Using Multiple Links to Increase the Performance of Bandwidth-Intensive UDP-Based Applications

Kristian Evensen* Dominik Kaspar* Audun F. Hansen* Carsten Griwodz*† Pål Halvorsen*†
*Simula Research Laboratory, Norway, †University of Oslo, Norway,
email:{kristrev, kaspar, audunh, griff, paalh}@simula.no

***Abstract*—Networked devices often come equipped with multiple network interfaces, and bandwidth aggregation is one of the many possible benefits of using multiple interfaces simultaneously. Real-world networks introduce several challenges that have often been ignored by related work on bandwidth aggregation. The challenges include limited connectivity due to NAT-boxes, link heterogeneity and link variability.**

In this paper, we present a transparent solution for proxy-based bandwidth aggregation that is able to overcome the different deployment and link heterogeneity challenges present in real-world networks. Our focus has been on increasing the performance of bandwidth-intensive UDP-based applications, and through evaluation we show that our solution efficiently aggregates bandwidth and increases the in-order throughput. Previously, we introduced a multi-link UDP proxy solution that improves in-order throughput. This paper presents a significant extension and improvement in terms of support for middle-boxes (NAT), congestion control, a client-based resequencer and support for all operating systems.

I. Introduction

Today, an increasing number of end devices are equipped with multiple network interfaces, giving users the opportunity to simultaneously connect to independent Internet service providers. For example, laptops typically come equipped with both LAN- and WLAN-interfaces, while most smartphones are able to connect to both HSDPA- and WLAN-networks. Using multiple networks simultaneously offers many advantages. For example, the links can be used to provide redundancy or bandwidth aggregation. This involves merging several physical links into one logical link that is exposed to the application, offering more bandwidth and potentially higher throughput.

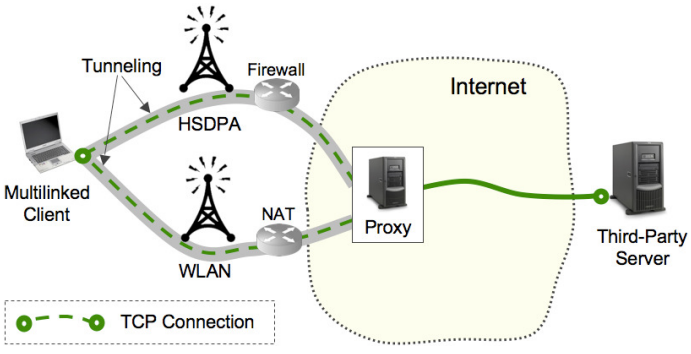

Fig. 1. Real-world multi-link scenario - End-to-end communication between a client and a server over independent access networks using a single TCP connection.

Multi-link communication and bandwidth aggregation have become a hot research topic the last couple of years, much thanks to the increased popularity of mobile devices. Solutions have been proposed on almost every layer of the network stack. However, we have chosen to focus on network-layer solutions. A network-layer approach to bandwidth aggregation can be made completely transparent to the higher layers, and implemented in such a way that the only required changes are routing configurations on the client.

To the best of our knowledge, existing bandwidth aggregation network-layer solutions have ignored the practical challenges introduced by a real-world network environment, such as the scenario presented in figure 1. Common incorrect assumptions include that network interfaces are always assigned globally reachable IP-addresses, the absence of NAT middle-boxes and cross-layer feedback from base stations. In addition, link heterogeneity is often not considered properly.

Based on our own experience and related work, we have defined the following three functional requirements that a bandwidth aggregation proxy-based solution intended for use in real-world networks must meet:

1) Work in the presence of NAT and other middle-boxes. Otherwise, the proxy will not be able to send packets back to the multi-homed client.
2) Stripe packets efficiently over links with different capacity and in a manner that avoids congesting the link. If not, the links will not be fully utilized, in addition to the proxy being unfair to other traffic.
3) Compensate for the delay difference between different networks and network technologies. Most applications process data sequentially and require it to arrive in-order.

