Exploiting Multipath Congestion Control for Fun and Profit

Matei Popovici and Costin Raiciu
University Politehnica of Bucharest

1. INTRODUCTION

Multipath TCP (MPTCP) is a drop-in replacement for TCP that can use multiple paths in a single transport connection to improve reliability and throughput. MPTCP has been recently standardized [4] and shown to offer throughput improvement in datacenter networks [12], wide-area transfers and for mobile clients. Recently, Multipath TCP adoption is gaining pace: it has been deployed by Apple on all IOS devices and Samsung (initially in Korea), as well as by OVH and Swisscom to bond DSL and LTE links.

The key ingredient of Multipath TCP is the congestion control algorithm [17] that aims to efficiently allocate network resources across different multipath flows. To achieve this goal, all multipath congestion controllers prefer less congested links: traffic sent via lossy paths will be reduced to a minimum (in theory) as long as there are less congested alternative paths. This enables MPTCP to get near-optimal throughput in network topologies with non-equal cost paths between endpoints, such as the BCube [5] or Jellyfish [15] datacenter network topologies and most operator networks.

MPTCP assumes the loss signal it receives from the network is an indication of congestion. In this paper we ask whether network operators can "game" the loss signal to convince multipath traffic crossing their network to travel on alternative paths. Doing so would reduce the amount of MPTCP traffic network operators carry, reducing their costs. Creating artifical loss to fool MPTCP is not trivial: the operator must take care not to penalize TCP traffic or MPTCP traffic that does not have good alternative paths, while obbeying net-neutrality requirements.

We show that the MPTCP congestion controller can be gamed with a technique we call *policy drop*. Policy drop adds artificial loss to *all* TCP and MPTCP traffic based on the client's access link speed and the connection RTT, and is thus net-neutral according to the current legislation. We have implemented policy drop both in simulation and in the Click modular router. Our experiments show that policy drop has no effect on the throughput of TCP or single-path MPTCP traffic, and significantly reduces its buffering delay for low-RTT connections. Policy drop is very effective for MPTCP traffic: when the client has two access links of the same speed and one enables policy drop, traffic on that link is reduced by 60%-80%.

In the second part of our paper we rely on game theory to understand the effect policy drop might have on the Internet ecosystem. Our findings indicate that there are economic advantages for operators that embrace policy drop, which could lead to this mechanism being used in practice.

As with other operator traffic engineering approaches, policy drop is highly controversial. While operators may view it as a useful tool to manage their traffic, users will be upset if policy drop is applied without prior consent, despite the fact that it is net-neutral. Beyond unveiling policy drop, this paper does not advocate for its deployment: it simply tries to understand whether it might work and whether there are economic incentives for people to use it. We also briefly discuss how users could detect or circumvent policy drop.

HotNets-XV, November 09-10, 2016, Atlanta, GA, USA
© 2016 ACM. ISBN 978-1-4503-4661-0/16/11…$15.00
DOI: http://dx.doi.org/10.1145/3005745.3005769

2. MULTIPATH CONGESTION CONTROL

Multipath congestion control aims to efficiently allocate resources in a network where connections span multiple paths. The stated goals for the Multipath TCP congestion controller are: a) do not harm existing TCP traffic, b) achieve at least as much throughput as TCP on the best path available to MPTCP and c) use efficient paths [17]. The starting point of all practical multipath congestion control algorithms is the theoretical work by Kelly and Voice that have proposed a class of congestion control algorithms where senders regulate the transmission rates based only on loss information received from the network[6].

The key idea behind the Kelly Voice algorithm is that senders send most traffic on the paths with lowest loss rates, and only probe other paths; this is reflected by the goal to use efficient paths. Based on the Kelly Voice ideas, a flurry of congestion control algorithms have been proposed for use with Multipath TCP: LIA [17], OLIA [7] or BALIA [11], to name only the most important ones. All these algorithms prefer paths with the lowest congestion, but also aim to ensure that MPTCP never receives less throughput than single path TCP on the same path. This goal is achieved by estimating the throughput TCP would receive on one path and ensuring that MPTCP achieves at least that in aggregate.

To understand how Multipath TCP congestion control works in practice, the example in Figure 1 shows a client with two 10Mbps DSL links from different providers. The client uses MPTCP to download data from a server. We run an experiment where we vary the number of competing TCP flows

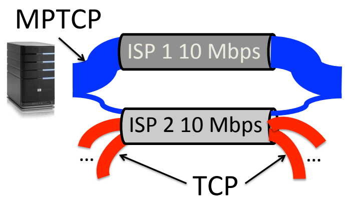


Figure 1: Resouce pooling example for MPTCP congestion control.

