On the Benefits of Applying Experimental Design to Improve Multipath TCP

Christoph Paasch[†] Ramin Khalili[‡] Olivier Bonaventure[†]

[†]ICTEAM, UCLouvain, Belgium
[‡]Telekom Innovation Laboratories / TU Berlin, Germany

ABSTRACT

Many scientific disciplines rely on *"Experimental Design"* to study various types of systems. Experimental design refers to a planned approach to experimentation that tries to provide statistical evidence to the outcome of experiments. The networking community rarely relies on such approaches, especially for real protocol implementations. Many improvements to protocols like TCP, including the recently proposed Multipath TCP, have been evaluated by considering a relatively limited set of simulations or experiments.

Multipath TCP increases the goodput of a data transfer by simultaneously using multiple interfaces. It also improves load balancing thanks to dedicated congestion control. By applying experimental design, we conduct a large set of measurements inside Mininet with the Linux kernel Multipath TCP implementation, to measure its bandwidth aggregation and load balancing. Thanks to the experimental design approach, we are able to highlight several limitations of this implementation. We identify heuristics that lead to lower than expected performance and propose improvements.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics—*Experimental Design*; C2.2 [**Computer-communication Networks**]: Network Protocols

Keywords

Experimental Design; Measurements; Multipath TCP

1. INTRODUCTION

"Experimental Design" is an approach to planned experimentation in order to answer different types of questions on a certain process or system [7]. Scientists in many disciplines follow this approach to have statistical confidence in their claims. The experimental design approach comprises a careful selection of the influencing factors of the system. Among these factors, the set of input parameters are chosen to allow an accurate view of the system's response. Finally, repeating runs allows to account for the variance of individual experiments.

CoNEXT'13, December 9-12, 2013, Santa Barbara, California, USA.

http://dx.doi.org/10.1145/2535372.2535403.

Multipath TCP (MPTCP) is a newly proposed extension to TCP [8, 9]. MPTCP allows the use of multiple interfaces for the transmission of a single data stream, without requiring any modifications among the applications and still being usable on today's Internet with all its middleboxes and firewalls. MPTCP achieves this by creating a TCP subflow for each interface and multiplexing the data segments among these subflows [22]. This allows to pool the resources of the interfaces, effectively increasing the goodput for the application. Further, MPTCP allows mobile nodes to vertically handover traffic from one interface to another, allowing them to seamlessly move data connections from a WiFi access point to the 3G connection [19].

An MPTCP implementation is a complex system, with many heuristics and algorithms influencing its performance [4, 22]. The congestion-control algorithm [26, 14] influences the sending rate of the individual subflows, the scheduler decides how to multiplex data among the subflows. Flow control provides another limitation to the sending rate. Many external factors further influence the performance of MPTCP [20, 4, 19, 6]. Especially the network's characteristics in terms of capacity, propagation delay, etc. It is very difficult to have a clear understanding of how MPTCP behaves in different heterogeneous environments.

In this paper we apply the experimental design approach to evaluate the Linux Kernel implementation of MPTCP in a wide range of heterogeneous environments. To the best of our knowledge, this is the first attempt to apply an experimental design approach to evaluate a real protocol implementation. We show that it is very beneficial to use such an approach, as it revealed previously unknown performance issues within MPTCP.

This paper is structured as follows. Section 2 describes the experimental design approach and the different choices that can be taken. Section 3 explains how this approach can be applied to the evaluation of a transport-layer protocol like MPTCP. Section 4 discusses our experimental results and describes how it allowed us to improve MPTCP's performance. The next section analyses the sensitivity of our experimental design. Finally, section 6 discusses the performance issues we identified and possible next steps.

2. EXPERIMENTAL DESIGN

Experimental Design refers to the process of executing controlled experiments in order to collect information about a specific process or system [7]. The system under experimentation is influenced by controllable or uncontrollable factors (see Figure 1). The system responds to these factors according to the laws of nature. The experimenter can observe these responses by collecting one or multiple outputs of the system, which provide the information necessary to understand the system.

