MPTCP PathFinder

Finding Your Way(s) to Aggregated Bandwidth

Jonas Karlsson, Per Hurtig,
Anna Brunstrom and Andreas Kassler

Faculty of Economic Sciences, Communication and IT

Computer Science

Karlstad University Studies | 2012:59

MPTCP PathFinder

Finding Your Way(s) to Aggregated Bandwidth

Jonas Karlsson, Per Hurtig,
Anna Brunstrom and Andreas Kassler

Karlstad University Studies | 2012:59

MPTCP PathFinder - Finding Your Way(s) to Aggregated Bandwidth

Jonas Karlsson, Per Hurtig, Anna Brunstrom and Andreas Kassler

RESEARCH REPORT

Karlstad University Studies | 2012:59

ISSN 1403-8099

ISBN 978-91-7063-472-7

Distribution:
Karlstad University
Faculty of Economic Sciences, Communication and IT
Computer Science
SE-651 88 Karlstad, Sweden
+46 54 700 10 00

Print: Universitetstryckeriet, Karlstad 2012

MPTCP PathFinder — Finding Your Way(s) to Aggregated Bandwitdh

Jonas Karlsson, Per Hurtig, Anna Brunstrom and Andreas Kassler
{jonas.karlsson, per.hurtig, anna.brunstrom, andreas.kassler}@kau.se

Department of Computer Science, Karlstad University, Sweden

13th December 2012

Abstract

Many networks are multi-path; mobile devices have multiple interfaces, data centers have redundant paths and ISPs forward traffic over disjoint paths to perform load-balancing. Multi-path TCP (MPTCP) is a new mechanism that transparently divides a TCP connection into subflows and distributes them over a host's network interfaces. While this enables multi-homed systems like e.g. smartphones to use several interfaces and thus different, and mostly disjoint, network paths for a single transmission, most end-systems are still single-homed. With one interface, standard MPTCP creates only a single subflow, making single-homed systems unable to benefit from MPTCP's functionality. In this paper we propose PathFinder, an MPTCP extension that tries to estimate the number of subflows required to fully utilize the network capacity, enabling single-homed hosts to reap the benefits of MPTCP. We evaluate MPTCP with PathFinder and compare its performance to standard MPTCP. The evaluation shows that PathFinder is able to open a limited but sufficient amount of subflows to significantly increase the throughput when compared to using standard MPTCP.

Contents

1 Introduction

Multi-path TCP (MPTCP) was recently proposed as an extension to TCP. It enables multi-homed end-systems to simultaneously use all their network interfaces from a single transport layer connection. This increases both robustness and throughput [1] [2]. Typically, MPTCP stripes a connection into one independent subflow per available network interface. MPTCP's default congestion control algorithm is designed to be fair if the communication paths happen to share bottleneck(s). Several researchers have shown that MPTCP leverages performance and robustness benefits for various multi-homed applications including datacenters [3] and vertical mobile/wifi handovers [4].

However, most end-systems are still single-homed, i.e. having one network interface. For such systems splitting flows over different interfaces is obviously of little use. This is unfortunate as today's networks, including most ISP, often use multi-path forwarding to load-balance traffic flows [5] [6] [7]. Thus, it would be possible for single-homed hosts to split a MPTCP connection into several subflows with the goal to have them forwarded over different paths within the network.

Several techniques have been suggested to accomplish striping using MPTCP for single-homed hosts. In [8] the amount of disjoint paths is conveyed to MPTCP using an optional DHCP field, informing MPTCP to open a corresponding number of subflows over a single IP address (using different port numbers for each subflow). An alternative solution has been suggested in [9] and [10] where middleboxes are used to transform regular TCP connections to MPTCP connections. Both approaches have severe drawbacks, requiring e.g. modifications of existing network infrastructure.

When load-balancing traffic on a per-flow basis it is important that the number of subflows is at least as many as the number of bottleneck disjoint paths. If the number of subflows is less, capacity will be unused and it will not be possible to effectively load-balance the traffic. Creating too many subflows is less of a problem since MPTCP's coupled congestion control will evenly distribute the load on all subflows. However, for the end-hosts using many subflows can have negative effects on CPU-load and memory consumption [11].

In this paper we present PathFinder, an algorithm that estimates the number of subflows required to utilize all bottleneck disjoint paths in the network based on knowledge already available to MPTCP. Basically, PathFinder works by opportunistically establishing new subflows as long as the average maximum observed throughput, in steady-state transmission, increases with a certain factor.

Compared to other solutions, PathFinder is a sender-only modification to MPTCP. Thus, there is no need for cross-layer signaling or network modifications for PathFinder to work. Furthermore, PathFinder is a general algorithm suitable for all types of networks where traffic might be routed over disjoint paths.

To evaluate PathFinder we implemented it in Linux 3.2 and conducted experiments in a wireless mesh network (WMN). WMNs are considered to be a new and promising technique to provide a wireless access network for mobile clients to connect to the Internet. For capacity reasons the traffic inside a WMN is often load-balanced over multiple paths. However, as the mobile clients rarely have more than one interface, there is only a single link between a client and the WMN, despite the WMN as a whole being multi-path capable. Our evaluation shows that PathFinder is able to estimate the amount of subflows required to saturate all bottleneck disjoint paths and thereby increases the throughput of a single MPTCP connection significantly.

