Performance Evaluation and Computer Simulation of Multipath Transmission Control Protocol

Taha Arabi[1], Kenz A. Bozed[2] and Amer Ragab Zerek[3]

[1]Higher Institute of industry technology, Computer Department
Tripoli- Libya
E-mail Tahaelarbi87@gmail.com

[2] Benghazi University, Faculty of Information Technology, Department of Computer System Design
Benghazi, Libya
E-mail kenz.bozed@uob.edu.ly

[3] Zawia University. Faculty of Engineering , EE Department.
Zawia Libya
E-mail anas_az94@yahoo.co.uk.

Abstract—**Multipath TCP has been proposed by the Internet Engineering Task Force (IETF) working group, in order to get benefit of the interfaces in the network devices as most of these network devices are equipped with more than one interface. MPTCP splits a single data stream across multiple paths. This has obvious benefits for reliability, and it can also lead to more efficient use of network resources. However, packet reordering at the destination and congestion control still a major problem in MPTCP. Several techniques have been proposed for congestion control for MPTCP, However, the packet reordering at the destination is not considered efficient packet reordering schemes can drastically improve the throughput for the MPTCP protocol. Therefore, in this work various available packet reordering techniques available for single path TCP are tested in the multipath situation. These algorithms are Duplicate Selective Acknowledgement (DSACK), Eifel and Forward Retransmission Timeout (F-RTO). Various network scenarios are simulated in a network simulator ns-3 and measurements are taken for various congestion control algorithms and path characteristics to see which algorithm works best for the multipath scenario.**

Keywords- ; Multipath TCP, CMT-SCTP and Protocol

I. INTRODUCTION

Technological developments in recent years have seen an increasing trend towards the integration of multiple network interfaces into mobile devices. For instance, most laptops are equipped with wired (Ethernet) and wireless (Wi-Fi) network interfaces. Similarly, the majority of smartphones and tablet PCs are able to connect to the internet through either Wi-Fi or their cellular network (3G). Given the importance of these, network operators rely upon the use of redundant paths between servers in datacentres, which protects against any link failures. All backbone networks are all mostly meshed. This means that multiple paths exist between two end points. It has been proposed that simultaneously using more than one path might improve the performance and the robustness of network connections [1].

Several multipath solutions have been proposed as a means of balancing the load between different paths and dynamically switching the traffic from congested or broken links to the best paths. The implementation of multipath connections has been suggested in different layers of the TCP/IP stack. Hacker *et al.*[2] proposed that can be implemented at the application layer, in order to simultaneously use many TCP sockets to distribute the data between these so called Parallel Sockets (PSockets). Other studies have proposed opening several sub-connections in the transport layer for the application of the Concurrent TCP (CTCP) [3,4].

The idea of using the transport layer to implement the multipath capabilities has been supported by other studies [5], because it is easier for the receiver to gather the characteristics of each path at this layer, such as capacity, congestion state and latency. This information may help to militate against congestion in the network by switching the packets from the congested and broken paths to the best paths. An Internet Engineering Task Force (IETF) working

group has proposed Multipath Transmission Control Protocol (MPTCP) in 2009. Their focus is to use the transport layer to improve network performance by finding ways to effectively split the packets between the available paths [6].

II. MULTIPATH TRANSMISSION CONTROL PROTOCOL (MPTCP)

MPTCP is the extended version of regular TCP that take advantage of the multiple paths between end points. MPTCP has been designed to utilise all the available resources to improve the network performance. It uses the available paths to send single data stream in such a way it is transparent to the application [6]. MPTCP is designed with the following three goals[7]:

Goal 1: MPTCP should be fair to regular TCP at shared bottlenecks. The Multipath TCP should utilise the same capacity as the TCP in a shared bottleneck without any consideration of the number of subflows used .

Goal 2: A multipath flow should not harm the other flows, which explain that

"A Multipath should not take up more capacity from any of the resources shared by its different paths than if it were a single flow using only one of these paths. This guarantees it will not unduly harm other flows" .

Goal 3: Balance congestion. A multipath flow should shift as much traffic as possible off the most congested paths .