Previously, we introduced a limited multi-link UDP proxy-based solution that aggregates bandwidth transparently, efficiently and improves in-order throughput [1]. This paper presents a complete solution with support for middle-boxes (e.g., NAT), congestion control, client-based resequencer and support for all major operating systems.

The proxy and client build a multi-link overlay network consisting of IP-tunnels established over each active interface at the client. The proxy is responsible for scheduling packets and doing bandwidth aggregation, while the client reduces reordering by buffering out-of-order packets and delaying their delivery to the higher layer until reordering is resolved.

We have made thorough evaluations of the new bandwidth aggregation solution, both in a controlled network environ-

ment, for different levels of bandwidth and latency heterogeneity, and with real-world wireless networks. Measurement results show that the proxy efficiently aggregates bandwidth, irrespective of link heterogeneity and dynamics, and increases the in-order throughput compared to when a single is used.

The rest of the paper is structured as follows. Section II presents related work, while section III describes the proxy and client. In section IV, we discuss the results of the different performance measurements, and our work is concluded in section V.

II. Related work

Multi-link communication and bandwidth aggregation have been popular research topics for many years, and solutions to client based aggregation of multiple network interfaces have been proposed on virtually all layers of the network protocol stack. Application layer bandwidth aggregation schemes are suggested in, for example, [2]–[5]. In [2], the FTP protocol is modified to create several connections, and the desired file is split into pieces that are requested when an interface is ready. Wang et al. [4] divide a single TCP-flow into several subflows that are sent over different interfaces, while in [5], we used HTTP pipelining to improve the performance of bulk, HTTP data transfer over multiple heterogeneous interfaces.

Most application layer solutions would work in real-world networks and meet all three functional requirements, as they build on established transport layer protocols. However, one drawback with application layer solutions is that they are tuned for one application or application type, making them unsuited when the goal is to do transparent bandwidth aggregation. Also, changes to the applications are required, which might not be desirable or even possible.

Transport-layer bandwidth aggregation solutions (e.g., [6]–[8]) also require changes to the applications, and depend on modifications to well-established protocols or brand new protocols. These will take years until wide-use or standardization. MPTCP [6] is a new TCP specification that IETF is currently working on, while [7] suggests a new multi-link TCP-extension, TCP PRISM, that makes use of a proxy to aggregate bandwidth.

We believe that a network-layer solution is most applicable, as the bandwidth aggregation can be made transparent to higher layers, and network-layer bandwidth aggregation approaches are presented in [9]–[11]. The approaches all make use of IP-tunnels and either stripe packets on a server or at a proxy, and are based on assumptions that are incorrect in real-world networks and does not coincide with field measurements we have made [12]. For example, the solution presented in [9] relies on link-layer feedback from base stations, while [10] assumes that the difference in RTT can be equalized by adjusting the packet size.

Finally, link-layer bandwidth aggregation solutions (e.g., [13], [14]), known as packet striping, stripes data across several physical channels. Another example of a link-layer solution to utilizing multiple links is presented in [15], where parity codes are applied across multiple channels instead of packets, thereby increasing resilience. However, striping data through multiple heterogeneous networks and to different IP-addresses requires additional and complex changes.

III. Transparent bandwidth aggregation

When aggregating bandwidth at the network layer, a common technique is to use a proxy and a client. Working on the network layer allows for changes to be transparent to the higher layers (including the actual applications running on top), while a proxy removes the need to change the server.

In this section, we present a transparent proxy-based bandwidth aggregation solution. The focus has been on downstream UDP-traffic and all three functional requirements, defined in introduction, are met. In addition, the client can be implemented for any major OS, making it platform independent.

A. Enabling multiple links in real-world networks

By default, even though a client machine has multiple network interfaces and is connected to several networks, one interface is regarded as the default and used by all applications. In order to transparently enable multi-link in real-world networks, our bandwidth aggregation solution uses virtual interfaces and IP-tunneling.

Virtual interfaces behave as normal network interfaces and are for example given a regular IP-address. However, unlike a normal network interface, the virtual interface is owned and controlled by a user-space application. This application receives all the data written to the virtual interface, and is responsible for sending it through an actual network. Virtual interfaces are typically used for IP-tunneling, and applications can make use of a virtual interface, for example, by explicitly binding to it, or desired routes can be configured to go through the tunnel.

