# Unleashing Tor, BitTorrent & Co.: How to Relieve TCP Deficiencies in Overlays (Extended Version)

Daniel Marks Florian Tschorsch Björn Scheuermann

Heinrich Heine University, Düsseldorf, Germany
Computer Science Department
Mobile and Decentralized Networks Group


TECHNICAL REPORT TR-2010-001
HEINRICH HEINE UNIVERSITY, DÜSSELDORF, GERMANY
COMPUTER SCIENCE DEPARTMENT

AUGUST 2010

# Unleashing Tor, BitTorrent & Co.: How to Relieve TCP Deficiencies in Overlays (Extended Version)

Daniel Marks Florian Tschorsch Björn Scheuermann

Mobile and Decentralized Networks Group
Heinrich Heine University, Düsseldorf, Germany
{marks, tschorsch, scheuermann}@cs.uni-duesseldorf.de

**Abstract**

Many overlay-based distributed applications employ TCP for the communication between peer nodes. Typically, the TCP connections of one peer will all share one physical Internet link. Using real-world network experiments, we demonstrate that this can lead to undesirable interactions between overlay links, resulting in oscillations and suboptimal throughput. Consequently, we argue that such effects should be taken into account in the design and deployment of overlay networks. In support of this case, we identify readily deployable countermeasures. In a first step, we show that simple traffic shaping mechanisms based on existing operating system QoS functionality can bring some relief. Yet, this alone is not fully effective if peers communicate bidirectionally, due to TCP ACK piggybacking. We thus suggest to set up the overlay's communication structure to avoid piggybacked ACKs, by separating bidirectional peer communication into two independent TCP connections. We demonstrate the effectiveness of this strategy and discuss viable migration paths for existing applications.

## I. Introduction

In recent years, a significant number of applications based on peer-to-peer overlay networks have been proposed. Many of them are now in widespread use on the Internet. Prominent examples are BitTorrent [1] for file distribution, Tor [2] for network anonymity, or the many existing flavors of P2P file sharing networks. Each of these systems has hundreds of thousands or even millions of active users.

Many such applications use TCP connections for the individual overlay links. Because a peer often communicates with many neighbors simultaneously, it is common that many TCP connections are active in parallel. This has important implications: the overlay nodes are Internet end systems, connected to the Internet over one single physical link. This link is shared by all TCP overlay connections to and from this peer. Consequently, overlay links are—very much unlike, e. g., the links of a router in a physical network—not independent.

In the design of overlay networks, it is therefore important to consider not only the data transfer within the overlay, but also potential interactions of overlay connections. Here, we put one particularly severe case of such interactions into focus. TCP connections can interfere with each other that originate from the same host, share this host's physical network connection, and generate both inbound and outbound bulk traffic on this link. These interactions cause throughput oscillations and repeated TCP slow-starts, and thus highly unsteady data transfers and suboptimal transfer rates.

This is a source of potentially significant performance problems in overlays, including widely deployed ones. In principle, it is known that such effects can occur, especially in the presence of asymmetric links [3]. However, despite being particularly problematic in settings often found in current peer-to-peer overlays, the effects have nevertheless been practically neglected in the scientific discussion of protocol designs.

The key points in this paper are the insight that overlay designs need to be aware of the implications of shared physical links on the one hand, and a discussion of how to deal with these implications on the other hand. To this end, we contribute a problem analysis based on real-world network experiments, and then focus on solution strategies. We aim for solutions with low intrusiveness and a realistic deployment perspective. In this spirit we contribute a practical approach that tackles the exposed performance problem by using readily deployable traffic shaping mechanisms in combination with simple modifications in the overlay structure.

We first argue that a form of ACK prioritization in the overlay nodes helps to improve unidirectional data transfers. This alone, however, does not yet lead to satisfactory results for bidirectionally used overlay links, because ACK piggybacking prevents ACKs from being handled separately. We thus suggest to substitute bidirectional overlay links by two unidirectionally used TCP connections, to thereby avoid ACK piggybacking. By implementing these proposals, we obtain significantly more steady data rates and throughput gains of 100 % and more.

This technical report is an extended version of [4]. Its remainder is structured as follows. We review related work and background topics in Sec. II. In Sec. III, we describe first experimental results and concretize the problem. We then show how ACK prioritization can help alleviate the problem for unidirectionally used connections in Sec. IV. In Sec. V, we discuss the case of bidirectional traffic and argue that the previously introduced measures should be combined with a directional separation of connections. We outline migration paths and implications for overlay designs in Sec. VI, before we conclude this paper in Sec. VII.

## II. Background and Related Work

In this section, we will first briefly revisit two widely used applications—BitTorrent [1] and Tor [2]—and some directions that have previously been pursued to improve their performance. In the role of case studies, these examples underline why the discussed effects are important for a broad family of overlay-based applications. Subsequently, we review previous results on interactions between multiple TCP connections in other contexts and the option of using alternatives to TCP.

### A. Examples of TCP-based overlays: BitTorrent and Tor

The aim of a BitTorrent overlay is to distribute a (typically large) file to a significant number of downloaders. For this purpose, an unstructured overlay is formed by establishing TCP connections between nodes that are interested in the same file. The nodes exchange information about locally already available parts of the desired file, and then share chunks of data with their neighbors over these TCP connections. This allows to utilize the peers' upload bandwidth and thus enables file distribution in a self-scalable manner. For details on the mechanisms used to set up the overlay and to coordinate the data exchange we refer the reader to [1]. Previous approaches to improve the performance of BitTorrent focused primarily on its incentive mechanisms and download strategies, i. e., on the application layer; examples are [5]–[8]. This is complementary to our work, as we focus on the observation that BitTorrent peers use a significant number of TCP connections in parallel, and on the resulting transport layer performance issues.