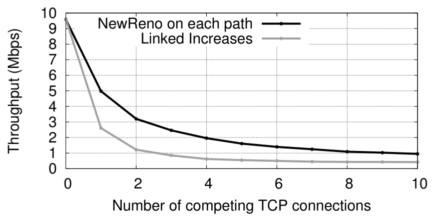


Figure 2: MPTCP congestion controller pulls traffic away from congested paths

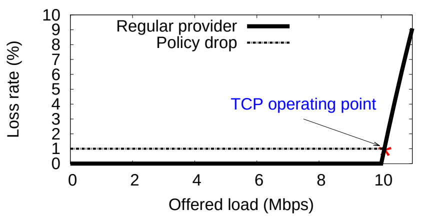


Figure 3: Policy drop effect on loss rates perceived by the client

at provider B and plot the total throughput achieved by the MPTCP connection when using the standardized Linked Increases congestion controller, or when using independent NewReno congestion control on each path. The results show that LIA utilizes both links when there is no competing traffic and achieves 20Mbps; when there is congestion, LIA shifts traffic away from the congested link responding to congestion much more severely than NewReno: the more congestion there is at provider B, the less traffic is sent there by the MPTCP customer.

This implies that provider B could intentionally underprovision its network and shift MPTCP traffic to provider A; indeed, this was one of the main concerns expressed by network operators when first presented with the MPTCP concept. It turns out these concerns are unfounded in practice: MPTCP adoption is gradual, and most traffic is still TCP, so underprovisioning would first affect the TCP users and drive them away from the operator. Secondly, congestion in the core network is seen as a sign of bad network planning: it generates crippling delays to all users and degrades customer satisfaction. Moreover, in the wide majority of access networks, congestion in not in the shared part of the network, but at the user access links (e.g. at the BRAS for DSL networks). In conclusion, our MPTCP customer would receive 20Mbps of throughput from most network operators today.

3. POLICY DROP

Network operators can't afford to underprovision their networks to convince Multipath TCP traffic to move to other networks, but we observe that they can achieve the same goal by using *policy drop*: *explicitly dropping a fraction of Multipath TCP traffic*. Policy drop will increase the loss rate seen by MPTCP subflows crossing a provider's network, tricking the MPTCP sender to shift traffic to other paths if these are available. Deploying policy drop must: not affect the performance of TCP or single-path MPTCP traffic, ensure that MPTCP always gets at least as much aggregate throughput as TCP on the best of its paths and be net-neutral.

The naive approach is to deploy a middlebox that scans for MPTCP traffic by examining options in TCP packets and drops some fixed fraction of them, e.g. 5%. It is obvious that this have the desired effect for MPTCP flows where other subflows exist that can carry traffic such as in Fig. 1, but it will not meet the requirements above:

- When an MPTCP connection only has a single subflow its rate will be drastically affected by 5% drop rate.
- When the alternative paths have low throughput, the user will get less than TCP on the best path.
- It discriminates MPTCP traffic and is thus not net-neutral.

To make policy drop deployable, the main goal is to ensure that the MPTCP sender can accurately infer the throughput single path TCP would get on the path, despite the fact that we are dropping a fraction of packets. To achieve this goal, we apply a different a drop rate per source-destination pair in a way that ensures base TCP throughput is not affected.

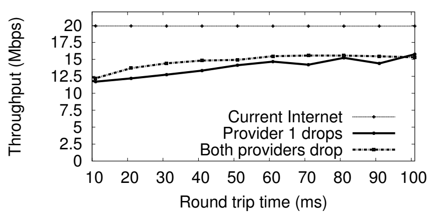
Assume that the bottleneck link in any connection is the client's access link with bandwidth B. In principle, any long-running TCP connection to/from any destination in the Internet will fully utilize the link and achieve throughput B. We can use the well known TCP throughput formula to estimate the loss rate of such a TCP connection as a function of B and RTT: $B = \frac{MSS}{RTT}\sqrt{\frac{8}{3p}}$. This estimate assumes the bottleneck buffer equals the bandwidth delay product of the TCP connection, but real buffers are in practice much larger to accomodate a wide range of RTTs: as B is constant, when RTTs are higher the loss rates will be lower.