The rest of this paper is structured as follows. Section 2 describes the main differences between TCP and MPTCP, as well as the benefits and drawbacks of using MPTCP instead of regular TCP. Section 3 presents the PathFinder algorithm and how it is integrated with MPTCP. Section 4 describes the environment in which PathFinder was evaluated and how the experiments were designed. Section 5 presents the results from the evaluation, showing PathFinder's ability to determine the amount of subflows to use and the impact of the different parameters on performance. Section 6 presents related work and section 7 concludes the paper.

2 Multi-Path TCP

This section shortly summarizes the key functionality of MPTCP and presents benefits and drawbacks when using it as a transport layer protocol.

2.1 Key Features

MPTCP is a major TCP extension that allows a single connection to be striped in separate subflows across multiple paths, transparent to applications. MPTCP's capability and parameters are negotiated whenever two TCP peers perform their initial handshake. After connection establishment, subflows can be created between peers. Additional subflows can be added by both peers. A new subflow can use the same pair of IP addresses as an earlier subflow, using different ports, or it can make use of any additional IP address that is available on either peer.

Some effort has been made to construct mechanisms to allow MPTCP to use several subflows on single-homed hosts [8] [9] [10], as we will further discuss in section 6. Research has also been conducted to establish the typical number of subflows to open, per interface, in specific environments like datacenters [3]. However, to the best of our knowledge, there is no general transport-level mechanism that is able to estimate the number of subflows to open. In section 3 we present PathFinder, an algorithm that tries to accomplish exactly that, using information already available to MPTCP.

When multiple subflows have been created, the sending peer is responsible for striping data across all subflows. To mitigate reordering caused by the striping, additional TCP options are used by the receiver to reconstruct the original send order. As each MPTCP subflow is a TCP connection of its own, it maintains its own sequence number space and congestion window so that it can adapt to conditions along the path. Although each subflow maintains and uses its own congestion control parameters, MPTCP tries to link all subflows and steer traffic away from congested paths by using the algorithm specified in equation 1 [12].

$$w_r = \begin{cases} w_r + \min\left(\frac{\alpha}{w_{\text{total}}}, \frac{1}{w_r}\right) & \forall \text{ ack on subflow } r, \\ w_r - \frac{w_r}{2} & \exists \text{ loss on subflow } r. \end{cases} \tag{1}$$

$$\alpha = w_{\text{total}} * \frac{MAX(\frac{w_{1\ldots n}}{RTT^2_{1\ldots n}})}{(SUM(\frac{w_{1\ldots n}}{RTT_{1\ldots n}}))^2}. \tag{2}$$

In equation 1, w_r is the congestion window on subflow r, w_{total} is the sum of the windows on all subflows and α determines the aggressiveness of all the subflows. α is calculated according to equation 2 [12]. There are two key parts to this algorithm. First, by making the window increase depend on the total window size, subflows that have large windows increase faster than subflows with small windows. This actively moves traffic from more congested paths to less congested ones and thereby load-balances the network. Second, by adapting α, MPTCP can compensate for different RTTs and can ensure that if all the subflows of a connection traverse the same bottleneck, they will compete fairly with a regular TCP flow. However, if the subflows encounter multiple unloaded paths, one connection can fill them all. The design of the coupled congestion control algorithm has been detailed in [13].

2.2 Benefits

By using multiple subflows that share a common congestion controller, MPTCP implements transport-level resource pooling. For non-trivial network topologies, resource pooling has several benefits, e.g.:

1. *Increased Robustness*

   If path failure occurs when multiple paths are used simultaneously, MPTCP will automatically react by diverting traffic through the paths that still work. Basically, the failure of a path looks like severe congestion to the transport-layer, and MPTCP's reaction is to shift traffic to other paths using its congestion controller (see above). MPTCP also improves robustness in mobility scenarios by enabling handovers, using separate simultaneous subflows for the old and new paths.

2. *Better Control of Congestion*

   Using multiple subflows simultaneously, MPTCP's congestion control will automatically balance traffic over the available paths to avoid congestion. Compared to TCP's congestion control, which only lowers the sending rate over congested paths, MPTCP actually moves traffic from such paths. Furthermore, MPTCP also enables network operators to perform fast traffic engineering using ECN-markings to steer traffic away from certain links.

3. *Maximized Utilization*

   The most obvious benefit of using multiple paths simultaneously is the throughput increase resulting from the aggregated capacities of the paths.

2.3 Drawbacks

Striping a connection over several independent subflows requires support functionalities. For instance, if the subflows of a MPTCP connection traverse paths with different characteristics, data can be reordered. This type of reordering does not affect transport-level performance, as each subflow is an independent TCP connection. However, to reconstruct the original send order an extra level of sequence numbering is introduced. This apporach has several consequences in terms of performance. First, MPTCP generally requires more memory as the degree of buffering increases with the number of subflows [11]. Second, stream reconstruction and other new processes introduced by MPTCP require more processing power than regular TCP [11]. However, using less subflows than available bottleneck disjoint paths has, as described before, a negative impact on throughput. Therefore, it is important to keep the number of subflows close to the number of bottleneck disjoint paths.