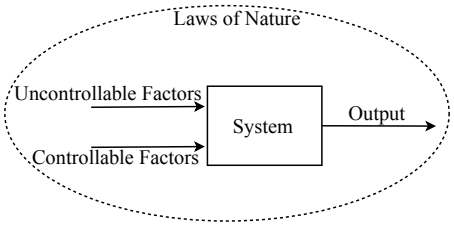

Figure 1: The system's output is influenced by a number of factors and obeys to the laws of nature [3]

Running experiments in computer science (and networking research) is peculiar in the sense that the output is often more deterministic (in absence of external factors, like temperature, time, etc.) [23]. Further, depending on the system, experiments may be cheap in terms of time required, thus allowing a large number of experiments. In this section, we give an overview of the different steps of experimental design and its particularities with respect to computer experiments.

2.1 Objective

The first step is to define the objective pursued by the experiments. Kleijnen et al. defines in [15] three different types of questions that may be answered through experimentation:

- *"Develop a Basic Understanding"* - This allows to have an overview of the system's behavior, uncover problems within the system or confirm expectations.
- *"Finding Robust Decisions or Policies"* - The goal is to find the correct configuration of the system to produce a desired output, while taking into consideration the influence of uncontrollable factors.
- *"Comparing Decisions or Policies"* - This process allows to estimate the behavior of the system with respect to a specific set of factors.

The objective defines the system that is under test and the outcome we want to measure. Once the objective is defined, the influencing factors and the way of measuring the output of the system must be also defined.

2.2 Factors

In experimental design, the system under test is often very complex. Many factors can influence the output of a system (see Figure 1). They may be of different kinds, controllable and uncontrollable. Among the controllable factors, those which influence the output of the system have to be selected. The uncontrollable factors may also influence the output of the system and are the reason for the variance of the system's output. To reduce the impact of the uncontrollable factors, an experiment should be repeated multiple times. This allows to extract the central tendency of the response by calculating the mean or median.

Optimal real-world experiments would require that the experiments are run simultaneously, to reduce the effect of the uncontrollable factors. However, in computer experiments this constraint can often be neglected as the output of the system is deterministic, allowing a sequential execution of the experiments [15].

2.3 Design of the experiment

The design of the experiment influences the input parameters that are selected to conduct the experiments. In [5, 17], the desirable properties that such parameter sets should have are discussed.

If the behavior of the system is meant to be modeled by a first-order polynomial model, fractional designs are a good fit [5]. These designs distribute the parameter set along the edges. Further, orthogonality is a desirable criterion of experiment designs, as it ensures that the sets are uncorrelated and thus allows to decide if a factor should be part of the model or not [15].

However, sometimes it is not possible to assume a first-order polynomial model of the response. It may be that there is no prior knowledge of the system's response surface, or that the system has a rather stochastic nature. In this case, space-filling designs are good choices [16]. Space-filling designs don't only sample at the edges, but distribute the parameter sets equally among the whole factor space. A space-filling design can be generated with different algorithms. Santiago et al. propose in [24] the WSP algorithm which distributes the sets equally among the space. It is based on a uniform random sampling of the input parameters and eliminates excess points according to a minimum-distance criterion. The WSP algorithm has particularly good space-filling properties, even in high-dimensional spaces [24].

3. EXPERIMENTING WITH MPTCP

In this section we describe how the experimental design approach can be applied to the experimentation with a transport protocol like MPTCP. For this purpose, we need to define our objective, determine our design factors and experiment design.

3.1 Objective

We are interested in a performance analysis of MPTCP to verify whether it fulfills its two main design goals [21]:

- *Improve throughput: MPTCP should perform at least as well as regular TCP along the best path.*
- *Balance congestion: MPTCP should move traffic away from congested paths.*

We evaluate the performance of MPTCP for a wide range of parameters and pinpoint the scenarios where these goals are not met. We can also use this framework to validate the performance of MPTCP as modifications to certain algorithms within the protocol are being done.

We execute our approach by using Mininet [11] which allows us to easily create a virtual network and run experiments between Mininet hosts using the v0.86 Linux Kernel implementation of MPTCP[1]. The benefit of using Mininet is that the results are reproducible and do not require a large number of physical machines. Compared to simulations, Mininet allows us to use the real MPTCP implementation and not a model of the protocol.