It follows that the highest possible policy drop rate δ that does not affect TCP throughput is the one computed from the formula above, where the RTT is smallest: $\delta = p$, where p is derived from the equation above. Thus, $\delta = \frac{3MSS^2}{8B^2RTT^2}$. The network operator can readily compute δ: it knows B since it is policy dropping one of its clients' traffic and can estimate the RTT with small overhead.

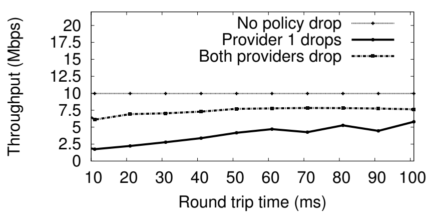
To better understand the effect of policy drop, we show in Figure 3 the loss rate experienced by the client as a function of offered load on the path. In a standard network loss rate will be zero as long as the offered load is smaller than the available capacity (10Mbps in this example), and TCP will increase its window until it fills the pipe and creates a small amount of loss (1% loss rate in this example). Policy drop will drop 1% of all packets *regardless of the offered load*.

Policy drop should not affect regular TCP clients, as they are "wired" to obtain 10Mbps throughput for the specified loss rate. However, MPTCP clients that have alternative paths will observe the higher loss rate and pull traffic away from this path as long as the loss rate on the alternative path is smaller. We study these hypotheses next.

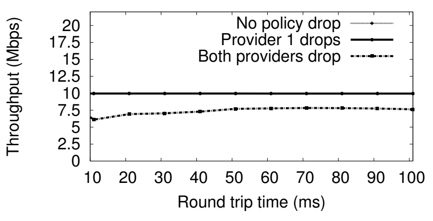
Simulation study of policy drop. We have implemented policy drop in the htsim simulator [17] and ran simulations to understand the effects it has on client throughput in a wide



(a) Total client throughput



(b) Throughput obtained from Provider 1



(c) Throughput obtained from Provider 2

Figure 4: Effect of RTT on the efficiency of policy drop.

range of scenarios. In all our experiments the client is dual-homed and the bottlenecks are the client access links.

Consider the case when the two access links have the same speed (10Mbps) and assume that the bottleneck buffers are provisioned for Internet-wide RTTs (200ms). The policy drop rate depends on the connection RTT: smaller RTTs result in higher policy drop loss rates. To understand how good policy drop is in forcing traffic away from provider 1, we vary the connection RTT from 10 to 100ms and measure the throughput for three scenarios: when no provider drops, when provider 1 alone drops, and when both providers drop.

We first examined the behaviour of TCP traffic [1] in this scenario: TCP throughput *is the same with or without policy drop*. However, policy drop has a beneficial effect on the RTT, decreasing the average RTT from 150ms to 35ms. This is because policy drop reduces the sender congestion window to the minimum required to keep the access link utilized. In contrast, TCP without policy drop will periodically fill the 200ms access link queue. Our experiments also show that Multipath TCP traffic running uncoupled congestion control (i.e. independent NewReno on each subflow) behaves similary to single path TCP: on each subflow throughput is unaffected and the RTT is reduced.

The behaviour of the Linked Increases algorithm is more interesting, and the results are given in Fig.4. First, notice that the total client throuhgput is lower than 20Mbps when policy is applied, but it is always greater than 10Mbps in all experiments, which means that the policy drop achieves its target of not breaking the MPTCP "do no worse than TCP" contract. When provider 1 uses policy drop alone, it gets a massive reduction in the traffic it carries for its client: 80% for small RTTs and around 50% for larger RTTs. When both providers shape, both reduce their traffic by 20%-30%.

Next, we wish to understand what happens when the speeds of the two access links are different. We fix the RTT at 20ms and the first access link speed at 10Mbps; provider 1 drops. We then vary the speed of the second access link from 1Mbps to 160Mbps, measuring the amount of traffic savings at Provider 1 in Fig. 5. The results show that little savings can be had when the speed of link 2 is small (1-5Mbps): this is because the loss rate on small links will be comparatively large, and the loss induced on link 1 is not enough to make the MPTCP congestion controller pull traffic away. In comparison, when the second link is at least twice faster than the policy drop link, its loss rate will be much lower, and policy drop will make 90% of traffic leave the first access link.