Our bandwidth aggregation client and proxy each create a virtual interface. The reason a virtual interface is needed at both sides, is that the data sent through the tunnel gets encapsulated and, thus, must be decapsulated. In addition to the additional network and transport layer header added when a packet is sent by the physical interface, a separate tunneling header is added. This header contains information needed by the client and the proxy. We use UDP as the tunnel transport protocol, and in our current implementation, the total per packet overhead is 24 bytes (including IP- and UDP-header). With a 1500 byte MTU, the overhead is 3.5%, which we believe is acceptable.

An overview of how multi-link is enabled and used for a client with two interfaces, is shown in figure 2. For each interface, an IP-tunnel is established between the client and the proxy, and by doing this, a multi-link overlay network is built. Data is then sent through these tunnels, and NAT hole punching is used to make our solution work in real-world networks. In order to maintain the tunnels and allow the client and proxy to communicate directly, the client sends probe packets to the proxy at given intervals, and the proxy replies. This forces NAT-boxes to keep a state for each ”connection“, allowing packets to flow in both directions. The reason the

NAT hole punching is initiated from the client, is that many NAT-boxes only create a state when the first packet was sent from within its network. Without NAT hole punching and the use of IP-tunnels, the proxy would in most cases not be able to send packets back to the client, as there would be no direct route. The packet resequencer and scheduler will be presented in the two following sections.

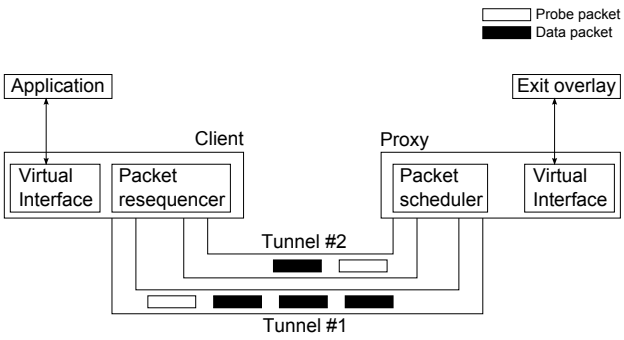

Fig. 2. An overview of our multi-link proxy achitecture running on a client with two active network interfaces

Applications that want to benefit from bandwidth aggregation can either explicitly be bound to the virtual interface, or routes can be configured to go through the overlay network. The proxy can then use source NAT (SNAT) to make sure packets destined for the clients are routed through it. SNAT rewrites the source IP of packets to be that of the proxy, and changes the destination IP back to the client's IP (the virtual interface) when (if) a reply arrives. In order to separate between packets that will and will not have their IP address rewritten, a mapping is used.

Even though our current implementation is for Linux (using TUN-devices [1]), libraries for creating virtual interfaces exist for most OSes, for example Windows and BSD. In other words, because our client does not use any Linux-specific features, it can be ported to any OS.

B. Bandwidth aggregation

In order to aggregate bandwidth efficiently, the proxy must be aware of the capacity of each link and use an intelligent packet scheduler. Otherwise, the proxy will congest the links, schedule packets incorrectly and not utilize the full capacity of each link. For example, if the bandwidth ratio between two links is 3:1 and packets are scheduled round-robin, only a third of the capacity of the fastest link is used.

To get a good estimate of the capacity of each link and be able to do congestion control, we chose to use CCID2 [16], as it is well suited for datagram-based applications that prioritize high bandwidth. CCID2 behaves like TCP's congestion control, and the proxy monitors each link's congestion window. The client acknowledges the packets it receives, and the proxy updates the size of the congestion window according to the same rules as in TCP, with some modifications due to the fact that UDP does not retransmit data.