The number of simultaneously used TCP connections is even higher in the overlay that lies at the heart of the Tor anonymization service [2]. Tor allows to set up anonymized TCP connections to arbitrary TCP-based services, for instance to a web server. The Tor client software offers a SOCKS proxy interface to applications like web browsers. To set up the requested connections, it relies on a back-end network of so-called onion routers. The onion routers—currently about 1 500, contributed and operated by volunteers—form an overlay, which is used by several 100 000 clients. A client's connection is repeatedly forwarded over a randomized sequence of onion routers in different administrative domains. This is combined with sophisticated cryptography. Tor thereby ensures that neither the final communication partner nor any of the onion routers can identify both end points of the connection.

Data exchanged over an anonymized TCP connection via Tor usually takes three hops through the onion router overlay. Between each pair of traversed Tor nodes, the data is tunneled over a TCP connection. Consequently, onion routers in Tor need to maintain a huge number of TCP connections in parallel—to other onion routers, to Tor clients, and to destinations of anonymized connections. They are thus particularly heavily affected by TCP interference effects.

Tor currently suffers from severe performance problems, which are subject to ongoing investigation [9], [10]. We believe that transport layer interactions are likely one of the key reasons. The existing literature on Tor performance improvements focuses either on alternative data encapsulations—for example TCP-over-DTLS transport [11] or IPsec tunneling [12]—or on application-layer problems like Tor's path selection and load balancing [9], [13]. All these approaches still employ TCP at some point; some, including [11] and [12], even increase the number of TCP connections per physical link. Thus, our results are relevant for all of them.

Besides BitTorrent and Tor, there are many other proposed and deployed TCP-based overlays, spanning the range from classical file-sharing applications like [14]–[16] to group communication protocols like [17]–[19]. Multiple proposals, including [20], [21], even aim to improve the performance of classical end-to-end unicast transport by sending the data over multiple hops through an overlay. These "split TCP" approaches substitute one "long" connection by multiple "short"

overlay hops.[1] For all these systems, we may expect that they are affected by the effects described here; the countermeasures that we discuss are applicable in all these cases.

### *B. TCP interactions and alternative protocols*

TCP interactions of a kind that is similar to what consider here were described in the early 1990s already. Zhang and Clark [22] and Clark et al. [23] analyzed situations where bottlenecks in the core network are traversed by multiple TCP connections between different pairs of end systems. They argued that "ACK compression" effects can lead to oscillations. Mogul [24] subsequently demonstrated that this can actually occur in practice. Back then, the impact on peer-to-peer overlays (with a concentration of potentially many active TCP connections at individual overlay nodes) and the implications for their design have—quite understandably at that time—not been considered.

Later, TCP interaction effects have received some attention in the context of asymmetric Internet connections. A good overview of early work in this area is given by Balakrishnan et al. in RFC 3449 [3]. This also includes a review of a number of solution ideas, most of which require complex modifications of the TCP implementations in the end systems and/or of the intermediate routers. Here, we show that significant problems exist even in entirely symmetric settings. Moreover, we are interested in solution strategies that can be easily implemented without deep modifications in routers or end systems.

One of the solution strategies that has been suggested for asymmetric Internet connections is to handle TCP acknowledgments with higher priority in node interface queues. This was suggested by Kalampoukas et al. [25] and was subsequently adopted in some practical settings. It has, however, barely been taken up in the scientific discussion. The key advantage of such an approach from today's perspective and in our specific context is that it is relatively non-intrusive: it can be realized based on existing mechanisms of many current operating systems. We will therefore transfer this to our problem setting and take it up as our first solution step.

It is of course conceivable to employ alternative transport protocols instead of TCP. Many have been proposed, some are even specifically tailored to the needs of peer-to-peer overlays; examples include SST [26] and CUSP [27]. Clearly, integrating such a transport protocol is much more intrusive than the solution strategies we propose. Moreover, existing overlay-tailored transport protocols do not take the deficiencies discussed here into account. They typically heavily borrow from TCP with respect to congestion control, and will therefore exhibit similar problems.

For the specific case of BitTorrent, it has been proposed to use UDP in conjunction with the LEDBAT congestion control mechanism [28] as an alternative transport mechanism for BitTorrent traffic [29]. The primary aim of this proposal is to enhance the fairness between BitTorrent traffic and other applications on the same host; at the same time an improved general performance is intended. Even though this proposal attacks a different problem than the one pointed out here, it clearly underlines that switching to a complex new transport protocol in a deployed application

[1]Note that despite the similarity in terminology this is very different from the "splitting" of an overlay connection into two parallel TCP connections for "forward" and "reverse" data traffic discussed here.

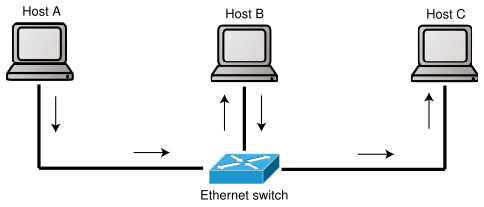



Fig. 1. Experiment setup.

is a challenging process and takes years to manifest: even though the introduction of LEDBAT into BitTorrent has started in 2008, after two years most BitTorrent clients still do not support the protocol, and stable implementations are only beginning to become available on a broader basis. Our proposals made here follow a much less intrusive and thus hopefully much more easily deployable direction.