3 PathFinder

In this section we present how PathFinder estimates the number of subflows to saturate all bottleneck disjoint paths, and how the algorithm interacts with MPTCP.

3.1 Overview

The PathFinder algorithm builds on two basic assumptions:

1. The core network load-balances traffic on a per-flow basis.

2. If two subflows share a path they will achieve similar throughput over that path as if only a single subflow was using that path [12].

These assumptions are valid in any network that do per-flow load-balancing and the transport layer is using a congestion control algorithm that fulfills the three goals of MPTCP's coupled congestion control [12].

Building on these assumptions, PathFinder tries to determine the appropriate number of MPTCP subflows to open by opportunistically opening new subflows as long as they contribute to the total throughput. Due to this and to be able to opportunistically exploit new paths being available in the network, i.e., when the load-balancer opens additional paths to ease congestion, PathFinder will open, at least one, more subflow than the amount of currently available bottleneck disjoint paths.

When competing traffic leaves the network the total throughput can increase, which gives PathFinder an erroneous indication that a new path is available in the network. Competing traffic that enters the network can also temporally create erroneous indications, as it takes time for the network to stabilize. Intuitively, this could be eliminated by measuring the throughput of each subflow, e.g. see if it is the last subflow that contributed to the total throughput or not. This is, however, problematic, as there is no way of knowing if it is the last opened subflow that is routed over a new path or if the load-balancer have moved an older subflow to a new path.

3.2 Algorithm

In the following we describe our PathFinder algorithm, which tries to open new subflows as long as they contribute to a throughput increase. The general functionality of the algorithm is to compare the throughput before and after opening a new subflow. If the maximal throughput, after opening a new subflow, has increased more then a predefined threshold, an additional subflow is opened.

To compensate for traffic variations, we have used a twofold approach; first we require a predefined increase of the throughput and, secondly, we require the average throughput to be stable over a certain time period to open a new subflow. Therefore, PathFinder introduces two variables (β, γ) to MPTCP. The β variable governs the required throughput increase to allow the establishment of new subflows. This allows for short term throughput fluctuations without influencing the opening of new subflows, at the cost of reduced sensitivity to discovering paths that only marginally increase the total throughput. β also caters for the cases where the bandwidth measurements show a slightly increased throughput due to reductions in competing traffic and not due to new paths being opened.

γ controls how responsive vs. accurate PathFinder should be in case of traffic fluctuations. It controls the time (in RTTs) where the average throughput increase must be higher than β in order to open a new subflow. This both smooths transient spikes and gives the traffic more time to converge. It thereby improves the accuracy of the throughput measurements at the expense of swiftness of opening new subflows.

Although both β and γ indirectly control how many subflows MPTCP will open, the implications of them are slightly different. β limits the total amount of subflows that can be opened with respect to the capacity and number of paths available. Whereas γ, only postpones the creation of subflows but not limits the total amount of subflows over time.

In the following we give a detailed description of the PathFinder algorithm. Given the variables β and γ, the PathFinder algorithm is implemented as outlined in algorithm 1; For every RTT of the main subflow, PathFinder checks if all subflows are in steady-state, i.e., congestion avoidance (line 1).

Algorithm 1 PathFinder (PF)

```
INIT()
    // RTT counter for the probe period
    $PF_{probe} = -1$
    // Maximum average bandwidth seen
    $BW_{max} = 0$
    // Aggregated Bandwidth during the probe period
    $BW_{probe} = 0$

PATHFINDER()
     // for every RTT of the main subflow
 1   if $MPTCP_{subflow}[1..n] == steadystate$
 2       $BW_{cur}$ = BANDWIDTH($MPTCP_{subflow}[1..n]$)
 3       if $BW_{max} == 0$
             // First time do not wait $\gamma$ RTTs
 4           $BW_{max} = BW_{cur}$
 5           OPENSUBFLOW() // $MPTCP_{subflow}[n+1]$
 6       else
 7           if $PF_{probe} > 0$
                 // in probe phase
 8               $PF_{probe} = PF_{probe} - 1$
 9               $BW_{probe} = BW_{probe} + BW_{cur}$
10           if $PF_{probe} < 0$ and $BW_{cur} > BW_{max} * \beta$
                 // start new probe phase
11               $BW_{probe} = BW_{cur}$
12               $PF_{probe} = \gamma$
13           if $PF_{probe} == 0$
                 // probe phase finished
14               $PF_{probe} = -1$
15               $BW_{avg} = BW_{probe}/(\gamma + 1)$
16               if $BW_{avg} > BW_{max} * \beta$
17                   $BW_{max} = BW_{avg}$
18                   OPENSUBFLOW()
```

If so, when entering the algorithm for the first time there is no data available for comparison. Therefore, when the main subflow has reached steady-state for the first time (line 3) we immediately open a new subflow.