[1] http://vtun.sourceforge.net/tun/index.html

The packet scheduler is summarized in algorithm 1, and is based on the one used by TCP PRISM [7]. Even though that solution is built for TCP, the congestion controls are similar and the same ideas and challenges apply. When a packet destined for the client arrives at the proxy, the proxy finds all links that have room in their congestion window. If no links have room in their congestion window, the packet is dropped (line 5-8) to avoid potentially causing congestion. Otherwise, the link with the most available capacity (relative to all links) is chosen (line 10-16). The capacity metric is the free space in each link's congestion window.

Algorithm 1 Packet scheduler

```
1:  max_capacity = MIN_VALUE
2:  scheduled_link = None
3:  links = [set of links with an open congestion window]
4:
5:  if links == Empty then
6:      drop packet
7:      return None
8:  end if
9:
10: for all links do
11:     if capacity_link > max_capacity then
12:         max_capacity = capacity_link
13:         scheduled_link = link
14:     end if
15: end for
16: return scheduled_link
```

C. Increasing in-order throughput

Even though bandwidth aggregation increases the available bandwidth, it does not guarantee a higher in-order throughput. That packets arrive in-order is important because most applications process data sequentially.

In a multi-link scenario, most of the packet reordering is caused by latency heterogeneity. With our new bandwidth aggregation solution, we have chosen to compensate for reordering at the client by resequencing packets. This is a similar approach to that used by MPTCP [6]. Our packet resequencer is summarized in figure 3, and works by buffering out-of-order packets at the client.

When a packet is sent from the proxy, it is given two sequence numbers. One is a global sequence number that is valid across all links, while the other is local to one link. The global sequence number is used to detect reordering. Packets are sent in-order from the proxy, so sorting packets according to the global sequence number will tell if reordering has occurred or not. Reordered packets are held in a buffer until the missing packet(s) arrive. In our current implementation, there is no limit on the buffer size, but this should be added if memory is an issue. Packets could then for example be written to the virtual interface when they are pushed out of the buffer.

The local sequence number is used to determine if packets are lost. We assume that there is no internal reordering on a link, so missing local sequence numbers are interpreted as packet loss. If packet loss is detected, the missing global sequence number(s) is ignored and in-order packets are released to the virtual interface.

To avoid deadlocks and head of line blocking, a timeout is used. This timeout is calculated in the same way as TCP's

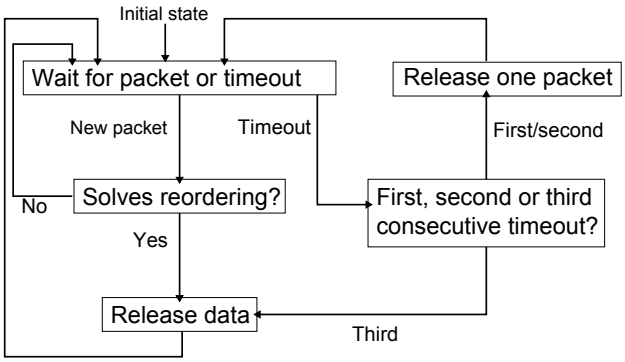

Fig. 3. A state-diagram showing how the resequencer works

Retransmission Timeout (RTO) [17], except that the one way delay (OWD) is used. The OWD is used because it provides a good estimate for the worst-case time between sending a packet and its reception at the client. When the timeout expires for the first and second time, one packet is released. When it expires for the third consecutive time, all packets held in the buffer are released. This is done to reduce the probability of releasing large bursts of out-of-order data.

IV. Performance measurements

To properly evaluate the performance of our bandwidth aggregation proxy, we were interested in its performance in a fully controlled environment, and in the real world. The latter gives an impression of how the proxy will benefit potential users, as well as how it performs with a combination of dynamic bandwidth and latency heterogeneity. A controlled environment allows us to isolate and control each type of heterogeneity, and look at how they affect the performance.

A. Controlled network environment

1) Testbed: Our controlled network environment testbed consisted of three machines, each running Linux 2.6.31-14. The machines were connected directly to each other using 100 Mbit/s Ethernet, and one machine was used as both proxy and sender, the second emulated link delay (when needed), while the third was the multi-link-enabled client. Our own tool was used to generate the constant bitrate stream that was used to measure the performance of our solution. To emulate bandwidth and RTT, the network emulator *netem* [2] was used, together with the hierarchical token bucket.

