A new approach to failure recovery in MPTCP

Internet Algorithmics

CSE 222B

Spring 2012

Project Report

Submitted by :

Sumit Dhoble

Rakesh Varna

Apurva Kumar

1. Introduction

The 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 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.

MultiPath TCP (MPTCP) is one such ongoing 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. The recovery mechanism in MPTCP in not yet well defined and currently relies on traditional TCP recovery mechanisms of time-out or fast retransmissions. In this report, 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 (NEG_ACK), which notifies the sender of the packet that path is broken and it should use other paths if available. A NEG_ACK field is added to each packet and as soon as a router detects a broken link for the concerned packet, it swaps the source and destination address in the packet and sends it back to the source to inform the source of the failure. The source can then retransmit the packets sent along the broken link along other links based on available capacity instead of doing waiting for timeouts and before taking action.

In section 2, we provide a little background on MPTCP. In section 3, we discuss the current failure detection and recovery mechanisms in TCP and MPTCP and their shortcomings. We define our problem statement in section 4 and in section 5, we discuss the proposed idea along with the principles we used from Network Algorithmics course. Simulation methodology is discussed in section 6, experiments and results are detailed in section 7. We conclude in Section 8.

2. MPTCP: Goals and Background

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 wifi devices, the load on a single connection from each of these devices has increased as well. It is the TCP Congestion Control that has kept Internet running so far as it matches 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 are now relying 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 (approx 95%) 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. Over the past three years, the MPTCP working group of the IETF has been developing multi path extensions to TCP [5] 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.

3. Current failure detection and recovery mechanisms in TCP and MPTCP

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

3.1 Routing via alternate link

In this case, if a forwarding link fails at a router, router sends the packet on alternate link. But if that link is congested, then 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 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. Having said that, this forwarding mechanism can still be used even with our proposal if the ISP feels that is acceptable.

3.2 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 (generally twice the 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.

4. Problem Statement

We illustrate our problem statement with a very simple example. We use the following notation: ‘S’ denotes the source node and ‘D’ the destination. ‘M’ and ‘N’ are two intermediate consecutive routers along one path. We assume that the source and the destination have multiple IP addresses with corresponding pair of addresses connecting them through a separate path. 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 is 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 unnecessary, particularly because there are other sub flows on which the packets could be sent. In our next section, we explain our proposed solution for this problem.

5. Proposed Solution:

5.1 Protocol modifications:

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.

- Each packet has a field NEG_ACK. If NEG_ACK = true, then this packet was not delivered to the destination.

- For each destination, source node maintains a set of paths that are known to be dysfunctional.

- If a router is unable to forward a packet because of link failures, the router sends the same packet back to the source by swapping source and destination IP address fields of that packet. In addition to that, NEG_ACK is set to true.

- When a packet received by an end-node has NEG_ACK set to true, the node retransmits that packet along an alternate path that is known to be connected.

- While sending a packet, a table lookup is used 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 (<< 1), a copy of the packet is sent on a path that is not known to be dysfunctional. This avoids packet failures and also helps to check whether a path that was broken is now functional.

- When an end-node receives an ACK through a dysfunctional path, that path is removed from the set of paths that are known to be dysfunctional.

5.2 Network Algorithmic Principles:

We apply the following principles in our proposal:

Principle **P10**: Passing Hints in protocol headers

> Sending NEG_ACK in a packet is the hint we send to the source node to notify that this packet could not be sent.

Principle **P12**: Add state

Each end-node maintains a list of paths that are known to be dysfunctional. We call this list of paths *unreachables*. This list represents the current state of the connectivity of the entire network in a concise form.

Principle **1**: Avoid obvious waste.

Instead of sending a separate packet with a NEG_ACK, we send the same packet with swapped source and destination fields. This avoids waste in two ways. The router does not have to generate a separate packet and the source does not has to regenerate the packet to send over another flow. It just reverses the source and destination fields and sets NEG_ACK to false.