Once we have at least two subflows (line 6); PathFinder check if it is in the probing phase (line 7), if so, decrement the probe counter by one (line 8) and add the current bandwidth to the aggregate probe measure (BW_{probe}). If not in the probe phase, check if the current total bandwidth of all subflows (line 2)

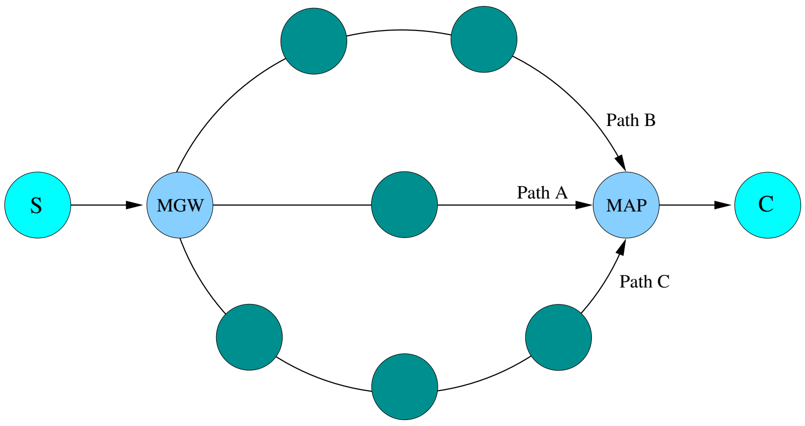

Figure 1: Experimental WMN Topology

is larger than the threshold ($BW_{max} * \beta$) (line 10). If so we store the current bandwidth estimate (line 11) and start a new probe phase consisting of γ RTTs.

After that, PathFinder checks if the probe phase is complete (line 13). If so, PathFinder computes the average bandwidth (BW_{avg}) during the probe period (line 16). If the average bandwidth is greater than the threshold ($BW_{max} * \beta$), the average bandwidth is stored as the maximum average bandwidth seen (BW_{max}) (line 17) and a new subflow is opened (line 18).

4 Evaluation of PathFinder

Several environments are suitable for evaluating PathFinder, e.g. access networks and data-centers. In this paper a wireless mesh network (WMN) is used. WMNs often provide significant path diversity inside the core to avoid bottlenecks. However, client access to WMNs is often restricted to a single interface, making it very suitable for PathFinder. In the following subsections we describe how WMNs work and our experimental environment.

4.1 MPTCP in Wireless Mesh Networks

Traditionally, wireless clients connect to access points connected to the Internet via a wired back bone or core network. WMNs can replace the wired architecture with a wireless core network. For capacity and deployment reasons a WMN is normally built as a flat hierarchy with multiple possible paths between the nodes. Each node is also equipped with multiple radio interfaces tuned to different channels. However, clients typically attach to one WMN access point at a time, creating a single last hop link despite the network as a whole being multi-path capable.

The link layer capacity of a WMN is often similar to the client access capacity, as they use a similar MAC/PHY layer. Therefore, it is not uncommon that bottlenecks are located inside the WMN. This has spurred the development of specialized wireless multi-path routing protocols for load-balancing the traffic over the WMN and reducing the possibilities for bottlenecks to form [14].

With flow-based forwarding, when the amount of flows inside a WMN is less than the number of available paths, it is not possible to evenly distribute traffic which in turn increases the risk of bottlenecks. This is shown in e.g. [15] where a full utilization of the network requires at least as many flows as there are available bottleneck disjoint paths. The key to an efficient utilization is therefore to open at least as many subflows as there are bottleneck disjoint paths.

4.2 MPTCP Experiments

The topology used for the experiments is shown in figure 1. As shown, the client (C) is connected to the WMN via a mesh access point (MAP) and can reach the server (S), connected to a mesh gateway (MGW), using three alternate paths. All link segments were configured on orthogonal channels to create three interference disjoint paths, excluding the access links.

To conduct the experiments we used ns-3 [16] in emulation mode to support the use of real end-hosts over a simulated network. The end-hosts ran Ubuntu 12.04 using Linux kernel 3.2 with the MPTCP implementation from [11] extended with the PathFinder algorithm. The simulated network used the default ns-3 IEEE 802.11 MAC/PHY layer extended with a flow-based forwarding algorithm based on the layer 2.5 routing (L2.5R) algorithm [14]. In this paper, L2.5R was configured to forward different MPTCP subflows over the different paths in a round-robin fashion, starting with the shortest available path. Furthermore, the network was configured with a high capacity access link between the client and the gateway that did not limit the performance. Thus the number of bottleneck disjoint paths in this topology are three. Each experiment was repeated 30 times.

5 Experimental Results

First, we evaluate the performance of standard MPTCP and compare it with the performance of MPTCP with PathFinder.

5.1 Using Standard MPTCP

When using standard MPTCP (one subflow) it is impossible to utilize the entire system capacity as only a single path will be used. This is shown in figure 2, where the solid line depicts the total system throughput as a function of the number of MPTCP connections. We evaluated MPTCP, using both one subflow and three subflows. Using three MPTCP connections with one subflows or one MPTCP connection with three subflows will create the same amount of flows in the network (utilizing all bottleneck disjoint paths) and