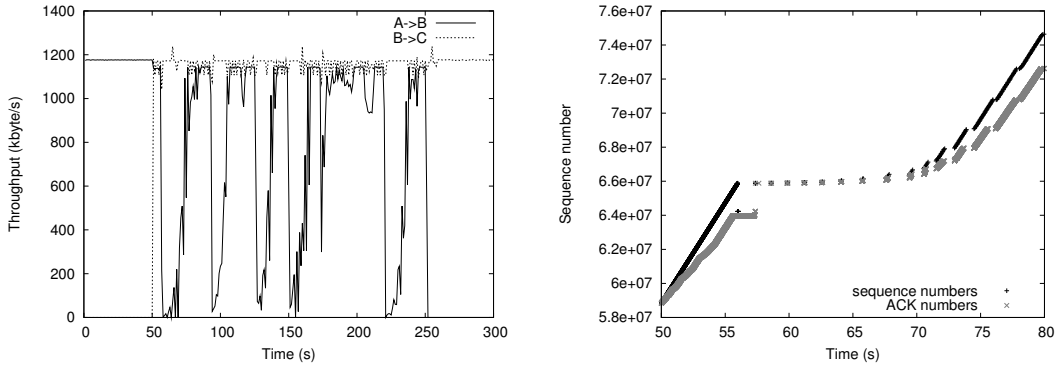
## III. Interference Between Overlay Connections

Overlay nodes are Internet end systems. Therefore, all their TCP connections to all their communication partners share one physical Internet link. As discussed above, they often exchange data with a significant number of other hosts in parallel. In this section, we will analyze the interplay between TCP data and ACK segments on the physical Internet link. We will show why the overlay nodes' Internet end system character has significant—and non-obvious—impact on the overlay network's performance.

### *A. Experiment setup*

To illustrate the effect we are dealing with, we perform a structurally simple experiment with real network nodes. It is illustrated in Figure 1. Three Linux (kernel 2.6.26, CUBIC TCP) hosts $A$, $B$, and $C$ are connected via a 10/100BaseTx Ethernet switch (3Com SuperStack 3 Switch 4400). To resemble typical Internet links more closely, we have configured all Ethernet interfaces to run at 10 Mbps full-duplex. Two TCP connections are set up: one from $A$ to $B$ and one from $B$ to $C$. Both connections thus share the link between $B$ and the switch, just like an overlay node's connections share its Internet link.

Starting from the beginning of the experiment for a total of 250 seconds, $A$ continuously transmits bulk data to $B$. 50 seconds later and likewise for a total duration of 250 seconds, $B$ starts a bulk data transfer to $C$. Consequently, there are three phases in the experiment: for the first 50 seconds, only the connection $A \rightarrow B$ is active. From second 50 to second 250, both TCP connections transfer data (from $A$ to $B$ and from $B$ to $C$). During this period, $B$ therefore has both incoming and outgoing TCP data transfers. From second 250 to second 300, finally, only the transfer $B \rightarrow C$ is active.




(a) Throughput in a single experiment run. (b) Sequence and ACK number progression of $A \rightarrow B$.

Fig. 2. TCP interactions in the experiment.

### *B. Result analysis*

Figure 2(a) shows the application-layer throughput over time in one single experiment run (calculated over 1 s intervals). During the first and last 50 seconds, when only one of the connections is active, the throughput is just as one would expect: due to the protocol overheads at the various layers, slightly less than 10 MBit/s. In the second phase of the experiment, however, when both incoming and outgoing transfers to/from $B$ are active, the picture is drastically different.

Evidently, the problematic connection is the incoming one: even though the link between $B$ and the switch is a full-duplex link with 10 MBit/s in both directions, connection $A \rightarrow B$ does not manage to make full use of this bandwidth. Instead, the achieved throughput is unsteady, it oscillates heavily. The oscillations set in right after $B \rightarrow C$ starts to transfer data.

The reason lies in the outgoing queue of node $B$'s link. In this queue, there are (1) the TCP data segments for connection $B \rightarrow C$ and (2) the ACKs of connection $A \rightarrow B$. In fact, we observe that when $B \rightarrow C$ sets in, this queue increases drastically in length, resulting in an increased queueing delay. This queueing delay also affects the acknowledgments for $A \rightarrow B$. This effect is clearly visible if we take a detailed look at the sequence number progression for $A \rightarrow B$ during one of the oscillation cycles. We show this in Figure 2(b). It visualizes the sequence numbers of outgoing data segments and incoming ACK segments at node $A$ over time, for a 30 s time interval immediately after $B \rightarrow C$ starts to transfer data.

The time until an acknowledgment for a transmitted data segment arrives increases rapidly as the outgoing queue at $B$ grows. This can be seen from the longer and longer horizontal gap between outgoing data segments and incoming ACKs with the same sequence number between seconds 50 and 55. The RTT of $A \rightarrow B$ rapidly increases to more than one second. Around second 55.5, a segment loss occurs. Because of the long queue at $B$ it takes a very long time until this loss is detected and the gap is filled in by a fast retransmit.

