e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

A FASTER FAILURE RECOVERY MECHANISM FOR MULTIPATH TCP

Apurva Kumar*[1], Sudheer Kandula*[2], Shilpa Priyadarshini*[3]

*[1]Principal Software Engineer, Walmart Labs, Sunnyvale, CA, USA.

*[2]Senior Software Engineer, Nvidia, Santa Clara, CA, USA.

*[3]Principal Product Manager, Silicon Valley Bank, Santa Clara, CA, USA.

DOI : https://www.doi.org/10.56726/IRJMETS45391

ABSTRACT

In recent years, there has been an effort to change the core transmission protocol of the internet from TCP to MPTCP to better utilize the changed physical infrastructure. Multi-Path TCP (MPTCP) is relatively a new standardized transport protocol that enables nodes to utilize multiple network interfaces. However, with multiple paths comes the risk of network failures too. Link failures are quite common but they can be tackled more efficiently in MPTCP. In this paper, we present a technique that uses negative acknowledgment to accomplish faster transmission of packets. We then evaluate the proposed technique and show that it can transfer packets up to three times faster with a negligible increase in network congestion.

Keywords: MPTCP, TCP.

I. INTRODUCTION

Internet landscape has been changing continuously since its inception. Endpoints today are connected by multiple paths within a complex network. The advent of multi-homing has challenged the traditional single path usage of TCP. The situation is more prominent in datacenters where many sub flows exist between end points. Using multiple paths for the same connection provides multiple benefits. It can alleviate congestion in the network by offloading the packets to an alternate sub flow and also increase the available bandwidth for each source and destination since the channel capacity across all paths can be viewed as a single resource pool. These benefits have attracted research attention and considerable work is being done in order to develop protocols that can achieve the required objective while integrating seamlessly with existing infrastructure.

MPTCP is one such move that enables the simultaneous use of multiple IP addresses or interfaces while providing a uniform TCP a-like interface to the application. Thus, it is capable of managing sub-flows through the multiple network interfaces with no modification to the application. Also, this creates an interesting possibility that, if the failure along one path is detected early, the packets sent through that sub-flow can be retransmitted along another path. As of now MPTCP has now been standardized since 2013 and now is part of Linux Kernel 5.6 or newer with v1 version. It has been adopted by Apple, Samsung and few other companies [11].

The recovery mechanism in MPTCP is not yet well defined and currently relies on traditional TCP recovery mechanisms of time-out or fast retransmissions. In this paper, we present a novel failure detection and recovery mechanism that greatly improves performance with little modification to the current protocol. We introduce the notion of Negative Acknowledgement (NEGACK), which notifies the sender of the packet that path is broken and it should use other paths if available. A NEGACK field is set on a packet as soon as a router detects a broken link for the concerned packet. This is accomplished by swapping the source and destination address in the packet and sending it back to the source to inform the source of the failure. The source can then retransmit the packets sent along the broken link through other paths based on available capacity instead of waiting for timeouts before taking action.

In section II, we provide a little background on MPTCP, the current failure detection and recovery mechanisms in TCP and MPTCP and their shortcomings. We also define our problem statement and discuss the proposed idea in section II. Simulation methodology is discussed in section III, experiments and results are detailed in section IV. We conclude in Section V.

When TCP/IP was initially designed, hosts had a single interface and only routers were equipped with several physical interfaces. As the number of hosts and load increased heterogeneously in terms of company servers, intranet, home users, data centers, tablets, smartphones and other Wi-Fi devices, the load on a single connection from each of these devices has increased as well. TCP Congestion Control has managed to keep the Internet running so far as it matches load to available capacity on small time intervals and relies on external loopback mechanism on larger time intervals [8]. To improve connection reliability in case of link failures, these traffic sources now rely on multiple IP connections/interfaces. Therefore, today most of these hosts are now equipped with more than one interface, what is now called multi-homing. End-users have come to expect that using multi-homed hosts will increase both reliability and performance.

In practice this is not always the case, as the majority (approximately 90%) [9] of the total Internet traffic is still driven by TCP and TCP binds each connection to a single interface. This suggests that TCP is not an ideal protocol for efficiently and transparently using the IP connections and interfaces available on a multi-homed host [9]. Over the past three years, the MPTCP working group of the IETF has been developing multi path extensions to TCP that enable hosts to use multiple paths available through multiple interfaces/IP addresses, to transfer the packets that belong to a single connection. The main motives and goals for MPTCP are: to be deployable and usable without making any significant changes to existing Internet infrastructure, to be usable by unmodified applications and to be stable and congestion-safe over the wide range of existing Internet paths [5].

