of path failure. The authors of [3] suggested using a constant RTO instead of exponential back-off in order to improve TCP performance. This scheme attempted to distinguish between route failures and network congestion by keeping track of number of timeouts. If an acknowledgment times out second time, it is attributed to the route failure and the packet is retransmitted without changing the RTO value. The analysis carried out in [3] also showed a significant improvement in TCP performance. The authors of [3] observed that the protocol is only applicable for a wireless network MANET environment due to the fact that the concept of route failures can be applied only in the case of MANET. In a wired network scenario, the packet losses are caused mainly due to network congestion. Another major factor that needs additional research in the case of the proposal presented in [3] is the claim that the successive packet drops are due to route failure. In a dense network scenario, with multiple nodes attempting to transmit at the same time, it is quite possible that the acknowledgment timed out due to network congestion itself.

One way to improve performance in ad hoc networks is to deploy multipath routing. In a sufficiently large network, a source node can route packets to the destination using multiple paths, which could be established using intermediate nodes. Most of the routing protocols used in ad hoc networks maintain a single path for every destination. Therefore, for every path failure, the routing protocol wastes a significant amount of time in rediscovering the path. In addition, other paths that might be available between the source and the destination are underutilized, thus wasting network resources. In ad hoc networks, paths are short-lived due to node mobility. Hence, maintaining only a single path for any destination may result in frequent route discoveries. Maintaining multiple paths between the source and the destination will ensure the availability of at least one path during link failure. Recently, many routing protocols have been developed to support multipath routing. While some of these protocols are based on a link-disjoint model, others work on the basis of a node-disjoint model.

Multipath routing improves the probability of path availability between the source and the destination. However, it does result in variable round-trip time (RTT), which does not work well with TCP flows. Packets flowing through the paths with a higher RTT tend to time out, forcing the sender to retransmit the packets. In addition, multipath routing causes out-of-order delivery of packets, which results in duplicate acknowledgments and additional delay at the receiving end (due to packet realignment). Recent research efforts on the performance evaluation of the TCP over a multipath environment have revealed that TCP performance actually degrades in a multipath environment [4]. In this paper, the authors propose extensions to the existing TCP to support a multipath ad hoc network environment. These extensions are based on the principle of multiple connections between the source and destination at the transport layer, similar to the idea proposed in [5]. Since each connection takes a distinct route to reach a destination, single-link failure will not affect the performance of the network to a large extent. Analysis carried out by the authors indicates that the proposed extensions improve TCP performance to a large extent, compared to the normal TCP.

The remainder of this paper is organized as follows. In Section 2, the authors review the related research work involving the TCP and ad hoc networks. Section 3 discusses the performance of the TCP in multipath ad hoc networks. Section 4 presents the proposed protocol, that is, TCP-M. Section 5 shows an analysis of the proposed protocol and compares it to the existing TCP. Conclusions are presented in Section 6.

2. RELATED WORK

While the TCP is the most widely used transport layer protocol, it has been proven that it does not perform well in an ad hoc network environment. Link unavailability (mainly due to mobility) is considered to be one of the main reasons for this performance degradation. However, the TCP assumes that performance degradation is due to congestion and resorts to congestion-avoidance techniques. Many researchers have suggested different methods to address the performance issue associated with the TCP and ad hoc networks. The authors of [1] proposed feedback-based extensions to the TCP and termed the new protocol TCP-ELFN. In their protocol, the sender probes the network to check the path status. In the TCP-ELFN protocol, neighbors take an active part and send notifications of link failure whenever a path becomes inactive. At this instance, the sender freezes its congestion window, thereby minimizing the effect of packet drops on overall throughput.

TCP-F is another similar proposal employing a feedback mechanism. Here, the intermediate nodes notify the sender about the link failure after which the sender freezes the congestion window. The difference between the two is that instead of the sender probing for network status, the intermediate nodes or neighbors intimate the sender whenever a link becomes active. One of the major issues in this protocol is the reliability of the feedback mechanism. It does not specify any measures to deal with lost feedback messages.

Another approach is to fix the RTO timer value. The fixed RTO scheme was primarily proposed to overcome the problem of a long restart latency caused by the exponential back-off algorithms during link failures. Normally the RTO is doubled for every retransmission timeout, and this continues until it reaches 64 times the original timeout value. After this, the timeout value remains constant until any one of the packet is acknowledged. According to the fixed RTO scheme, the RTO is frozen until the path becomes active again.

Sundaresan et al. suggested using the ad hoc transport Protocol (ATP) for mobile ad hoc networks [6]. After studying issues with the TCP in ad hoc networks, the authors have taken a new approach using ATP to improve the performance of the transport layer protocol in ad hoc networks with the introduction of a rate-based transmission scheme rather than window-based transmission. The authors also have introduced other techniques such as quick start, separating the congestion control mechanism from the reliability, a composite parameter that considers the effect of transmission, and a queuing delay of the followed path.

Sundaresan et al. revealed that, even though the slow-start mechanism used in the TCP may be effective in wired networks, it has several drawbacks in an ad hoc network environment. Although the slow-start mechanism uses an exponential increase in the congestion window, this method is not aggressive enough, especially because of the short-lived nature of links in ad hoc networks, where the packet drop is due to link failure. Also, when the link becomes active again, the slow-start mechanism wastes a significant amount of time to reach the normal transmission rate, whereby it can transmit at the capacity rate almost immediately after becoming active. The authors who suggested ATP introduced a quick mechanism; whereby after a packet drop or at the time of connection establishment, the sender sends a synchronization (SYN) packet to the receiver. The intermediate nodes update the values of the parameters Q_t, queuing delay, and T_t, transmission delay, for the link. When the receiver receives the SYN packet, it replies by an acknowledgment (ACK) packet with the values of Q_t and T_t. After receiving the ACK packet, the sender determines the bandwidth of the path and begins transmitting at the same rate instead of exponentially increasing the transmission rate.

Rather than using window-based transmission, the ATP uses a three-step rate-based data transmission. This scheme consists of an increase phase, whereby the sender checks the feedback rate from the receiver. If this rate is greater than the current rate, then the sender increases the transmission rate. The threshold is kept as the current rate to allow flows with lower rates to increase more aggressively than flows with higher rates. If the feedback rate from the receiver is less than the current transmission rate, then the sender reduces the transmission rate to the value in the feedback. If the feedback rate is within a certain limit of the current rate, then the protocol maintains the current transmission rate.

Thus, the ATP tries to solve issues that are the result of the slow-start mechanism response of the TCP to congestion and the window-based transmission scheme. However, the ATP still fails to make optimal use of network resources since it does not have a mechanism for multipath routing, and also, data transmissions are scheduled by the timer at the sender; thus requiring timer overheads at the sender.

Although all of these schemes try to improve the performance of the TCP, they use only a single path to transmit the data from source to destination. Therefore, every time a path failure occurs, a considerable amount of time is wasted until a new path is reestablished. However, a number of other alternate paths that are not being used might be available in the network. In the case of path failure, being able to use multiple paths can improve the performance of a TCP since it can use an alternate path.

The routing protocol used at the network layer also plays an important role in ad hoc networks. A robust routing protocol increases path availability. Recent proposals for routing protocols maintain more than one path for the same destination in the route cache so that when the primary path fails, the secondary or backup path can be used immediately. In the case of a path failure, this helps to eliminate additional time wasted in the route discovery. Some of the multipath routing protocols are discussed below.

The multipath routing protocol called ad hoc on-demand multipath distance vector (AOMDV) [7] finds multiple-disjoint loop-free paths during the route discovery. The paths can be node-disjoint or link-disjoint and are selected on the basis of hop count. A node enters a path in the table, only if the new route has less hop count than the one already in the table.

When the protocol is configured for using a node-disjoint path, those paths with no common intermediate node are selected; whereas when the protocol is configured for using a link-disjoint path, those paths with no common intermediate link are selected. Using node-disjoint paths provides more granularity in path selection and guarantees more reliable paths. However, it also reduces the number of available paths, as compared to link-disjoint paths. The authors observed in their performance evaluation that, in most cases, the link-disjoint paths have satisfactory performance. The AOMDV maintains different next hops for different paths.

Unlike in the single-path routing protocol called ad hoc on-demand distance vector (AODV), in AOMDV the intermediate node does not simply drop the duplicate route request (RREQ) packet but examines it to see if it gives a node-disjoint path to the destination. If so, then the node checks to see if the reverse path to the source is available. If this is also true, then the path is added in the table. In the case of a link-disjoint path, the node applies a slightly lenient policy and replies to a certain number of RREQs, which come from disjoint neighbors. The unique next hop guarantees the link-disjoint path. Although not a part of the basic implementation, the AOMDV can use multiple paths simultaneously for data transmission.