[1]The behaviour is the same for MPTCP with single subflow

All the experiments so far were run with perfect RTT information. To understand how sensitive policy drop is to RTT estimation errors, we vary the RTT used when computing the policy drop ratio (estimated RTT). We use a 20ms base RTT and 10Mbps links, and only show the effects at Provider 1 using policy drop in Figure 6. The figure shows that underestimating the baseline RTT (i.e. propagation delay) decreases regular TCP throughput, though not drastically: a 12ms estimated RTT reduces TCP throughput by 20%. Over-estimating the RTT decreases the traffic offloading benefits of policy drop for MPTCP traffic. With a 25% overestimation, only 50% of the traffic is pushed away, and a 50% overestimation only achieves a 10% traffic reduction.

Finally, we also explored what happens when the access link is not the bottleneck, and the bottleneck is the provider uplink to the core network. In our scenario a 1Gbps access link is shared shared between 100 to 500 customers, each with 10Mbps access links. Each client runs one TCP connection with a 20ms RTT, and only one client is subject to policy drop. In Figure 7 we plot the throughput obtained by the policy dropped client as a fraction of the other clients' throughput. The results show that policy drop has almost no effect on TCP traffic when the core link is not bottlenecked; when we load the link the throughput of the customer subject to policy drop is reduced by 10% to 20%.

Implementation. We have implemented policy drop as a Click modular router element [8]. Our implementation does not hold per-flow state, and treats each packet independently. Our element relies on two in-memory arrays that store:

- Customer link speed information, with one 1B per customer. We assume the provider has 1 million customers so the total table is 1MB in size.
- RTT estimation information. We store a single RTT estimate per /24 IP prefix, resulting in 16 million entries. The RTT map aims to capture underlying base RTTs to different destinations, and thus changes rarely. Our policy drop element takes it as input.

When a packet from the Internet to the local network arrives, our code uses the destination address to directly index the link speed array, finding the client's access speed, and uses the most significant 24 bits of the source address to get an RTT estimate from the RTT array. Using this information, the policy drop probability is computed for this con-

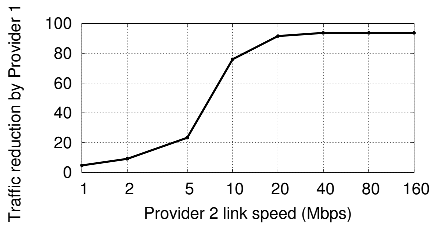


Figure 5: Provider 1 can shift more traffic if Provider 2 access link is faster

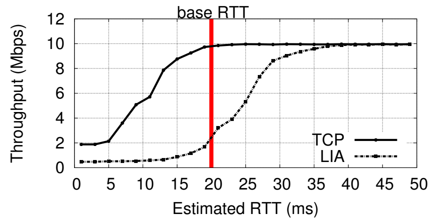


Figure 6: RTT estimation errors may decrease TCP throughput or reduce policy drop gains.

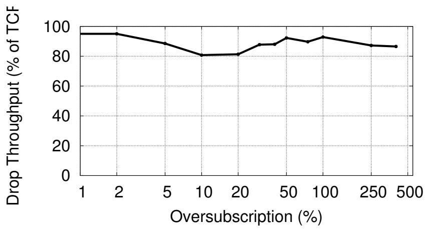


Figure 7: Policy drop has little effect on TCP when bottlenecks reside om provider uplinks.

nection and a random decision is made whether to drop this packet or not. Our implementation only drops data packets; it never drops TCP handshake messages to avoid the resulting lengthy timeouts and their associated impact on application performance. When the transport is ECN-enabled, the packets are marked instead with the CE codepoint.

The RTT information can be estimated offline and supplied to the policy drop module. The simplest option is to actively sample a fraction of client-to-Internet packets, store them in a map and compute the RTT when the corresponding ACK packets arrive. In our experiments, we manually filled the RTT map according to the experiment we were running; we plan to implement the adaptive version and to perform Internet-wide measurements in the future.

We deployed our prototype in a local cluster on three 6-core Intel Xeon machines with 8GB of RAM running Linux 3.36 kernel patched with MPTCP version 0.88. One host acts as the client, one as the policy-drop router and one as the server. The client and server boxes can communicate via two paths, one that passes the policy drop router and one that avoids it. We used dummynet to shape traffic. In all our experiments we use NewReno congestion control at endpoints (instead of Cubic) for TCP traffic, and the LIA congestion control algorithm for MPTCP traffic.