MPTCP does provide reliability to demanding applications such as file transfer, video streaming, VoIP, Internet games, IPTV but it still relies on costly timeouts in case of failures as explained in following section.

1. Current Failure Detection and recovery mechanism in TCP and MPTCP

Although, MPTCP working group has suggested the congestion control mechanism [4] for MPTCP, to the best of our knowledge, there has been no proposal for failure detection at a router, at which packet cannot be forwarded on alternate paths to the destination. MPTCP still relies on TCP for failure detection and recovery. TCP has multiple ways to discover network failure and as we shall show that these methods are either ill-suited or inefficient for quick recovery in MPTCP. Currently TCP has following methods to handle failure:

1.1 Routing via alternate link

In this case, if a forwarding link fails at a router, the packet is sent on an alternate link. But if that link is congested, this forwarding may prove costly to that flow and can cause further congestion on that link. Also, it may not be possible to forward along the alternative link in order to ensure quality of service. On the other hand, if it was possible to notify the sending host of this failure then the sender can retransmit the data along alternate paths available to it. The existing mechanism does not take advantage of the inherent source routing opportunity available. Doing so can avoid the possible network congestion scenario as explained above. This forwarding mechanism can be used even with our proposal if the ISP feels that it is acceptable.

1.2. Timeouts

The sender maintains a sliding window to track unacknowledged packets with each such packet having an associated timestamp. The sender waits for a predefined time that equals twice the round-trip time (RTT) and retransmits unacknowledged packets in the sliding window. If the RTT is not small, waiting for twice the interval is costly and clearly inefficient for MPTCP as it has alternative paths to the destination.

II. METHODOLOGY

We illustrate our problem statement with a very simple example. Suppose our source node is S and destination node is T and S-T path has two intermediate routers - M and N. Further our source has multiple IP addresses i.e., our source S has multiple paths to the destination via these IP addresses. In TCP, if a link failure occurs between intermediate routers M and N, M will route the packet to destination through some alternate path if it exists. In case an alternate path does not exist, a failure will be known to source S only after the timeout occurs. But since there are no other paths currently available in TCP, the source S has no other option than to wait for link M - N to be up again. On the other hand, in MPTCP, failures can occur on individual sub flows but if the source knows that a sub flow Y is down, waiting for timeout is time consuming, particularly because there are other sub flows on which the packets could be sent. So, in our next section we explain our proposed solution for this problem.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

2. Proposed Solution

2.1 Protocol Modification

We propose the following modifications in the existing MPTCP protocol to make the sender aware of path failures and enable it to retransmit the lost packets on an alternate path.

1. Each packet will have a field NEG ACK. If NEG ACK is equal to true, then this packet was not delivered to the destination.
2. When a router is unable to forward a packet on any of the possible links, it sends the same packet back to the source by swapping source and destination IP address fields of that packet. In addition, NEG ACK is set to true.
3. For each destination, source node maintains a set of paths that are known to be dysfunctional.
4. When a packet received by an end-node has NEG ACK as true, then the source enqueues that packet on a path which is known to be connected.
5. While sending a packet, the sender does a table look up to check whether there is a list of dysfunctional paths being maintained for the destination address. If the lookup fails, then a conventional MPTCP packet sending algorithm is used. Otherwise, we have to choose from a set of functional paths. Whenever a packet is being sent on a path that is known to be functional, with a probability $p \ll 1$, a copy of the packet is sent on a path that is not known to be dysfunctional.
6. When an end-node receives an ACK from a dysfunctional path, that path is removed from the set of paths that are known to be dysfunctional.

2.2 How does this reflect improvement?

In traditional MPTCP, if a router cannot forward a packet, the source node has to wait for a timeout or for a packet with a higher sequence number to reach the destination. When a packet with a higher sequence number reaches destination, the destination sends an ACK of the last packet unto which all packets have been received. Thus, it could take a long time before the source realizes that the packet has not been delivered. Sending a NEG ACK can solve this problem. Whenever a link is up, it could take a long time for faraway routers and end-nodes to realize that packets can now be sent over another route. Sending a copy of a packet occasionally solves this problem. If an ACK was received for a path that was in dysfunctional paths for an address, then it implies that the link which was broken earlier is now functional.