3.2 Factors

The performance of a transport protocol like MPTCP is influenced by various factors, such as bandwidth limitations, propagation delay, queuing delay, loss, etc. [2]. Further, memory constraints on either of both hosts will limit the TCP window size, additionally influencing the performance [4, 25]. Among these factors, one must distinguish between the quantitative ones (e.g., loss-rate between two hosts) and the qualitative ones (e.g., congestion-control being used). For each of the selected factors, the domain must be selected accordingly. We consider the following factors in our study:

[1]Our scripts and the Mininet virtual images are available at http://multipath-tcp.org/conext2013

Capacity Our evaluation of MPTCP targets environments of regular users, whose access speed may range from mobile networks to FTTH. We fix the range of our capacity from 0.1 to 100 Mbps.

Propagation delay The propagation delay is the round-trip-time between the sender and the receiver over an uncongested path. Measurement studies have shown that the delay may be up to 400 ms [27] . We set the delay to a domain between 0 ms and 400 ms.

Queuing delay The buffers at the bottleneck influence the queuing delay [10]. A perfect Active Queue Management (AQM) algorithm at the bottleneck router would not add any additional delay, whether a badly sized buffer may add a huge amount of queuing delay [10, 1]. We only consider tail-drop queues whose size is set to add a queuing delay between 0 ms up to 2000 ms. We leave the evaluation of different queuing policies like RED or Codel for future work.

Loss [18] shows that the loss probability over the Internet is very low (between 0 and 0.1%). In wireless or mobile networks, the loss probability may be considerably higher. We consider environments where the loss ranges from 0% to 2.5%.

Congestion Control We consider the two MPTCP congestion-control schemes: the Coupled congestion control [21, 26] which is the default one and the recently proposed OLIA [14].

These factors and their considered ranges allow to cover the principal environments that MPTCP might face when being used over the Internet. Additional factors and/or broader ranges are left for future work.

3.3 Design of the experiment

We cannot be sure of the nature of the response surface of Multipath TCP. Hence, we choose a space-filling design to cover a wide range of scenarios and correlations among the factors. It allows us to avoid making any assumptions on the behavior of MPTCP (cfr. Section 2.3). The drawback is that we need to run a large number of experiments in order to fully cover the factor space, but thanks to Mininet we are able to quickly perform these experiments. We use the WSP algorithm to generate the parameter sets in the space-filling design.

4. EVALUATION

This section shows the experiments we conducted and how they helped us to identify previously unknown performance issues with MPTCP.

4.1 Aggregation Benefit

We study whether MPTCP satisfies its first design goal (improve throughput), by quantifying the aggregation benefit (as defined by Kaspar in [13]). The aggregation benefit is expressed as a function between -1 and 1. If MPTCP performs as good as the path with the highest goodput, the aggregation benefit will be 0. If MPTCP perfectly aggregates the capacities of all paths, the aggregation benefit will be 1. -1 means that MPTCP achieves zero goodput.

Let S be a multipath aggregation scenario, with n paths. C_i is the capacity of the path i and C_{max} the highest capacity among all paths. If we measure a goodput of g with MPTCP, the aggregation benefit, $Ben(S)$, is given by [13]:

$$Ben(S) = \begin{cases} \dfrac{g - C_{max}}{\sum_{i=1}^{n} C_i - C_{max}}, & \text{if } g \geq C_{max} \\ \dfrac{g - C_{max}}{C_{max}}, & \text{if } g < C_{max} \end{cases}$$

	Low-BDP		High-BDP	
Factor	Min.	Max.	Min.	Max.
Capacity [Mbps]	0.1	100	0.1	100
Propagation delay [ms]	0	50	0	400
Queuing [ms]	0	100	0	2000

Table 1: Domains of the influencing factors for the measurement of aggregation benefit.

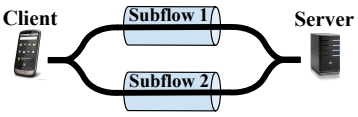

Figure 2: Our topology to evaluate aggregation benefit: MPTCP creates one subflow across each bottleneck. This allows to evaluate multipath-scenarios like a mobile phone connecting to WiFi/3G at the same time.