Our experiments targeted two aspects: first, we re-ran setups from simulation and cross-checked the results, finding that the simulator and experimental results matched. Next, we measured the throughput of our Click implementation running over netmap[13], to understand how expensive policy drop is to deploy in practice. We generated different-sized packets with the `pktgen` tool. The table below shows the throughput of policy drop and contrasts it to simple Click forwarding on the same machine. The results show that policy drop can forward almost 10Mpps on two cores, and that it reaches 10Gbps line-rate with 128B packets.

Packet size	64B		128B	
CPU Cores	1	2	1	2
Forwarding	8.5Mpps 5.7Gbps	13.2Mpps 8.87Gbps	8.2Mpps 10Gbps	8.2Mpps 10Gbps
Policy drop	6.8Mpps 4.57Gbps	9.5Mpps 6.38Gbps	6.8Mpps 8.2Gbps	8.2Mpps 10Gbps

4. POLICY DROP IMPLICATIONS

Our performance analysis shows that policy drop is effective at pushing MPTCP traffic away when there are alternatives, and that a single box can sustain almost 40Gbps of throughput per box. In this section we wish to understand whether Internet providers will deploy policy drop, assuming they make rational choices based on their expected profit. For policy drop to be used, the traffic reduction it provides must be greater than the cost of the policy-drop machines.

The cost of servers for policy drop grows linearly with the amount of traffic that is processed. If policy drop is effective, total operator traffic will decrease by as much as 80%; hardware is only needed to handle the remaining traffic. If no traffic reduction is achieved, then the hardware would have to handle all traffic and would cost significantly more. We thus expect the attractiveness of policy drop to depend on the amount of MPTCP traffic that can be shifted.

We rely on game-theory to understand whether operators will use policy drop or not, and under what circumstances.

Preliminaries. In our model, n providers interact with m clients. Each provider i offers a single fixed x_i Mbps subscription and each client has a *type* TCP or MPTCP, with a preference for low or high bandwidth. A provider's decision to policy drop traffic is $\delta_i \in \{0,1\}$. $\overline{\delta} = (\delta_1, \ldots, \delta_n)$ is a set of actions - one for each provider. An action for client j is $\overline{q}_j = (q_{1j}, \ldots, q_{nj})$, where $q_{ij} \in \{0,1\}$ is client j's option for provider i. $Q = (\overline{q}_1, \ldots, \overline{q}_m)$ is a set of actions - one for each client. We start by introducing our utility model.

The client utility trades off *price* and *throughput*. Internet subscription prices are strongly correlated with the provider costs of carrying client traffic to the Internet. To keep our model simple, we assume that subscription prices only depend on the providers' *transit cost* and the cost of purchasing servers to run policy drop. We analyzed real ISP subscription prices [1] as well as transit costs reported in [2], and chose $\gamma \cdot \sqrt{x}$ as a model for the transit cost supported by any provider carrying x Mbps to the Internet[2]. We estimate policy dropping costs at 1 eurocent/Mbps/month.

MPTCP clients select at least two providers to ensure high availability; TCP clients select a single provider as they do today. Let us first look at the utility of a MPTCP client which selects subscriptions x_1 and x_2 of two no-dropping ISPs:

$$\beta_j \cdot \log(1 + x_1 + x_2) - \gamma(\sqrt{x_1} + \sqrt{x_2})$$

The expression $\beta_j \cdot \log(1 + x_1 + x_2)$ measures the payoff from the combined throughput $x_1 + x_2$ — a client with a

[2]With γ selected as to best fit the romanian transit costs from [1]

higher β_j will prefer higher-bandwidth subscriptions. The expression $\gamma(\sqrt{x_1}+\sqrt{x_2})$ is the price of the subscriptions. Note that the utility function is *concave* w.r.t. throughtput.

We now move to a more general setting where ISPs have made dropping decisions $\overline{\delta}$ and clients made provider selections Q. The **throughput** of a TCP client will allways be equal to the subscription he acquired. For MPTCP clients, the throughput is $th_j(Q,\overline{\delta}) = \sum real_i(\overline{\delta})$ i.e. the sum of the *actual throughput* received from each provider i, as influenced by policy drop.

The table below illustrates all possible values $real_1(\overline{\delta})$ and $real_2(\overline{\delta})$ for any two providers (denoted here as 1 and 2) selected by the same MPTCP client[3]. We assume w.l.o.g that subscription $x_1 \geq x_2$. For simplicity of analysis, the values below assume an ideal MPTCP congestion controller that pushes all traffic away from the congested link, with the exception of negligible probing traffic[4].