Split multipath routing (SMR) [8] is another multipath protocol, which attempts to utilize the available network resources in an effective manner. SMR is also an on-demand routing protocol, which finds and uses multiple-disjoint paths. The SMR uses a per-packet allocation scheme to distribute data packets among different paths of the active session. This scheme helps in utilizing network resources and preventing congestion at a node under heavy-load conditions. The protocol operates as follows.

Being an on-demand routing protocol, the SMR source broadcasts the RREQ packet only when the route to the destination node is not present in the route cache. The RREQ packet contains the source ID and sequence number that uniquely identifies the source. When the intermediate node receives the packet, it records its node ID in the packet header and further forwards that packet. The intermediate node forwards any duplicate RREQ packets that come from different links and have a hop count less than the earlier-received RREQ packet. Also, the intermediate node does not send the source route reply (RREP) packet, even if it knows the path to the destination. This helps avoiding paths with common links. Although more than two paths can be selected in the SMR implementation, the receiver selects two disjoint paths. Upon receiving the first RREQ packet, the destination replies to the source by sending the RREP packet with the complete path in the packet. Then the destination node waits a certain amount of time and receives more RREQ packets. Then it selects the route that is maximally disjoint with the path already sent to the source. In the case of multiple-disjoint paths, the path with the shortest hop count is used. In the case of a tie, the path received first is used. When an intermediate node is unable to forward the packet to the next hop, it assumes a link failure and sends the route error (RERR) packet in the upstream direction to the source. Upon receiving the RERR packet, the source deletes every entry in the route table that uses the broken link. After this, if the second path to the

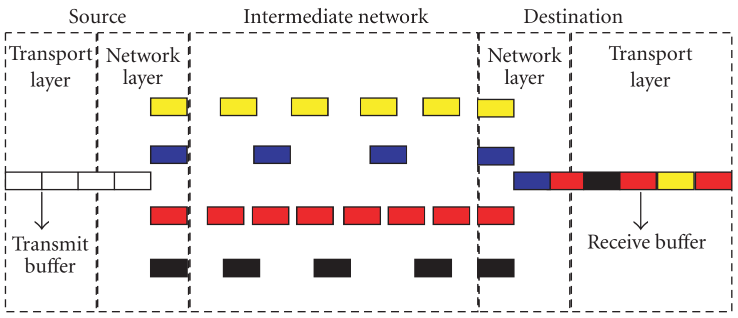

FIGURE 1: Packet forwarding in a multipath environment.

destination is available, the source uses the remaining route for data transfer or it restarts the route discovery.

Thus, the SMR allows two routes to be used simultaneously for data transmission and provides optimal use of network resources. Choosing the second route disjoint with the first one reduces the possibility of both routes failing at the same time and hence giving greater path availability.

In the next section, the authors discuss the issues involved in using the TCP in a multipath ad hoc network environment.

3. TCP PERFORMANCE IN MULTIPATH AD HOC NETWORKS

The TCP was originally designed for wired networks where the packet drop is generally assumed to be due to network congestion. Therefore, whenever there is a packet drop, the TCP goes into fast-retransmit mode and appropriately reduces the congestion window. If the path is unavailable for an extended period of time and packets are still being dropped, the TCP enters the slow-start mode, and the window size is reduced to one. This can be justified in the case of wired networks where congestion is the primary reason behind a packet drop. However, when the TCP is used for mobile ad hoc networks, its performance suffers. Here, the links are short-lived and prone to errors, and generally the packet drop is due to link failure. In this case, even after reducing the window size, if the packet sent is dropped, the TCP enters the slow-start mode and reduces the window size to one packet. As pointed out by the authors of [6], if the path is reestablished at this stage, the TCP takes a longer time to come out of slow-start and attain normal transmission capacity of the path, thus wasting path capacity during this time.

With the traditional TCP implementation in place, even using multipath routing will not improve the situation. As seen in Figure 1, in the traditional implementation, the TCP maintains a single buffer and congestion window for every connection. When packets are routed through different paths, a packet drop in any one path (which is heavily congested) triggers a change in TCP behavior. This automatically pushes the TCP into the congestion-avoidance mode, thus reducing the rate of data transmission and the congestion window size. This diminishes the advantages of multipath routing to a great extent.

Another important factor to be considered is the value of retransmission timeout (RTO). When multiple paths with different RTT values are present, it is better to maintain different RTO values for each of these paths. This ensures a guaranteed delivery of the packets to their destination and also reduces the number of duplicate acknowledgments. Although different RTT values result in an out-of-order delivery of packets at the destination, this is better than losing packets or increasing traffic due to duplicate acknowledgments.

Recently, the authors of [9] proposed transmission control protocol-persistent packet reordering (TCP-PR) in which they suggested a mechanism to handle out-of-order packet delivery in MANETs. The TCP-PR does not rely on duplicate acknowledgments to detect packet loss in the network. Instead, it maintains a timer for each transmitted packet. This timer is set when a packet is transmitted, and if the sender does not receive the acknowledgment before the timer expires, the packet is assumed to be dropped. Hence, since this protocol does not reply on duplicate acknowledgment, reordering at the receiver does not degrade TCP-PR performance.

The TCP-PR maintains two lists: a *to-be-sent* list, which contains all the packets that are waiting to be transmitted, and a *to-be-ack* list, which stores all the packets whose acknowledgments are pending. When an application has to send a packet, it puts the packet in the *to-be-sent* list. When the packet is transmitted, a time stamp is applied to the packet, and it is removed from the *to-be-sent* list and stored in the *to-be-ack* list. The packet is removed from the *to-be-ack* list when it is acknowledged. If the acknowledgment for the packet is not received before the time stamp expires, the packet is assumed to be dropped and placed in the *to-be-sent* list again for transmission.

Another transport layer protocol proposed for ad hoc networks, multiflow real-time transport protocol (MRTP) [5], is primarily used for real-time data transmission. This protocol uses multiple flows at the transport layer and multiple paths at the network layer.

Figure 2 describes the operation of the MRTP scheme. Packets are split over different flows for the transmission. Each packet carries the timestamp and the sequence identifier so it can be reassembled at the receiver. The receiver also keeps track of the QoS parameters like the jitter, packet

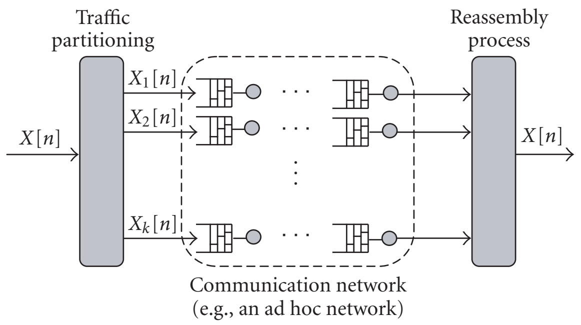

FIGURE 2: MRTP operation mechanism [5].

loss, and the highest packet sequence number received and sends this information to the sender in a receiver report (RR) packet, which can be sent through any flow. This helps the sender in maintaining the QoS parameters. The MRTP uses flow IDs for each flow, and either the sender or the receiver can delete a flow that is broken. The MRTP uses the underlying protocol multiflow real-time transport control protocol (MRTCP), which establishes multiple flows at the transport layer.

The MRTP tries to use multiple paths in the network. Although the protocol uses the multiple paths wisely, it cannot be used for non-real-time transmission, that is, for reliable data transmission, since it does not have any mechanism for packet retransmission and is mainly used for real-time data transmission.

In this paper, the authors propose using multiple TCP flows per connection. Each of these TCP flows can be routed through different paths and reassembled at the destination. In the next section, the authors explain the protocol in detail.

4. TCP MULTIFLOW

Traditionally, when using the TCP in a multipath environment, a single TCP connection is opened between two communicating nodes. Datagrams are sent from the TCP layer to the network layer, where the routing protocol decides the scheduling of packets over different available paths. As pointed out earlier, information about the number of paths available from the source to the destination is hidden from the TCP layer. In the proposed scheme, the authors suggest providing the information about the number of paths available between the source and the destination to the transport layer (TCP layer in this case). This will enable the TCP layer to decide upon the optimum number of connections required between the source and destination to transfer the given data. This scheme ensures optimum utilization of network resources and improves overall network performance.

4.1. *Protocol description*

Figure 2 represents a logical view of the traditional implementation, and Figure 3 represents the logical view of the proposed protocol. In a traditional implementation, when an application wants to communicate with a remote destination node, the transport layer establishes a single connection with the destination and allocates one transmitter/receive buffer along with a single congestion window. In a normal multipath environment, only the network layer will know the number of available paths. When load balancing is enabled, the network layer intelligently (using some sort of scheduling algorithm) forwards the packets belonging to the same connection through multiple paths. This helps to reduce the load on the best path and improve the network utilization; however, a single packet drop in any one path will result in the TCP going into the congestion-avoidance mode and dropping down the data transmission rate on all other stable paths. Depending upon the number of packet drops, the TCP will take a long time to restabilize. This severely affects the throughput of the application.