Our setup evaluates MPTCP in a scenario (Figure 2) where the hosts establish two subflows between each other. We consider this as the common scenario (e.g., a client having two access networks like WiFi/3G). In order to measure the aggregation benefit, the Mininet-hosts create an iperf-session using the v0.86-release of MPTCP, which creates one subflow per bottleneck-link. The iperf-session runs for 60 seconds to allow the flows to reach equilibrium.

We study two types of environments: low Bandwidth-Delay-Products (BDP) and high-BDP. Low-BDP environments have relatively small propagation and queuing delays. In a high-BDP environment, the maximum values for the propagation and the queuing delays are very large. In a first run we only consider 3 factors per bottleneck, namely the capacity, propagation delay and queuing delay. For this first run we do not add the loss-factor as the MPTCP-specific congestion controls have a very specific behavior in lossy environments (as can be seen at the end of this section). The exact specifications of each environment can be found in Table 1.

As we consider 2 paths, each being influenced by 3 factors, we have a 6-dimensional parameter space. We generate the parameter sets by using the WSP space-filling design, resulting in about 200 individual experiments. We have limited ourself to 200 experiments, as our Mininet environment is able to run these within 4 hours. This allows us to quickly obtain the results of the experiments. In order to cope with possible variations, we repeat each parameter set 5 times and use the median to extract the central tendency of the aggregation benefit.

Effect of receive-buffer sizes.

The performance of MPTCP is influenced by the receive-buffer sizes of the end hosts [22]. We evaluate the impact of a fixed receive buffer on the aggregation benefit in the low-BDP and the high-BDP environments. In Figure 3 we show the aggregation benefit's mean (with its standard deviation) and the median, 25% and 75% percentiles as well as the degree of dispersion. We see that the larger the receive buffer, the larger is the aggregation benefit. If the receive buffer is small, the MPTCP session is flow limited and thus cannot use the full capacity of the links, which reduces the aggrega-

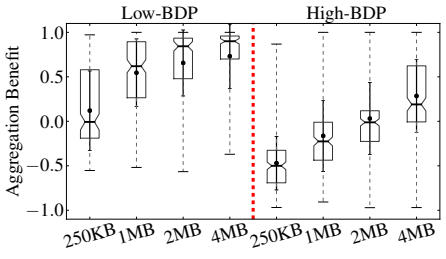

Figure 3: The aggregation benefit increases with the buffer size. If the buffer is small, MPTCP is limited by the receive window and does not fully utilizes the network's capacity.

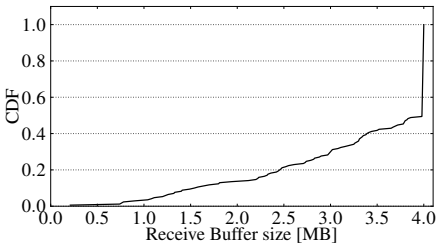

Figure 4: With auto-tuning, 50% of the experiments are able to consume less than 4MB of receive buffer in the low-BDP environment.

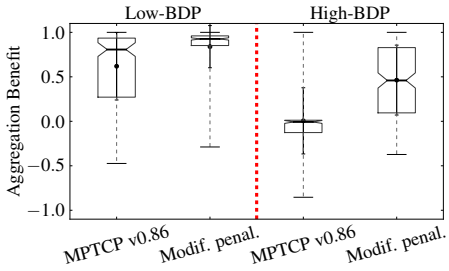

Figure 5: MPTCP v0.86 has a weak performance with auto-tuning. Our modification to the penalization algorithm significantly improves the aggregation benefit.

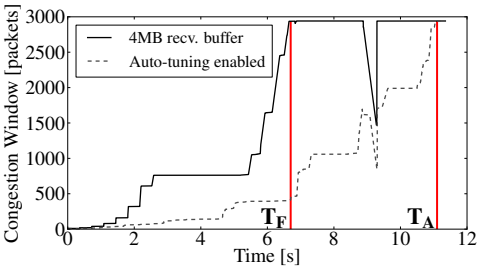

Figure 6: During the slow-start phase of an MPTCP subflow, the congestion window increases much slower when auto-tuning is enabled.