δ_1	δ_2	$th_j(Q,\overline{\delta})$	$real_1(\overline{\delta})$	$real_2(\overline{\delta})$
0	0	x_1+x_2	x_1	x_2
1	0	x_1	x_1-x_2	x_2
0	1	x_1	x_1	0
1	1	x_1	$x_1\frac{x_2}{x_1+x_2}$	$x_1\frac{x_1}{x_1+x_2}$

The **subscription prices** are also influenced by policy drop — we assume an idealised provider model where charging is done on the actual carried traffic and not on the *declared* subscription size. Hence, $price_{ij}(Q,\delta)$ is the cost of the *real traffic* transited by provider i divided by the number of i's clients, including policy dropping costs:

$$price_{ij}(Q,\delta) = \frac{\gamma\sqrt{\sum_{1\leq j\leq m} real_i(\overline{\delta})\cdot q_{ij}}}{\sum_{1\leq j\leq m} q_{ij}} + \delta_i dropcost_i \quad (1)$$

The expression $\delta_i \cdot dropcost_i$ models the dropping cost (when it occurs) and is linear in the carried traffic.

We are now ready to combine the throughput and price expressions into a unified, type-independent client utility:

$$U_j(\overline{\delta},Q) = \beta_j \cdot \log(1+th_j(Q,\overline{\delta})) - \sum price_{i,j}(Q,\overline{\delta})$$

4.1 The game

While it is clear that providers have a strong incentive to policy-drop in order to reduce costs, we would like to see if ISPs can use dropping to offer more attractive client subscriptions, instead of securing some unilateral benefit.

We analyse the *state* of an idealised ISP market *at equilibrium*. To this end, we use the *strategic game model* [10, Chapter 2.1.] — where, similarly to the *rock-paper-scissors* game, actions are played *simultaneously*. In an equilibrium of such a game, no individual player can unilaterally improve his utility, if the actions of all other players are fixed[5].

[3]The throughput expressions can easily be lifted to the case when k out of $n > k$ providers policy drop

[4]our LIA experiments show that probing on higher loss paths is 15% even in the best case; other congestion controllers probe less

[5]This corresponds to the Nash Equilibrium of a strategic game, c.f. [10, Chapter 2.1.]

We take a few basic assumptions: clients and providers are only affected by subscription sizes and dropping; they are *rational* and *self-interested* — they behave in such a way as to maximise utility. Moreover, *ISPs know the client utility model* and exploit this knowledge when deciding to policy drop. To capture this, we use a *Stackelberg game* [16] consisting of two stages (or two strategic sub-games): ISPs make the first move by deciding whether to use policy drop and clients respond by chosing the best ISPs.

We start with **the client stage**. Suppose ISPs have already made the dropping decisions $\overline{\delta}$. A *best-response* Q^* of the clients is a Nash Equilibrium dependent on $\overline{\delta}$. Since such a response need not be unique, let $NE(\overline{\delta})$ designate the set of equilibria induced by $\overline{\delta}$.

We turn to **the provider stage**. ISPs know that clients will always reply with a *best-response*. We take this into account when formulating the provider utility $\mu_i(\overline{\delta}) = \frac{\sum_{Q^*} s_i(Q^*)}{|NE(\overline{\delta})|}$, where $s_i(Q) = 1$ if provider i is selected in Q, and 0 — otherwise. Providers count the equilibria in which they have clients and divide the amount by the total number of equilibria. Thus, $\mu_i(\delta)$ is the probability that provider i is selected by some client j, if each Nash Equilibrium $Q^* \in NE(\overline{\delta})$ is equally likely. The equilibrium $\overline{\delta^*}$ of the provider stage is a subgame-perfect equilibrium (**SPE**) [10].

4.2 Results

We compute SPE for a range of scenarios, and examine the best client move for each such equilibrium. As determining SPE is computationally hard [9] our experiments are limited to games with a small number of clients and providers.

We consider subscriptions of 3, 10 and 100 Mbps. We refer to 3 and 10 Mbps providers as *low-bandwidth* and to those of 100 Mbps as *high-bandwidth*. A clients' type is x-y where $x \in \{TCP, MPTCP\}$ and $y \in \{low, high\}$ (different β in the utility function).