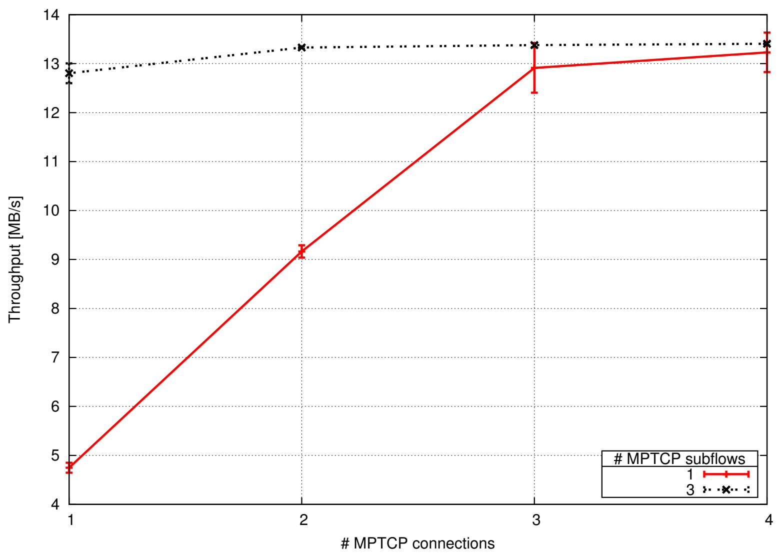

Figure 2: The average aggregated throughput using standard MPTCP. The number of MPTCP connections where varied between one and four, using one and three MPTCP subflows. Each point in the graph is an average of 30 repetitions. The errorbars indicate the 95% confidence intervals.

therefore also achieve a similar throughput. Increasing the number flows in the network beyond three, e.g. using four MPTCP connections or two MPTCP connections with three subflows, can slightly increase the total throughput further. An analysis of this behavior is out side the scope of this paper and we will in the rest of this paper use the term “Optimal” for MPTCP with three subflows and “Standard” to denote MPTCP with one subflow.

5.2 Using MPTCP with PathFinder

With knowledge of the underlying topology it is trivial to configure MPTCP to open enough subflows to utilize all bottleneck disjoint paths. However, in the general case it is difficult at the end node to know the entire network topology and thus hard to configure MPTCP accordingly. Thus, it is crucial to be able to automatically estimate the number of subflows required without global network knowledge.

5.2.1 Estimating the Available Number of Subflows

Figure 3 shows the number of subflows opened by the PathFinder algorithm. We varied the β variable, while γ was set to zero for these experiments. The graph shows the accuracy of PathFinder given different traffic loads, ranging from no competing traffic to five simultaneous MPTCP connections. One key observation is that when the amount of MPTCP connections (and flows)

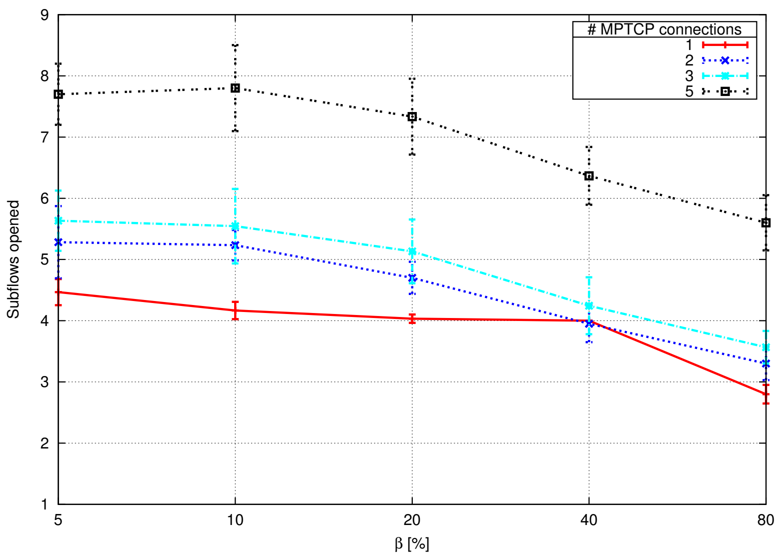

Figure 3: The average amount of estimated subflows per MPTCP connection. The β variable controlled the throughput increase required to open a new subflow, while γ was set to zero for these experiments. Each point in the graph is an average of 30 repetitions. The errorbars indicate the 95% confidence intervals.

increases, PathFinder overestimates the number of subflows. The reason for this is that traffic fluctuations, due to congestion and MPTCP's congestion control, are larger than the β threshold, especially for small values of β.

The goal of PathFinder, as stated in section 3, is to conclude the estimation when all subflows have reached steady state and the addition of a new subflow does not increase the total throughput. At that point the amount of subflows should correspond to at least the number of disjoint bottleneck paths plus one. Opportunistically opening one subflow more than the number of disjoint bottleneck paths, enables PathFinder to detect when a load-balancer starts utilizing additional paths in the network. Thus, for our topology at least four opened subflows per MPTCP connection is the goal.

As indicated by figure 3 PathFinder fulfills this goal for a single MPTCP connection when $10 < \beta \leq 40$. For $\beta \leq 10$ or with competing traffic, the traffic variations make PathFinder slighlty overestimate the number of subflows required. For $\beta > 40$, PathFinder underestimates the available number of paths as the required throughput increase can not be reached when adding new subflows.