In the proposed protocol, when an application running on the source node requests the transport layer (in this case, TCP layer) to establish connection with a remote destination, the transport layer sends a message (similar to *ioctl ()* function with a request type of *SIOCGRTCONF*) to the network layer (interlayer messaging) requesting the number of paths to a particular destination. After receiving the request, the network layer looks for available routes for the given destination in the routing table. If it finds the routes in the routing table, it sends the number of routes available to the transport (TCP) layer as a reply (similar to *rtc_returned* in the case of an *ioctl ()* call). If there is no route in the routing table corresponding to the requested destination address, the network layer broadcasts a route request to the network (same as a normal route request). Once routes to the destination address are installed in the routing table, the network layer updates the transport (TCP) layer with the requested information. This route request by the network layer does not introduce an extra overhead to the network because the route request is just preponed. At this point, the TCP can set up connections according to the number of paths available (one connection per path). Data transfer between the source and destination is then divided into the number of flows (one for each connection), and one flow is assigned to each connection.

4.2. *TCP connection establishment*

For the purpose of this protocol, the authors divide the TCP layer into two parts (Figure 4). The first part, called the global connection manager (GCM), is responsible for communication with the upper layers, establishing connection with the remote destination, packet reordering, and packet scheduling. The second part, called the data transmission manager, consists of multiple TCP processes, which are child processes of the GCM. The data transmission manager is similar to the normal TCP layer and handles data delivery to the destination. Except for packet reordering, it performs all functions of a normal TCP layer.

When the TCP layer obtains the number or available routes to the destination, it initiates the three-way handshaking process by originating multiple SYN messages with

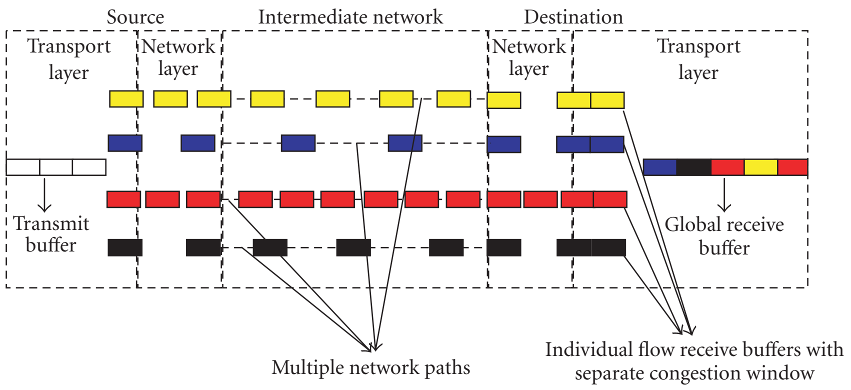

Figure 3: Logical representation of TCP-M.

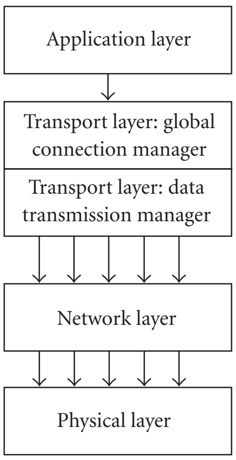
Application layer
Transport layer: global connection manager
Transport layer: data transmission manager
Network layer
Physical layer

Figure 4: Logical partitioning of the TCP layer.

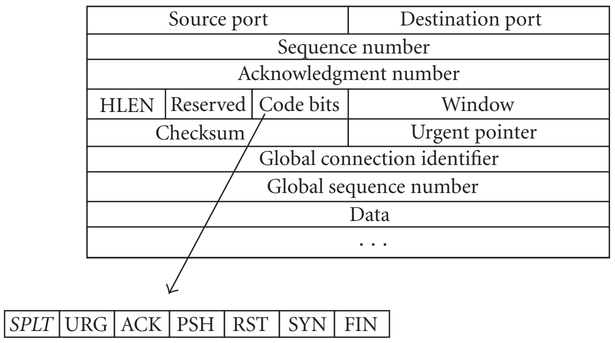

Figure 5: TCP header for the proposed protocol.

respect to the number of available paths. Each SYN message contains a different source port address but a single destination port address. This is consistent with the case where a single host opens multiple connections with the server. The destination node responds to the SYN messages like the normal TCP does, and connections between the source and destination nodes will be established. Each of these connections will have different connection identifiers (typically the TCP uses the IP address and port address, both source and destination, as the connection identifier). In addition to the normal connection identifier, the proposed protocol uses another connection identifier, that is, global connection identifier (4 bytes), which will be used by the destination node for reordering the packets.

4.3. *Data transfer process*

Once the connection is established between the source and the destination, the GCM starts acting as a scheduler. Based on feedback information (collected periodically from each individual connection), the GCM schedules the data on different connections. While transferring data to the child processes, the GCM also sends the original sequence number (4 bytes) of the datagram and a connection identifier (4 bytes) as arguments. These two pieces of information will be embedded in the options field of the TCP header and will help the receiver in reordering the packets.

When the child processes obtain the data, they form the TCP header similar to the normal TCP process. As mentioned earlier, they embed the original sequence number and the connection identifier information in the options field of the TCP header. In order to inform the destination node that the datagram is part of split connections, the proposed protocol suggests borrowing a bit from the reserved bits in the original TCP header. The borrowed bit will be called the *SPLT* bit. When the *SPLT* bit is set, it indicates to the receiver that the packet is a part of split connections. Figure 5 shows the TCP header information for the proposed protocol.

Each connection between the source and the destination will be associated with its own buffer space. This allows the TCP process to handle the flows independent of each other during congestion. If one path experiences congestion or failure, traffic corresponding to that path could be forwarded using other active paths without affecting other TCP flows. In addition, each of these flows maintains its own congestion window, independent of other flows.

When datagrams reach the network layer, based on the source and destination port pair, the network layer forwards

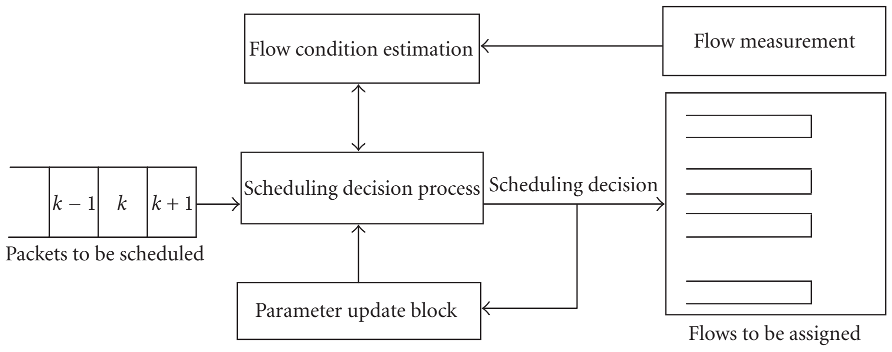

Figure 6: Block diagram of scheduling algorithm.

the packets through different paths. The authors assume that the network layer is enabled with load-balancing techniques to use different paths for different connections between the same source and destination pair. The discussion of implementation of multipath routing with load balancing is beyond the scope of this paper. While it is possible that two flows could be assigned to the same path, they will be treated as two different connections. On the other hand, two flows assigned to the same path might lead to unfair sharing of bandwidth, which also is beyond the scope of this paper. The proposed scheme works better in the presence of multiple paths between the source and the destination. The presence of disjoint paths (either node-disjoint or path-disjoint) to the destination is preferred in order to minimize the risk of performance degradation due to one link. In addition, this will also help in avoiding overloading and unfair sharing of a link or path.

While sending acknowledgments to packets, the receiver treats each split flow as a separate flow and sends acknowledgments accordingly. Acknowledgments are handled only by the data transmission layer, and the GCM will not have any control over this process. As mentioned earlier, the GCM is also responsible for reordering datagrams and presenting data to the higher layers. When the GCM obtains data from the child processes, it first checks for the *SPLT* flag. If the *SPLT* flag is set, it checks for the actual sequence number and the global connection identifier in the options field of the TCP header. Based on these two pieces of information, the GCM reorders the datagrams from different child processes and presents the data to the higher layers. As the sender gets individual acknowledgments for each split flow, it can keep track of the packet losses on the individual flows. If a packet is dropped, the sender can identify the flow in which the packet has been dropped and enter the congestion-avoidance mode only for that flow. In the case of a path failure, the sender can stop sending packets along that path and only use active paths until the old path is reestablished.