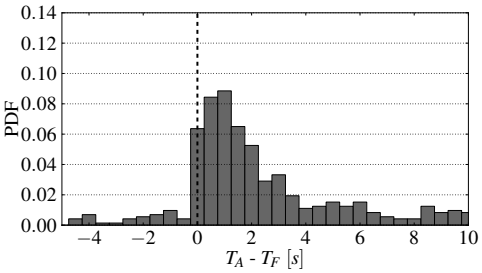

Figure 7: With auto-tuning enabled, MPTCP is slower to increase the congestion window during slow-start compared to a fixed receive buffer at 4 MB.

tion benefit. The results are shown when coupled congestion control [26] is used. In this scenario, OLIA [14] performs similarly to coupled congestion control.

Effect of enabling auto-tuning.

The Linux TCP stack does not use a fixed receive buffer. Instead, it includes an auto-tuning algorithm [25] that adapts dynamically to the size of the receive buffer to achieve high goodput at the lowest possible memory cost. The current MPTCP implementation sets the buffer to $2 * \sum_i^n bw_i * RTT_{max}$, to allow aggregating the throughput of all the subflows [4].

Enabling auto-tuning should reduce the memory requirements of the MPTCP connection. Figure 4 reports the maximum receive buffer used in the low-BDP environment for each set. Indeed, we observe that on 50% of the experiments the receive buffer remains below 4 MB, effectively reducing the memory used by the connection.

However, enabling auto-tuning can also lead to a huge performance degradation with MPTCP. Figure 5 depicts the aggregate benefit of MPTCP v0.86 when auto-tuning is enabled, capping the buffer to Linux's default of 4 MB. We observe that in high-BDP environments the aggregation benefit is significantly smaller than if the receive buffer is fixed at 4 MB (cfr. Figure 4). Effectively, 80% of the experiments have an aggregation benefit below 0. We observe this performance degradation in low-BDP environments too: 25% of the experiments have an aggregation benefit below 0.1.

The auto-tuning at the receiver will make the receive buffer start at a small value at the beginning of the connection, increasing it as the sender's subflows evolve during their slow-start phase. MPTCP evaluates the receive-buffer size every RTT_{max} in order to estimate the sending rate of the peer. $\sum_i^n bw_i * RTT_{max}$ represents the amount of data the peer sends during one RTT_{max}-interval. Multiplying this by 2 to achieve the recommended receive buffer of [4] should allow the sender to increase its sending rate during the slow-start phase. However, subflows whose RTT is smaller than RTT_{max} will more than double their sending rate during an RTT_{max}-interval, effectively making the sender limited by the receive window. As the subflows evolve through their slow-start phase, the announced window will continue increasing and eventually be large enough. Hence, these receive-window limitations are only transient and should not prevent users from achieving a high transmission rate.

MPTCP's reaction to transient receive-window limitations is overly aggressive because of the *"penalization"* mechanism proposed in [22]. This mechanism handles receive-window limitations due to round-trip-time differences among the subflows (e.g., in WiFi/3G environments). When the flow is limited by the receive window, it halves the congestion window of the subflow who causes this receive-window limitation and sets its slow-start threshold to the current congestion window. If an MPTCP-connection experiences the transient receive-window limitations while one of its subflows is in slow-start, the penalization algorithm will give this subflow a false view of the path capacity by adjusting its slow-start threshold to a smaller value.

We propose to modify the penalization algorithm. It should not adjust the slow-start threshold when a subflow is in its slow-start phase. Figure 5 shows how this small modification to the penalization algorithm improves the performance of MPTCP. In the low-

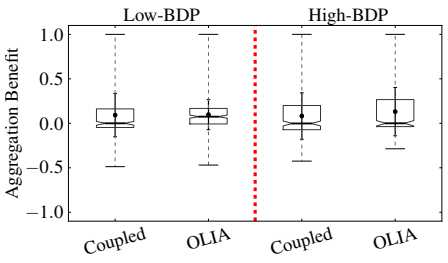

Figure 8: The MPTCP congestion controls move traffic away from congested (aka lossy) paths. Only the less-lossy path is used for MPTCP.