2) Bandwidth aggregation: A good bandwidth aggregation depends on an efficient packet scheduler and good congestion control, the proxy has to accurately estimate the capacity of each link and select the ”correct” one. Otherwise, if for example a slow link is prioritized, the full capacity of the fast link will not be used. To measure how efficiently the proxy aggregates bandwidth, two sets of experiments were performed. For all the results presented in this section, the server sent a 10 Mbit/s UDP-stream to the client, and the sum of the available bandwidth at the client was always equal to 10 Mbit/s. We experimented with different bandwidths as well, and saw similar results to those presented here.

[2]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

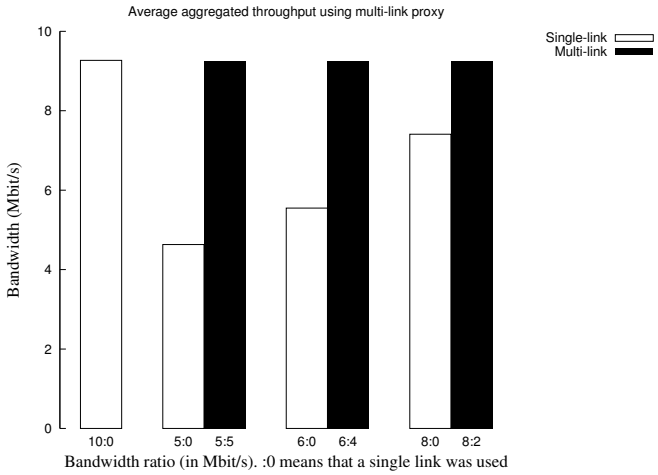

Fig. 4. Achieved bandwidth with fixed bandwidth heterogeneity

In the first series of tests, the proxy was faced with different levels of bandwidth heterogeneity (the RTT was left untouched and was <1 ms). The results are shown in figure 4. Ten one minute long tests were run for each heterogeneity level and the proxy achieved close to the same bandwidth as a single 10Mbit/s link, and gave a significant performance increase over a single link for all levels of heterogeneity. In other words, the congestion control accurately estimated the capacity of each link, which enabled the packet scheduler to make the right scheduling decisions.

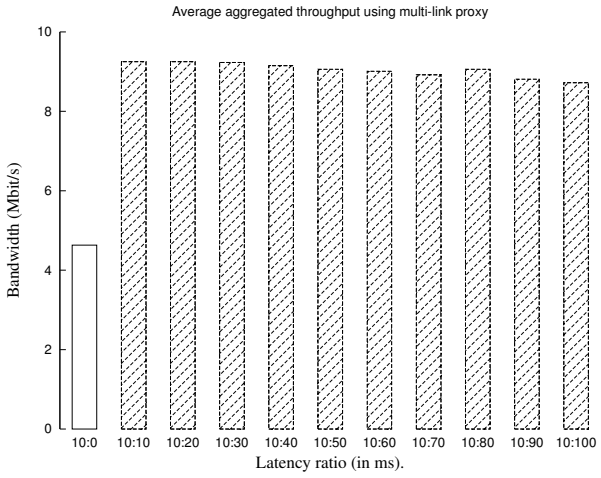

Fig. 5. Achieved bandwidth with fixed latency heterogeneity

The purpose of the second series of tests was to see how latency heterogeneity affects the effectiveness of the bandwidth aggregation. Each link had a bandwidth of 5 Mbit/s, to avoid potential side-effects caused by bandwidth heterogeneity, while the RTT of one link was set to 10 ms, and the other assigned an RTT of r ms, with $r \in \{10, 20, \ldots, 100\}$. As with bandwidth heterogeneity, ten one minute long tests were

run for each heterogeneity level, and the results are shown in figure 5. The proxy significantly improved the performance compared to a single 5 Mbit/s link.

