Multipath bonding at Layer 3

Author(s):
Bednarek, Maciej; Kobas, Guillermo B.; Kühlewind, Mirja; Trammell, Brian

Publication Date:
2016

Permanent Link:
https://doi.org/10.3929/ethz-b-000119911 →

Originally published in:
http://doi.org/10.1145/2959424.2959439 →

Multipath bonding at Layer 3

Maciej Bednarek
ETH Zurich

Guillermo Barrenetxea
Kobas
Swisscom

Mirja Kühlewind
ETH Zurich

Brian Trammell
ETH Zurich

ABSTRACT

Recent work has applied Multipath TCP proxies to the problem of bonding a customer's multiple access interfaces to the Internet, in order to augment available bandwidth, especially in areas with marginal fixed connectivity. However, such proxies only apply to TCP traffic, and while UDP-based media streams can be tunneled through bonded TCP connections, this would lose the advantages of loss-tolerant media-oriented transports. We therefore propose an approach to do interface bonding at layer 3, design a scheduling algorithm to shift traffic between fixed and mobile lines, implemented Linux-based bonding gateways, and tested them within a testbed on Swisscom's production DSL and LTE networks.

CCS Concepts

•**Networks** → **Middle boxes / network appliances; Packet scheduling;**

Keywords

multipath; bonding; UDP; DSL; LTE

1. INTRODUCTION

Multipath TCP (MPTCP) [6] allows an endpoint to simultaneously use multiple paths over multiple interfaces (e.g., wifi and LTE) for a single TCP session. As MPTCP does not require any modification in the application, it can be used any time if both the client and the server support it. However, as MPTCP is not yet widely deployed, another approach to provide its advantages involves the creation of multipath tunnels between MPTCP proxies [3], which run all traffic between the proxies over the multiple paths between them. When placed on a customer's access router, the proxy allows the customer's TCP traffic to benefit from the aggregation the DSL and the mobile network link capacity, and for network providers to offer better service than would be available with DSL alone. For this reason, Various network providers are currently experimenting with use of multipath proxies to increase the bandwidth they can offer to their customers by aggregating the DSL and the mobile network link capacity on a customer's access router.

An MPTCP proxy can only be applied to TCP traffic, however. Running other traffic such as IPTV or VOIP applications over UDP in a multipath TCP tunnel can lead to poor performance. We therefore propose a new approach for multipath bonding at layer 3, which is independent of the transport protocol and therefor applicable to UDP traffic as well. The proposed design aims to minimize loss while fully utilizing the available bandwidth and avoiding reordering.

We targeted our prototype at a specific scenario: a customer in an area with marginal, low-bandwidth DSL connectivity but with good mobile connectivity, where a single traffic flow (e.g. high quality video) momentarily exceeds available DSL capacity. Here, the strategy is to keep the DSL link full, and to use the LTE link to provide excess capacity when necessary. The degrees of freedom of our architecture, however, allow it to be generalized to other scenarios. The architecture is described in section 2. The primary problem in building such a generic proxy is to send each packet over the appropriate interface. The design of the scheduling algorithm (in section 3) is therefore a key contribution of this work.

We implemented the proposed system in a lab testbed using a DSL connection and an LTE interface. Our testbed evaluation (section 5) demonstrates that our prototype can reach our goals of high utilization with low loss within our target scenario, even with dynamic network and cross-traffic conditions.

1.1 Related Work

The present work can be understood as a generalization of [3] to layer 3 and optimized for our described scenario where the fixed access network does not provide enough capacity to serve a bandwidth-consuming (multimedia-)service. In addition, Deutsche Telekom and Huawei have proposed a related approach to the present work, describing extensions to Generic Routing Encapsulation (GRE) to enable signaling for tunnel setup [8], but does not describe a scheduling algorithm for deciding which packets go over which tunnel. These extensions could be used to implement our approach over GRE.

