Dynamic TCP Proxies: Coping with Disadvantaged Hosts in MANETs

Tu Ouyang
Case Western Reserve University
Cleveland, Ohio 44106
Email: tu.ouyang@case.edu

Shudong Jin
Case Western Reserve University
Cleveland, Ohio 44106
Email: jins@case.edu

Michael Rabinovich
Case Western Reserve University
Cleveland, Ohio 44106
Email: misha@eecs.case.edu

Abstract

Applications in mobile ad-hoc networks can suffer from poor link quality and degraded network services. In particular, standard TCP over low-quality, long routing paths, can have disproportionally low throughput as the result of a double penalty: long end-to-end round-trip time—due to long path length, and frequent congestion window backoff due to packet losses that could not be masked by the link-layer retransmissions. To address this problem, we leverage a well-known concept of dynamic TCP proxies but propose to dynamically *install these proxies along a routing path based on the quality of the links comprising the path. By enclosing poor quality path segments between a pair of TCP proxies, we isolate the effect of these segments on the rest of the network and improve the overall performance. We further utilize cross-layer optimization to select proxy locations without a separate signaling protocol, by leveraging existing messages in the underlying routing protocol. We demonstrate the potential benefits of this dynamic TCP proxies mechanism, and illustrate the relevance of host mobility level on this design choice.*

1. Introduction

A mobile ad-hoc networks (MANET) is a self-configuring network of mobile hosts connected by wireless links. The network topology may change rapidly and unpredictably, and the network connectivity may exhibit heterogeneity at different parts of the network. Often in a network there are *disadvantaged* hosts (and the associated users) who experience poor network services due to various reasons, such as communication interference from external sources, congestion along prevalent routing paths, or bottlenecks at upstream transit nodes. The negative effect of disadvantaged hosts in MANETs extends far beyond their associated users. Indeed, not only do these hosts have limited capacity to relay packets for other hosts, but the resulting packet losses cause end-to-end retransmissions by upper-layer protocols, increasing the wireless channel contention along the entire paths and degrading the entire network performance.

This work is supported by Lockheed Martin Corp.

To address these issues, we rely upon the following observation. It is well known that standard TCP (Reno version) throughput is inversely proportional to the end-to-end delay [18], and in MANETs, the throughput is directly affected by the hop-distance [13]. However, disadvantaged hosts that are far from the data sources will incur a double penalty for both the long hop-distance, and the long delay and packet loss at the bottleneck along the path. Our proposed solution then builds upon a concept of TCP proxies, which has long been used in a variety of contexts to improve end-to-end TCP performance. A TCP proxy is a transit node that acts as a TCP end-point to either side of the communication path; the resulting *split TCP* connection can have higher overall throughput by reducing the round-trip time on each side. In particular, Kopparty et al. [15] attempted to improve the throughput of end-to-end TCP connections in MANETs by utilizing TCP proxies along the routing path. They position a TCP proxy every three nodes along the routing path, so that a long path is divided into a sequence of short segments of TCP connections.

However, we recognize that the benefits of TCP proxies in a realistic MANET depend crucially on *where* on the path the proxies are positioned. Indeed, by strategically placing TCP proxies along the path, we can encompass a trouble area with TCP proxies at both sides, isolating the rest of the path from its adverse effects. Moreover, we can obtain similar performance benefits with fewer proxies. This is important for several reasons. First, processing transit packets at the higher layer by proxies involves performance overhead (although we did not quantify this overhead in our preliminary experiments). Second, the set of TCP proxies creates an overlay path that remains pinned for the duration of the connection. As nodes change their locations due to mobility, this overlay path may become highly suboptimal. We show experimentally that an excessive number of proxies can lead to significant performance degradation in high mobility scenarios. While this observation suggests that it might be beneficial to *migrate* proxies of an ongoing connection (see Section 3.2), it is also clear that one should place TCP proxies judicially.

Further, the beneficial positions for TCP proxies on the path depend on the network topology and conditions, which change in a MANET in time with node movements. Thus,

we provide a mechanism and heuristics for any transit node to be designated as a TCP proxy *dynamically* based on previously observed network performance in its vicinity. For instance, it might be desirable to have a TCP proxy at each side of an area with low-quality or congested communication links. In addition to reducing the end-to-end round trip time and hence improving the effective throughput, these proxies will relieve the rest of the path from having to carry end-to-end retransmissions. This will reduce the wireless channel contention in those parts of the network and improve the overall throughput of the network for concurrent communication between other hosts.