A small decrease in the aggregated bandwidth can be observed as the heterogeneity increases, indicating that the latency heterogeneity will at some point have a significant effect. This is expected with a TCP-like congestion control, like the one we have implemented. As the latency increases, the growth rate of the congestion window decreases. During the initial phase or when packets are lost, the congestion window and thereby throughput will grow at a slower rate.

3) In-order throughput gain: Increasing the in-order throughput is important for most applications, as they depend on processing data sequentially. With the bandwidth aggregation solution presented in this paper, reducing reordering is the responsibility of the resequencer at the client. For measuring the in-order throughput, three sets of tests were run. Two were the same as for bandwidth aggregation, while, in the third, we combined the different bandwidth and latency heterogeneities.

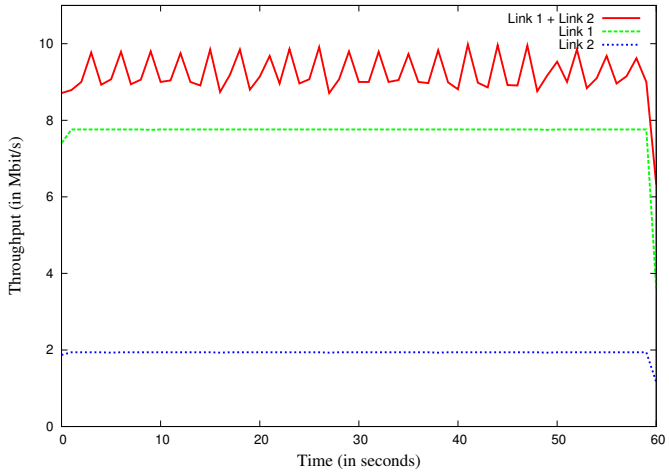

Fig. 6. Achieved in-order throughput with a bandwidth ratio of 8 Mbit/s:2 Mbit/s (equal RTT). The throughput was measured for every second.

One representative sample of the in-order throughput for the most severe bandwidth heterogeneity, 8:2, is shown in figure 6. The in-order throughput increased significantly, however, a bursty pattern can be observed. This is caused by the bandwidth heterogeneity. Due to the difference in capacity, the fast link was allocated a larger share of the packets. When a packet that solved reordering arrived over the slow link, the buffer often contained several packets waiting to be released to the virtual interface and the application, causing the spikes.

Figure 7 displays the measured in-order throughput for a case of worst-case latency heterogeneity (10ms:100ms). As with bandwidth heterogeneity, the in-order throughput was significantly better than that of a single link.

However, a distinct pattern can be seen also in this graph. When the link with the highest RTT got congested and the proxy invoked congestion control, it took a significant amount of time before the aggregated throughput grew back to the previous level. As with the decrease in aggregated bandwidth,

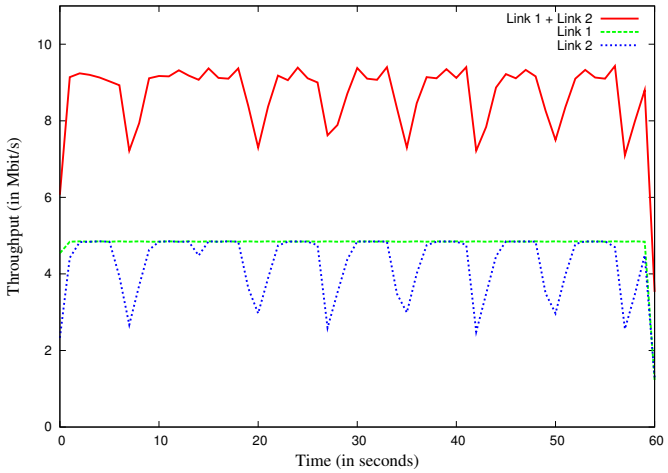

Fig. 7. Achieved in-order throughput with a latency ratio of 10 ms:100 ms (equal bandwidth). The throughput was measured for every second.

this is caused by the increased RTT affecting the growth rate of the congestion window at the proxy. Also, the reason there are no significant throughput spikes, is that the bandwidth was homogeneous (we wanted to isolate the effect of latency heterogeneity), so close to the same number of packets were sent over each link. Thus, there were few out-of-order packets in the buffer.