4.4. Packet-scheduling algorithm

TCP-M uses a packet-scheduling algorithm for scheduling the packets on different flows. The packet scheduler is a part of the GCM. It schedules packets based on current information about queue size, delay, and available capacity for each flow. The scheduler assumes that each flow is a single entity. In addition, it also assumes that information about a connection's queue size, queuing delay, and available capacity of each flow is provided to the scheduler. The scheduler then selects a flow in order to minimize the delay experienced by the packet.

Figure 6 shows the block diagram for the packet scheduler. In here, the flow measurement block keeps monitoring the status of each TCP flow setup by the sender for transmitting data. At regular intervals, the flow measurement block updates the flow condition estimation block about the status of different TCP connections. Typical update parameters include RTT for each connection, number of packets handled by each connection in that time duration, number of retransmissions during the time interval, and the current state of the TCP congestion window. The flow condition estimator block acquires these parameters and calculates packet share for each connection. It then updates the scheduling decision process with the new traffic share information.

The scheduling decision process is responsible for forwarding the packet towards destination using a specific connection. It uses the traffic share information provided by the flow condition estimation block in deciding the connection for each packet. Once the decision is made, the scheduling decision process marks the packets accordingly and sends it to the appropriate connection. In order to reduce the effect of packet reordering at the destination, the scheduler forwards consecutive packets through the same connection (as per the traffic share calculated by the flow condition estimator). The parameter update block updates the scheduling decision process block if the connection chosen at that instance runs out of buffer space. Scheduling decision process updates the flow condition estimation block with the same information. This helps the scheduler to avoid packet drops at the source and also helps the scheduler to reduce the overall delay.

4.5. Traffic share calculation

In the proposed scheduling algorithm, traffic share is calculated based on three different parameters, that is, instantaneous RTT, instantaneous packet drop probability, and current status of TCP congestion window. While considering the routing table metric for each path will also help in deciding the best path, the authors assume that any change in the routing metric will also be reflected in the TCP parameters like RTT and packet drop probability.

The flow condition estimation block obtains parameters like instantaneous RTT, number of packets transmitted by a particular connection, and number of retransmissions in the given time interval. Based on these parameters, the flow condition estimator first calculates the packet drop probability:

$$\text{Packet drop probability for path } i(p_i) = \frac{\text{Total number of retransmissions during the time period for path } i}{\text{Total number of packets transmitted through path } i \text{ during the time period}}. \tag{1}$$

Once the packet drop probability is found for each path, the flow condition estimation block calculates the geometric distribution of the ratio of packet drop probability of path i and minimum packet drop probability:

$$\text{Packet\_Drop\_Ratio\_sum} = \sum_{i=1}^{i=n} \frac{p_i}{p\_\min}, \tag{2}$$

where $p\_\min = \min(p_1, p_2, \ldots, p_n)$ is the minimum packet drop probability among all the paths and n is the total number of paths available between the source and destination.

Similar to packet drop probability, the flow condition estimator also calculates the geometric distribution of the ratio of RTT of path i and minimum RTT among all the paths (RTT_min):

$$\text{RTT\_Ratio\_sum} = \sum_{i=1}^{i=n} \frac{\text{RTT}_i}{\text{RTT\_min}}. \tag{3}$$

Now the traffic share for path i for time instance t $(TS(i,t))$ is calculated as

$$TS(i,t) = \begin{cases} \alpha * \left(\dfrac{\text{RTT\_Ratio}}{\text{RTT\_Ratio\_sum}}\right) + (1-\alpha) * \left(\dfrac{\text{Packet\_Drop\_Ratio}}{\text{Packet\_Drop\_Ratio\_sum}}\right) & \text{if CW is stable or increasing,} \\ \min\Bigg[\Bigg(\alpha * \left(\dfrac{\text{RTT\_Ratio}}{\text{RTT\_Ratio\_sum}}\right) & \\ \quad +(1-\alpha) * \left(\dfrac{\text{Packet\_Drop\_Ratio}}{\text{Packet\_Drop\_Ratio\_sum}}\right)\Bigg), \text{DataRate}_i\Bigg] & \text{if CW is decreasing due to congestion.} \end{cases} \tag{4}$$

Here α represents the weight to be assigned for the RTT ratio, RTT_Ratio represents the ratio of RTT for path i and the minimum RTT (RTT_min), and Packet_Drop_Ratio represents the ratio of packet drop probability for path i and the minimum packet drop probability $p\_\min$.

Now the overall traffic share for any path i (N_i) is calculated as

$$N_i = \sum_{t=1}^{t=z} TS(i,t) * N_t, \tag{5}$$

where z is the total number of time intervals and N_t is the total number of packets transmitted during the time interval t.

In the following sections, the authors discuss and analyze the performance of the proposed protocol and compare it with the TCP single-path and traditional multipath approaches.

5. PROTOCOL ANALYSIS

In this section, the authors analyze the proposed protocol performance with respect to delay and throughput. In addition, they compare the proposed protocol performance with that of the traditional single-path and multipath TCP.

Consider a sample network with m nodes in the network arranged in a random fashion. Let n be the average number of distinct paths between any two nodes. Let RTT_i be the round-trip time of the ith path and T_s the time period within which the TCP expects an ACK from the destination. Let p_i be the loss probability of the ith path. As described in [10], the TCP connection setup time can be estimated using the loss probability and RTT of the path as

$$t_{\text{Setup}} = \text{RTT} + 2T_s\left(\frac{1-p}{1-2p} - 1\right), \tag{6}$$

where

$$\text{RTT} = \min\left(\text{RTT}_1, \text{RTT}_2, \text{RTT}_3, \ldots \text{RTT}_n\right) \tag{7}$$

is the minimum RTT among all the available paths, and p is the loss probability of the path corresponding to the minimum RTT.

In an ideal situation, with no packet losses, the total time required to transfer N packets from source to destination depends on the maximum congestion window size, time required to reach the peak transmission rate, and number of packets itself. Equation (8) [10] represents the total time required to transmit N packets from source to destination. As can be observed from this equation, when the total number of packets to be transmitted is greater than the total number of packets that could be transmitted until the congestion window (*cwnd*) reaches the maximum congestion window size ($W_{\max}$), the time required to transmit all the packets also depends upon the maximum congestion window size.

$$T_{\text{nl}} = \begin{cases} \left[2\log_2\left(\dfrac{2N+4+3\sqrt{2}}{2\sqrt{2}+3(2)^{5/8}}\right)\right]\text{RTT} & \text{if } N \le N_{\exp}, \\[2ex] \left[n_{w\max} + \left[\dfrac{N-N_{\exp}}{W_{\max}}\right]\right]\text{RTT} & \text{otherwise,} \end{cases} \tag{8}$$

where $n_{w\max}$ is the number of rounds required by the TCP to reach congestion window of $W_{\max}$, and $N_{\exp}$ is the expected number of packets transmitted until the TCP congestion window reaches the maximum congestion window size $W_{\max}$,

$$n_{w\max} = \left[2\log_2\left(\frac{2W_{\max}}{1+\sqrt{2}}\right)\right],$$

$$N_{\exp} = \left[2^{(n_{w\max+1})/2} + 3(2)^{(4n_{w\max}-3)/8} - 2 - \frac{3\sqrt{2}}{2}\right] + W_{\max}. \tag{9}$$

Equation (10) represents the corresponding data transfer time when the TCP flow experiences a single packet loss. Here, $t_{\text{nl}}(i)$ represents the time required to transmit first y packets without any drops, $t_{\text{lin}}(a, b)$ represents the time required to transmit a packets during the congestion-avoidance mode with a congestion window size of b, and $E[TO]$ represents the timeout period experienced by the TCP flow. In the case of fast retransmit, TCP experiences timeout for congestion windows less than 4 [11]. TCP will recognize a packet drop only when it receives multiple duplicate acknowledgments (typically 3) [12]. When congestion window is less than 4, there cannot be 3 duplicate acknowledgments. The only way to identify a packet drop is due to timeout of an acknowledgment,

$$t_{\text{sl}} = \begin{cases} \left[t_{\text{nl}}(y) + E[To] + t_{\text{lin}}(N-k-1, 2) + 1\right]\text{RTT} & \text{if } y \le 6, \\ \left[t_{\text{nl}}(y) + 1 + t_{\text{lin}}(N-k, n) + 1\right]\text{RTT} & \text{if } n_{\max}(y) - y < 3 \\ \left[t_{\text{nl}}(y) + 1 + t_{\text{lin}}(N-k, n)\right]\text{RTT} & \text{otherwise} \end{cases} \quad \text{if } y > 6, \tag{10}$$

$$t_{\text{lin}}(a,b) = \begin{cases} \left[\dfrac{a - x(x+1) + b(b-1)}{x+1}\right] + 2x - 2(b-1) & \text{if } a \le W_{\max}(W_{\max}+1) - b(b-1), \\[2ex] \left[\dfrac{a - W_{\max}(W_{\max+1}) + b(b-1)}{W_{\max}}\right] + 2W_{\max} - 2(b-1) & \text{otherwise,} \end{cases} \tag{11}$$

$$E[TO] = TO\frac{2\left(1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6\right)}{1-p}.$$