Finally, a key problem is proxy selection without adding extra signaling overhead. In our design, the source of an end-to-end connection selects proxies at the TCP connection establishment time. To do that, the source collects the information about network conditions through a cross-layer optimization mechanism, by utilizing the signaling messages of the underlying routing protocol. For instance, in the dynamic source routing (DSR) protocol [14], which is one of the most widely adopted routing protocols for MANETs [5], we extend the route discovery messages to piggyback link quality information.

We describe the implementation of the proposed mechanisms and present preliminary results from our ns-2 simulation experiments. The results demonstrate the potential performance gain by using our dynamic TCP proxies approach, as well as the impact of high node mobility. This paper represents our preliminary work in this direction by motivating the idea and showing its promise and challenges. We outline various directions for future work throughout the paper.

2. Dynamic TCP Proxies

In designing our dynamic TCP (DTCP) proxy mechanisms, we take the approach of cross-layer optimization. We obtain information on link quality from the data link layer, and propagate this information using extended route discovery messages at the network layer. We then use this information at the transport layer to improve end-to-end performance.

2.1. Link Layer Extensions

We infer a disadvantaged link by two link-layer characteristics at each node on the path: the link-layer transmission queue size and the MAC layer retransmission count. The former reflects the load on the transit node, which affects the routing delay; the latter indicates the congestion and interference on the link.

Our mechanism extends the link layer (and MAC sublayer) to calculate, using the exponentially weighted moving average method, the smoothed estimates of two metrics: the transmission queue length and retransmission count. The smoothed estimates are calculated in a standard way, similar to the approach used in TCP for round-trip time estimation. Let AVG_i be the smoothed estimate before the transmission of the i-th packet, and S_i be the instantaneous sample value of the metric at the time of the transmission of the i-th packet. We set

$$AVG_{i+1} \quad = \quad \alpha * AVG_i + (1 - \alpha) * S_i, \qquad (1)$$

where the coefficient α determines the rate of the exponential decay; we use α =0.85 in our experiments.

2.2. Network Layer Extension

We extend the DSR protocol to collect the above link quality metrics without extra signaling messages. DSR is a reactive routing protocol, where communication between a pair of nodes is preceded by a route discovery phase. When a node wishes to communicate to another node, it broadcasts a Route Request (RREQ) packet into the network. As RREQ packets flood through network, they record the path they have taken. Once a RREQ reaches a node that is either the destination or known the path from itself to the destination, this node will send back to the source a route reply packet (RREP) containing the whole path. The source will cache the route from the first received RREP message and use this route for subsequent communication.

To enable the collection of link quality information, we extend DSR as follows. First, we make every node append its smoothed estimates of the transmission queue length and MAC layer retransmission count, along with its IP address, to every RREQ message it processes. Thus, RREQ messages accumulate the link quality information of the paths they discover. Second, we modify the DSR route cache maintained by each node so that whenever a node is present in the DSR route cache, its associated link-layer quality metrics are also maintained in the cache. This allows a node to include the per-hop link quality information of the full path any time it constructs a RREP response, even if it completes the path from its cache. Third, we extend the interface between the network and transport layers to let the transport layer obtain from the network layer the per-hop link quality information of the discovered route to the destination. This allows the TCP layer at the source to select which nodes on the path should become TCP proxies as described in the next section. Forth, we insert a check in the network layer to identify TCP SYN packets and to further check an optional TCP header in these packets to see if the current node should become a TCP proxy (see Section 2.3). If this is the case, the network layer performs a special upcall to the TCP layer and passes the packet to the TCP layer. Fifth, our design calls for an API to install a packet filter in the network layer to capture subsequent TCP packets for which the current node acts as a proxy and to

pass these packets up to the TCP layer. This API would be used by the TCP layer when a transit node becomes a TCP proxy.