This is a large number of related scheduling approaches in literature, including scheduling schemes proposed for MPTCP, e.g. [9]. While we only propose a scheduling that controls the bandwidth usage on each link, these proposal often also take higher layer information at the endpoint into account to de-

cide which packet should be scheduled over which path [13]. While it would be easy to make our proposed algorithm flow-aware to ensure that, for smaller flows, all packets of one flow would be router over the same link, we did not further consider this extension in this paper, as he have concentrated here on a scenario where one flow already would need more capacity than provided by the fixed network link, in our case DSL.

Further, there is an even larger set of proposals that discuss multipath routing, in various scenarios and based on different assumptions. Our approach differs from previous multipath routing work, such as [1, 10]: as we address a different goal with fixed preference, full utilization of a lower-cost fixed link while using a higher-cost wireless link to handle additional demand, therefore our approach can be much simpler than a generalized approach. The same is true for more generalized approaches as presented in [5, 2] as well as bandwidth aggregation in vehicular networks [11]. While these approaches and architectures concentrates on scheduling, The scheduling on our approach is simple, but the intelligence of our system lies in the outbound bonding box that performs the reordering.

We would further like to note that on-going work focuses in addition on middlebox and path signaling mechanisms [12]. If such mechanism would be in place, they could further be used to provide additional guidance for our multipath bonding proxy. E.g. if it would be know that a certain flow is not sensitive for re-ordering, our approach could be further simplified.

2. BONDING ARCHITECTURE

Our architecture consists of two gateways, one at the customer site and one operated by the access network provider, connected by at least two tunnels. One of these tunnels runs over a terrestrial (DSL) internet connection, and one over a mobile (LTE) connection; the approach could, however, be easily generalized to n access interfaces. The proxies build a "bonded" interface across these two connections. Each proxy consists of two components: an "ingress" which accepts traffic, assigns it to one of the two bonding interfaces based on interface conditions, and schedules its transmission; and an "egress" that takes traffic from the two bonding interfaces, merges it in the correct order, and sends it out. Each gateway therefore acts as a transparent proxy. The customer gateway is designed either to be integrated into Customer Premises Equipment (CPE) with multiple access interfaces, or to be deployed as a separate middlebox connected to CPE for each interface. The provider gateway is connected to the Internet and/or to provider-hosted services (such as VoIP or video on demand).

The customer side gateway sends upstream traffic to the provider's bonding server, and the provider's gateway sends downstream traffic to the customer side bonding server. The two proxies cooperate with each other to determine link conditions on each of their bonded interfaces, as input to the scheduling algorithm used by each gateway. This arrangement is shown in Figure 1.

The ingress uses a scheduling algorithm that, for each incoming packet, decides which tunnel to use. This ingress adds global (per bonded gateway pair) and local (per interface) sequence numbers to each packet. These sequence numbers are used by the opposite gateway's egress to emit outgoing packets in order and detect loss on both links. The

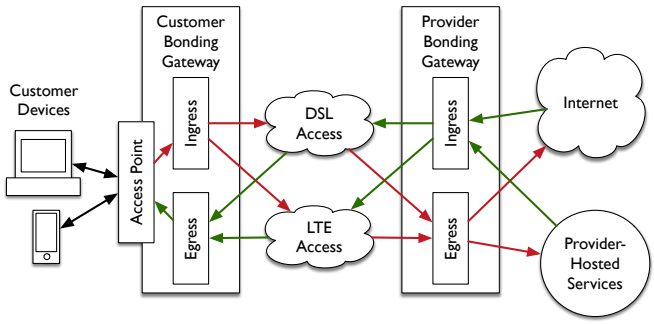

Figure 1: Architecture

egress periodically reports loss information back to the peer gateway, as input to the scheduling algorithm.

When splitting a single traffic flow across multiple access links, the delay over the total bonded link is limited to the maximum delay across the access links. In the case of LTE and DSL, as we target, this delay difference can be quite high. The egress must have enough buffering capacity to hold packets from the lower delay link so they can be resequenced before transmission.