In the event of multiple packet losses, the total data transfer time also depends upon the time interval between packet drops. Equation (12) [10] represents the delay involved in transmitting N datagrams across the network in the presence of multiple packet losses. Here, $t_{\text{sl}}(y)$ represents the time required to transmit the first y packets with a single loss, and $t_{\text{fr}}(l)$ represents the time delay involved in transmitting remaining packets in the congestion-avoidance mode. The point to be noted here is the fact that once the TCP enters the congestion-avoidance mode, the congestion window will be set to half of its previous value. Hence, the total time required will be much higher than the single packet loss case:

$$\begin{aligned} t_{\text{ml}}(N) = {} & E\{t_{\text{sl}}(m-1)\} + E\{(M-2)t_{to}(D_{\text{ave}})\} \\ & + E\{(M-2)t_{\text{fr}}(D_{\text{ave}})\}, \end{aligned} \tag{12}$$

$$t_{\text{fr}}(l) = \begin{cases} \left[2 + t_{\text{lin}}\left(D_{ave} - k, \left[\dfrac{h}{2}\right]\right)\right]\text{RTT} & \text{if } h - j < 3, \\[2ex] \left[1 + t_{\text{lin}}\left(D_{\text{ave}} - k, \left[\dfrac{h}{2}\right]\right)\right]\text{RTT} & \text{otherwise,} \end{cases}$$

$$t_{to}(l) = \left[E[TO] - I(j > 1) - t_{\text{lin}}(D_{\text{ave}} - h, 2)\right]\text{RTT}. \tag{13}$$

Here, M represents the number of loss occurrences while transmitting N packets and D_{ave} represents the average number of packets transmitted between two successive losses. Using (6), (8),(10), and (12), the total time required to transfer N packets across the network can be estimated as

$$\begin{aligned} T_{\text{transfer}}(N) &\\ = t_{\text{setup}} &+ (1-p)^N t_{\text{nl}}(N) \\ &+ p(1-p)^{N-1} E\{t_{\text{sl}}(N)\} + t_{\text{ml}}(N) + t_{\text{dack}}. \end{aligned} \tag{14}$$

Equations (8), (10),(12), and (14) were developed by the authors of [10] for analyzing TCP performance primarily in a single-path environment. However, this could be extended to analyze TCP performance in a multipath environment.

5.1. *TCP performance in multipath environment*

In a multipath environment, normal TCP implementations suffer performance degradations due to the fact that from the TCPs perspective, it is still a single path, that is, multipath routing is transparent to the TCP layer. Hence, a packet drop in one path affects the overall performance of the entire system.

5.1.1. *TCP connection setup time*

The following equation:

$$t_{\text{Setup}} = \text{RTT}_i + 2T_s\left(\frac{1-p_i}{1-2p_i} - 1\right) \tag{15}$$

represents the TCP connection setup time in the case of a multipath environment. As can be observed from this equation, the TCP setup time depends upon the RTT and packet drop probability of the path assigned by the network layer. This is similar to the normal single-path environment where the TCP connection is established using the best available path, that is, the path suggested by the network layer. For purposes of this analysis, the authors assume that the path suggested by the network layer has the lowest RTT. Then the total connection setup time is similar to the single-path routing scenario. The connection setup time also depends upon the packet drop probability p_i of the path selected for the connection setup process.

5.1.2. *Data transmission delay*

While the TCP setup time in the case of a multipath routing environment is similar to the single-path routing environment, the data transfer delay varies significantly. Equation (16) represents the corresponding total data transfer time for N packets. In this case, p_i represents the maximum packet drop probability among all the paths that are used to transmit the data. As one of the paths being used becomes congested/unavailable, it adversely affects the performance of the entire TCP flow, irrespective of the performance demonstrated by other paths. In the case of ad hoc networks, path unavailability leads to another route discovery process, which further deteriorates ad hoc network performance:

$$\begin{aligned} T_{\text{transfer}}(N) = t_{\text{setup}} &+ (1-p_i)^N t_{\text{nl}}(N) \\ &+ p_i(1-p_i)^{N-1} E\{t_{\text{sl}}(N)\} \\ &+ t_{\text{ml}}(N) + t_{\text{dack}} + t_{\text{recorder}}, \end{aligned} \tag{16}$$

$$p_i = \max(p_1, p_2, p_3, \ldots, p_n). \tag{17}$$

Compared to the single-path data transfer, multipath data transfer also induces additional delay in terms of packet reordering. In a multipath environment with load balancing capabilities, it is highly likely that packets travel through various paths to reach a destination. Depending on the amount of delay experienced by the path, the packets may reach the destination out of order. One of the functionalities of the TCP layer is to check for packet sequence numbers and arrange them in the proper order so that data can be presented to higher layers. Packet reordering requires that datagrams wait in the queue for some time before all the packets (of a particular sequence) arrive at the destination. This delay is referred to as packet reordering delay (t_{reorder}).

Another important aspect to note here is the fact that in a multipath environment, the RTT is calculated as the average RTT of all available paths. This is because datagrams from source to destination could get routed through one path while the acknowledgments could take a different path. Hence, the effective RTT is the average RTT of the forward path and the reverse path:

$$\text{RTT} = \frac{\sum_{i=1}^{n} \text{RTT}_i}{n}. \tag{18}$$

5.2. *TCP performance in proposed scheme*

5.2.1. *TCP connection setup time*

In the case of the proposed scheme, multiple connections will be set up between source and destination. The setup process will be complete only after the connection using the slowest path is complete. Hence, the total time involved in setting up the connection can be expressed as

$$t_{\text{setup}} = \max\left(\text{RTT}_i + 2T_s\left(\frac{1-p_i}{1-2p_i} - 1\right)\right), \tag{19}$$

where RTT_i represents the round-trip time of the path chosen for ith connection, and p_i represents the corresponding packet drop probability. Compared to the normal multipath routing scheme, the proposed scheme takes longer time in setting up TCP connections.

5.2.2. *Data transmission delay*

Data transmission delay is one of the major areas where the proposed scheme gains over the traditional multipath scheme. Contrary to the traditional multipath scheme, the proposed scheme establishes multiple connections at the transport layer itself. This is done in cooperation with the network layer, with the understanding that the network layer forwards the packets belonging to different connections through different paths. The amount of traffic to be forwarded through an individual path is decided based on the RTT, packet drop statistics, and queue status corresponding to that path at that instance. Since the scheduler changes the traffic share through individual paths adaptively, the effect of one path failure will be minimal on overall TCP performance. Equation (20) represents the expected time delay in transmitting N packets from source to destination:

$$\begin{aligned} T_{\text{transfer}}(N) = {} & t_{\text{setup}} + t_{\text{dack}} + t_{\text{reorder}} \\ & + \sum_{i=1}^{n} \Big((1-p_i)^{N_i} t_{\text{nl}}(N_i) \\ & \qquad + p_i(1-p_i)^{N_i-1} E\{t(N_i)\} + t_{\text{ml}}(N_i) \Big). \end{aligned} \tag{20}$$

Here, N_i represents the number of packets transmitted through path i, and p_i represents the packet drop probability of that path. Since these connections are set up independent of each other, a packet drop in one path will not affect flow on the other. When a packet is dropped, the TCP enters the congestion-avoidance mode only for that flow. This reduces the overall data rate by a small fraction, as compared to traditional multipath routing where a single packet drop results in a large data rate reduction (almost 50%). However, the packet drop in one path may result in more packets being forwarded through the other paths, resulting in higher resource utilization in that path. However, the effect of such rerouting is minimal on overall TCP performance, as compared to TCP behavior in a traditional multipath environment. While the proposed protocol requires the TCP to reorder the packets (out-of-order delivery of packets is common in a multipath environment), the authors argue that the delay associated with packet reordering will have lesser impact on overall TCP performance, as compared to reduction of the congestion window. Also, the effect of out-of-order delivery in the proposed protocol is not similar to traditional TCP as the connection is managed by individual flows where packets are delivered in order. Reordering has to be done only when data has to be delivered to the upper layers.

While the proposed scheme improves overall TCP performance, it is associated with certain overhead such as additional memory, reordering delay at the receiver, and flow-splitting delay at the transmitter. In order for the TCP to handle each of the flows separately, a larger buffer space is required. Each flow must be associated with a separate buffer space that is similar to the original TCP connection. Hence, in the proposed scheme, the memory requirement can be calculated as

$$\text{Memory} = n * M_{\text{TCP}}, \tag{21}$$

where n is the number of connections established, and M_{TCP} is the memory required by the original TCP connection. With recent advances in the portable communication devices, the authors assume that allocation of additional memory should not be a great concern. Also, based on available system resources, it is possible to restrict the number of paths to be used in the proposed protocol. Optimizing the buffer size would also improve the performance at a lower cost.