2.3. TCP Proxy Selection and Connection Management

When the TCP source needs to connect to the destination, it decides which intermediate nodes on the path to the destination should act as TCP proxies. To this end, the TCP layer first queries the network layer for the route information, which would be returned either from the DSR route cache or as the result of the route discovery protocol. The algorithm for selecting the proxies uses a combination of (the smoothed estimates of) both the queue size and retransmission count of a node. We currently use a simple sum of the two metrics as a combined metric, although other choices can also be explored. We call it the "proxy-selection metric". The higher the value of the proxy selection metric of a host the more disadvantaged this host is. The source host executes the following procedure:

1) Calculate the difference of the proxy-selection metric between any two adjacent nodes. If the difference exceeds a threshold, the node with the smaller metric is selected to become a proxy. The intuition behind this selection is that a jump in the proxy selection metric indicates a boundary of a disadvantaged area, and then the node with the smaller value will be last node *outside* the trouble spot. For example, in Figure 1, the nodes 3 and 9 will have much lower values of their proxy selection metrics than, respectively, neighboring nodes 4 and 8, indicating the boundary of the disadvantaged area. At the end of this step we have a set of proxies. The size of this set can vary from 0 to $N/2$, where N is the path length. The threshold value is an implementation parameter. (In our experimental evaluation, we set it to 20). This design choice is based on the observations that if a node's neighbor has a dramatically higher proxy-selection metric, this node can be a good proxy to reduce the impact of the bottleneck (i.e., the neighbor).
2) Check the entire path to see if there exist a continuous path section between two proxies (or source/destination) comprising three or more nodes without a proxy in-between. If so, select the middle node as a proxy. Repeat this process until no such path exists (thus, there will be a proxy at most every three nodes). We chose this heuristic to allow apple-to-apple comparison with [15]. However, from our own experiments, we found that in a contiguous congestion area, we can have chains of up to 5 regular nodes without affecting the throughput (see the twelve-node experiment described in Section 3). This suggests we can be more judicial in proxy deployment in some areas. More sophisticated heuristics in this regards should be explored in the future.

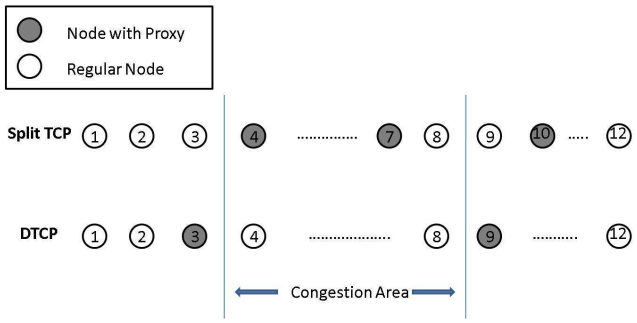

Figure 1. A comparison of proxy selection in Split TCP and DTCP.

After the source selects the proxies, it inserts the proxy list into an optional TCP header of the SYN segment. Every node receiving this modified SYN packet would check the optional header to see whether its address appears to be the first element of proxy address list. If yes, it becomes a proxy. It then will (a) respond to the source with a TCP SYN-ACK segment, and (b) attempt to establish the TCP connection with the destination by sending out a new SYN packet, with the first element (corresponding to itself) removed from the proxy list. The new proxy node will then intercepts all subsequent TCP packets in this connection defined by the source address, source port, (ultimate) destination address and destination port, buffering these packets as needed on each side of the connection.

There is another option when the intermediate node receives a SYN packet and its address appears in the proxy address list: the node may choose not to become a proxy based on its own newest information (queue size, retransmission count, or other metrics). Instead, it would just let the SYN packet pass through with the removal of its address from the optional header. In other words, the proxy selection by the source would be treated only as a hint by the intermediate nodes. This approach needs further exploration in the future.

In summary, our design of TCP proxies divides an entire end-to-end TCP connection into multiple sections. The proxies are chosen and will perform most functionality of a regular TCP source/destination. This design was convenient as the first step in our study since it makes the path sections rather independent of each other. However, we acknowledge there is also a weakness in this design. Like most other split TCP approaches, our current prototype does not support full end-to-end reliability semantics of TCP since without end-to-end acknowledgments at the TCP layer, the burden of providing end-to-end reliability rests with an application. One future direction is to provide these semantics by layering end-to-end acknowledgments

on top of our current mechanism. These acknowledgments would be mostly piggybacked on our current proxy-to-proxy acknowledgments.

2.4. Implementation in *ns-2*

We have completed the ns-2 implementation of the above TCP proxy mechanism and the necessary extensions to lower layers, using ns-2 version 2.31. We modified the DSRAgent and FullTCPAgent (NewReno), as well as the link layer and MAC sub layer, to accommodate our design.