3. SCHEDULING ALGORITHM

The scheduling algorithm at the ingress is based on a Weighted Round Robin (WRR) [7] scheduler. At a given point in time, a certain fraction of traffic is assigned to each bonded interface (either the mobile or fixed line) based on the weights for each.

These weights are calculated dynamically based on the loss ratio reported back from the peer's egress, under the assumption that loss is an indication that a link is at its capacity. Since LTE access is more costly than DSL, one of our goals is to fill the fixed link first, and use the mobile link for excess traffic demand only. Therefore, the scheduling algorithm we implemented uses a constant fixed link weight and only adapts the mobile link weight; a generalization of this approach to meet other goals could use both loss rates and adapt both weights independently.

The global sequence number is used to detect loss, and the local sequence number identifies the link on which the loss occurred. A loss is assumed when an expected packet has not arrived at the egress before a timeout T_{lto}. The egress reports the number of losses pkt_{lost} within a given interval T_{report} back to the ingress. For our evaluation, we only used fixed-size packets and therefore we only feed back the number of lost packets. However, the proposed algorithm could easily be adapted to reflect the actual packet size.

Our algorithm consists of three parts: Adaptive Weight Increment (AWI), Initial Weight Increment (IWI), and Delayed Weight Decrement (DWD), detailed in the subsections below. The algorithm is controlled by the k parameter to AWI, which controls the responsiveness of the AWI stage and thereby the responsiveness to traffic and link condition changes; and $t_{dwdwait}$, which controls the responsiveness of the DWD stage, and thereby the aggressiveness of the algorithm with respect to other traffic on the fixed link. In section 5 we show how tuning these parameters changes the algorithm's behavior.

3.1 Adaptive Weight Increment (AWI)

Upon the reception of a report that indicates that loss occurred, AWI updates the weight for the mobile link:

$$w_{mobile} \mathrel{+}= k * \frac{pkt_{lost}}{pkt_{sent}} * w_{fixed} \quad (1)$$

The increment of weight w_{mobile} is linearly proportional to the ratio of packets lost in the last reporting interval T_{report} vs. sent packet as counted by the gateway since the last report was received, for the fixed line only. k is a control parameter to adjust the steepness of the weight increase and thereby the responsiveness to loss. w_{fixed} is a constant value, however, the value itself determines the granularity of the packet scheduling as well as responsiveness. A small value allows only for a small number of ratios of packet distributions, while a large value needs longer to converge.

3.2 Initial Weight Increment (IWI)

In Eq. 1 above, a very small loss fraction would not change w_{mobile} despite congestion on the fixed link. However, in a situation where congestion is just arising, it is important to react quickly. Therefore, when w_{mobile} is zero but loss is reported, IWI increases w_{mobile} by the number of lost packets.

w_{mobile} is initially zero, and is clamped to a maximum value $w_{mobilemax}$. This clamp keeps IWI from overshooting and shifting too much traffic to the mobile link.

3.3 Delayed Weight Decrement (DWD)

After no loss has been reported for T_{dwd}, DWD decrements w_{mobile} by one for each interval T_{report} in which no loss has been reported. This shifts load back to the fixed line without inducing loss by shifting the load too quickly. As loss reports are only received every T_{report} milliseconds, T_{dwd} must be a multiple of T_{report}.

4. IMPLEMENTATION

We implemented the described approach using two Linux machines based on Debian Wheezy for experimental evaluation. We intercept packets using the `libnetfilter_queue` userspace library[1], which forwards packets from a specific Netfilter kernel queue to a userspace program. Each packet in the Netfilter queue is associated with an *id* - used to issue a verdict and release a packet from the userspace program back to the kernel: `NF_ACCEPT` accepts the packet for forwarding, `NF_DROP` discards it, and `NF_QUEUE` passes it to another queue.

4.1 Packet Mangling

The bonding gateway registers an outgoing queue in the `OUTPUT` chain of the Netfilter architecture. There two types of incoming packets in this queue: data packets to be scheduled for multipath bonding and control packets from the egress, containing loss reports. The latter are consumed as input for the schedule and discarded, while data packets are mangled to add the needed sequence number information and will then be forwarded to the scheduler. While a standards-based implementation of our approach would use the Generic Routing Encapsulation (GRE) Sequence Number and Key fields [4] could be used for this purpose, we added a custom header between the UDP header and payload.