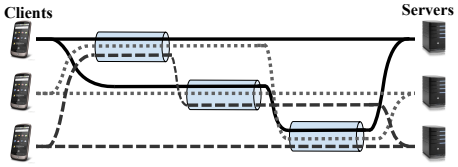

Figure 9: Three bottlenecks are used to evaluate MPTCP's load-balancing performance. Each MPTCP session has a one-hop and a two-hop subflow.

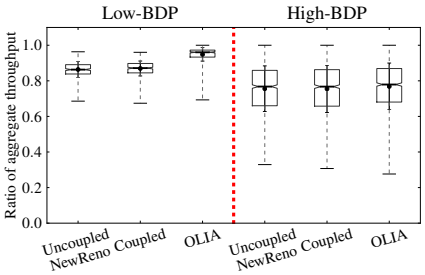

Figure 10: In the low-BDP environments, OLIA is able to efficiently move the traffic away from the congested paths.

BDP environments, more than 75% of the experiments achieve an aggregation benefit of 0.85 or higher. And even the experiments in the high-BDP environments have their median increased up to an aggregation benefit of around 0.5. Long data transfers with MPTCP are now less vulnerable to transient receive-window limitations and achieve a high aggregation benefit.

Effect of transient receive-window limitations.

Our modification to the penalization algorithm mitigates the effect of the transient receive-window limitations for long flows. However, the fundamental problem is still there. Transient receive-window limitations slow down the increase rate of the congestion window during slow-start. We picked a specific set of parameters where these transient receive-window limitations are particularly apparent. The first bottleneck has a capacity of 1 Mbps, while the second has a capacity of 50 Mbps. Both bottlenecks have a high propagation delay and queueing delay. A plot of the congestion-window's evolution of the fastest path during the slow-start phase is shown in Figure 6[2]. When auto-tuning is enabled, the congestion window increases much slower, reaching its maximum at time T_A, about four seconds later compared to a fixed receive buffer at 4 MB (T_F).

We measure the difference between T_A and T_F in our high-BDP environment among the 200 parameter sets generated with the space-filling design. Figure 7 illustrates the PDF of these differences. Ideally, the difference between T_A and T_F should be zero. However, this is only true for 6% of the experiments. For a large portion of the experiments the congestion window reaches its maximum between 500ms and 2000ms faster if auto-tuning is disabled.

This could have a negative impact on flow-completion time of short flows, as the connection is receive-window limited during its slow-start phase. An ideal auto-tuning algorithm should not prevent the sender from increasing the sending rate - even during slow-start.

Effect of losses.

In this section, we study the effect of transmission losses on the performance of MPTCP. This could represent wireless environments where losses occur due to the fading. We model lossy links by adding a loss factor to each bottleneck, effectively increasing the parameter space to a total of 8 factors. These loss-factors range from 0 to 2.5% and are set individually on each bottleneck. The two MPTCP congestion controls (Coupled and OLIA [26, 14]) both try to move traffic away from congested paths. As these loss-based congestion controls interpret a loss as congestion, they move traffic away from lossy subflows. Figure 8 shows that Coupled and OLIA have mostly an aggregation benefit of 0. This confirms that Coupled and OLIA only push traffic on the less lossy of the two subflows, thus moving almost all traffic to the best path.

4.2 Congestion Balancing

In this section, we analyze whether MPTCP satisfies its congestion-balancing design goal. For this purpose, we study the performance of MPTCP in the scenario of Figure 9. The network contains three bottlenecks, and the end-hosts create a total of three MPTCP sessions passing by this network. Each session creates two subflows, one crossing a single-bottleneck and the other passing through two bottlenecks. As discussed in [26], to balance the congestion and hence maximize the throughput for all MPTCP sessions, no traffic should be transmitted over the two-hop subflows. The bottlenecks are influenced by the capacity, propagation delay and queuing delay, effectively emulating the low-BDP and high-BDP environments form Table 1. With three bottlenecks and three factors per bottleneck, we have effectively a 9-dimensional parameter space. We generate about 400 parameter sets with the WSP space-filling algorithm and start iperf-sessions for each MPTCP session.