5.3 How does this reflect an 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 for the last packet up to which all packets have been received. Thus it could take a long time before the source gets to know that the packet has not been delivered. Sending a NEG_ACK could 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 can help solve this problem. If an ACK was received for a path that was in unreachable for an address(which would be the source in that packet), then it implies that the link which was broken earlier is now functional.

On the basis of these two hunches, 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.

5.4 Are we faster than Fast retransmission?

Fast Retransmit is an enhancement to TCP which reduces the time a sender waits before retransmitting a lost segment [11]. 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.

6. Implementation and simulation methodology

We built upon an existing network simulator called Java Network Simulator (JNS) [6]. We implemented the MPTCP agents at the source and destination, the required modifications in the protocol mentioned in section 5 and the network topologies used in the simulations. The implementations allowed us to simulate the network running on MPTCP protocol with and without our proposal.

For purpose of simulation, our proof of concept was the idea that our modified MPTCP protocol helps in sending more packets in a given period of time. We simplified some constraints to ease the simulation process.

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.

7. Experiments and Results

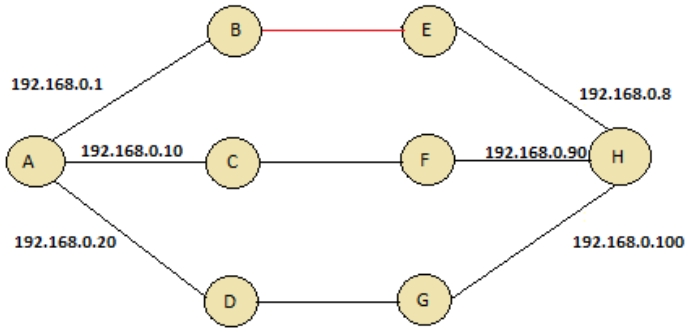
7.1 Experiment 1:

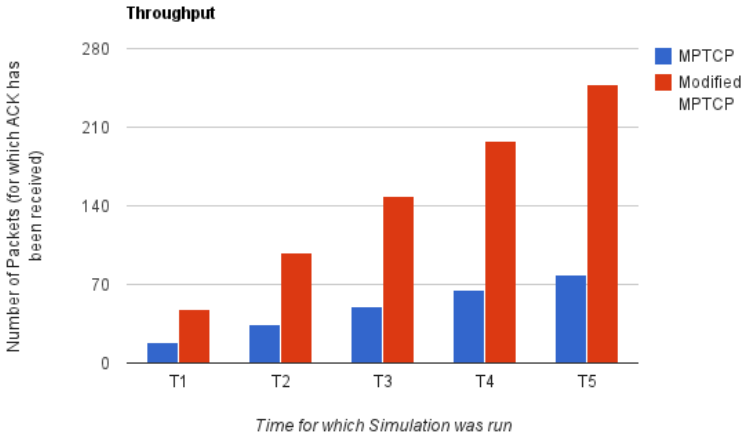
7.1.1. Simulation Setup:

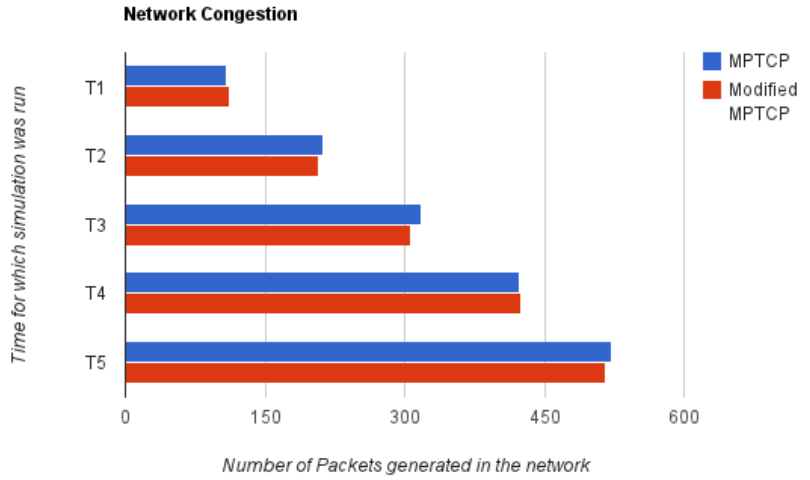
For the first experiment, we had a simple network topology as shown below. A is the source and H is the destination. Source and destination have different IP Addresses for each path. In this simulation, link EH goes down after a while. We defined throughput in the network as the number of packets that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

network. We, therefore, defined congestion as the total number of packets that are generated in the network. Plots for throughput and congestion are shown below.