In particular, the instantiation of a TCP proxy on a node is implemented as follows. We add an extra procedure into DSRAgent's packet processing operation. The modified DSRAgent checks each incoming packet. If it is a TCP SYN, the DSRAgent checks if the optional header containing the proxy list exists, and if so, whether this node is on the list. If the node is supposed to become a proxy, DSRAgent creates two new instances of TCPAgent. One TCPAgent, which is a new object we implemented by modifying the existing ns-2 TCPAgent, will handle any further intercepted incoming packets of this connection, mimicking the TCP end-point on the preceding path section. The other TCPAgent connects to the destination (by sending out the SYN packet to the destination; it could be again intercepted by the next TCP proxy), and sends out the packets belonging to this TCP connection towards the final destination.

Thus, a TCP proxy consists of two TCPAgent objects and a buffer between them, with the DSR routing agent playing an important role in intercepting and demultiplexing packets. We make the source code of this implementation publicly available here[10].

3. Experimental Results

The purpose of our simulation experiments is to demonstrate the potential performance gain by using dynamic TCP proxies, as well as possible detrimental effects in some cases, for example, in highly mobile environment. For this purpose, we consider two extreme scenarios: first an ideal, stationary network configuration, and then a network with high node mobility. We focus on environments with long routing paths, which are characteristic of MANETs connecting troops on a battlefield, members of search-and-rescue or disaster relief teams, etc. These environments are also likely to contain disadvantaged areas that we target in our approach.

3.1. Performance in Stationary Environment

In this experiment, we consider a stationary network. The network is deployed on a 3000mX2000m area, with evenly spaced 60 nodes forming a regular rectangular grid. The source and the destination nodes are chosen manually

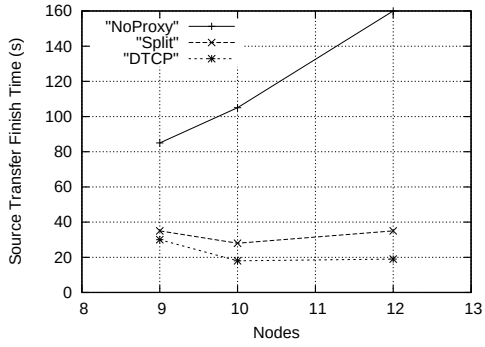

Figure 2. Source Transfer Finish Time Versus Hops Using 3 Algorithms

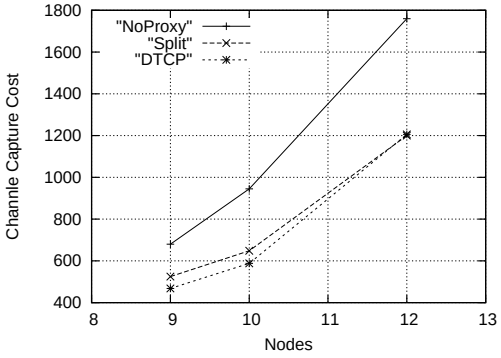

Figure 3. Channel Capture Cost Versus Hops Using 3 Algorithms

to ensure a path of a desired length. We introduce disadvantaged hosts on the path from source to destination by injecting a number of persistent flows of cross traffic, each at an average rate of 50KBps. The size and location of the disadvantaged area differ in different scenarios. After a 10-second warm-up period (allowing nodes to acquire the link quality metrics), the source initiates an FTP/TCP connection to send a 1.44MB file to the destination. Both source and destination use the NewReno version of TCP.

We run simulations with different path lengths between the source and destination – 9 nodes, 10 nodes, and 12 nodes. The congested area in these three scenarios is set as follows: the congested area in the 9-node and 10-node scenarios only includes the 9th node; in the 12-node experiment, the congested area extends, inclusively, from 4th node to 8th node.