MPTCP Architecture

The transport layer in the MPTCP is divided into two sublayers. The main function of the upper sublayer is to gather connection management information, such as reordering packets and establishing connections. The second, lower sublayer is responsible for the management of the available subflows, enabling them to be seen as one single TCP flow [6, 17]. The MPTCP architecture [8] is described in Figure 1.

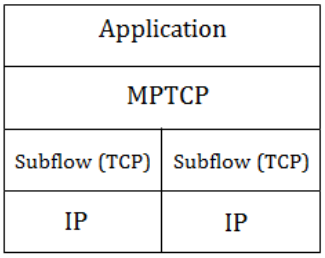

Figure 1: MPTCP Architecture

In order to facilitate the reordering of packets received out of sequence by the receiver, MPTCP has been designed to have two spaces for sequence numbers, while each sublayer has one sequence number. The first space is for the subflows (at the subflow level), with each having its own sequence number, which is the same as the standard TCP sequence number. The main purpose of this sequence number is to control the data flow in each subflow, as well as to distinguish the bytes sent in the subflows. The second sequence number is used at the connection level. Its main function is to reorder the received packets before they are sent to the application layer [6]. The mapping between the two levels is conducted at the connection level [9].

Main Mechanisms

Some new TCP options have been used in MPTCP protocols in order to exchange data between two end points. These options are as follows [19]:

MPC (Multipath Capable): The main function of this option is to inform the other end points that the sender is able to send or receive using the MPTCP protocol.

DATA FIN (Data Finished): This option is used by the sender to inform the receiver that the transmission is finished and to close the Multipath connection.

ADD and **REMOVE** IPv4 addresses: The main purpose of this option is to inform the end point of the availability of new addresses, or to disconnect from an existing one.

JOIN: In case of the availability of new addresses between two end points, this option is used to establish a new subflow between them.

DSN (Data Sequence Number): This option is responsible for mapping between the two levels (subflow level and connection level)

Starting MPTCP Connection

A regular TCP socket is opened by the application, thereby starting the connection by initiating the first subflow. At this point, the MPTCP connection is similar to the regular TCP connection. However, if the two end points support Multipath TCP, additional subflows that resemble the classical TCP connection can be combined into the existing MPTCP session. This session then appears as a single connection to the applications at both sides [6], rather than being in a separate connection. Multiple paths can be identified by the presence of multiple addresses at hosts [7].

The outgoing data is scheduled and the incoming data from all paths is then reordered as shown in Figure 2.

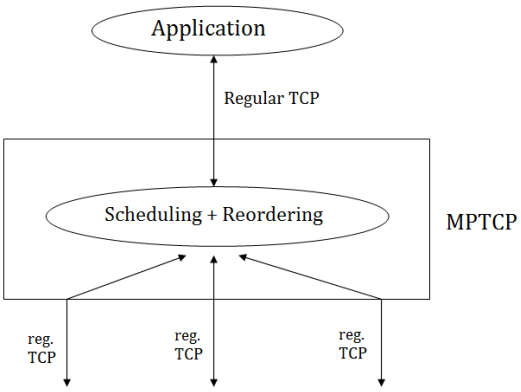

Figure 2: MPTCP is transparent to both the applications and network

When the client wants to start a new MPTCP connection as shown in Figure 3, it sends a SYN segment with a MPC option to inform the server that it is capable of performing an MPTCP connection. If the server supports Multipath TCP connection, it replies with a SYN+ACK segment that includes the MPC option. The client then replies with an ACK segment to complete the three way handshake [9].