[1]http://www.netfilter.org/projects/libnetfilter_queue/

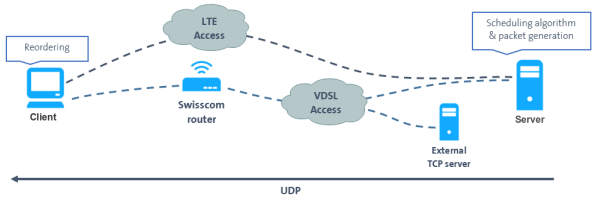

Figure 2: Laboratory setup.

4.2 Scheduling

The scheduler checks if packets are available in the incoming queue. After making a scheduling decision it applies a netfilter mark (`fwmark`) to each packet that is mapped to the right output queue using iptables. The scheduler also counts the number of packets sent on each interface, in order to calculate the weights.

4.3 Reordering

The egress implements a Netfilter queue which is hooked in the PREROUTING chain that intercepts all incoming UDP packets, forwarding these to a reordering function. If the sequence number is the next expected number in a flow ($last\_accepted + 1$) or the first one of a new flow, the packet is forwarded directly and the sequence number stored as *last_accepted*. If any packets are currently buffered, we check if $last\_accepted + 1$ matches the sequence number of the first packet in the queue. If so, this packet is forwarded as well, *last_accepted* updated, and the next packet in the queue checked, until the queue is empty or no match was found. If a packet with a larger sequence number than expected arrives, it is timestamped and buffered. Packets with a sequence number lower than *last_accepted* are discarded, as they have been assumed to be lost.

Further, for any out-of-order packets we compare the timestamp of the first packet in the queue to the current system time (as provided by the linux `time.h` library). If the difference is larger than T_{dwd}, we assume that the packet we are waiting for is lost and forward the first packet in the queue, respectively update *last_accepted*, and check the next packet in the queue for a match. This mechanism automatically detects multiple missing packets whenever the loss occurred in a bulk.

The original packet is then recreated and enqueued for further forwarding. This simplifies implementation, and make it possible to selectively disable reordering in a future implementation, e.g. for transports known to be tolerant of out-of-order delivery.

5. EVALUATION

For the evaluation we performed two sets of experiments: Experiments with a single UDP flow to evaluate the impact of different input parameters for k and T_{dwd}; and experiments to assess the algorithm's ability to operate in a more dynamic environment with multiple starting and stopping UDP flows as well as with adaptive TCP cross-traffic on the fixed line.