In each scenario, we compare three algorithms: regular TCP without proxies, Split TCP with a proxy every three nodes, and Dynamic TCP Proxies (referred to as DTCP below). We consider three performance metrics: source transfer finish time, overall transfer finish time, and overall channel capture cost. In particular, the overall channel capture cost is calculated as follows. We first compute, for each path section

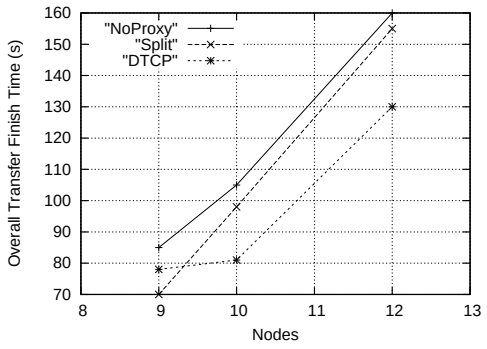

Figure 4. Overall Transfer Finish Time Versus Hops Using 3 Algorithms

(note that regular TCP has a single section), the product of its transfer time and its hop-distance. We then take the sum of these computed products for all path sections. Intuitively, this metric defines the total amount of time when any link is engaged in the current TCP transfer. Note that this is an aggregate metric indicating the overall effect of the transfer on the network, as a particular link will not be continuously busy with an ongoing TCP transfer. The results are shown in Figures 2–4.

Figure 2 shows that both Split TCP and DTCP allow the source to finish its transfer to the first proxy much quicker than regular TCP. This promises significant benefits in the overall network effectiveness because the nodes and links on the first path section become quickly available to be used by other, unrelated, connections. Furthermore, Figure 3 shows that both Split TCP and DTCP have an advantage over regular TCP in terms of the overall channel capture costs as well, although this advantage is less dramatic than was the case with the previous metric. Finally, Figure 4 demonstrates that both Split TCP and DTCP accomplish the overall transfer faster than regular TCP, confirming the intuition that proxies should improve TCP throughput.

To compare between Split TCP and DTCP, we examine the simulation scenarios and results more carefully. In both scenarios with 9 nodes and 10 nodes on the path, we set up cross traffic through the 9-th node (i.e., the congested area). Regardlessly, Split TCP places one proxy every 3 nodes, i.e., at node 4 and node 7. Our dynamic TCP proxies mechanism is designed to place a proxy just outside of the congested area. Thus, it places proxies at node 4 and node 8 in both scenarios.

In the 9-node experiment, Split TCP is slightly better than DTCP in terms of the overall transfer time because the former divides the path more evenly into sections. From this perspective, this is the most unfavorable scenario for DTCP. Still, DTCP has a slight advantage over Split TCP even in this scenario in terms of channel capture (as shown in Figures 2 and 3), promising better overall efficiency for competing flows. However, in the 10-node experiment, DTCP is better for all metrics, including the overall transfer finish time. The reason is that DTCP-produced path sections differ less in length in this case, yet DTCP localizes the effect of the disadvantaged host (node 9) better.

The third scenario with 12 nodes, which was illustrated earlier in Figure 1, might be more interesting, and it show more performance gain for our dynamic proxies mechanism. Here, an array of nodes (from node 4 to node 8) all experience congestion due to cross traffic. Our dynamic TCP proxies mechanism wisely places proxies only at node 3 and node 9, while Split TCP places proxies at nodes 4, 7, and 10. The overall channel capture of both Split TCP and DTCP is similar as shown in Figure 3. However, Figure 4 shows a significant advantage of DTCP over Split TCP in total transfer time. Indeed, Split TCP approaches regular TCP according to this metric but is over 19% worse than DTCP. We suspect it is not a good idea to designate already overloaded nodes as proxies. This experiment also suggests that using fewer proxies wisely (2 proxies with our dynamic TCP proxies mechanism) can achieve competitive or even better performance than using more proxies (3 proxies with Split TCP).

3.2. Impact of Mobility

In this experiment, we would like to investigate the potential impact of high mobility on TCP proxies mechanisms. Intuitively, TCP proxies "pin" an overlay route from source to destination, and in a high-mobility environment this route may diverge significantly from the underlying topology.

In our simulation, 120 nodes in a mobile network are uniformly spread in a 3000mX2000m area. Each node will undergo a series of random movements, similar to the random waypoint model [4], but without pauses between two consecutive movements. For each movement, the speed is chosen uniformly between 1 and 100m/s, and the direction of movement is randomly decided. Therefore, the simulated network has a very high degree of mobility. We create different network scenarios (by using different seeds for random number generators) in order to discern meaningful trends. For each scenario, in the beginning, a source/destination pair is chosen randomly and a TCP connection to transfer a 1.44M file is initiated between them. In each scenario, we run simulations for: (1) regular TCP without proxies, (2) TCP with a proxy every 5 nodes, and, (3) TCP with a proxy every 3 nodes.