Network Topology







7.1.2 Analysis:

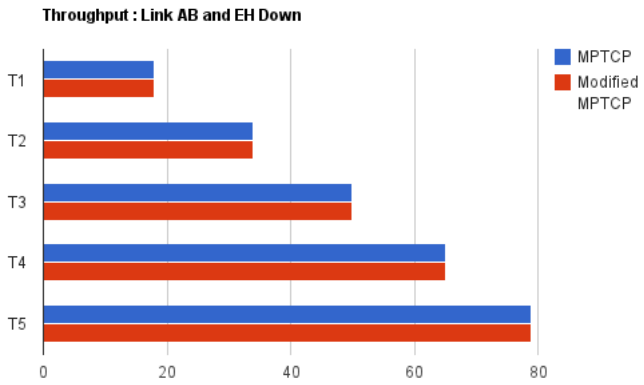
We can see that there is a remarkable difference in the number of packets being reliably received at the destination. This is because in MPTCP, even though source never becomes fully aware of the fact that one of the paths has become dysfunctional. Consequently it keeps sending packets over that path as before and this degrades performance.

More surprising is the result that network congestion in unmodified MPTCP is a bit more than in modified MPTCP. The reason for this is also the same. In modified MPTCP, we do not send packets over broken paths and hence somehow make up for extra NEG_ACK packets being sent over the network.

7.2 Experiment 2:

7.2.1 Simulation Setup:

In this experiment, the topology is the same. Instead of one link going down, we simulate two simultaneous link failures A-B and E-H. The reason behind this simulation is that in the previous simulation, no timeout can occur in Modified MPTCP because NEG_ACK reaches the source before timeout time (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.



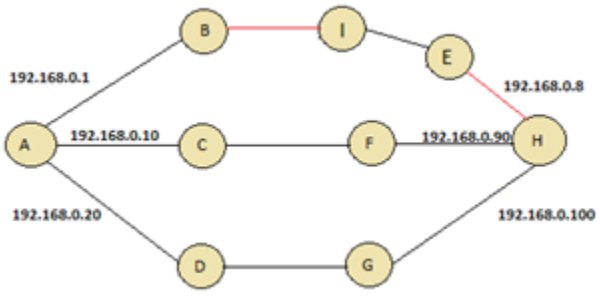
7.2.2 Analysis:

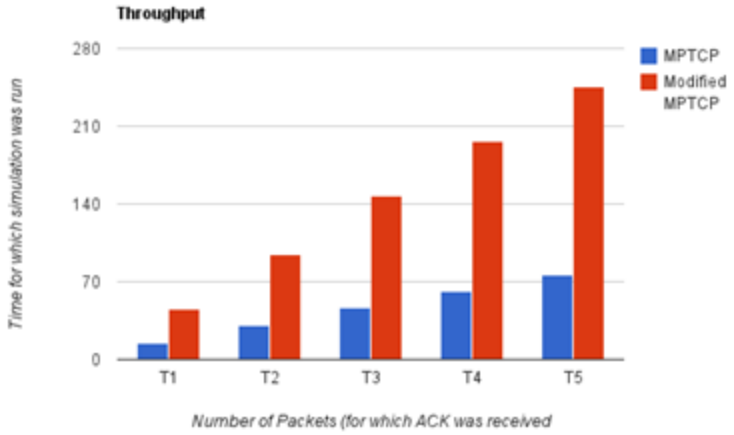
This result is not surprising and indirectly acts like a *sanity test* for our simulation implementation. A-B link is connected to source and when that link goes down, source is aware of the fact that this path is not functional. This is in sharp contrast to Experiment 1. Thus we see that both protocols perform **equally** well.