5.1 Experimental setup

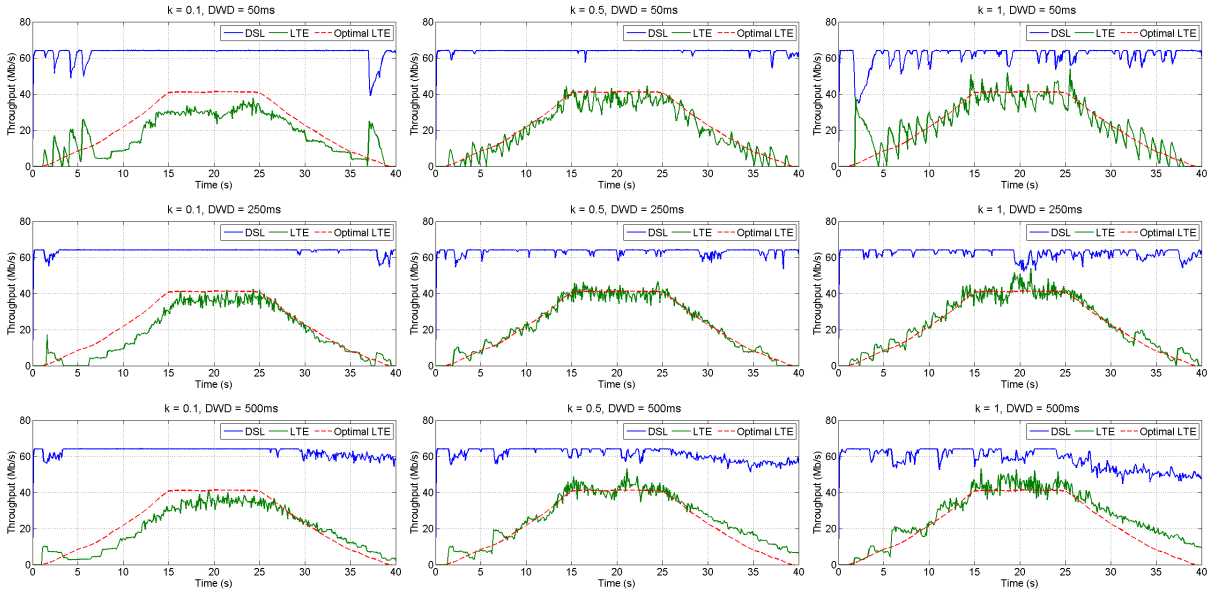

Figure 3: Per-link throughput for $k = \{0.1, 0.5, 1\}$ and $T_{dwd} = \{50ms, 250ms, 500ms\}$.

Figure 2 shows the basic platform used for the evaluation in all scenarios. Note that in this evaluation, we only send traffic from the server to the client; the ingress is colocated with the server, and the egress with the client.

The server generates UDP traffic directly, while TCP cross-traffic is generated by large file transfer from a well connected public server (`cdimage.debian.org`) 50ms away from the client. All generated UDP packets are 1492 bytes long (28 bytes UDP/IPv4 header, 4 bytes for the global sequence number, and 1460 bytes of dummy payload).

Each endpoint has two interfaces: a mobile LTE dongle and one DSL line connected to standard Swisscom router (Stargate). The available bandwidth on DSL link is shaped between the CPE and the Digital Subscriber Line Access Multiplexer (DSLAM) to a maximum rate of 64 Mb/s. The LTE access is provided using Swisscom's Huawei E3276s LTE stick with a maximum rate of about 60Mb/s. The average latency between client and server for the DSL line was measured around 13ms and remained stable, whereas the latency for mobile LTE access varied between 25 - 45ms.

For each evaluation in the subsections below, we chose a fixed $w_{fixed} = 50$, and an loss reporting interval $T_{report} = 50ms$.

5.2 Single flow

To demonstrate the impact of k and T_{dwd}, we initially evaluate a scenario with only one UDP flow. The flow starts with a sending rate of 63 Mb/s, then increases its rate exponentially every 200ms for 15 seconds, until it reaches a maximum of 105Mb/s. It holds this constant sending rate for 10 seconds and afterwards decreases the rate logarithmically to finally get back to the initial data rate of 63Mb/s after another 15 seconds.

$w_{mobilemax} = 45$ in order to avoid exceeding the maximum LTE link capacity in our scenario. This is necessary especially for evaluation with large k values ($k = 1$) which can causes an abrupt increase of w_{mobile} during IWI. The experiment was run for k values of 0.1, 0.5, and 1 and T_{dwd} of 50ms, 250ms, and 500ms.

Figure 3 shows the achieved throughputs on each link where the red, dashed line depicts the ideal throughput for zero loss on the fixed line. With a small values k of 0.1 not enough traffic is shifted to the mobile link, leading to permanent losses with an average loss rate of 6.25%, 4.72%, and 4.1% for the different T_{dwd} values, respectively. However, the larger k is, the more the traffic load oscillates between the two links. A value of 1 basically means that as soon as any loss is reported, we will start shifting traffic to the mobile link. Due to the IWI scheme that we need for scenario where new traffic flows start as we will show later, we bascially overshoot, shift too much traffic such that we underutilize the fixed line, and consequently start to shift traffic back. A similar behavior can be observed for low traffic rates with a small k value as the impact of a small number of losses is higher in these cases.