Also due to the long queueing delay at $B$, $A$ continues to receive duplicate ACKs for a very long time after the fast retransmit. In the sequence number plot, it can clearly be seen that $A$ is ultimately

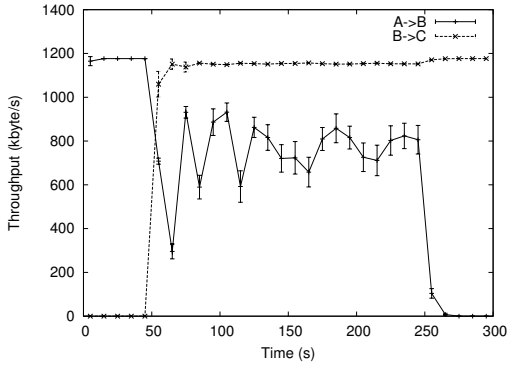



Fig. 3. Throughput average over 100 simulation runs.

forced into a slow start. Because of the still very long queue, the recovery takes a long time. In the figure, it is evident that the transfer during the slow start is not continuous, but ACKs arrive (and new segments are released) in "batches". If the current congestion window is exhausted, the sender is forced to wait until the next batch of acknowledgments makes it through the outgoing queue at $B$; this picture matches the "ACK compression" effect described in [22], [23]. This happens over and over again, and causes the oscillations.

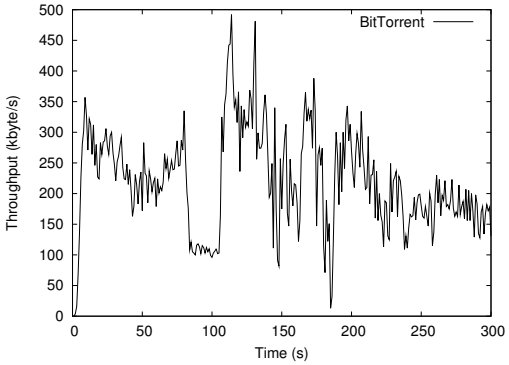
These oscillations result in a significant loss of bandwidth. To obtain a statistically solid picture, we have performed 100 independent runs of the above experiment. In Fig. 3 we show the measured TCP throughput of connections $A \rightarrow B$ and $B \rightarrow C$. The data points are throughput averages over 10 s intervals, the error bars show 90 % confidence intervals. These results show that the average throughput from $A$ to $B$ decreases by approximately one third as soon as the transmission from $B$ to $C$ starts. The bumps in the $A \rightarrow B$ throughput average line occur because the throughput breakdowns tend to happen at similar points in time, especially during the first 100 seconds of bidirectional traffic.

### *C. Deployed overlays*

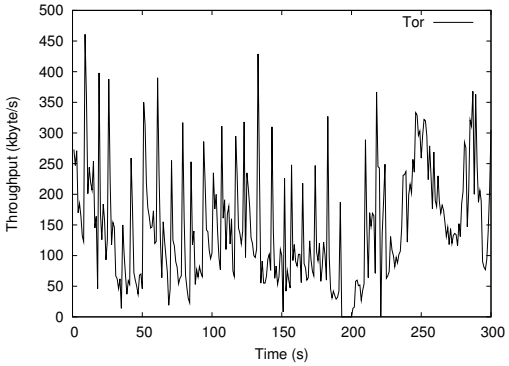
Beyond isolated and tightly controlled laboratory settings it obviously becomes more difficult to rigorously determine the impact of specific factors. In large-scale overlays in the wild, we do not have full control over all peers and their links, and therefore we cannot ultimately prove the presence or absence of the above demonstrated effects. There is, however, at least indicative evidence that such interactions do occur.

Clear signs for transport layer problems in the Tor overlay can be found in Loesing's Tor performance measurements [10]. There, it is reported that only half of the available bandwidth is effectively used. An explanation for this finding is not provided in [10]. However, it apparently fits the picture that we see in our experiments reported above.

A look at in-the-wild BitTorrent and Tor traces corroborates the real-world existence of TCP interaction effects further. We downloaded a Gentoo Linux installation CD image via a Tor-anonymized HTTP connection (Tor version 0.2.2.8-alpha) and recorded the throughput over time. In




(a) A BitTorrent connection.




(b) A Tor connection.

Fig. 4. Examples of TCP throughputs in real-world overlays.

a similar manner, we monitored a BitTorrent download (with KTorrent version 3.2.4). Throughput-over-time plots are shown in Figure 4, one from a connection between two BitTorrent peers (Fig. 4(a)) and one from the HTTP download via Tor (Fig. 4(b)). A multitude of external, non-controllable factors contribute to the exact behavior of such connections. However, some key features of Figure 2(a) are clearly present. In particular, we notice significant throughput oscillations and sharp throughput drops at several points in time. Similar to what we see in Figure 2(a), connections sometimes break down and may be stalled for several seconds (in the specific plots shown here this happens particularly drastically for the Tor connection before second 200). We take this as an indication that TCP interference effects do in fact occur.

## IV. Traffic Shaping and ACK Prioritization

So far, the effects that we demonstrated in the previous section have not been taken into consideration in the design of overlay networks. However, related phenomena are known in the context of TCP performance on asymmetric links (like, e. g., ADSL Internet access links) [3]. In particular, it has been observed that TCP download speed in asymmetric settings suffers in the presence of significant upload activity, for very similar reasons. A possible remedy is to prioritize TCP ACK segments in the outgoing traffic stream by using traffic shaping mechanisms [3], [25]. This technique is an

important first step towards solving the throughput oscillation problem. In this section, we therefore take it up and adapt it to our needs.