We use two metrics to compare performance: transfer finish time and total number of DSR route discovery requests. Transfer finish time is the time when the destination receives the last packet. DSR route discovery requests represent pure overhead and should be minimized. To study the overall route discover overhead, we sum up the number of route

discovery requests by the source, as well as all proxies, if any.

We collected results for 15 different scenarios. Table 1 shows the performance of the three algorithms in the first three scenarios. It illustrates a trend that more proxies increase the transfer time and DSR route discover overhead. In all but one scenario, regular TCP has the best transfer time, and in more than 75% of the scenarios, TCP with a proxy every 5 nodes has better transfer time than with a proxy every 3 nodes. In addition, there are also more route discovery requests when more proxies are used.

To explain these results, we note that when nodes are mobile, especially when they move rather quickly, the locations of the end-hosts and proxies can change in the course of the TCP transfer. Therefore, an initial, good proxy selection scheme might degrade. In the extreme case, the end-hosts and proxies could be randomly placed anywhere, and the sheer effects of introducing more proxies is to increase communication cost with no benefit. For instance, it might be possible for the source and destination hosts to move close to each other, but for the end-to-end TCP connection to still traverse proxies that are far away from both end-hosts.

Table 1. The effect of TCP proxies in a mobile environment

Scenario	Proxy Positions	Tot. Transfer Time	Number of Route Discovery Reqs
1	*No Proxy*	40	21
	Every 5 nodes	90	44
	Every 3 nodes	110	113
2	*No Proxy*	60	15
	Every 5 nodes	400	68
	Every 3 nodes	380	144
3	*No Proxy*	45	19
	Every 5 nodes	55	46
	Every 3 nodes	85	142
...	...	...	...

To summarize, in highly mobile networks, it could be harmful to have fixed proxies in the network. Our dynamic TCP proxies scheme selects proxies more judicially than Split TCP and can be less vulnerable to node mobility. Still dynamic proxy selection based on the *current* state of the network will be necessary to fully overcome this problem. This is especially important for long-lived TCP connections—even with moderate node mobility, the initial proxy selection may become a poor choice later. We envision an approach and implementation which can dynamically *relocate* proxies in the middle of a long TCP transfer to adapt to a high-mobility environment. Yet a number of relevant questions need to be answered, for example, (1) how much benefit we will get from relocating proxy? (2) how much relocating overhead we have to pay? and (3) what are the criteria to make the decision to relocate a proxy, and achieve a balance between performance and overhead?

4. Related Work

Our work builds upon a well-known concept of TCP proxies, which have been applied in a variety of contexts to improve TCP throughput (see, e.g.,[1], [12], [6], [15], [16], [8] and references therein). The general performance of TCP in the presence of proxies has been studied in, e.g., [11].

There are alternative ways to improve communication efficiency, for example, in the context of Internet, CHART system[2] makes use of several fixed nodes of overlay network as intelligent middle routers. Forwarded messages are encapsulated in UDP within the routing overlay. End-to-end reliability and congestion control are maintained by TCP at source and destination nodes. Some of these techniques, such as UDP communications between proxies, could be applied in our DTCP method.

The closest to ours is the approach by Kopparty et al. [15], which attempts to improve the throughput of end-to-end TCP connections in MANETs by positioning a TCP proxy every three nodes along the routing path. Our approach selects proxy locations dynamically based on the observed network characteristics.

Our approach uses a combination of link-level retransmission count and transmission queue length to measure a wireless link quality and to dynamically select TCP proxy locations. A number of other metrics have been proposed in the context of improving ad-hoc routing performance. Bononi et al. [3] proposed to improve multipath ad-hoc routing based on cross-layer measurements of path statistics reflecting the size and congestion level of each path. Mitra et al. [17] proposed discovering stable paths using predicted future sleep times of mobile nodes. Draves et al. [9] compared several metrics in a static wireless network and found the expected transmission count, or ETX [7], to perform the best for routing purposes. An advantage of our metric is that the underlying data is readily available in the link layer (whereas, e.g., all the metrics considered in [9] require probe traffic). However, quantitative analysis of various metrics for proxy placement is an open question for future work.