We evaluate this scenario and show the relation between the aggregated goodput of all MPTCP sessions, compared to the theoretical upper bound in Figure 10. We compare the performance of OLIA and Coupled congestion control with the case that uncoupled NewReno congestion control[3] is used. We observe that OLIA is able to move traffic away from the two-hop subflows in low-BDP environments and hence efficiently uses the capacity available in the network. Coupled fails to provide any load balancing in the network and performs similarly to uncoupled NewReno. This observation confirms previous discussions about performance issues with coupled congestion control [14]. In high-BDP environments, even OLIA fails to provide a good congestion balancing. This is because the large BDP makes one of the three MPTCP sessions become receive-window limited if the three bottlenecks have different characteristics. This flow cannot benefit of OLIA's load-balancing algorithms and leads to suboptimal network utilization.

[2]The Figure shows that even the flow whose receive buffer is fixed at 4 MB experiences a stall in the congestion-window's increase rate (between 2.5 and 5.5 seconds). This is due to a very specific issue of the Linux TCP window handling and has been fixed by http://patchwork.ozlabs.org/patch/273311/. Unfortunately the Linux MPTCP-stack does not yet include this recent fix.

[3]Uncoupled NewReno [12] represents the case where regular TCP congestion control is used on the subflows. It increases the congestion windows of each subflow irregardless of the congestion state of the other subflows that are part of the MPTCP session.

5. SENSITIVITY ANALYSIS

The number of experiments executed within the parameter space influences the accuracy of the results. The more sets explored, the better the accuracy will be. However, CPU-time constraints limit the number of experiments that can be executed. The sensitivity analysis allows to confirm that the number of experiments conducted per parameter space is sufficient to have an accurate view of MPTCP's performance. This can be achieved by generating different space-filling designs, and comparing the 5th, 25th, 75th and 95th percentiles and median among each of these designs.

To evaluate the aggregation-benefit we used three influencing factors per bottleneck link, effectively creating a 6-dimensional parameter space. 200 sets were generated, using the WSP space-filling algorithm. We generate 5 different space-filling designs of comparable size for the sensitivity analysis. Each design explores different sets among the parameter space. Comparing the percentiles and the median of the aggregation benefit for each of the designs has shown that the standard deviation is very low. Relative to the range of the aggregation benefit $(-1, 1)$, it ranges between 0.1% and 2.62%. We can conclude that running 200 experiments is sufficient to have a good overview of the aggregation benefit of MPTCP in our 6-dimensional parameter space. In the load-balancing environment we observe a simimlarly low standard deviation ranging from 0.008% to 1.4%.

6. DISCUSSION

Transport layer protocols like MPTCP, are complex systems, whose performance is influenced by numerous factors. To explore the impact of these factors we use a space-filling design. Our first attempt at applying *"Experimental Design"* techniques to the evaluation of MPTCP allowed us to discover previously unknown performance issues within the Linux Kernel implementation of MPTCP.

We were able to modify the penalization algorithm and validate over our 6-dimensional parameter space, that this modification indeed improves the aggregation benefit. Thanks to the sensitivity analysis, we are confident that the proposed modification works well within the considered domain of each factor.

Further, we identified an issue within MPTCP concerning the auto-tuning algorithm during the slow-start phase. The receiver is not able to accurately estimate the sender's increase rate while its subflows are in slow-start. This makes the sender suffer from transient receive-window limitation issues. A solution for this problem is far from straight forward, as a good compromise between memory utilization, responsiveness and throughput must be found.

One of MPTCP's many benefits is its load-balancing capabilities by coupling the congestion controls. We validated that the load balancing works well in low-BDP environments. However, in high-BDP environments the load balancing does not work well as some subflows are being limited by the receive buffer. This effectively prevents OLIA to balance the load among the subflows.