The key idea is easy to understand: the outgoing interface queue is replaced by separate queues for (large) data packets and (small) acknowledgment packets. Acknowledgments are then forwarded to the interface with higher priority. This avoids that the ACKs suffer from long delays due to many large data segments from other connections. It can therefore be expected to counter the effect observed in the previous section. Typically, this modification is combined with a marginal artificial bandwidth throttling applied to the outgoing link. This ensures that the queue builds up before this bottleneck and thus under the control of the traffic shaping module.

Such mechanisms are actually not uncommon in practice. They can be configured using the standard Linux kernel's class-based queueing features and the iproute2 package [30], similar means exist in other operating systems. In fact, there are software packages designed to optimize the performance of hosts with asymmetric Internet connections (examples are [31], [32]), and even some DSL home routers provide corresponding features. However, even though early scientific literature like [3] indicated the need for more detailed investigations, and despite increasing practical deployment prospects, there is surprisingly little scientific discussion on these techniques.

We have set up such an ACK prioritization mechanism in host $B$ in our experimental setting above and assessed its performance impact. We attached a filter to the Linux kernel's so-called root qdisc to identify TCP ACK segments with a total size of less than 64 bytes. The matching packets form one traffic class, another traffic class contains all other outgoing traffic. The two traffic classes are served with different priorities and different bandwidth limits. Specifically, the ACKs are served at a higher priority than the remaining traffic. Furthermore, we assign the remaining traffic queue a fraction of 0.9 of the link bandwidth and combine this with a 5 % bandwidth throttling over both classes. As mentioned above, the latter tweak ensures that the prioritization mechanisms hold sway in the outgoing link queueing system.

Figure 5 shows the results obtained with this modification. The figure corresponds to Figure 3, but now with ACK prioritization enabled. In this simple scenario, the problem is apparently solved. Now both connections make close-to-optimal use of the available bandwidth, and the throughput remains stable throughout the experiment.

## V. Bidirectionally Used TCP Connections

So far, our experiments and explanations have focused on TCP connections with unidirectional data traffic. In many overlay networks, however, TCP connections are used bidirectionally. For instance, BitTorrent's fairness mechanisms explicitly encourage connections to transfer data in both directions at the same time; in Tor many anonymized connections may be multiplexed over a single overlay link. This makes bidirectional data transfer likely to occur. In fact, this characteristic distinguishes typical overlay communication from many other commonly used protocols: be it HTTP, mail transport protocols, or media streaming—in most client-server protocols one communication direction clearly dominates at any point in time.

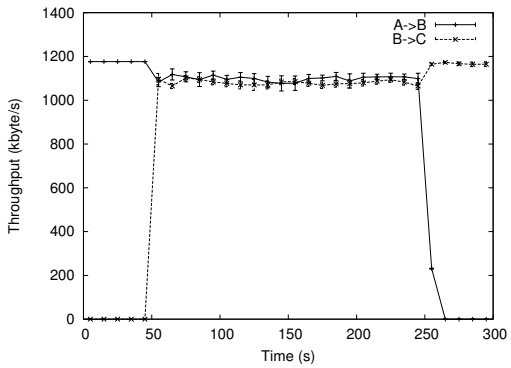



Fig. 5. Impact of ACK prioritization on simultaneous independent up- and downloads.

In case of bidirectional communication, TCP uses piggybacked acknowledgments—this saves (a little) bandwidth. However, it also means that ACKs are not separate packets that can be identified and preferred as described in the previous section. Thus, in case of bidirectional TCP traffic with piggybacked acknowledgments, we should not expect ACK prioritization alone to perform as well as it did for unidirectional traffic.

### *A. Remaining problems with bidirectional traffic*

To see what happens in the case of bidirectional traffic, we have again performed experiments in the same setting as above. Host $B$ still communicates with hosts $A$ and $C$ over one TCP connection each, but now both connections transfer data in both directions simultaneously. Again, the communication between $A$ and $B$ starts at second 0 and ends after 250 seconds, and the connection between $B$ and $C$ sets in after 50 seconds and ends at second 300.

Figure 6 shows the achieved throughput over time for the bidirectional data streams without ACK prioritization. We show the throughput from $A$ to $B$ and from $C$ to $B$—as we have seen before, the incoming data transfer towards $B$ is the problematic direction. In the figure, we see a massive decrease in throughput to around 200 kB/s when both connections are active. There are also high throughput variations.

If we repeat the experiment with the ACK prioritization mechanisms described above, the result are surprising at a first glance: despite piggybacked ACKs, the throughput deterioration virtually vanishes. Both connections now achieve half of the throughput of a single connection, i. e., they share the bandwidth fairly and make (almost) full use of it (note that, unlike in our previous experiments, each direction of $B$'s link is now shared between two data transfers, not only between data and ACK streams). This is shown in Figure 7(a).