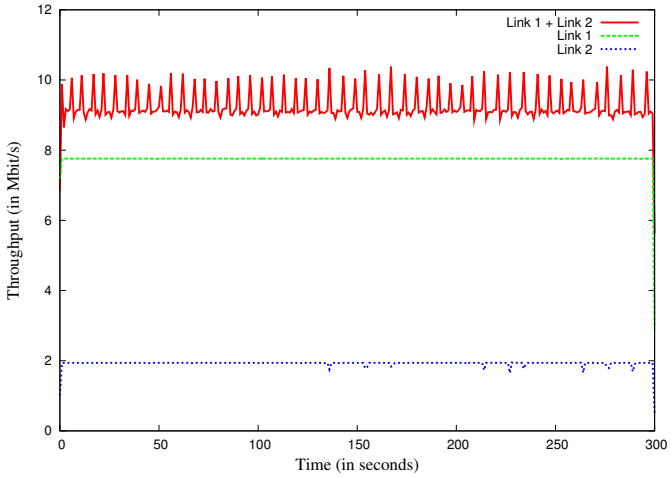

Fig. 8. Achieved in-order throughput with a combination of bandwidth and latency heterogeneity (8 Mbit/s, 10 ms RTT and 2 Mbit/s, 100 ms RTT). The throughput was measured for every second.

In figure 8, we show the results from one experiment with the combination of the worst-case bandwidth and latency heterogeneity. A more bursty traffic pattern can be observed, which was caused by the low bandwidth link also having the highest RTT. Reordering occurred more frequently, and when a packet that solved reordering arrived over the slow link, more packets were waiting in the out-of-order buffer than in the tests with only bandwidth heterogeneity. The reason for the lack of the throughput drops seen in figure 7, was that less traffic went over the high RTT-link. When looking at the logs, we see that the drops were present, however, they are less visible and did

not have a significant effect on the throughput.

B. Real-world networks

1) Testbed: To get an impression of how our bandwidth aggregation solution performs in real-world networks, and in the presence of dynamic bandwidth and latency heterogeneity, we also measured the performance when the client was connected to one public WLAN (placed behind a NAT) and one HSDPA-network. The average bandwidth and measured RTT of the networks were 4 Mbit/s / 25 ms and 2.5 Mbit/s / 60 ms, respectively. The same application was used to generate the network traffic as in the other experiments.

2) Results: As in the experiments performed in the controlled network environment, the sender sent a 10 Mbit/s stream to the client. The average aggregated bandwidth when the client was connected to real-world networks was 5.52 Mbit/s, which is a significant improvement over using only WLAN (which measured an average of 3.34 Mbit/s).

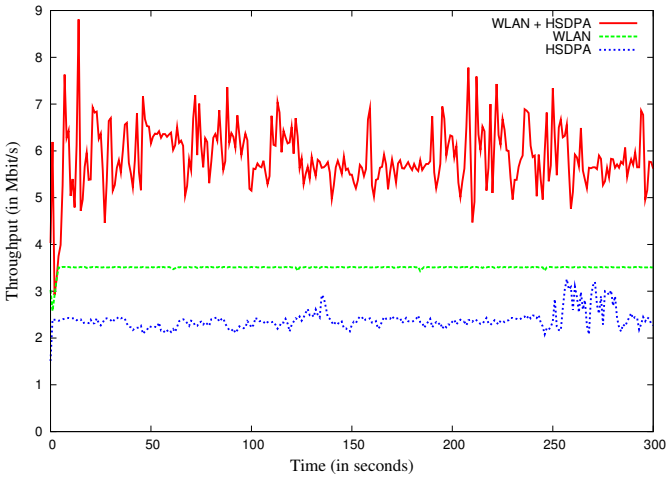

Fig. 9. The in-order throughput experienced in real-world networks. The throughput was measured for every second.

The measured in-order throughput for one experiment is shown in figure 9. As can be seen, the aggregated throughput was significantly more bursty than in the experiments performed in the controlled network environment. This is caused by the dynamic behavior of the links, as well as the combined heterogeneities. A higher number of out-of-order packets will often be buffered at the client, so when reordering is resolved, a larger amount of data will be delivered to the application.