Further, the larger T_{dwd} is, the slower our approach adapts to decreasing traffic demands (as between 25 and 40 s) by shifting traffic back to the fixed line. However, this counteracts the oscillation that we have observed with high k values, and less oscillation also leads to lower loss rates. Unfortunately, it also decreases the utilization on the fixed line.

For $k = 0.5$ and $k = 1$ with $T_{dwd} = 50ms$, we still have a loss rate of 1.79% and 1.14%. However, loss rates are below one percent for the other combinations: 0.53% and 0.29% with $T_{dwd} = 250ms$ as well as 0.34% and 0.15% with $T_{dwd} = 500ms$.

We have shown here that k and T_{dwd} provide a trade-off between agressiveness and responsibility, and therefore can be used to adapt our proposed algorithm to the desired goals of the operator in a given scenario. Selection of $k = 0.5$ and $T_{dwd} = 250ms$ or $T_{dwd} = 500ms$ provides the best fit to optimal load with acceptable loss for the single flow scenario. As we aim to always utilize the fixed line while most of our traffic is rather loss- tolerant, we use this parameter setting in further evaluation scenarios.

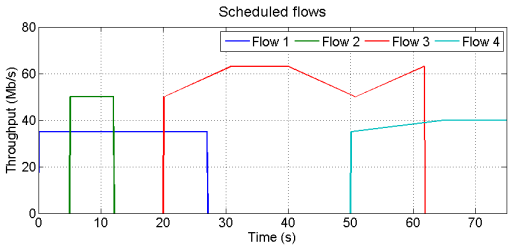

Figure 4: Generated UDP input traffic demand.

5.3 Multiple UDP flows

To demonstrate the ability of our proposal to adapt to changing traffic situations, we start and stop various UDP flows as shown in Figure 4. Input parameters are $k = 0.5$ and $T_{dwd} = 250$ as these values have shown a good trade-off between loss and link utilization in our previous evaluation. Throughput is calculated every 100ms.

We only perform per-packet scheduling, as opposed to per-flow scheduling, as our target scenario is to handle one flow that needs a higher data rate than provided by the fixed line. In a scenario with multiple smaller flows, one could schedule on a per-flow basis to avoid the additional delay of the mobile link for flows that are small enough to fit on the fixed line link.

Figure 5 shows the achieved per-link throughput, overall generated data rate and in-order throughput, as well as number of losses and and the changes in w_{mobile} observed during the experiment.

The excess load moves quickly from the fixed to the mobile link whenever the scheduled data rate exceeds the maximum capacity on the fixed line which is clearly visible when new UDP flows are being turned on (5s, 20s and 50s). Every time a UDP flow is terminated, the algorithm shifts the load back. This happens rather slowly as we use a quite high T_{dwd} value of 250ms, e.g. it required about 5 seconds to reduce load from 11Mb/s to 0Mb/s on the mobile link (12s - 17s).

The comparison of the output rate and the in-order throughput as provided by our approach (second plot from the top) gives an insight on the amount of re- ordering that happens, in this case with a timeout of 50ms.

Further, bursts of losses occur whenever a new flow starts, as our algorithm needs at least one interval of T_{report} time to detect the changed traffic demand and react accordingly. However, IWI shifts the traffic quickly and a low loss rate is maintained afterwards.

5.4 Single TCP flow

In this section we evaluate the behavior of our proposed scheme in the presence of greedy TCP traffic which due to it's adaptive congestion control aims to fully utilize the available link capacity while still being friendly to cross traffic that allocates capacity on the same link. In this scenario, there is a trade-off that allows the operator to decide if it what to to support this kind of cross traffic by shifting more UDP traffic to the mobile network, or prioritize his own UDP-based service over the adaptive TCP traffic. We show that this trade-off is configurable given the parameters our algorithm provides.