The reason for this improvement becomes clear upon closer examination of the traffic. Even though both sides continuously generate traffic at the application layer, there is, in fact, a small number of non-piggybacked acknowledgments. Their presence is easily explained: consider the situation where one side temporarily exhausts its window, but incoming data from the other end

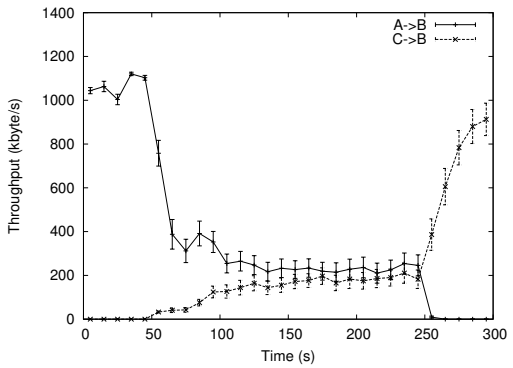



Fig. 6. Bidirectional communication without ACK prioritization.

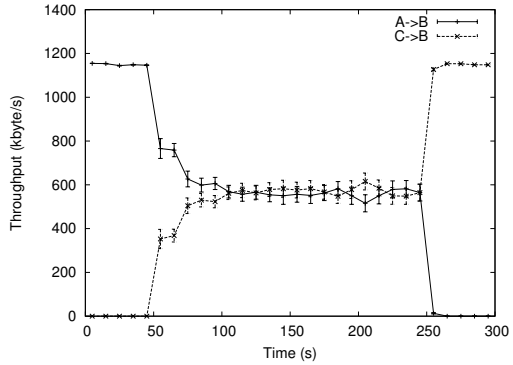
of the connection is to be acknowledged. In that case, despite a non-empty outgoing buffer, non-piggybacked acknowledgments will be generated. Such acknowledgments are handled by the prioritization mechanism; effectively, they are brought to the front of the queue. Upon their arrival, they serve as cumulative ACKs. This apparently suffices to remedy the throughput decrease.

A more detailed look, however, reveals that this remedy is treacherous: the average throughput increases, but strong throughput oscillations still remain. This is not evident in the multi-run averages in Figure 7(a), but upon examination of the individual connections' behavior the still unsteady behavior becomes apparent. We show one typical example in Figure 7(b). Significant oscillations are clearly visible.

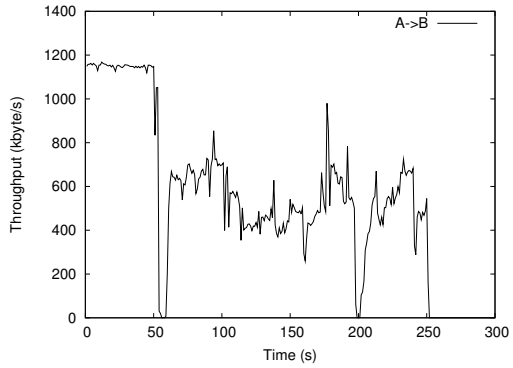
Particularly problematic are the frequent multi-second throughput breakdowns, like those in Figure 7(b) around seconds 55 and 200. To name one example where this is fatal, consider BitTorrent's incentive mechanisms [1]. BitTorrent peers monitor the current download rates from all their neighbors. After each 10 second interval they re-decide how they spend their upload capacity. The neighbors from which the highest download rates were recently achieved are rewarded by uploading to them, others are "choked", i. e., they receive no data. The effectiveness of this mechanism is vital for BitTorrent's fairness and efficiency [6], [8]. Transport-layer effects as we see them here may lead to a good sharing partner being choked. The respective neighbor will then react by choking, too, so that data transfer in both directions is halted at the application layer—potentially for a long time. Even one single throughput breakdown may therefore have a large impact on the effectiveness of the sharing mechanism.

### *B. TCP connection separation*

Clearly, prioritizing the small number of individual ACKs that occur despite bidirectional traffic is not enough. However, piggybacked ACKs cannot be prioritized as described above, because they cannot easily be separated from the data packets—at least not by standard operating systems means. We therefore argue that overlay designs should take this into account by avoiding to use a single TCP connection for longer data transfers in both directions simultaneously. If bidirectional data




(a) Multi-run averages.




(b) Individual connection ($A \rightarrow B$).

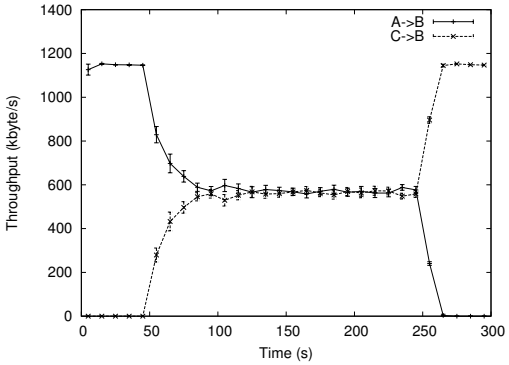
Fig. 7. Bidirectional communication with ACK prioritization.

exchange between the same pair of peers is required, a simple solution is to separate incoming and outgoing traffic into two different TCP connections.

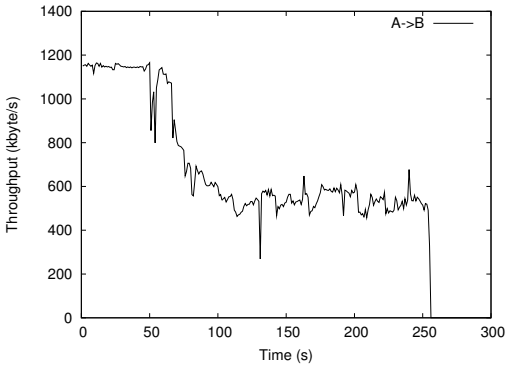
We have verified the efficacy of this proposal in another experiment. Now, two TCP connections are set up between hosts $A$ and $B$, and another two connections between $B$ and $C$. Each of these connections is used in one direction; transmissions start and end as in the previous experiments. Apparently, piggybacked acknowledgments will no longer exist in this setting. This implies a slightly higher overhead for the separate ACK segments, but in return the prioritization mechanisms become fully effective again.