In theory when all paths are equal in terms of capacity and the network is unloaded, β can be configured according to equation 3.

$$\beta = \frac{100}{|P|-1}, P \text{ set of bottleneck disjoint paths.} \tag{3}$$

In our topology, where these assumptions almost holds [1] for one MPTCP connection, the optimal $\beta \approx 50\%$. However, setting β to such a high value would limit PathFinder to only detect three paths, according to equation 3. Actually, [7] shows that load-balanced environments often contain five or less partially disjoint paths between sender and receiver. Using equation 3, five or less paths gives us: $|P| = 5 \Rightarrow \beta = 25\%$.

However, in practice where not all paths are equal, β can be seen as a parameter that weigh the computational cost of adding a new subflow to the benefit of finding a new path and thereby achieve higher throughput. Therefore, combining a $\beta \approx 20$ with an appropriate setting of γ would enable PathFinder to open a sufficient number of subflows while requiring each subflow to at least add a 20% throughput increase.

As mentioned earlier, with a higher amount of MPTCP connections the traffic dynamics can cause the throughput of individual MPTCP connections to have an average variance larger than β. This causes an overestimation of the number of available paths. To mitigate this problem we can increase the robustness of PathFinder by increasing γ, which controls how many RTTs an average throughput increase given by β has to be stable before a new subflow is opened. This makes PathFinder more robust to temporary traffic fluctuations.

In figure 4, we show the average number of opened subflows per MPTCP connection as a function of γ for different number of MPTCP connections using a fixed β of 20%. For a single MPTCP connection, PathFinder reaches the goal of four subflows when $0 < \gamma \leq 8$. Although, PathFinder with five MPTCP connections overestimates the number of subflows, the amount of overestimation is reduced with an increasing γ.

5.2.2 Performance Impact of Subflow Estimation

In figure 5, we show the average throughput as a function of β using $\gamma = 0$. In our topology, for every path that is not found, throughout is reduced by $\approx \frac{1}{3}$. Therefore, there will be a slight performance hit when using < 3 connections, as it takes time before all paths can be utilized. Recall that PathFinder always starts by using one subflow and adds subflows by a rate of maximum $\frac{1}{(\gamma+1)}$ subflow per RTT. The time it takes to open all (required) subflows is outlined in equation 4, where P is the set of bottleneck disjoint paths, T_{open} is the time to open all subflows, RTT_{main} is the round trip time for the main subflow and $T_{SS_{subflow[x]}}$ is the time it takes for subflow x to enter steady state.

$$T_{open} = T_{SS_{subflow[1]}} + (T_{SS_{subflow[2..n]}} + \gamma * RTT_{main}) * (|P|-2), |P| > 1 \tag{4}$$

[1]As the paths have different lengths, the TCP throughput varies slightly (< 13%) between the paths.

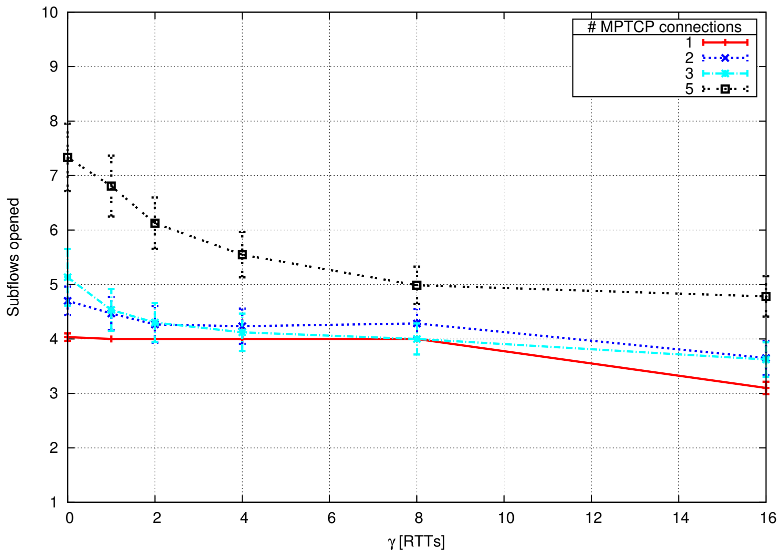

Figure 4: The average amount of estimated paths per MPTCP connection. β was set to 20%, while γ was varied. Each point in the graph is an average of 30 repetitions. The errorbars indicate the 95% confidence intervals.

In our topology the second flow was opened after $\approx 5.5s$ and the third after $\approx 12s$ when $\gamma = 0$. Due to the delay before opening new subflows, there will be a slight performance hit compared to using ≥ 3 MPTCP connections, as here all paths will be fully utilized from the start. In the graph, the throughput for one MPTCP connection is not impacted while $\beta < 80$. After this point, the throughput is reduced since on average not all paths can be utilized. This can also be observed in figure 3 where $\beta \geq 80$ results in less than three subflows being opened.

Throughput wise, overestimation is less of a problem since the throughput will be equally divided among the subflows sharing one path. This is evident from figure 5, where with five MPTCP connections the network always will contain more (sub)flows than the number of available paths but there is no reduction in throughput due to this. The throughput when using PathFinder is close to the optimal with MPTCP with three subflows and much higher than using standard MPTCP with one subflow. However, as shown in [11], the memory and CPU consumption increases with the number of subflows, making it preferable to reduce the overestimation to a minimum.