On the basis of these two hypotheses, we believe that in the modified MPTCP protocol, end nodes would be able to send more packets because end nodes are more aware of the current state of the network. The delay between the actual state of the network and the state known to the end node is less in the new protocol.

2.3. Is this approach faster than fast retransmission?

Fast Retransmit is an enhancement to TCP which reduces the time a sender waits before retransmitting a lost segment [10]. Instead of waiting for timeout, the sender retransmits based on three duplicate acknowledgements for the same packet. This results in much faster data transfer during failures/ lost packets. At first glance, this might seem to completely solve the problems we highlighted. But there are a couple of nuances; our proposal is at least as fast as Fast retransmit and our mechanism does not send packets over dysfunctional paths. In fast retransmit, the sender might end up sending the packets again over the broken path thus causing a time-out. We therefore claim our proposal to be superior.

III. MODELING AND ANALYSIS

We simplify some of the constraints for the purpose of simulation. In our initial setup, we have a single source and a single destination. We differentiate between different paths by different ISP networks. Source and destination are connected by three different paths (networks) and have different IP addresses for each of these networks. We verify whether we have received all the packets at the destination by using sequence numbers. We generate packets with random data but label them increasing with sequence numbers. We then simulate a multi-source setup.

IV. RESULTS AND DISCUSSION

4.1 Experiment 1

4.1.1 Simulation Setup

For the first experiment, we have a simple network topology as shown below. A is the source and H is the destination. Source and destination have different IP Addresses for each path. In this simulation, link EH goes down after a while. We define throughput in the network as the number of packets that were confirmed to be received by the destination i.e., number of packets for which ACK successfully reached source. We expect that sending NEG ACK packets might increase congestion in the network. We define congestion as the total number of packets that are generated in the network. Plots for throughput and congestion are shown below.