In a **TCP game** with one *TCP-low* client, one 3, 10 Mbps subscriptions and two 100 Mbps subscriptions we observe — as expected — that policy drop is not a good choice.

MPTCP with uniform subscriptions. To observe the effects of dropping, we compute SPE for the game G_{drop}, with four identical 10 Mbps providers and two *MPTCP-lo* clients. We illustrate the best client move as a $m \times n$ matrix where lines represent clients, rows represent providers and each cell i,j represents clients' i option for provider j. The four equilibria $\overline{\delta}_x$, $x \in \{a,b,c,d\}$ are shown in Fig. 8.

In each SPE, one provider is better off not dropping and will always be selected by all clients. Each of the other three providers are better off dropping. They offer almost-zero bandwidth for an almost-zero cost. Since our subscription price function p is concave, *the price for a bigger subscription $x_1 + x_2$ is allways smaller than that of any combination of smaller subscriptions x_1 and x_2*. To maximise utility, clients choose the subscription closest to their bandwidth needs and the cheapest subscription available as backup.

Intuitively, $p(x_2)$ is an *availability cost* which MPTCP

$$\overline{\delta}_a = (0,1,1,1) \quad \overline{\delta}_b = (1,0,1,1) \quad \overline{\delta}_c = (1,1,0,1) \quad \overline{\delta}_d = (1,1,1,0)$$

$$\overline{\delta}_a = (\ 0\ ,\ 1\ ,\ 1\ ,\ 1\ ) \qquad Q_a^1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

Figure 8: SPE of G_{drop} and the best client response to $\overline{\delta}_a$

clients are happy to pay. For instance, in Q_a^1 (Fig. 8 - right), both clients receive $0 + 10$Mbps throughput; 10 Mbps is received from the no dropping provider (the first one), and 0 Mbps — from the dropping provider (in Q_a^1, the third and forth provider). Any policy dropping provider has equal chances of being selected by some client. There are 9 possible best-moves for the client for each $\overline{\delta}_x$, and each dropping provider will be selected in 5 such moves, yeilding a selection probability of ≈ 0.55. If a provider would try to compete with the no dropping provider — hence change his action to "no dropping" — he would have only 0.5 probability of being selected. This shows why each $\overline{\delta}_x$ is stable.

How do these observations scale to large Internet markets? Suppose a new ISP would like to enter a market with uniform subscriptions having m potential clients. The new ISP has to decide whether to compete with the no dropping or the dropping providers from the market. Our experiment shows that, if at least one no dropping provider exists, then the rational decision for the new ISP is to perform dropping[6].

Heterogeneous MPTCP clients. We now move to the game G_{mdrop}, where clients are no longer identical. We have one *MPTCP-lo* and one *MPTCP-hi* client. There are one $3, 10$ and two 100 Mbps subscriptions. We use two 100 Mbps ISPs instead of one to observe their competitive behaviour.

We obtain three SPE $\overline{\delta}_e = (0,0,0,0), \overline{\delta}_f = (1,1,0,0)$ and $\overline{\delta}_g = (0,0,1,1)$. In the first, the *MPTCP-lo* client selects the 3 and 10 Mbps subscriptions, while the *MPTCP-hi* client selects the 3 and 100 Mbps subscriptions. This is the best outcome for the low-band providers as both get selected.

If any low-band provider would policy drop to reduce costs, it would force the *MPTCP-lo* client to select him (for almost 0 cost) and the 100 Mbps provider. This outcome is not stable as it would force the other low-subscription provider to behave in the same way in order to be competitive. This is reflected by the equilibrium $\overline{\delta}_f$. Here, surprisingly, the high-bandwith providers do not drop. If one did policy drop, it would greatly increase the price of the associated low bandwidth subscription, resulting in higher total price for the client.

The SPEs $\overline{\delta}_f$, $\overline{\delta}_g$ depend on subscriptions and client values β_j, thus it is hard to generalise this result directly. However, *dropping as a means to offer cheap subscriptions* remains a viable ISP move irrespective of the market setting.

From TCP to MPTCP. Our analysis shows that policy drop makes no sense when traffic is TCP, and is desirable when all traffic is MPTCP. The natural question is how much MPTCP traffic must there be until it makes sense to use policy drop?

To answer this question, we consider that a percentage p of the clients' traffic is MPTCP while the rest is TCP and compute a threshold condition at which policy drop becomes attractive to providers. We find that, for a provider to policy drop when the amount of MPTCP traffic is low it must offer a low bandwidth subscription. We omit details for brevity.