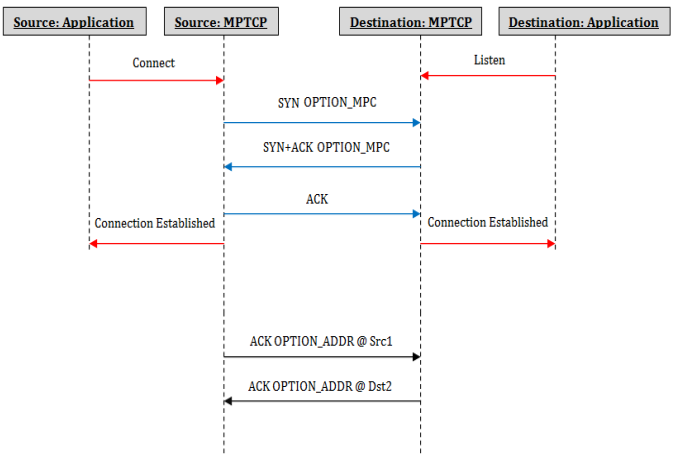

Figure 3: Establishing MPTCP Connection

Reliability and Retransmissions

In case of packet loss, MPTCP proposes a retransmission mechanism that sends the lost packets in a different subflow from the one in which it is originally sent. This can be done by using data sequence mapping, which allows the data with the same data sequence number to be re-sent on a different subflow. However, this approach also requires the original segment to also be sent in its original subflow in order to maintain the integrity of the subflow "middleboxes could reply old data, and / or could reject holes in subflows" [8], and this retransmission will be ignored by the receiver. There are no specific mechanisms for handling retransmissions in MPTCP protocol specifications, meaning that they depend on the local policy. According to Nguyen [9], the use of different subflows for segment retransmission and then sending it in its original subflow is likely to waste bandwidth. However, this may result in the reduction of application-to-application delays.

Terminating an MPTCP connection

MPTCP terminates connection through the use of a mechanism that resembles regular TCP. In this case, if the sender sent all the data and wants to inform the destination that there is no more data to send, they will send a DATA_FIN segment that affects the subflow level. The destination will then reply with a DATA_ACK segment when all packets have been correctly received at the connection level. In the case of packet loss, the sender will maintain the connection until the retransmission is finished and the receipt of all data has been acknowledged. Only then will the connection be terminated [6].

MPTCP Congestion Control

The algorithms discussed above cannot be used in a multipath context, because this would result in a multipath flow receiving twice the throughput of a single flow which is not fair [10]. Therefore, Zhou *et al.* [11] have changed the AIMD parameters in terms of (α, β), and have analyzed the congestion algorithm Dynamic AIMD (DAIMD) [12]. The congestion control has been coupled to MPTCP in order to ensure that the overall bandwidth is shared between the multipath subflows and single TCP; and that the aggregation of single flows should be the same as a multipath flow.

Although, [13] proposed the use of the EWTCP algorithm to ensure fairness in multipath connections, this approach would not use network recourses efficiently. Another solution has been proposed by [14]. The idea of this approach is for the MPTCP to move the traffic along less congested paths. In examining this idea, the IEFT working group has designed the Multipath TCP to achieve the following goals [15, 16 , 18].

Goal 1: MPTCP should be fair to regular TCP at shared bottlenecks.
Goal 2: A multipath flow should not harm the other flows.
Goal 3: Balance congestion.

The first two goals ensure the fairness at the bottleneck, where the third goal is to perform resource

pooling. In addition to these three goals, Tuan *et al.*[15] have proposed an energy aware congestion control algorithm for Multipath TCP ecMTCP. This could be achieved by employing both the energy-aware multipath congestion control algorithm and an end to end cost measurements model. However, while this model has demonstrated the ability to shift traffic from high energy cost paths to lower cost paths, it has not yet been validated in real life scenarios.

Uncoupled TCP

Uncoupled TCP is the core TCP algorithm for Additive Increase Multiplicative (AIM) decrease. In this approach, once the connection starts, the congestion window increases exponentially, and then after a retransmission timeout [16]:

- Increase w_r by $1/w_r$ per ACK on path r
- Decrease w_r by $w_r/2$ per packet loss on path r

Where:

w_r = congestion window
r = path

Fully Coupled

Fully coupled algorithms use the same approach as the standard TCP congestion control for increasing the congestion window. However, in cases of packet loss, decreasing the congestion window is dependent on the total window size for all the subflows [17]:

- Increase w_r by $1/w$ per ACK on path r
- Decrease w_r by $\max(w_r - w/2)$ per packet loss on path r

Where:

$$w = \sum_r w_r$$

Linked Increase

In linked increase algorithms, each subflow maintains its own congestion window [18].