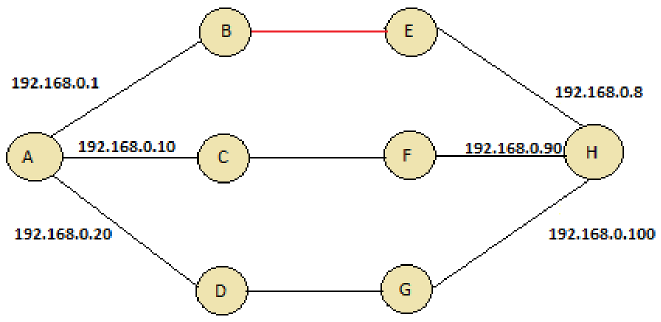

Figure 1: Network topology

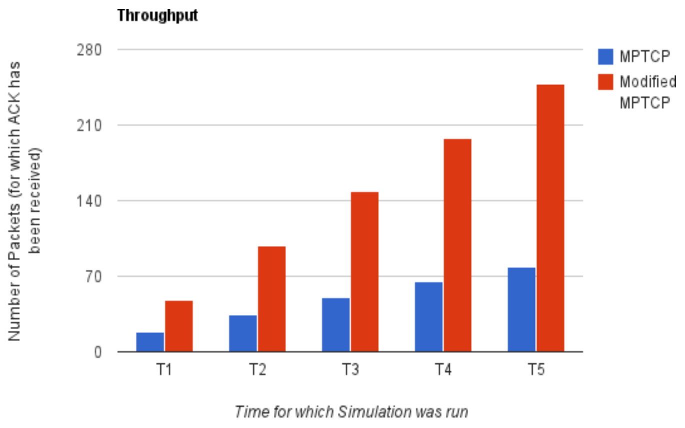

Figure 2: Throughput comparison for topology in Fig. 1

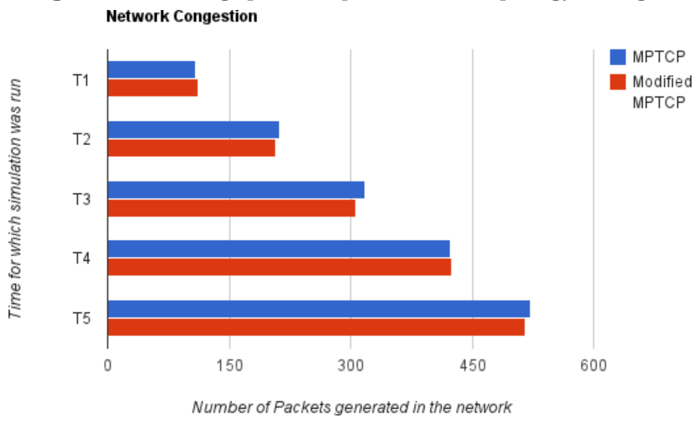

Figure 3: Network Congestion comparison for topology in Fig. 1

4.1.2 Analysis

We see that there is a remarkable difference in number of packets being reliably received at the destination. This is because, in unmodified MPTCP, the source never becomes fully aware of the fact that one of the paths has become dysfunctional. Consequently, it keeps sending packets over that path as before and this degrades performance. More surprising is the result that network congestion in unmodified MPTCP a bit more than in modified MPTCP. The reason being the same, in modified MPTCP, we do not send packets over dysfunctional paths and hence makes up for extra NEG ACK packets being sent over the network.

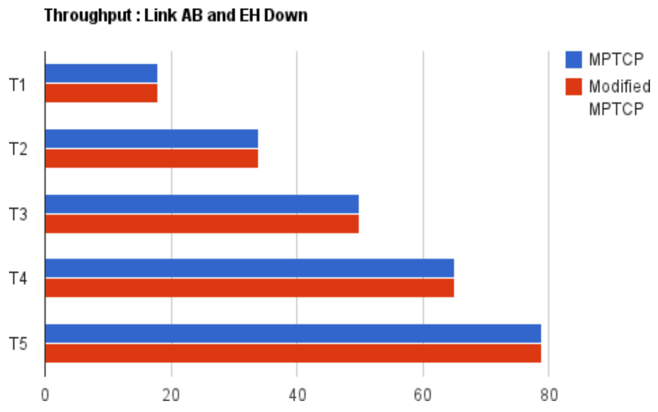

Figure 4: Throughput comparison for topology in Fig. 1 when link A-B and E-H are down

4.2 Experiment 2

4.2.1 Simulation Setup

In this experiment, the topology is the same as Fig. 1. Instead of a single link failure, we simulate two simultaneous link failures i.e. A-B and E-H. In the previous simulation, no timeout can occur in modified MPTCP because NEG ACK reaches the source before timeout interval (2*RTT). In this case, some NEG ACKs will not reach the source as A-B is down as well and hence timeout will occur in both protocol implementations.

4.2.2 Analysis

The A-B link is connected to source and when that link goes down, the source is aware of the fact that this path is not functional. Therefore, in both protocols, source does not send over broken path once it is known that the path has gone down. This is in sharp contrast to Experiment 1. Thus, we see that both protocols perform equally.

4.3 Experiment 3

4.3.1 Simulation Setup

In this experiment, one more node I between B and E (Fig. 5) is added and after a while, links B-I and E-H go down.

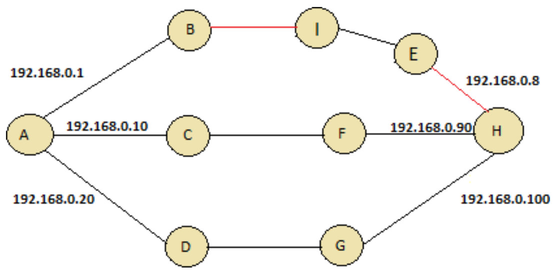

Figure 5: A modified network topology of Figure 1 with a new node I in between B and E.

4.3.2 Analysis

We simulated this topology to determine what happens when some NEGACKs are lost when multiple links fail. The difference is very small. This is because only a few NEGACKs are lost and the source timed out on these packets and resent. Interestingly, the source does send more packets along B but NEGACKs for these reach the source fairly quickly because B-I is down. Thus, this path is recorded dysfunctional and this obviates resending along this path in future. Therefore, we do not see a significant drop in throughput gain compared to experiment 1.

4.4 Experiment 4

4.4.1 Simulation Setup

In this experiment, we used 2 sources instead of one and the rest of the topology is the same as in Experiment 1 e., Figure 1. As in the case of Experiment 1, link E-H goes down.