Because β does not delay the opening of new subflows. β has only a marginal impact on the throughput, as long as it does not limit the amount of opened subflows to less than the number of bottleneck disjoint paths. γ does not limit the total amount of subflows found, but will add a delay according to equation 4, before all subflows are opened. During this time not all paths can be utilized, which reduces the total achievable throughput, especially for

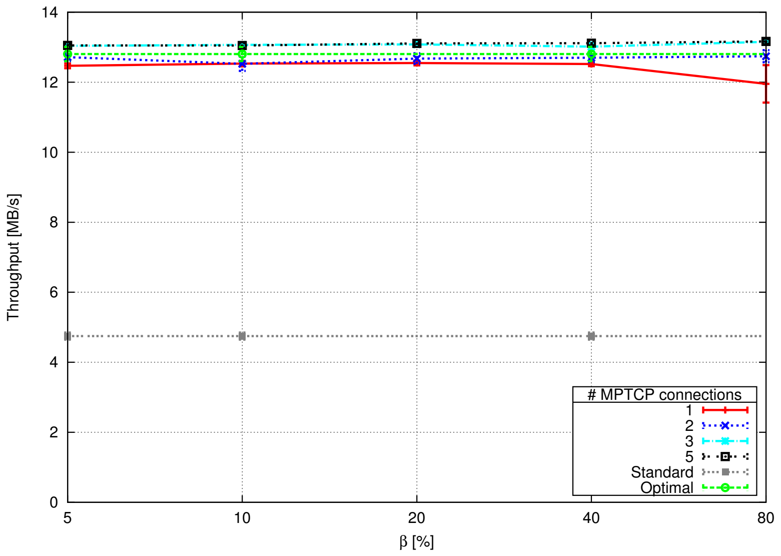

Figure 5: The average aggregated throughput. The β variable controlled the throughput increase required to open a new subflow, while γ was set to zero for these experiments. Each point in the graph is an average of 30 repetitions. The errorbars indicate the 95% confidence intervals.

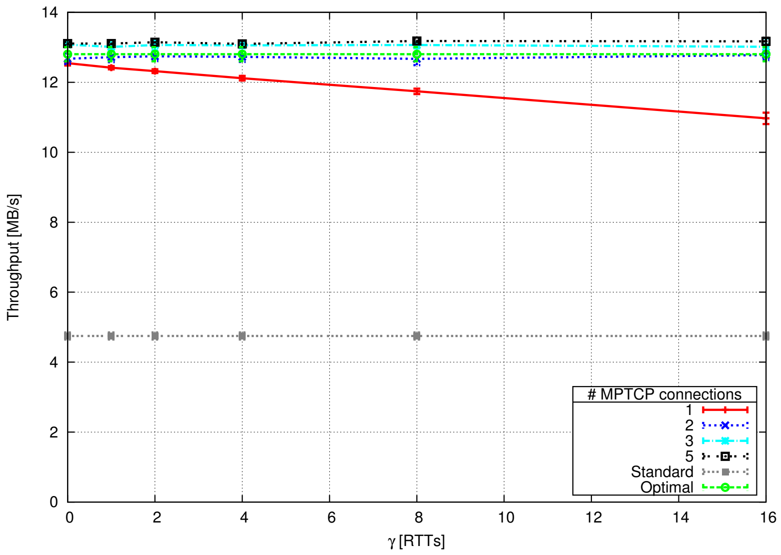

Figure 6: The average aggregated throughput. β was set to 20%, while γ was varied. Each point in the graph is an average of 30 repetitions. The errorbars indicate the 95% confidence intervals.

short connections. In figure 6, we show the average aggregate throughout as a function of γ using $\beta = 20$ for different number of MPTCP connections. When increasing $\gamma > 1$, the throughput for one MPTCP connection is reduced, even though the amount of opened subflows is unchanged until $\gamma \geq 8$. In our topology, increasing γ from 0 to 8 also increases the average time it takes to open three subflows by three times. However, when the amount of MPTCP connections is increased there is less or no drawback of using a high $gamma$, since each connection creates at least one subflow.

6 Related Work

Several techniques have been suggested to accomplish traffic striping for single-homed hosts. In [8] the amount of disjoint paths is conveyed to MPTCP using an optional DHCP field, informing MPTCP to open a corresponding number of subflows over a single IP address (using different port numbers for each subflow). An alternative solution has been suggested in [9, 10] where middleboxes are used to transform regular TCP connections to MPTCP connections.

There are several problems with the above approaches. Using the DHCP approach in [8], both DHCP servers and clients need to be updated and cross-layer information needs to be conveyed to MPTCP. Furthermore, this approach requires more intelligence to be put into the core network. The middlebox approach in [9, 10] adds a lot of problems. First, parts of the transport-layer are pushed into the core network causing well-known "middlebox problems" like poor transport security, mobility problems, tight coupling between end-host and core network and also complicates further MPTCP development, e.g. using specific MPTCP-socket options.