V. Conclusion

In this paper, we have presented a proxy-based bandwidth aggregation solution that improves the performance of bandwidth-intensive UDP-based applications. Virtual interfaces and IP-tunnels are used to build a multi-link overlay network. The proposed solution works in the presence of NAT and can therefore be deployed in real-world networks, and the proxy does congestion control to avoid congesting the links and affecting other traffic.

The solution consists of a proxy and a client, and was thoroughly evaluated in a controlled network environment and in real-world networks. The results from the experiments show that the proxy was able to achieve almost ideal bandwidth aggregation, and that the in-order throughput was significantly increased compared to when a single link was used. In the future, we plan to look at different ways of reducing reordering and the bursty delivery of in-order data to the virtual interface, and thereby to the application. We also plan to add support for more transport protocols, and analyze the performance of different congestion controls and packet schedulers.

Acknowledgements

We would like to thank Paal E. Engelstad for useful discussions and comments.

References

[1] K. Evensen, D. Kaspar, P. Engelstad, A. F. Hansen, C. Griwodz, and P. Halvorsen, "A network-layer proxy for bandwidth aggregation and reduction of IP packet reordering," in *IEEE Conference on Local Computer Networks (LCN)*, October 2009.

[2] M. Allman, H. Kruse, and S. Ostermann, "An application-level solution to TCP's satellite inefficiencies," in *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*, Rye, New York, USA, 1996.

[3] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: video streaming with wwan striping," in *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*. New York, NY, USA: ACM, 2006, pp. 327–336.

[4] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath live streaming via TCP: scheme, performance and benefits," in *ACM CoNEXT*. New York, NY, USA: ACM, 2007, pp. 1–12.

[5] D. Kaspar, K. Evensen, P. Engelstad, and A. F. Hansen, "Using http pipelining to improve progressive download over multiple heterogeneous interfaces," in *International Conference on Communications (ICC)*, 2010.

[6] IETF, "MPTCP Status Pages," Online: http://tools.ietf.org/wg/mptcp.

[7] K.-H. Kim and K. G. Shin, "PRISM: Improving the Performance of Inverse-Multiplexed TCP in Wireless Networks," *IEEE Transactions on Mobile Computing*, 2007.

[8] J. Iyengar, K. Shah, P. Amer, and R. Stewart, "Concurrent multipath transfer using sctp multihoming," in *SPECTS*, 2004. [Online]. Available: www.cis.udel.edu/ amer/PEL/poc/pdf/SPECTS04-iyengarCMTwithSCTPmultihoming.pdf

[9] K. Chebrolu and R. R. Rao, "Bandwidth aggregation for real-time applications in heterogeneous wireless networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 388–403, 2006.

[10] D. S. Phatak, T. Goff, and J. Plusquellic, "IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping," *Computer Networks*, 2003.

[11] K. Chebrolu, B. Raman, and R. R. Rao, "A network layer approach to enable TCP over multiple interfaces," *Wireless Networks*, vol. 11, no. 5, pp. 637–650, 2005.

[12] D. Kaspar, K. Evensen, A. F. Hansen, P. Engelstad, P. Halvorsen, and C. Griwodz, "An analysis of the heterogeneity and IP packet reordering over multiple wireless networks," in *IEEE Symposium on Computers and Communications (ISCC)*, 2009.

[13] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," *ACM SIGCOMM*, 1996.

[14] A. C. Snoeren, "Adaptive inverse multiplexing for wide-area wireless networks," in *GLOBECOM*, 1999.

[15] J. Chesterfield, R. Chakravorty, I. Pratt, S. Banerjee, and P. Rodriguez, "Exploiting diversity to enhance multimedia streaming over cellular links," in *INFOCOM*, March 2005. [Online]. Available: http://citeseer.ist.psu.edu/720915.html

[16] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," IETF RFC 4341, October 2006.

[17] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," IETF RFC 2988, November 2000.