Figure 8(a) shows the throughput of the data streams towards $B$ over time, averaged over 100 experiment runs with 90 % confidence intervals; Figure 8(b) is a throughput-over-time plot for one single representative connection. Here, we observe a significantly more stable throughput. It varies much less over time, and multi-second breakdowns like above do not occur.

Note that by applying our TCP connection separation we do not deteriorate the fairness between other traffic on the same link—because TCP congestion control works independently for both directions of a connection, the behavior of the connections with respect to fairness remains unchanged if connections are directionally split.




(a) Multi-run averages.




(b) Individual connection ($A \rightarrow B$).

Fig. 8. Bidirectional communication with separate TCP connections.

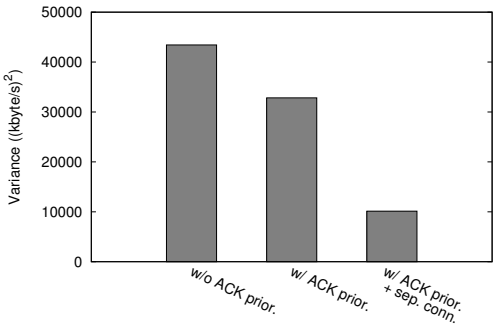



Fig. 9. Throughput variance.

To underline that the observed improvements are indeed statistically significant, we also performed a different evaluation of the experimental results from this section. In Figure 9, we quantify the long-term throughput stability: how strong are the throughput variations after fair throughput sharing has initially been reached? To this end, we consider all throughput samples for data flows towards $B$ from all 10 s time slots between seconds 100 and 250 from all experiment runs. For this set of samples, we calculate the sample variance. A high variance means that the throughput samples are spread over a large range of values, a low variance means that the throughput is generally more balanced. It can therefore be used as a measure for the smoothness of the data transfer.

In the figure, the variance for the experiments without ACK prioritization and with bidirectionally used TCP connections, for the experiments with ACK prioritization only, and for the experiments with separate per-direction TCP connections are shown. The results corroborate and quantify what the individual connection plots in Figures 7(b) and 8(b) already indicated: ACK prioritization brings with it a clear benefit, but only with the per-direction separation of TCP connections can undesirable effects be avoided with full effectiveness.

## VI. Design Implications and Migration Paths

In the design of overlay networks, the TCP transport layer has mostly been perceived as a black box. This point of view is definitely not enough. Undesired interactions between TCP overlay links may incur performance penalties and unexpected effects like throughput breakdowns. Consequently, we argue that researchers and practitioners must take the transport-layer implications of their overlay designs into consideration.

Fortunately, relatively simple strategies yield significant benefits. First, we suggest that overlay application developers encourage their users to make use of ACK prioritization on their Internet link. As mentioned before, this is not difficult, since readily usable software packages like [31] or [32] exist. This technique and its implementations are originally intended for use on asymmetric links, but our results clearly demonstrate their effectiveness also in symmetric settings.

Of even greater significance from a protocol design perspective, though, are the implications for the bidirectional use of overlay links. Our advice for overlay designers is simple: avoid bidirectional traffic on TCP connections whenever the amount of data exceeds that of a few control messages. There are multiple strategies how overlays could take this advice into consideration. The most consequent one would be to design the communication structure of the overlay in a way where each overlay link has a fixed direction, and communication in the opposite direction occurs seldom or never. This is not as unrealistic as it may appear at a first glance. In fact, some existing overlay structures exhibit similar properties (for entirely different reasons, though): just consider tree-based overlay streaming solutions [17], [19] or the always clock-wise message forwarding in the Chord DHT [33].

More easily viable for the migration of many existing overlay designs, however, is a pragmatic approach: setting up two separate TCP connections between each neighboring pair of peers—one for each direction—, as we did in Sec. V-B. The application-layer protocol could otherwise remain unchanged, so this can easily be incorporated in future protocol versions: if two peers establish a connection and, during the initial handshake, find that both support the feature, a second connection is set up. Otherwise, for backward compatibility, one connection is used in both directions. The overlay will incrementally profit if more and more peers are upgraded to the new protocol version. For BitTorrent, for example, this appears to be a viable and promising option.

The drawback of this approach is that it increases the number of open sockets. Depending on the number of open connections and on operating system limits, this may or may not be a problem. Tor routers, for instance, must already deal with sometimes thousands of open sockets, and clearly

approach the limits of some operating systems [9]. For such cases, we recommend a dynamic strategy: the current usage of TCP connections can be monitored. If significant bidirectional traffic on a connection is detected, a second connection is set up and the traffic is directionally separated. If one direction remains unused for a certain time period, one of the connections can be closed again.

## VII. Conclusion

Simultaneous inbound and outbound TCP traffic is common in peer-to-peer overlays. In this paper, we argued that this can cause interactions between overlay links, leading to resource underutilization and high throughput variability. Based on real-world network experiments, we analyzed the problem and discussed how to overcome it in the context of existing overlay networks. Our focus was on solutions that are minimally intrusive and readily realizable, and are thus realistic to deploy.