- Increase w_r by α/w per ACK on path r , with $\alpha = 1/|Si| * RTT_r/RTT\ Si$
- Decrease w_r by $w_r/2$ per packet loss on path r

Where:

Si = The aggregation of all subflows
|*Si*| = The number of subflows
RTT *Si* = The RTT of the best path

The summation of all congestion windows across all paths allocates more of the increase to those subflows with a larger window. In other words, traffic is pushed from more congested to less congested paths. α is designed to examine all the path properties, such as congestion window and RTT, then summarise the total throughput and compare it against the throughput of a single TCP [17].

RTT Compensation

When the subflows have different RTT, RTT compensation can be used to ensure fairness and resource pooling [18]:

- Increase w_r by $\min(\alpha/w, 1/w_r)$ per ACK on path r

  , with $\alpha = 1/|Si| * RTT_r/RTTSi$
- Decrease w_r by $w_r/2$ per packet loss on path r

Packet Reordering Techniques

In the multipath context, packet reordering is a well-known phenomenon. Packets are sent along different paths, each of which has different end to end delays and other link characteristics. This means that packets will often arrive late, potentially leading to a large number of out of sequence packets at the destination. This can result in the degradation of network performance, and create a serious problem for MPTCP when reassembling packets at the connection level rather than the subflow level [10].

A packet scheduling algorithm at the sender side has been proposed as a viable way to decrease the degradation of the throughput. In order to measure the RTT, the sender uses the time stamp method to calculate retransmission timeout (RTO). Each time the receiver sends an acknowledgement, the sender acquires an instant sample RTT value. Future RTTs can then be estimated based on the measured sample RTT values, enabling the time needed for the packets to arrive at the receiver to be estimated in each path [19]. This should result in large packets being sent along the fastest paths and the small packets along the slower paths, with a consideration of the sequence number of each packet.

According to Ford *et al.*[6] "MPTCP uses a 64-bit Data Sequence Number (DSN) to number all data sent over the MPTCP connection. Each subflow has its own 32 bits sequence number space and an MPTCP option maps the subflow sequence space to the data sequence space".

One possible solution for this challenge was proposed by Dreibholz *et al.* [20]. This suggested approach involved packets being scheduled to use an optimally chosen series of paths. These multiple paths would enable RTT compensation for paths that have dissimilar RTTs and serve to increase the overall throughput of the network. This solution would require two buffers, with a capacity at least equal to the bandwidth of the link in order to increase the cost of packet reordering. Another option has been suggested by *et al.* Tsai [10], who propose the use of a multipath transmission control scheme (MTCS), handling any out-of-order packets in the transmission through a bespoke packet scheduling policy. If successful, this scheme would offer the advantage of not requiring a large buffer, while still improving the throughput of real time data transmission along multiple paths.

The effect of this is that estimating the time needed for a packet to arrive at the receiver may constitute an effective packet scheduling technique for use in MPTCP. However, this technique is unable to detect packet loss and the retransmission process can be done slower [19]. This means that these types of algorithms are unsuitable for reliable connections, meaning those situations in which all the packets should be arriving at the receiver without any packet loss. The following sections discuss the three packet reordering algorithms (Duplicate Selective Acknowledgement Algorithm DSACK, Forward Retransmission Timeout Algorithm F-RTO and Eifel Algorithm) used for regular TCP. These algorithms will be examined with MPTCP under different scenarios and topologies using Network Simulator 3 ns-3.6.

Duplicate Selective Acknowledgement Algorithm (DSACK)

The Duplicate Selective Acknowledgement (DSACK) is a proposed extension for the Selective Acknowledgement (SACK) for TCP [21]. When using DSACK, the first block of SACK option should specify the sequence number for a received duplicate segment that triggers the ACK. If the sender receives three duplicate acknowledgements, it retransmits the lost segment, stores the cwnd value and finally switches to the congestion avoidance phase. If the retransmitted segment is acknowledged twice, the sender infers that the retransmission is spurious and begins a DSACK slow start to the stored value of the congestion window.