Common for the above mentioned solutions is that they unnecessarily try to exactly match the amount of disjoint paths with an equal amount of subflows. The key is instead to not underestimate the amount of available bottleneck disjoint paths, as that will limit the performance significantly. To overestimate the amount of disjoint paths is, however, not as detrimental to the network performance as MPTCP's coupled congestion control will balance the traffic among the subflows accordingly. However, for the end-hosts an overestimation can have negative effects on CPU-load and memory consumption [11]. It is therefore still important to limit the overestimation and respect the hardware limitations of the end-nodes. With PathFinder the cost of adding a new subflow can be weighted against the benefit of opening all required subflows by setting an appropriate β parameter based on equation 3.

7 Conclusions and Future Work

MPTCP is a new mechanism that transparently divides a TCP connection into subflows and distributes them over a host's network interfaces. While this enables multi-homed systems to use mostly disjoint paths for a single transmission, most end-systems are still single-homed. To enable single-homed systems to reap the benefit of MPTCP, we in this paper propose PathFinder, a general algorithm for estimating the number of subflows required to fully utilize the network capacity. We evaluate PathFinder by using a Linux 3.2 kernel implementation of the algorithm and conduct experiments using real traffic sent over a simulated WMN topology. The evaluation shows that PathFinder is able to open a limited but sufficient amount of subflows to fully utilize all bottleneck disjoint paths. This increases the throughput significantly when compared to using standard MPTCP, while at the same time limiting the overhead of subflow processing.

Improving the accuracy of PathFinder by increasing γ can lead to a slight delay before all subflows are opened, which is mostly affecting short MPTCP connections. As future work we have left to evaluate the possibility to automatically adapt γ (and/or the sampling rate) depending on the length of the MPTCP connection.

References

[1] A. Ford, C. Raiciu, M. Handley, S. Barré, and J. Iyengar. Architectural guidelines for multipath TCP development. *Internet RFCs, ISSN 2070-1721, RFC 6182*, March 2011.

[2] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. Technical report, Internet Engineering Task Force, June 2012. draft-ford-mptcp-multiaddressed-09.

[3] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of the ACM SIGCOMM Conference*, Toronto, Canada, August 2011.

[4] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. Exploring Mobile/WiFi handover with multipath TCP. In *Proceedings of the ACM SIGCOMM Workshop on Cellular Networks (CellNet '12)*, Helsinki, Finland, August 2012.

[5] Jon C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6), December 1999.

[6] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2), April 2007.

[7] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the Internet. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC '07)*, San Diego, USA, March 2007.

[8] R. Winter and A. Ripke. Multipath TCP support for single-homed end-systems. Technical report, Internet Engineering Task Force, July 2012. draft-wr-mptcp-single-homed-03.

[9] T. Ayar, B. Rathke, L. Budzisz, and A. Wolisz. TCP over multiple paths revisited: Towards transparent proxy solutions. In *Proceedings of the IEEE International Conference on Communications (ICC '12)*, Ottawa, Canada, June 2012.

[10] T. Ayar, B. Rathke, L. Budzisz, and A. Wolisz. A transparent performance enhancing proxy architecture to enable TCP over multiple paths for single-homed hosts. Technical report, Internet Engineering Task Force, February 2012. draft-ayar-transparent-sca-proxy-00.

[11] C. Raiciu, C. Paasch, S. Barré, A. Ford, M. Honda, F. Duchène, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *Proceedings of the 9th USENIX Symposium of Networked Systems Design and Implementation (NSDI '12)*, San Jose, USA, April 2012.

[12] C. Raiciu, M. Handley, and D. Wischik. Coupled congestion control for multipath transport protocols. *Internet RFCs, ISSN 2070-1721, RFC 6356*, October 2011.

[13] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*, Boston, USA, March/April 2011.

[14] S. Avallone, I.F. Akyildiz, and G. Ventre. A channel and rate assignment algorithm and a layer-2.5 forwarding paradigm for multi-radio wireless mesh networks. *IEEE/ACM Transactions on Networking*, 17(1), February 2009.

[15] J. Karlsson, P. Hurtig, A. Brunstrom, A. Kassler, and G. Di Stasi. The interaction between tcp reordering mechanisms and multi-path forwarding in wireless mesh networks. In *Proceedings of the 8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob '12)*, Barcelona, Spain, October 2012.

[16] T. Henderson, S. Roy, S. Floyd, and G.F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator (WNS2 '06)*, Pisa, Italy, October 2006.

MPTCP PathFinder

Many networks are multi-path; mobile devices have multiple interfaces, data centers have redundant paths and ISPs forward traffic over disjoint paths to perform load-balancing. Multi-path TCP (MPTCP) is a new mechanism that transparently divides a TCP connection into subflows and distributes them over a host's network interfaces. While this enables multi-homed systems like e.g. smartphones to use several interfaces and thus different, and mostly disjoint, network paths for a single transmission, most end-systems are still single-homed. With one interface, standard MPTCP creates only a single subflow, making single-homed systems unable to benefit from MPTCP's functionality. In this paper we propose PathFinder, an MPTCP extension that tries to estimate the number of subflows required to fully utilize the network capacity, enabling single-homed hosts to reap the benefits of MPTCP. We evaluate MPTCP with PathFinder and compare its performance to standard MPTCP. The evaluation shows that PathFinder is able to open a limited but sufficient amount of subflows to significantly increase the throughput when compared to using standard MPTCP.

ISBN 978-91-7063-472-7

ISSN 1403-8099

Karlstad University Studies | 2012:59