On the fixed line link, we start one greedy TCP flow at

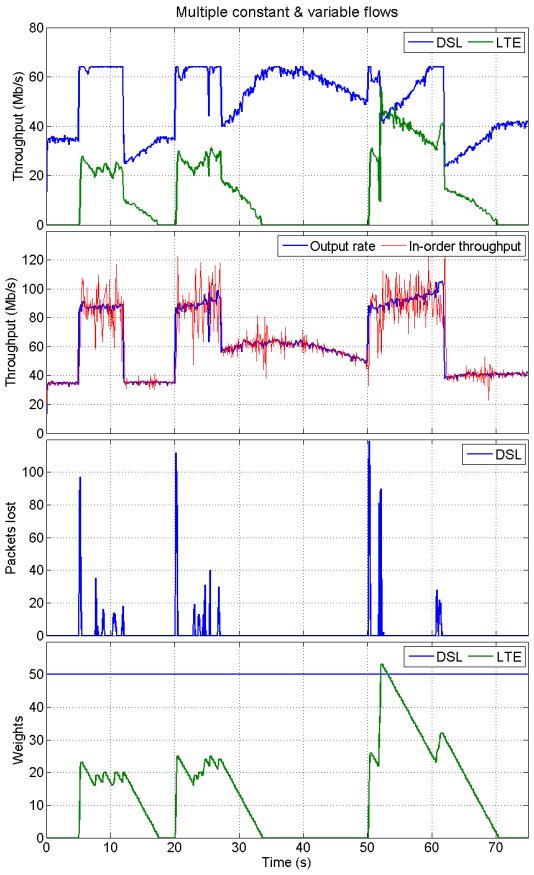

Figure 5: Behavior with multiple UDP flows.

the beginning of our test and then start one UDP flow after 10s with a constant data rate of 41.5Mb/s (for 200k packets, until 68.7s) which is lower than the maximum capacity provided by the fixed line and a second UDP flow around 88.7s (20 seconds after the end of first UDP flow) with a constant data rate of 63Mb/s (for 400k packets, until 165.1s), approximately the fixed-line capacity. Further, $w_{mobilemax} = 50$ in this scenario to ensure that UDP traffic will not be completely shifted to the mobile link. As it can be seen in Figure 6 the TCP connection is able to fully utilizes the available bandwidth of the DSL link if no cross traffic is present.

In Figure 6 we show the results for a k value of 1 and different T_{dwd} values of 50ms, 500ms, 1000ms. In all three plots the initial loss burst that is caused by the UDP flow startup leads to a shift of the traffic load to the mobile link. However, the smaller T_{dwd} is, the faster traffic shifts back to the fixed link. Only with a high value of T_{dwd} can we permanently shift some of the UDP traffic to the mobile link and leave spare capacity for the TCP traffic. Therefore T_{dwd} can be used by the operator to decide how TCP-friendly the algorithm should be. Further note that the achieved sharing is also influenced by the aggressiveness of the congestion control algorithm(s) used by the TCP traffic.

We also ran experiments with multiple TCP flows. E.g. in a scenario where we first start one UDP with a rate of 62 Mb/s and then start one or two TCP flows after 10s, of course we can see that the two TCP flows are more aggressive and therefore are able to grab more of the capacity (even