DSACK can be used to detect the loss of all ACKs in a window. When the ACKs in a window of successfully delivered segments are lost, this will cause a timeout to occur. Effectively, the first ACK received after the timeout will carry DSACK information that would not be conveyed by the SACK mechanism alone [22]. However, the DSACK algorithm is unable to avoid the retransmission of a full window of segments after a spurious timeout occurs. A DSACK-enabled sender is only able to identify a timeout as being a spurious RTT after the occurrence of a spurious retransmission, which occurs at the time when the ACK of the first (spurious) retransmitted segment arrives at the sender. When that RTT elapses, the receiver will be unaware that an entire window of segments is continuously causing timeouts [22]. This drawback is addressed by the Forward RTO Recovery algorithm, which is represented in Figure 4.

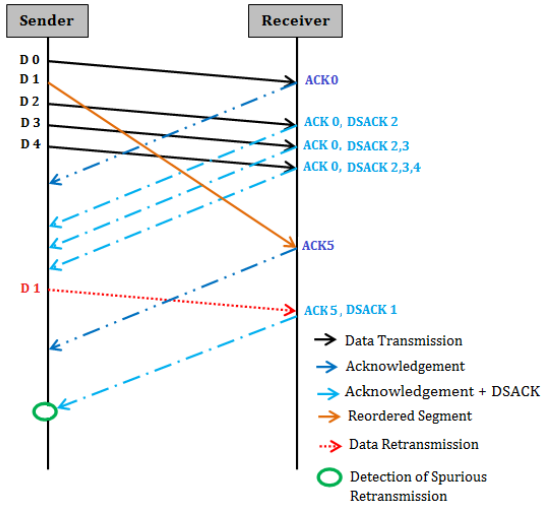

Figure 4: DSACK Algorithm

3.4.1 Forward Retransmission Timeout Algorithm (F-RTO)

Sarolahti *et al.* [23] proposed that TCP senders use the F-RTO algorithm to detect spurious retransmission timeouts and to thereby avoid the need to retransmit a whole window. When the TCP retransmits a segment after a timeout, F-RTO starts sending new data instead of retransmitting more old segments. At the same time, F-RTO monitors the next two incoming ACKs in order to determine whether or not the timeout was spurious. If either of these is a dupe ACK, the F-RTO algorithm aborts and the TCP then performs conventional RTO recovery [24]. If the first two ACKs after the timeout both acknowledge new data, the F-RTO algorithm identifies the timeout as being spurious and continues sending new data. In other words, F-RTO does not prevent single spurious retransmissions from occurring, but is effective in the prevention of further unnecessary retransmissions after a spurious timeout.

3.4.2 Eifel Algorithm

The Eifel algorithm has been proposed as a means of improving the performance of TCPs in the case of spurious retransmissions (covering both cases of spurious timeouts and spurious fast retransmits) [25]. The Eifel algorithm uses the timestamp option to distinguish an original segment from a retransmission. When the sender retransmits a segment, its timestamp is stored. Then, the receiver sends an ACK with a corresponding timestamp, which the sender compares against the stored timestamp. If the ACK timestamp is greater than the stored one, the sender can ensure that this is the original segment. If it is smaller, the retransmission can be considered spurious [26]. The Eifel algorithm solves this problem by saving the cwnd and ssthresh when the first retransmission occurs and then restoring them upon the detection of a spurious retransmission [25]. Figure 5 illustrates how the Eifel algorithm functions.