5. Conclusions

This paper describes a novel approach to improve the performance of a mobile ad-hoc network in the presence of disadvantaged hosts, which can be hosts in a high-interference or contention area, busy hosts with a large packet transmission queue, or distant hosts with a poor line-of-sight connectivity. Our approach extends the split

TCP mechanism that divides a long routing path into short sections, each with its own TCP end-point called a TCP proxy. However, unlike existing split TCP approaches, our approach determines TCP proxy locations dynamically based on the network conditions at the time of the connection establishment. This allows us to effectively isolate disadvantaged portions of the routing path, shielding the remaining network areas from their detrimental effect. We further utilize a cross-layer optimization mechanism to select the TCP proxy locations without extra signaling overhead, by leveraging the messages exchanged by the underlying routing protocol to convey the link quality information. Our preliminary simulation study shows the promising benefits of our approach. Our study also indicates that a large number of TCP proxies can have a detrimental effect in a high-mobility environment. This provides additional motivation for our approach that places TCP proxies judicially and suggests a further direction of work to investigate a possibility of dynamic migration of TCP proxies in an ongoing connection when the network topology and conditions change due to node mobility.

Acknowledgment: The authors would like to thank Krishna Shastry of Lockheed Martin Corp. for valuable discussions throughout this work.

References

[1] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)*, pages 136–143, Los Alamitos, CA, USA, May 30–June 2 1995. IEEE Computer Society Press.

[2] A. Bavier, L. Peterson, J. Brassil, R. McGeer, D. Reed, P. Sharma, P. Yalagandula, A. Henderson, L. Roberts, S. Schwab, et al. Increasing TCP Throughput with an Enhanced Internet Control Plane. In *Proc. IEEE Military Comm. Conf (Milcom'06),(Washington DC)*, pages 1–7.

[3] L. Bononi and M. Di Felice. Performance analysis of cross-layered multipath routing and MAC layer solutions for multi-hop ad hoc networks. In *Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, pages 190–197. ACM New York, NY, USA, 2006.

[4] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Comm. and Mobile Computing (WCMC) Special Issue on Mobile Ad Hoc Networking*, 2(5):483–502, 2002.

[5] I. D. Chakeres and C. E. Perkins. Dynamic MANET on-demand routing protocol. *IETF Internet Draft*, 6 2006.

[6] Reuven Cohen and Srinivas Ramanathan. Using proxies to enhance TCP performance over hybrid fiber coaxial networks. Technical Report HPL-97-81, HP Labs, 1997.

[7] Douglas S. J.De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A highthroughput path metric for multihop wireless routing. In *Proceedings of MobiCom*, pages 114–128.

[8] S.M. Das, H. Pucha, and Y.C. Hu. Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks. *Ad Hoc Networks*, 5(6):680–703, 2007.

[9] Richard Draves, Jitendra Padhye, and Brian Zill. Comparison of routing metrics for static multi-hop wireless networks. In *SIGCOMM*, pages 133 – 144.

[10] Dynamic tcp proxies source code. http://ipl.eecs.case.edu/DTP/.

[11] Navid Ehsan and Mingyan Liu. Modeling tcp performance with proxies. *Computer Communications*, 27(10):961 – 975, 2004.

[12] Z.J. Haas and P. Agrawal. Mobile-TCP: an asymmetric transport protocol design for mobile systems. *ICC*, pages 1054–1058 vol.2, 1997.

[13] Gavin Holland and Nitin H. Vaidya. Analysis of tcp performance over mobile ad hoc networks. In *MOBICOM*, pages 219–230, 1999.

[14] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomas Imielinski and Hank Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[15] Swastik Kopparty, Srikanth V. Krishnamurthy, Michalis Faloutsos, and Satish K. Tripathi. Split TCP for mobile ad hoc networks. In *Proceedings of GLOBECOM*, volume 1, pages 138– 14, 2002.

[16] M. Luglio, M. Y. Sanadidi, M. Gerla, and J. Stepanek. On-board satellite split tcp proxy. *IEEE Journal on Selected Areas in Communications*, 22(2):326–344, 2004.

[17] Pramita Mitra, Christian Poellabauer, and Shivajit Mohapatra. Stability aware routing: Exploiting transient route availability in manets. In *HPCC*, pages 508–520, 2007.

[18] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 28(4):303–314, 1998.