This paper has demonstrated the benefits of applying experimental design to evaluate and improve the reference implementation of an IETF protocol. Applying it to real implementations enables us to understand the impact of implementation heuristics such as auto-tuning that are often neglegcted in simulation models.

Two directions of further work are clearly open. First, we encourage other networking researchers to apply experimental design techniques to validate their new protocols and algorithms across a wide range of parameters. Second, we plan to perform more detailled experiments with Multipath TCP to include other factors such as background traffic, advanced queuing and packet discard disciplines, flow durations,...

7. ACKNOWLEDGMENTS

This research has received funding from the European Union's Seventh Framework Programme FP7/2007-2013 under Trilogy 2 project (grant agreement 317756), the EU project CHANGE (FP7-ICT-257422) and the IAP BESTCOM network. Many thanks go to Bernadette Govaerts and Catherine Rasse from the UCL-SMCS for their consultancy while investigating the Experimental Design. Finally, we thank our anonymous reviewers as well as our shepherd Richard Mortier for their constructive feedback.

8. REFERENCES

[1] M. Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1), 2012.

[2] M. Allman and A. Falk. On the Effective Evaluation of TCP. *ACM SIGCOMM Computer Communication Review*, 29(5), 1999.

[3] Ö. Andersson. *Experiment! Planning, Implementing and Interpreting*. Wiley, 2012.

[4] S. Barre, C. Paasch, and O. Bonaventure. Multipath TCP: From Theory to Practice. In *IFIP Networking*, 2011.

[5] G. Box and N. R. Draper. *Empirical Model Building and Response Surfaces*. John Wiley & Sons, 1987.

[6] Y-C Chen, Y-S Lim, R. Gibbens, E. Nahum, R. Khalili, and D. Towsley. A Measurement-based Study of Multipath TCP Performance over Wireless Networks. In *ACM SIGCOMM conference on Internet measurement*, 2013.

[7] R. A. Fisher et al. *The Design of Experiments*. Number 5th ed. Oliver and Boyd, London and Edinburgh, 1949.

[8] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC6182, March 2011.

[9] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC6824, January 2013.

[10] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *ACM Queue*, 9(11), 2011.

[11] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible Network Experiments using Container-based Emulation. In *ACM CoNext*, 2012.

[12] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC6582, April 2012.

[13] D. Kaspar. *Multipath Aggregation of Heterogeneous Access Networks*. PhD thesis, University of Oslo, 2011.

[14] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J-Y. Le Boudec. MPTCP is not Pareto-Optimal: Performance Issues and a Possible Solution. In *ACM CoNext*, 2012.

[15] S. Kleijnen, J.and Sanchez, T. Lucas, and T. Cioppa. State-of-the-art Review: a User's Guide to the Brave new World of Designing Simulation Experiments. *INFORMS Journal on Computing*, 17(3), 2005.

[16] M. Morris and T. Mitchell. Exploratory Designs for Computational Experiments. *Journal of Statistical Planning and Inference*, 43(3), 1995.

[17] R. Myers and D. Montgomery. *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*. Wiley, 2009.

[18] H. X. Nguyen and M. Roughan. Rigorous Statistical Analysis of Internet Loss Measurements. *IEEE/ACM Transactions on Networking*, 38(1), 2012.

[19] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. Exploring Mobile/WiFi Handover with Multipath TCP. In *ACM SIGCOMM workshop CellNet*, 2012.

[20] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *ACM SIGCOMM*, 2011.

[21] C. Raiciu, M. Handley, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC6356, October 2011.

[22] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *USENIX NSDI*, 2012.

[23] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and Analysis of Computer Experiments. *Statistical science*, 4(4), 1989.

[24] J. Santiago, M. Claeys-Bruno, and M. Sergent. Construction of Space-Filling Designs using WSP Algorithm for High Dimensional Spaces. *Chemometrics and Intelligent Laboratory Systems*, 113, 2012.

[25] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. *ACM SIGCOMM Computer Communication Review*, 28(4), 1998.

[26] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *USENIX NSDI*, 2011.

[27] B. Zhang, T. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang. Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. *IEEE/ACM Transactions on Networking*, 18(1), 2010.