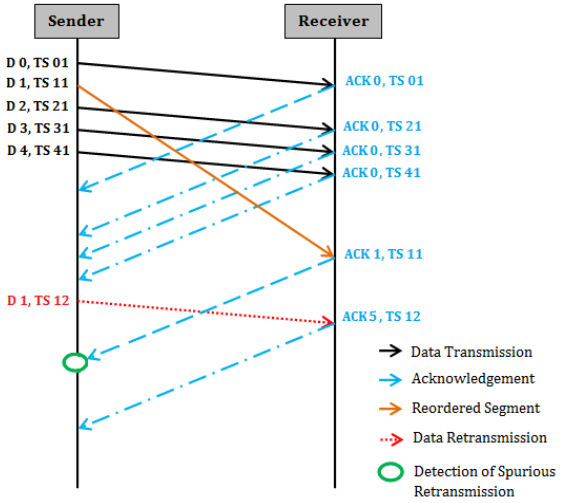

Figure 5: The Eifel Algorithm

When the segment is retransmitted, the source stores the ssthresh and the congestion window at that time, enabling solutions to be found to problems arising from spurious retransmission. Essentially, if this kind of retransmission is detected, the sender is able to restore the stored value of the ssthresh and the congestion window. [25] have shown that the Eifel algorithm can enhance the performance of the TCP. However, this approach is ineffective when either the original or the retransmitted signals are reordered [26].

SIMULATION SCENARIOS

In order to test the packet reordering algorithms DSACK, Eifel and F-RTO, which already implemented in ns-3 simulation tool, the mpTopology.cc has been modified and two scenarios have been implemented. In the first scenario the two nodes (client and the server) are directly connected with two different subflows namely subflow 0 and subflow 1. The first link with 5Mbps data rate and the second link with data rate of 2Mbps both links have a path delay of 100ms. The two interfaces in the client side have the IP addresses 10.1.1.1 and 10.1.2.1 and the IP addresses for the interfaces in the server are 10.1.1.2 and 10.1.2.2. The first Scenario is shown in Figure 6:

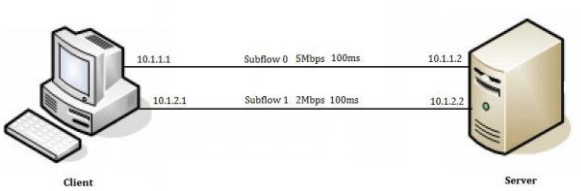

Figure 6: The First Scenario

In the second scenario all the three links have different path delay as well as data rate. The two nodes are directly connected with three different subflows namely subflow 0, subflow 1 and subflow 2. The first link has a data rate of 5Mbps with delay of 100ms, the data rate of the second link is 2Mbps and the delay is 10ms and the third link has 7Mbps data rate with a delay of 50ms. The three interfaces of the client have the IP addresses 10.1.1.1, 10.1.2.1 and 10.1.3.1 and the IP addresses for the interfaces in the server are 10.1.1.2, 10.1.2.2 and 10.1.3.2. The second scenario is shown in Figure 7:

Figure 7: The Second Scenario

RESULTS & DISCUSSION

It can be seen from Figure 8 that when using the linked increases congestion control algorithm the throughput of using DSACK stays the same. However, the throughput of both the Eifel and FRTO algorithms is decreased when they detects a spurious retransmission and the throughput when no packet reordering algorithm is used remains the same as when using uncoupled congestion control algorithm.

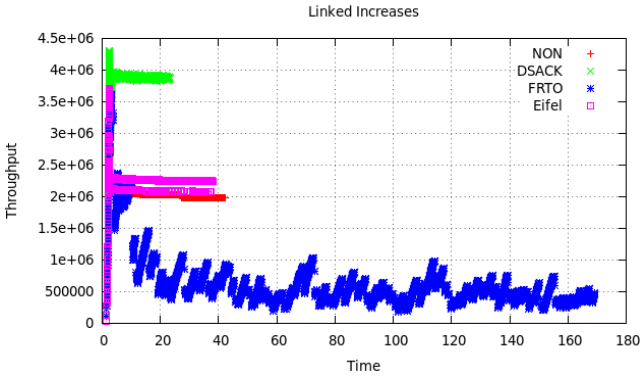

Figure 8: The Global Throughput of All Subflows Using Linked Increases Congestion Control

Figure 9 shows the overall throughput when using uncoupled congestion control of three subflows between the two nodes. It can be seen that the throughput is increased and the best result is the throughput for DSACK as it reaches about 10 Mbps and the throughput of both FRTO and Eifel algorithms is 40 less than the one when no packet reordering algorithms have been used. As a result both the Eifel and FRTO do not improve the performance of the connection.