5. RELATED WORK

Previous work has explored whether network operators can glean more information about the aggregate MPTCP connection just by looking at one subflow [14]. The work also exploits the resource pooling behaviour, albeit in a passive way, to determine which subflows are part of MPTCP connections with multiple subflows, and estimate how good those subflows are. The detection method is intended for offline analysis because it is very expensive computationally as it requires maintenance of per-flow state. Our approach does not try to detect alternative subflows; instead, it applies policy drop to all traffic, and only the traffic that has alternative paths will move away.

6. DISCUSSION

Policy drop is a technique that tricks MPTCP traffic to take another path and is a simple way for network operators to reduce the traffic they carry. it is likely that there will be takers for this technology, if one only looks at the huge market in DPI, application optimization and other network functions, including the recent push towards NFV. Since policy drop is applied to all traffic, it obbeys US net neutrality laws. Furthermore, our economic incentive analysis shows that clients might actually prefer a cheap second link for availability purposes, even if they know the provider uses policy drop.

Circumvention. Can endpoints avoid policy drop? It is certainly possible to detect it, and perhaps blow the whistle on policy dropping operators. If clients actively monitor their loss rate and their received throughput, they can obtain results similar to Figure 3. Detecting policy drop is trivial because there is a significant loss rate even at low throughputs.

Policy drop can be neutralized if endpoints simply give up on multipath congestion control altogether, and run independent congestion control on each path instead. If this were the case, policy dropping providers would incur the high cost of policy drop equipment and not see any bennefit in traffic reduction. This could be used as a "stick" by users to convince providers against policy drop.

The trouble with changing the congestion controller is twofold. First, the servers are running the congestion control algorithms for downlink traffic and the clients have no control over it. Secondly, this also nullifies all the resource pooling benefits of Multipath TCP that benefit not only networks but users alike. Previous work has shown these benefits in the context of datacenters [12] and to achieve smooth handovers in dense Wifi networks [3].

[6]For n dropping providers there are $m(n-1)$ outcomes where the ISP is *not* selected, out of n^m possibilities. The probability that the ISP is *not* selected is $\approx \frac{m}{n^{m-1}} \rightarrow 0$ for large m.

Acknowledgements

This work has been partly funded by University Politehnica of Bucharest (UPB-GEX 2016, award 331) and by the Superfluidity (671566) and SSICLOPS (644866) projects funded by the European Commission under its Horizon 2020 program.

7. REFERENCES

[1] Broadband operators and tariffs. http://point-topic.com/wp-content/uploads/2013/02/Broadband-tariffs-complete-database-Q4-2012.xlsx. Accessed: 2016-02-29.

[2] How transit works, what it costs & why itâĂŹs so important. http://blog.streamingmedia.com/2014/02/transit-works-costs-important.html. Accessed: 2016-02-29.

[3] A. Croitoru, D. Niculescu, and C. Raiciu. Towards wifi mobility without fast handover. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 219–234, Berkeley, CA, USA, 2015. USENIX Association.

[4] Ford, Alan and Raiciu, Costin and Handley, Mark and Bonaventure, Olivier. RFC6824: TCP Extensions for Multipath Operation with Multiple Addresses. https://tools.ietf.org/html/rfc6824.

[5] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of ACM SIGCOMM 2009*.

[6] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *SIGCOMM Comput. Commun. Rev.*, 35(2):5–12, Apr. 2005.

[7] 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 *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 1–12, New York, NY, USA, 2012. ACM.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[9] S. Leyffer and T. Munson. Solving multi-leader common-follower games. *Optimization Methods and Software*, 25(4):601–623, 2010.

[10] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[11] Q. Peng, A. Walid, and S. H. Low. Multipath tcp algorithms: Theory and design. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, pages 305–316, New York, NY, USA, 2013. ACM.

[12] 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 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[13] L. Rizzo. netmap: a novel framework for fast packet I/O. In *Proceedings of USENIX ATC 2012*.

[14] M. Z. Shafiq, F. Le, M. Srivatsa, and A. X. Liu. Cross-path inference attacks on multipath tcp. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pages 15:1–15:7, New York, NY, USA, 2013. ACM.

[15] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers, randomly. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

[16] H. von Stackelberg. Market structure and equilibrium: 1st edition translation into english. *Bazin, Urch & Hill, Springer 2011, XIV*.

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proceedings of USENIX NSDI 2011*.