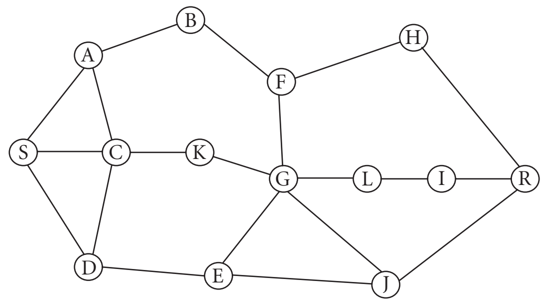

Figure 7: Sample ad hoc network with random topology.

While the protocol is associated with certain additional delays, it is assumed that these delays are very small and can be ignored, considering the performance benefits provided by the proposed protocol. Also, these delays will be heavily dependent on implementation.

Example 1. Consider an ad hoc network with nodes placed in a random manner (Figure 7). Let S be the sender node having infinite number of packets to send, and R the receiver node. Assume that all nodes are using the 802.11b channel access mechanism and can transmit data at the rate of 11 Mbps. From Figure 7, it can be seen that there are three distinct paths (S-A-B-F-H-R), (S-C-K-G-L-I-R), and (S-D-E-J-R), with different costs between the source and destination.

In the current scenario, assume that the optimal path (S-D-E-J-R) between S and R is over utilized and can offer only 4 Mbps of bandwidth for the TCP flow under test. If a packet size of 50 Kb is assumed, then the optimal path can carry nearly 80 packets per second, which can be considered as the maximum throughput attainable for the data transfer. However, if a packet loss occurs in the network, then the congestion window size is reduced to half. This, in turn, results in a reduction of the data transfer rate to half of its original value, that is, 40 packets per second. Subsequent packet drops further reduce the data rate. Hence, the average throughput achieved by the data transfer will be much less than the maximum throughput, that is, 80 packets per second and can be approximated to be around 40 packets per second.

An interesting point to note here is that even if multiple paths are considered between the source and the destination, the performance may not improve. Since only a single flow is used, a packet drop in any path will trigger a reduction in the data rate, thereby reducing throughput.

Now consider the proposed scheme. Assume the existence of three distinct paths between S and R, which is a fair probability in a large network. Let one of the paths be the optimum path, as in the last case, with a capacity of 80 packets per second, and let the other two have lower capacity, that is, 40 packets per second and 60 packets per second. The proposed protocol establishes three flows, each using one of the three available paths. Therefore, the maximum throughput of the data transfer is the cumulative capacity of all the paths, that is, 180 packets per second. This will be the upper limit for the throughput. Compared to the previous case, the throughput is much higher. If congestion occurs in the primary path, the data transfer rate in that path alone is reduced to half or even less. However, the overall throughput will still be more than 100 packets per second, which is higher than the maximum throughput attainable in the previous case. If all paths are considered to have congestion and their data rate is reduced to half, then the overall throughput of the proposed scheme is at 90 packets per second, which is higher than the throughput of the original scheme. The probability of all paths failing at the same time is considered to be less, especially if the paths are distinct. Therefore, if one of the paths fails, then it will not influence the congestion window of other paths.

6. RESULTS AND DISCUSSION

In this section, the authors compare the performance of the proposed protocol with that of normal single path and multipath TCP. The simulations were run using MatLab 7.0. Following values were considered for various parameters:

(i) RTT {40, 30, 20, 15, 45, 70};
(ii) packet drop probability {0.05, 0.018, 0.023, 0.25, 0.097, 0.086};
(iii) timeout value = 250 ms;
(iv) $T_S = 150$ ms;
(v) $H = 10$, $J = 4$.

Figure 8 presents the total data transfer time for all three approaches under different load conditions. From the figure, it can be inferred that the proposed protocol outperforms both single-path and multipath TCP in the case of higher packet drop probability scenarios. However, at low packet drop probability scenarios single-path approach works better. Figure 8(a) represents the total data delivery delay variation with respect to different data size when all paths have equal packet drop probability (0.05) and equal round-trip time (40 ms). Figure 8(b) also represents the same results as Figure 8(a), but with emphasis on the performance variation of single-path and normal multipath environment. From these two graphs, it is evident that when all paths have similar characteristics, TCP based on single path outperforms others. This is due to the fact that with single path, most of the packets get delivered in order and packet realignment time is very negligible. However, in the proposed scheme, delay due to packet reordering and delay due to packet scheduling play an important role and hence affect the overall performance compared to single-path and traditional multipath approaches.

Figures 8(c), 8(d), and 8(e) present the results corresponding to different packet drop probabilities. Figures 8(f), 8(g), and 8(h) present the total packet transfer delay for a packet drop probability of 0.15, round-trip time of 40 ms, and different window size. All the results presented confirm the fact that when all paths have similar characteristics, TCP single-path approach provides the best results. At the same time, it can also be observed from the results that as the packet drop probability increases, the proposed approach starts performing better compared to the traditional multipath approach. This is because each packet drop (in any path) results the traditional multipath TCP to get into congestion-avoidance mode. On the other hand, in the case of proposed approach, packet drop in one path affects only the performance of the TCP connection through that path. Also the scheduling algorithm will consider the affect of congestion while deciding the traffic share that needs to be sent via any path thereby reducing the overall effect of packet drops.

Figure 9 presents the total data transfer time in a normal situation (with different packet drop probabilities and different round-trip times) under different load conditions. The results presented in Figure 9 confirm the previous trend, that is, under low packet drop probability, TCP single-path approach provide better results compared to other two approaches. However, as the number of packets to be transferred increases and packet drop probability of the best path (lowest RTT path) increases, the proposed approach outperforms both TCP single-path and traditional multipath approaches. This is due to the fact that, in the proposed approach traffic is shared between all available paths based on their respective RTT (instantaneous), packet drop probability (instantaneous), and current status of the TCP connection. This ensures that the effect of packet drop in one path does not affect the overall data transfer process to a larger extent.

Figure 10 lists the proposed approach performance against TCP single-path and traditional multipath approaches under varying packet drop probability conditions. Total number of packets to be transferred was kept constant at 5000 packets and TCP window size was fixed at 16. The results indicate that as the packet drop probability of the best path (lowest RTT) increases, the TCP single-path and traditional multipath approaches suffer performance degradation to a larger extent compared to the proposed approach. This is because, as stated earlier, the scheduling algorithm in the proposed approach detects the congestion in one path and directs traffic through other paths. Even compared to the traditional multipath approach, TCP single path performs better due to the fact that after successive packet drops, the TCP enters congestion-avoidance mode quickly and enters the slow start mode. However in the case of multipath, before the TCP process realizes the packet drop, it would have already transmitted several packets which results in retransmission. Figures 10(e) and 10(f) present the performance of all the three approaches when the packet drop probability of a secondary path is varied. Again, similar to previous results, at low packet drop probability, TCP single-path approach performs better and as the packet drop probability increases, the proposed approach performs better than the other two.

While the above results indicate that the TCP single-path approach is better in the case of low packet drop conditions, it is not applicable in all situations. The above simulation results do not consider the resource availability along the path. Also, in the case of ad hoc networks, a route failure in the best path may result in high amount of delay for the data