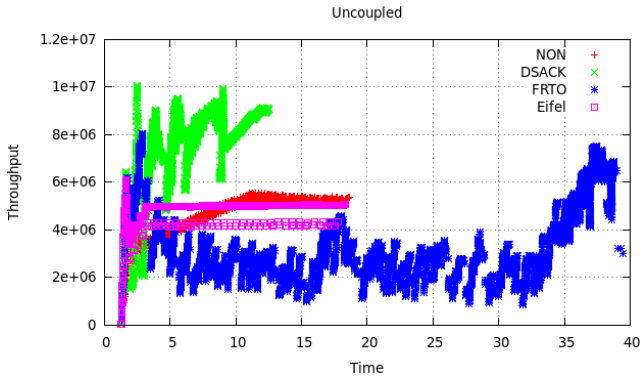

Figure 9: The Overall Throughput When Using Uncoupled Congestion Control

CONCLUSION

In this paper, MPTCP has been presented as an interesting multipath transport protocol; two different scenarios have been simulated using ns-3 to evaluate the performance and robustness of packet reordering algorithms, DSACK, FRTO and Eifel as they are already standardised and implemented in the current operating systems to optimize the performance of regular TCP. The three algorithms have been compared under two different scenarios and using four congestion control algorithms, uncoupled, linked increases, RTT compensation and fully coupled congestion control algorithms.

It can be seen from all the graphs that a good results but is not the best can be obtained when using DSACK packet reordering algorithm with RTT compensation congestion control algorithm in the first scenario when using two subflows. The Eifel algorithm can improve the performance of the connection only when using two subflows with uncoupled, linked increases and RTT compensation congestion control algorithms. The best throughput can be obtained in the scenario of three subflows is when using DSACK with uncoupled algorithm which is about 10Mbps. The throughput of FRTO increases exponentially in each scenario however; when a spurious retransmission is detected it decreases and stays in the recovery state.

Based in the simulation results it can be concluded that none of the three packet reordering algorithms is the optimal solution for the out of order packets in the MPTCP scenario, although they improve the performance of the connection. For that reason I propose to test other packet reordering algorithms in the in real implementation of MPTCP under three different scenarios i) point to point using three subflows ii) shared bottle neck iii) disjoined bottle neck.

REFERENCES

[1] B. Chihani and D. Collange, “A Multipath TCP model for ns-3 simulator”, Workshop on ns-3 held in conjunction with SIMUTools 2011, Barcelona/Spain, Dec 2011.

[2] Hacker, T.J.; Athey, B.D.; Noble, B., "The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network," Parallel and Distributed Processing Symposium, Proceedings International, IPDPS 2002, Abstracts and CD-ROM, pp.10 pp,, 15-19 April 2001.

[3] Yu Dong; Pissinou, N.; Jian Wang, "Concurrency Handling in TCP" Communication Networks and Services Research, 2007. CNSR '07. Fifth Annual Conference on, pp.255, 262, 14-17 May 2007.

[4] Sarkar, D., "A Concurrent Multipath TCP and Its Markov Model" Communications, 2006. ICC ‘06. IEEE International Conference on, vol.2, pp.615, 620, June 2006.

[5] Damon W.; Mark H.; Marcelo B., “The Resource Pooling Principle", ACM SIGCOMM Computer Communication Review, Vol. 38.5, pp 47-52, October 2008.

[6] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses" drat-ietf-mptcpmultiaddressed-12.txt, Internet-Draft, October 22, 2013.

[7] C. Raiciu; M. Handly; D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols" draft-ietf, Internet-Draft, October 2011.

[8] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[9] Sinh Chung Nguyen and Thi Mai Trang Nguyen, "Evaluation of multipath TCP load sharing with coupled congestion control option in heterogeneous networks", Global Information Infrastructure Symposium (GIIS), 2011.