7.3 Experiment 3:

7.3.1 Simulation Setup:

In this experiment, we add one more node I between B and E and after a while, links B-I and E-H go down together.





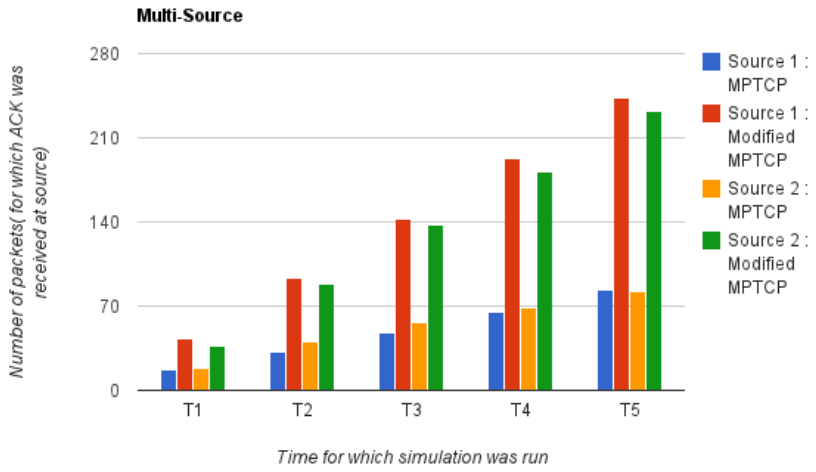
7.3.2 Analysis:

The purpose behind running this topology was that we wanted to find out what happens when some NEG_ACKs are lost because another link went down in the process. The difference was very small. This is because only a few NEG_ACKs were lost and the source timed out on these packets and resent. Interestingly, the source did send a few more packets along B but NEG_ACKs for these reached the source fairly quickly because B-I was down. Thus this path was recorded dysfunctional and this saves resending along this path in future. This is the reason that improvement in throughput is not too less than from experiment 1.

7.4 Experiment 4:

7.4.1: Simulation Setup:

In this experiment, we used two sources A and X instead of just A and the rest of the topology is the same as in experiment 1. As before, link E-H goes down.



7.4.2 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 NEG_ACKs. This did 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. We considered this as another sanity check for our implementation.

8. Conclusion and Future Work

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 do 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 NEG_ACKs into MPTCP seems an attractive option for various reasons. First, MPTCP has not been deployed yet and is still in a draft stage. Second reason counters the potential criticism of the protocol that it requires modification in router algorithm. The reason this argument will not affect modified MPTCP adversely is the fact that our modifications do not require to be done on 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.

We wish to implement fast retransmit 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 NEG_ACK packets in routers.

Another potential criticism for the proposed mechanism is the number of NEG_ACK packets that might float around in the network. A naïve solution would be to make the router remember the hosts to which it has already sent a NEG_ACK to. Clearly, this might consume large amounts of memory in a router with many edges. Instead, if each router along the path maintains the list, the router attached to the broken link will probably not receive packets from the same sender, since routers before it in the path can send the NEG_ACK back to the source. This is a much more scalable solution. Still, this requires that the routers check for the NEG_ACK field in addition to the source and destination fields. It will be interesting to see if this approach is feasible.

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.

9. References

[1]. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP - C. Raiciu,C. Paasch, S. Barré, Alan Ford, Michio Honda, F. Duchène, 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. Barré, C. Pluntke, A. Greenhalgh, D. Wischik and M. Handley, SIGCOMM 2011, Toronto, Canada, August 2011.

[3]. MultiPath TCP: From Theory to Practice - S. Barré, 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 Congetion Control :
http://en.wikipedia.org/wiki/Transmission_Control_Protocol#Congestion_control.

[10]. Deep Packet Inspection :
http://en.wikipedia.org/wiki/Deep_packet_inspection#Quality_of_service.

[11]. Fast retransmit
_http://en.wikipedia.org/wiki/Fast_retransmit.