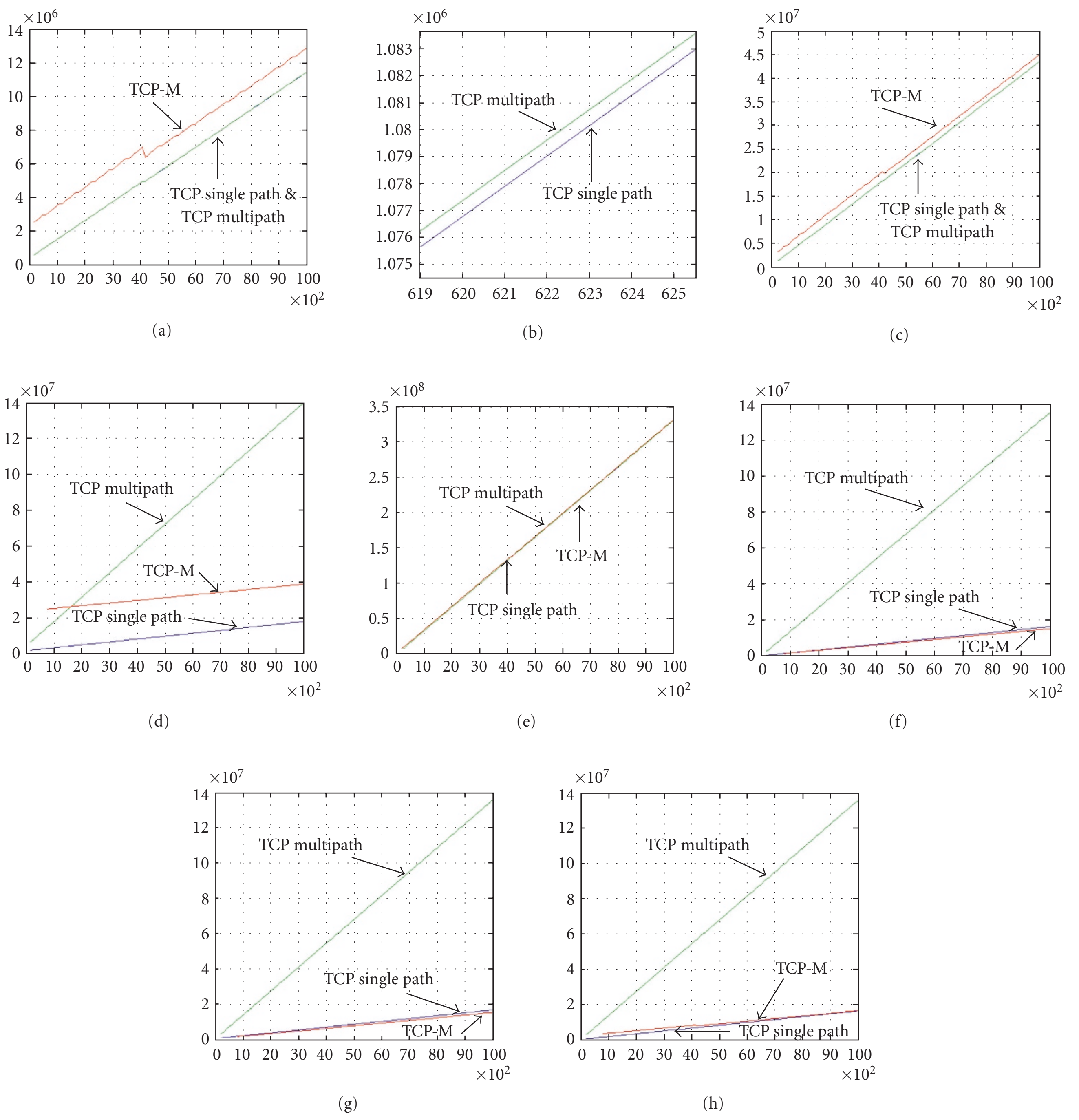

Figure 8: Total data transmission delay with respect to different traffic loads with all paths having same characteristics. (a) & (b) Packet drop probability = 0.05, TCP window = 16, RTT = 40 ms. (c) Packet drop probability = 0.1, TCP window = 16, RTT = 40 ms. (d) Packet drop probability = 0.01, TCP window = 16, RTT = 40 ms. (e) Packet drop probability = 0.3, TCP window = 16, RTT = 40 ms. (f) Packet drop probability = 0.15, TCP window = 16, RTT = 40 ms. (g) Packet drop probability = 0.15, TCP window = 32, RTT = 40 ms. (h) Packet drop probability = 0.15, TCP window = 64, RTT = 40 ms.

transmission as the source needs to discover the alternate path and reestablish the connection. Also, this kind of approach would result in underutilization of some links in the network. Compared to that, the proposed approach provides a balanced performance under all conditions. Under low packet drop scenarios, its performance is comparable to the TCP single-path approach and is better than the traditional multipath approach. Under highly unreliable network scenario, the proposed approach outperforms both TCP single-path and traditional multipath approaches.

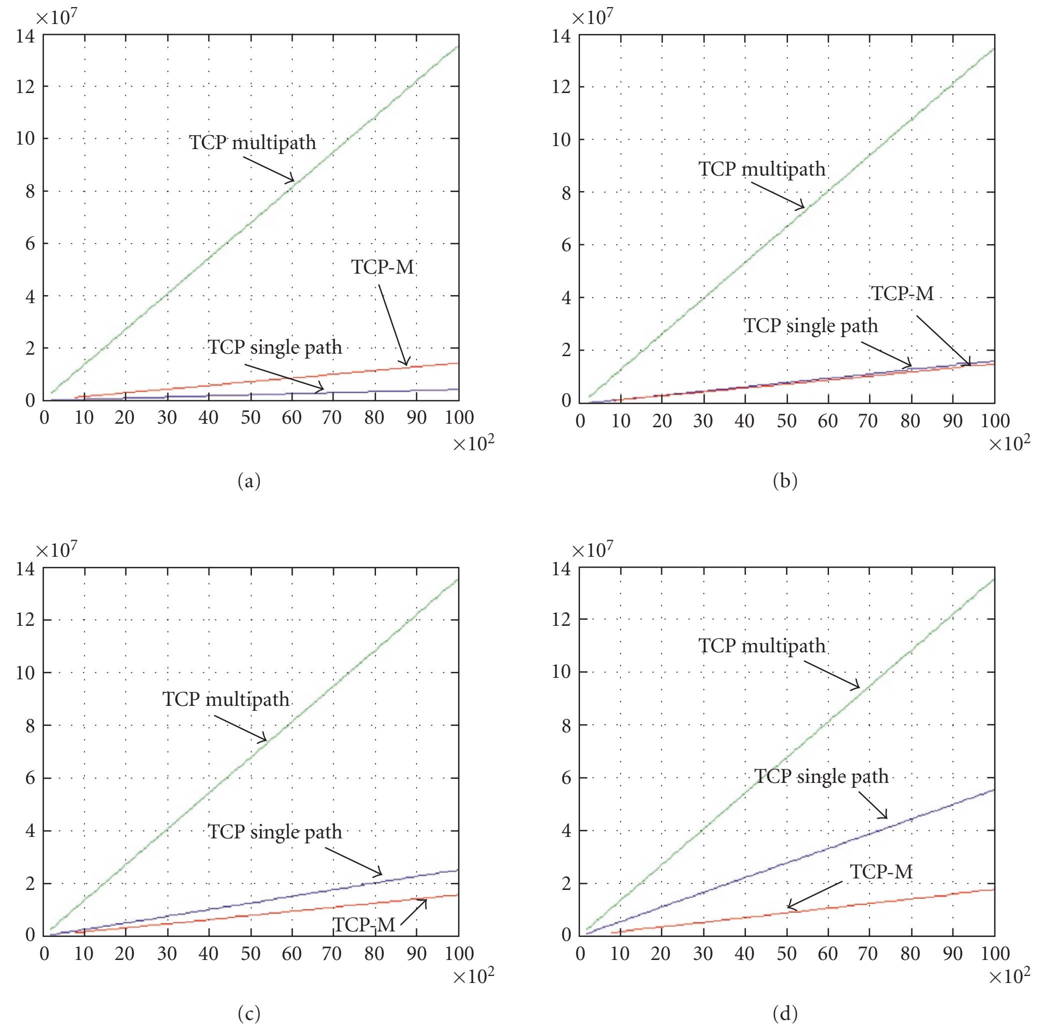

FIGURE 9: Total data transmission delay with respect to different traffic loads under general conditions. (a) Packet drop probability = 0.05, TCP window = 16, RTT = 40 ms. (b) Packet drop probability = 0.15, TCP window = 16, RTT = 40 ms. (c) Packet drop probability = 0.20, TCP window = 16, RTT = 40 ms. (d) Packet drop probability = 0.30, TCP window = 16, RTT = 40 ms.

7. CONCLUSION

In this paper, the authors have analyzed various parameters that affect the performance of TCP in an ad hoc network environment. Congestion and path nonavailability are two major factors that affect TCP performance. It was also observed that, in the presence of multiple paths, TCP performance degrades when one of the paths used for forwarding data drops a packet. In the current paper, the authors have proposed establishing multiple connections for every data transfer between the source and the destination. The proposed mechanism would be transparent to the application and session layers; however, it involves the transport layer in multipath routing scheme.

The analysis carried out by the authors indicates a significant improvement in overall performance. While the performance of the proposed protocol is slightly inferior to the standard TCP or TCP in the presence of multiple paths when the network is stable, it proves beneficial in the presence of network congestion and packet losses. This analysis also indicates that a packet drop in one of the paths does not affect the overall performance of TCP flow in a larger scheme. Even though the protocol has some additional costs in terms of memory and delay, the authors argue that performance benefits associated with the protocol overshadow costs. The authors also note that the protocol performs better in the presence of multiple paths between the source and the destination, and that the paths are disjoint.