[10] Tsai, M.-., Chilamkurti, N., Park, J.H. & Shieh, C. 'Multi-path transmission control scheme combining bandwidth aggregation and packet scheduling for realtime streaming in multi-path environment', Communications, IET, 4 (8), pp.937- 945, 2010.

[11] Dizhi Zhou; Wei Song; Yu Cheng, "A Study of Fair Bandwidth Sharing with AIMD-Based Multipath Congestion Control," Wireless Communications Letters, IEEE , vol.2, no.3, pp.299,302, June 2013

[12] Lin Cai, Xuemin Shen, Jianping Pan & Mark, J.W. 'Performance analysis of TCP-friendly AIMD algorithms for multimedia applications', Multimedia, IEEE Transactions on, 7 (2), pp.339-355, 2005.

[13] Honda, M., Nishida, Y., Eggert, L., Sarolahti, P. & Tokuda, H. "Multipath congestion control for shared bottleneck", Proc. PFLDNeT Workshop,2009 .

[14] Huaizhong Han, Shakkottai, S., Hollot, C.V., Srikant, R. & Towsley, D. 'Multipath TCP: A joint congestion control and routing scheme to exploit path diversity in the internet', Networking, IEEE/ACM Transactions on, 14 (6), pp.1260-1271, 2006.

[15] Tuan Anh Le, Choong-seon Hong, Razzaque, M.A., Sungwon Lee & Heeyoung Jung 'ecMTCP: An energy-aware congestion control algorithm for multipath TCP', Communications Letters, IEEE, 16 (2), pp.275-277, 2012.

[16] Sinh Chung Nguyen & Thi Mai Trang Nguyen "Evaluation of multipath TCP load sharing with coupled congestion control option in heterogeneous networks", Global Information Infrastructure Symposium (GIIS), Global Information Infrastructure Symposium (GIIS),p1-5, 2011.

[17] Raiciu, C., Handley, M. & Wischik, D. 'Coupled congestion control for multipath transport protocols', Draft-Ietf-Mptcp-Congestion-01 (Work in Progress), 2011.

[18] Wischik, D., Raiciu, C., Greenhalgh, A. & Handley, M. "Design, implementation and evaluation of congestion control for multipath TCP", Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, USENIX Association. p8-8 2011.

[19] Han Ah Kim, Bong-hwan Oh & Jaiyong Lee "Improvement of MPTCP performance in heterogeneous network using packet scheduling mechanism", Communications (APCC), 18th Asia-Pacific Conference on, Communications (APCC), 18th Asia-Pacific Conference on,p842-847, 2012.

[20] Dreibholz, T., Seggelmann, R., Tuexen, M. & Rathgeb, E. "Transmission scheduling optimizations for concurrent multipath transfer", Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT), Citeseer, 2010.

[21] Floyd, S., Mahdavi, J., Podolsky, M. & Mathis, M. 'An extension to the selective acknowledgement (SACK) option for TCP', 2000.

[22] Kaspar, D. "Multipath Aggregation of Heterogeneous Access Networks". Unpublished Phd. University of Oslo, 2011.

[23] Sarolahti, P., Kojo, M., Yamamoto, K. & Hata, M. 'Forward RTO-recovery (FRTO): An algorithm for detecting spurious retransmission timeouts with TCP', Internet Engineering Task Force RFC5682,2009 .

[24] Sarolahti, P., Kojo, M. & Raatikainen, K. 'F-RTO: An enhanced recovery algorithm for TCP retransmission timeouts', ACM SIGCOMM Computer Communication Review, 33 (2), pp.51-63, 2003.

[25] Ludwig, R. & Katz, R.H. (2000) 'The eifel algorithm: Making TCP robust against spurious retransmissions', ACM SIGCOMM Computer Communication Review, 30 (1), pp.30-36.

[26] Leung, K., Li, V.O. & Yang, D. 'An overview of packet reordering in transmission control protocol (TCP): Problems, solutions, and challenges', Parallel and Distributed Systems, IEEE Transactions on, 18 (4), pp.522-535, 2007.