Our results are important for both existing and future overlay designs. Interactions between transport layer streams are likely and should be taken into account when the communication structure of the overlay is designed. In particular, our results show that ACK prioritization combined with separate, per-direction TCP connections constitutes a promising path.

## References


[1] B. Cohen, "Incentives build robustness in BitTorrent," in *P2PEcon '03: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security '04: Proceedings of the 13th USENIX Security Symposium*, Aug. 2004, pp. 303–320.

[3] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry," RFC 3449 (Best Current Practice), Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3449.txt

[4] D. Marks, F. Tschorsch, and B. Scheuermann, "Unleashing Tor, BitTorrent & co.: How to relieve TCP deficiencies in overlays," in *LCN '10: Proceedings of the 35th Annual IEEE International Conference on Local Computer Networks*, Oct. 2010, to appear.

[5] C.-J. Wu, C.-Y. Li, and J.-M. Ho, "Improving the download time of BitTorrent-like systems," in *ICC '07: Proceedings of the IEEE International Conference on Communications*, June 2007, pp. 1125–1129.

[6] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *SIGCOMM '04: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Sept. 2004, pp. 367–378.

[7] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a BitTorrent networks performance mechanisms," in *INFOCOM '06: Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, Apr. 2006.

[8] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," Tech. Rep., Feb. 10 2006.

[9] R. Dingledine and S. J. Murdoch, "Performance improvements on Tor or, why Tor is slow and what we're going to do about it," Mar. 2009. [Online]. Available: https://www.torproject.org/press/presskit/2009-03-11-performance.pdf

[10] K. Loesing, "Measuring the Tor network from public directory information," in *HotPETs '09: 2nd Workshop on Hot Topics in Privacy Enhancing Technologies*, Aug. 2009.

[11] J. Reardon and I. Goldberg, "Improving Tor using a TCP-over-DTLS tunnel," in *USENIX Security '09: Proceedings of the 18th USENIX Security Symposium*, Aug. 2009.

[12] C. Kiraly, G. Bianchi, and R. Lo Cigno, "Solving performance issues in anonymization overlays with a L3 approach," Univ. degli Studi di Trento, Tech. Rep. DISI-08-041, Ver. 1.1, Sept. 2008.

[13] R. Snader and N. Borisov, "A tune-up for Tor: Improving security and performance in the Tor network," in *NDSS '08: Proceedings of the Network and Distributed System Security Symposium*, Feb. 2008.

[14] J. Liang, R. Kumar, and K. W. Ross, "The FastTrack overlay: A measurement study," *Elsevier Computer Networks*, vol. 50, no. 6, pp. 842–858, 2006.

[15] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *P2P '01: Proceedings of the 1st IEEE International Conference on Peer-to-Peer Computing*, Aug. 2002, pp. 99–100.

[16] O. Heckmann, A. Bock, A. Mauthe, and R. Steinmetz, "The eDonkey file-sharing network," in *INFORMATIK '04: Proceedings of the 34th GI Jahrestagung*, vol. 2, Sept. 2004, pp. 224–228.

[17] J. Janotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, "Overcast: Reliable multicast with an overlay network," in *OSDI '00: Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2000, pp. 197–212.

[18] F. Baccelli, A. Chaintreau, Z. Liu, A. Riabov, and S. Sahu, "Scalability of reliable group communication using overlays," in *INFOCOM '04: Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2004.

[19] G.-I. Kwon and J. W. Byers, "ROMA: Reliable overlay multicast with loosely coupled TCP connections," in *INFOCOM '04: Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2004.

[20] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. F. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1260–1271, Dec. 2006.

[21] H. Pucha and Y. C. Hu, "Overlay TCP: Ending end-to-end transport for higher throughput," in *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2005.

[22] L. Zhang and D. D. Clark, "Oscillating behavior of network traffic: a case study simulation," *Internetworking: Research and Experience*, vol. 1, pp. 101–112, 1990.

[23] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," *ACM SIGCOMM Computer Communication Review*, vol. 21, Sept. 1991.

[24] J. C. Mogul, "Observing TCP dynamics in real networks," in *SIGCOMM '92: Proceedings of the 1992 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 1992, pp. 305–317.

[25] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Improving TCP throughput over two-way asymmetric links: Analysis and solutions," in *SIGMETRICS '98 / PERFORMANCE '98: Joint International Conference on Measurement and Modeling of Computer Systems*, June 1998, pp. 78–89.

[26] B. Ford, "Structured streams: a new transport abstraction," in *SIGCOMM '07: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2007.

[27] W. W. Terpstra, C. Leng, M. Lehn, and A. Buchmann, "Channel-based unidirectional stream protocol (CUSP)," in *INFOCOM '10 Mini Conference: Proceedings of the 29th Annual Joint Conference of the IEEE Computer and Communications Societies Mini Conference*, Mar. 2010.

[28] S. Shalunov, "Low Extra Delay Background Transport (LEDBAT)," Mar. 2010. [Online]. Available: http://tools.ietf.org/id/draft-ietf-ledbat-congestion-01.txt

[29] A. Norberg, "uTorrent transport protocol," June 2009. [Online]. Available: http://www.bittorrent.org/beps/bep_0029.html

[30] B. Hubert, "Linux advanced routing & traffic control," http://lartc.org/.

[31] ——, "The wonder shaper," http://lartc.org/wondershaper/.

[32] cFos Software, "cFosSpeed traffic shaping," http://www.cfos.de/.

[33] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.