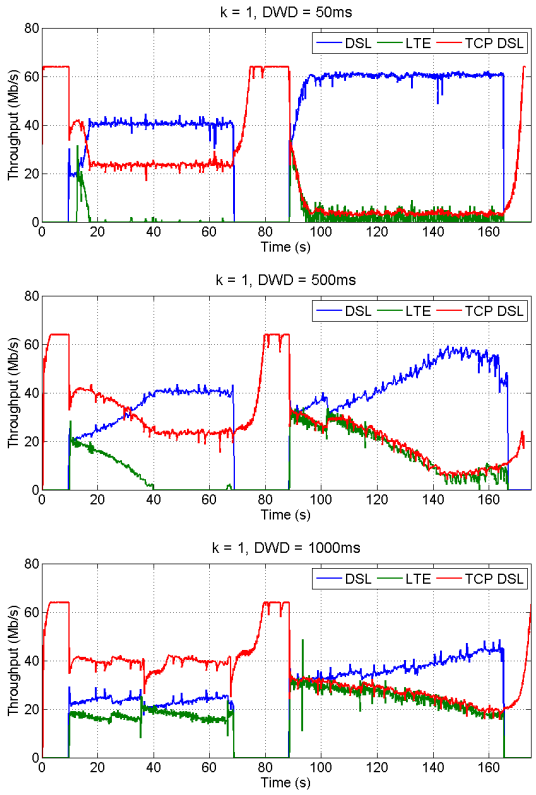

Figure 6: TCP cross-traffic on the fixed line.

with a T_{dwd} value of 500ms). In this scenario with a runtime of 88s and $k = 1$, a single TCP flow reaches an average rate of 7.79Mb/s while the sum of both TCP throughputs was almost twice as much with 14.30Mb/s. Further we ran a set of experiments with different value for k. As expected, k does not have significant influence on the sharing between TCP and UDP flows.

6. CONCLUSION AND OUTLOOK

In this work, we have demonstrated that a relatively simple scheduling algorithm can allow multipath bonding to handle non-TCP, loss-tolerant media traffic in dual DSL/LTE-connected access networks.

Future work in this area includes interoperation with presently deployed MPTCP proxies to simultaneously provide bonding for both TCP and non-TCP traffic, tuning our approach to interoperate with MPTCP's coupled congestion control. In addition, further work is planned to improve our algorithms with respect to the right adaptation steps and timing as well as additional evaluations to assess the latency and loss impact for real application, such as video streaming or even real-time media that is even more sensitive to delay.

Finally, we also work on schemes for explicit middlebox cooperation which would allow endpoints to indicate if reordering is required or not and therefore enable a better management given the tradeoff between bandwidth and latency in our proposed approach. Integrating these information in our approach enables further simplification as well as a more customized service treatment.

7. ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421, and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0268. The opinions expressed and arguments employed reflect only the authors' views. The European Commission is not responsible for any use that may be made of that information. Further, the opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

8. REFERENCES

[1] C. Cetinkaya and E. W. Knightly. Opportunistic traffic scheduling over multiple network paths. In *INFOCOM*, 2004.

[2] K. Chebrolu and R. Rao. Bandwidth aggregation for real-time applications in heterogeneous wireless networks. In *IEEE ToMC*, volume 4, 2006.

[3] G. Detal, C. Paasch, and O. Bonaventure. Multipath in the Middle(Box). In *ACM SIGCOMM HotMiddlebox'13*, 2013.

[4] G. Dommety. Key and Sequence Number Extensions to GRE. RFC 2890, IETF, Sep 2000.

[5] K. Evensen et al. A network-layer proxy for bandwidth aggregation and reduction of ip packet reordering. In *IEEE LAN*, 2009.

[6] A. Ford et al. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, IETF, Jan 2013.

[7] M. Katevenis et al. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. In *IEEE Journal on Selected Areas in Communications*, volume 9, 1991.

[8] N. Leymann, C. Heidemann, and X. Li. GRE Notifications. Internet-Draft draft-heileyli-gre-notifications-00, IETF, Oct 2013.

[9] C. Paasch et al. Experimental evaluation of multipath tcp schedulers. In *ACM SIGCOMM CSWS*, 2014.

[10] A. Qureshi and J. Guttag. Horde: Separating network striping policy from mechanism. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, MobiSys '05, pages 121–134, New York, NY, USA, 2005. ACM.

[11] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. Mar: A commuter router infrastructure for the mobile internet. In *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 217–230, New York, NY, USA, 2004. ACM.

[12] B. Trammell and J. Hildebrand. Evolving transport in the Internet. *Internet Computing, IEEE*, 18(5):60–64, Sept 2014.

[13] M. F. Tsai et al. Multi-path transmission control scheme combining bandwidth aggregation and packet scheduling for real-time streaming in multi-path environment. *IET Communications*, 4(8), 2010.