One of the disadvantages of the proposed protocol lies in its memory requirements, whereby each data transfer requires memory in multiples of the number of connections established. The authors assume that the memory allocation will not cause any issue as, at system level, the proposed approach is similar to having multiple simultaneous TCP sessions with the same destination. However, at the server end, this could cause performance issues if the TCP session lasts

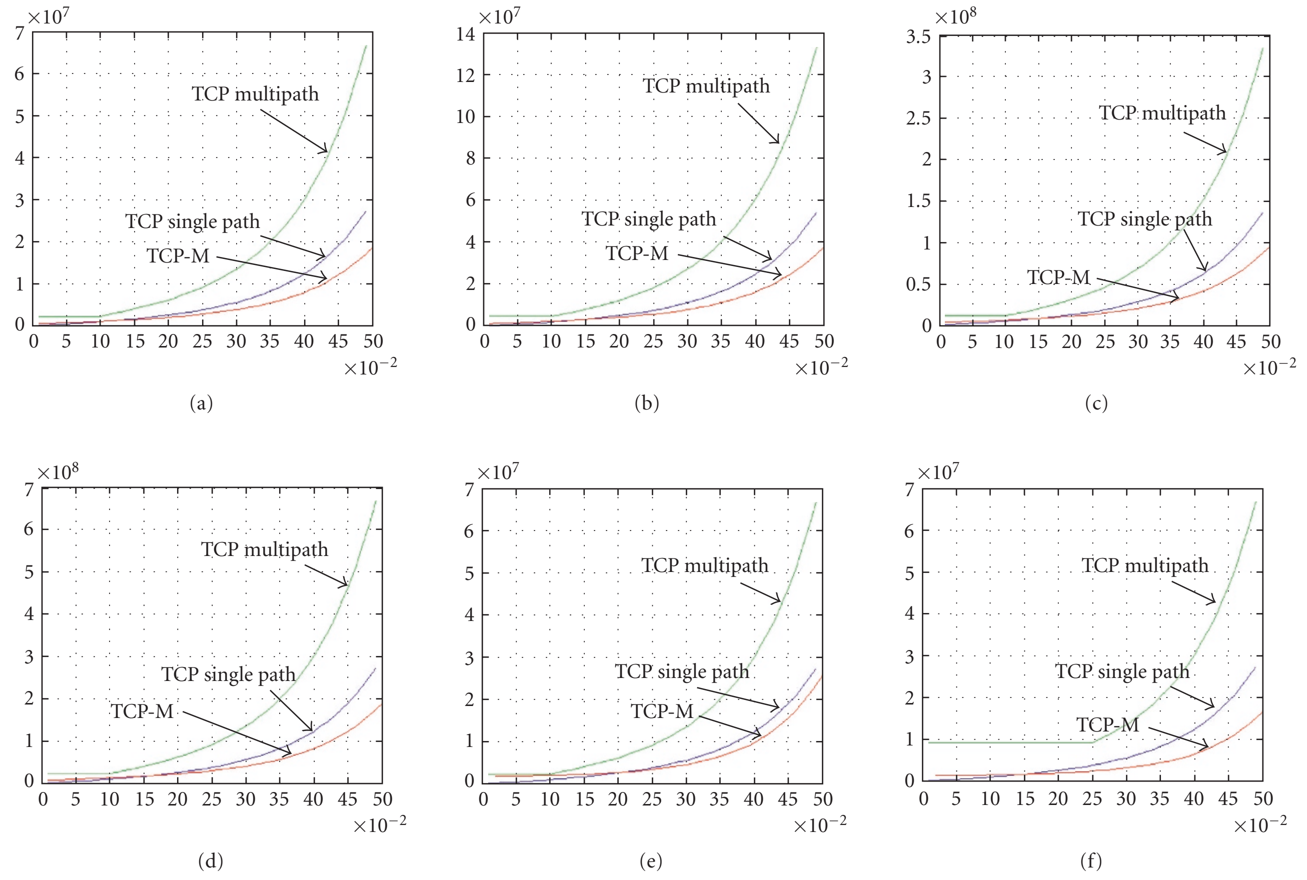

FIGURE 10: Total data transmission delay with respect to different packet drop probabilities under general conditions. (a) TCP window = 16, RTT (best path) = 40 ms. (b) TCP window = 16, RTT (best path) = 40 ms. (c) TCP window = 16, RTT (best path) = 40 ms. (d) TCP window = 16, RTT (best path) = 40 ms. (e) Packet drop probability (best path) = 0.15, TCP window = 16, RTT (best path) = 40 ms. (f) Packet drop probability (best path) = 0.25, TCP window = 16, RTT (best path) = 40 ms.

longer and the memory allocated for one session is not freed at due time. The fine tuning of the memory allocation policy is beyond the scope of this article. Another area where the proposed protocol introduces overhead compared to TCP single-path and traditional multipath approaches is control data. Compared to TCP single path and traditional multipath, the proposed approach will generate higher number of connection establishment/connection maintenance packets. But the number of packets generated during the data transfer still remains the same. As a matter of fact, the authors argue that the additional control packets generated in the proposed approach is negligible compared to the amount of retransmissions saved by the proposed approach.

In the proposed scheme the authors assumed that the delays at the source and destination sides due to modifications would be negligible, but it would be interesting to study these effects at the system level. Also, the proposed scheme requires modifications to the existing TCP implementation.

REFERENCES

[1] B. Atkin and K. P. Birman, "Evaluation of an adaptive transport protocol," in *Proceedings of 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '03)*, vol. 3, pp. 2323–2333, San Francisco, Calif, USA, March-April 2003.

[2] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A feedback based scheme to improving TCP performance in ad-hoc wireless networks," in *Proceedings of 18th International Conference on Distributed Computing Systems*, pp. 472–479, Amsterdam, The Netherlands, May 1998.

[3] T. D. Dyer and R. V. Bopanna, "A comparison of TCP performance over three routing protocols for mobile ad hoc networks," in *Proceedings of the 2nd International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, pp. 56–66, Long Beach, Calif, USA, October 2001.

[4] H. Lim, K. Xu, and M. Gerla, "TCP performance over multipath routing in mobile ad hoc networks," in *Proceedings of IEEE International Conference on Communications (ICC '03)*, vol. 2, pp. 1064–1068, Anchorage, Alaska, USA, May 2003.

[5] S. Mao, D. Bushmitch, S. Narayanan, and S. S. Panwar, "MRTP: a multi-flow real-time transport protocol for ad hoc networks," in *Proceedings of the IEEE Semiannual Vehicular Technology Conference (VTC '03)*, Orlando, Fla, USA, October 2003.

[6] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar, "ATP: a reliable transport protocol for ad-hoc networks," in *Proceedings of ACM International Symposium on Mobile Ad*

Hoc Networking and Computing (MOBIHOC '03), Annapolis, Md, USA, June 2003.

[7] M. K. Marina and S. R. Das, "On-demand multipath distance vector routing in ad hoc networks," in *Proceedings of the International Conference for Network Protocols (ICNP '01)*, pp. 14–23, Riverside, Calif, USA, November 2001.

[8] S.-J. Lee and M. Gerla, "SMR: split multipath routing with maximally disjoint paths in ad hoc networks," Tech. Rep., Computer Science Department, University of California, Los Angeles, Calif, USA, August 2000.

[9] S. Bohacek, J. P. Hespanha, J. Lee, C Lim, and K. Obraczka, "TCP-PR: TCP for persistent packet reordering," in *Proceedings of 23rd International Conference on Distributed Computing Systems*, pp. 222–231, Providence, RI, USA, May 2003.

[10] B. Sikdar, S. Kalyanaraman, and K. S. Vastola, "Analytic models and comparative study of the latency and steady-state throughput of TCP Tahoe, Reno and SACK," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '01)*, San Antonio, Tex, USA, November 2001.

[11] R. Morris, "Scalable TCP congestion control," in *Proceedings of the 9th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 3, pp. 1176–1183, Tel-Aviv, Israel, March 2000.

[12] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC2001, IETF, 1997, http://rfc.sunsite.dk/rfc/rfc2001.html.

Nagaraja Thanthry is a Ph.D. student in the Department of Electrical and Computer Engineering, Wichita State University. He has received his B.S. degree in electronics and communication from Mysore University, India, in 1997, and M.S. degree in electrical engineering from Wichita State University, in 2002. He is a student member of IEEE. His research interests include ad hoc networks, multiservice over IP, aviation data networks, quality of service, and security.

Anand Kalamkar has completed his M.S. degree in electrical engineering at Wichita State University in 2004. He received his B.S. degree in electrical engineering from Nagpur University, India, in 2001. He is currently working with AT&T as a Data Support Engineer. His research interests include ad hoc networks and quality of service.

Ravi Pendse is an Associate Vice President for Academic Affairs and Research, Cisco Fellow, and Director of Advanced Networking Research Center at Wichita State University. He has received his B.S. degree in electronics and communication engineering from Osmania University, India, in 1982, M.S. degree in electrical engineering from Wichita State University, in 1985, and Ph.D. degree in electrical engineering from Wichita State University, in 1994. He is a Senior Member of IEEE. His research interests include ad hoc networks, multiservice over IP, and aviation security.