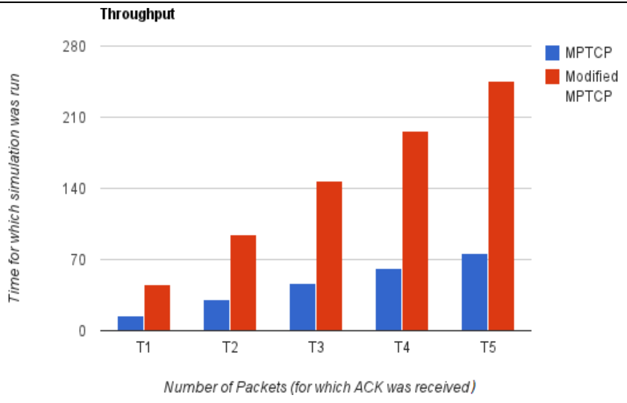

Figure 6: Throughput comparison for topology in Fig. 5 when link B-I and E-H are down

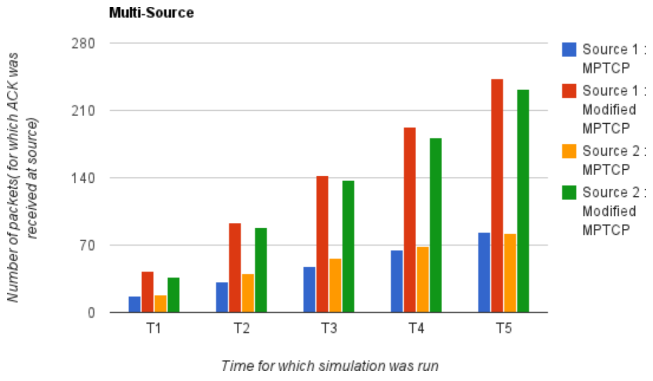

Figure 7: Throughput comparison for topology in Fig. 1 with two sources and when link E-H goes down

4.4.2 Analysis

The improvement is similar to Experiment 1 (single source). This is expected as long as network does not get clogged because of too many NEGACKs. This does not occur in two-source setup and hence the improvement over MPTCP is almost the same. It is worth noting that without any failure of links, both protocols produced exact same result on both sources.

V. CONCLUSION

In our simulations, we observed that modified MPTCP protocol improves performance by almost thrice at a negligible cost to network bandwidth. There are a few caveats. MPTCP algorithm did not have the facility to fast retransmit. Therefore, the result that we have could be upper bound to performance improvement that can be attributed to modified MPTCP algorithm. Nevertheless, incorporating the concept of NEGACKs into MPTCP seems an attractive option for various reasons. First, MPTCP has not been widely deployed yet and still in early phase of adoption [11] Second, our modifications do not require to be incorporated in all routers across the network at the same time. They could be deployed phase by phase and this will not affect usual transmission of packets. This counters the potential criticism of the protocol that it requires modification in router algorithm. We wish to implement Fast Retransmit [10] in future. We also wish to comply with current IETF draft on MPTCP and evaluate exact improvements. We would also like to evaluate our modifications for scalability and for different transmission priority for NEGACK packets in routers. It is important to note that our proposed approach is applicable wherever source routing is an option, MPTCP being a special case. If a source has multiple alternate paths to send packets and can choose between them, we believe our approach can improve performance during network failures.

VI. REFERENCES

[1] How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP - C. Raiciu, C. Paasch, S. Barr, Alan Ford, Michio Honda, F. Duchne, O. Bonaventure and Mark Handley. USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA), 2012.

[2] Improving datacenter performance and robustness with multipath TCP - C. Raiciu, S. Barr, C. Pluntke, A. Greenhalgh, D. Wischik and M. Handley, SIGCOMM 2011, Toronto, Canada, August 2011.

[3] MultiPath TCP: From Theory to Practice - S. Barr, C. Paasch and O. Bonaventure. IFIP Networking, 2011.

[4] Design, implementation and evaluation of congestion control for multipath - D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley. NSDI, 2011.

[5] MPTCP IETF Draft: http://datatracker.ietf.org/wg/mptcp/charter/.

[6] Java Network Simulator (JNS): http://jns.sourceforge.net/.

[7] Javis network animators: http://jns.sourceforge.net/.

[8] Multipath TCP Resources: http://nrg.cs.ucl.ac.uk/mptcp/.

[9] TCP Performance, The Internet Protocol Journal - Volume 3, No. 2.

[10] Fast retransmit: http://en.wikipedia.org/wiki/Fast retransmit.

[11] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan and J. Ott, "From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet," 2021 IFIP Networking Conference (IFIP Networking), Espoo and Helsinki, Finland, 2021, pp. 1-9,
doi: 10.23919/IFIPNetworking52078.2021.9472785.