Exploring TCP-friendly Realtime Streaming on Multiple Paths

Christopher Pluntke and Miguel Rio
{c.pluntke, m.rio}@ee.ucl.ac.uk

Abstract

We explore the question how to design and control a realtime streaming protocol on multiple paths with deadline constraints using forward error correction to compensate for packet loss. Sending rates on each interface are TCP friendly to make it fair to competing TCP traffic even if some of the paths share common bottleneck links in the network that are not directly observable. At the same time the protocol beneficially leverages the diversity of its paths to improve the user experience. The system architecture we propose is able to deliver higher goodput to the end user than current single path solutions and outperforms the current standard rate controller for TCP friendliness on multiple paths.

1 Introduction

Live streaming HD traffic usually involves a satellite truck that has to be out in the field close to the source of the transmission. Recently, mobile transmitters in a backpack that use multiple UMTS connections to deliver HD traffic have been introduced and shown to be able to competitive in terms of quality while enabling transmission in highly mobile scenarios and hard to access locations [1]. In our experiments with 3G data connections, we found that in many cases the packet drop probability was low and the per-packet latency was highly variable: it varied from one packet to the next, as well as over longer timescales. Figure 2(a) shows a measurement trace of ping RTTs taken while travelling by train to illustrate the short and long timescale variations.

For real-time video, a high latency packet is as good as lost, and so the main challenge in constructing a real-time mobile video service is to cope with variable latency. Since different 3G providers use different access networks, the RTTs observed on multiple 3G interfaces connecting to the same server are usually diverse such that high RTTs are not observed on all interfaces at the same time. This motivates the use of a multipath system with load balancing between different interfaces. We found however that existing solutions for data transport, both single-path and multi-path [2, 4, 5], do not cope well when packet latencies vary over multiple timescales. In this paper we describe our solutions.

2 Design Problems and Solutions

The main performance measure for realtime streaming is receiver goodput that meets the deadline while guaranteeing a maximum amount of total loss. In this section we give an overview of our system architecture that optimises this performance measure and describe the problems that drove us towards the final design.

2.1 System Architecture

The final system architecture is illustrated in Figure 1. A streaming source sends data packets with a preset delivery deadline to a selection of access servers using multiple outgoing interfaces. Each server can be contacted by multiple interfaces at the same time. We use UDP as underlying transport layer protocol. Since UDP does not guarantee delivery of all its data packets and also does not give any delay guarantees, we have to protect the system against lossy channels and channels with high and variable delay. By sending redundant information, lost or late data packets can be recovered from the extra data. This is called forward error correction (FEC). In this paper, we assume an error correcting code that works on blocks of packets, a so-called *block code*. A FEC encoder for a (n, m)-block code takes n source packets as input and converts them into m coded packets which are transmitted. If any n packets out of m arrive, the code is able to reconstruct the original n data packets. The first question that has to be addressed is how to set the group size m and the redundancy $m - n$ of the block code.

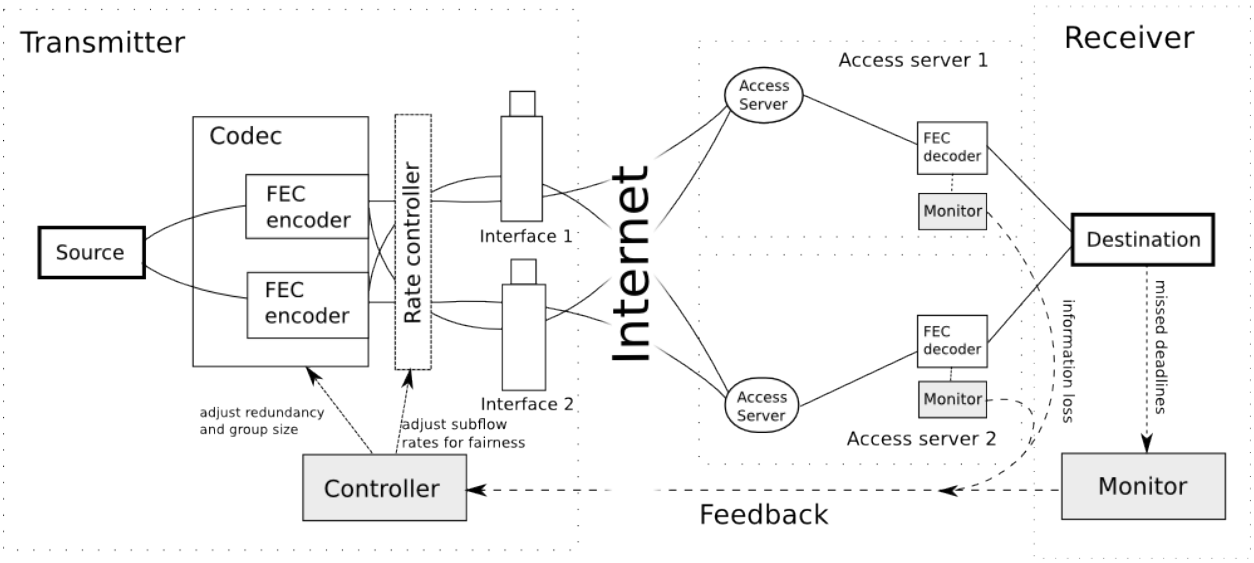


Figure 1: System architecture

We decided to use a FEC encoder and decoder between the sender and each access-server, so each access server can reconstruct the source packets individually and give feedback to the sender. An access server can be a dedicated content delivery network (CDN) server or a user in a peer-to-peer live streaming network which then takes care of the final dissemination of the data to the end users. The access servers give feedback to the sender about the slack time, i.e. the time difference between the arrival of the decoded packet at the receiver and the latest possible arrival imposed by the deadline, and lost packets.

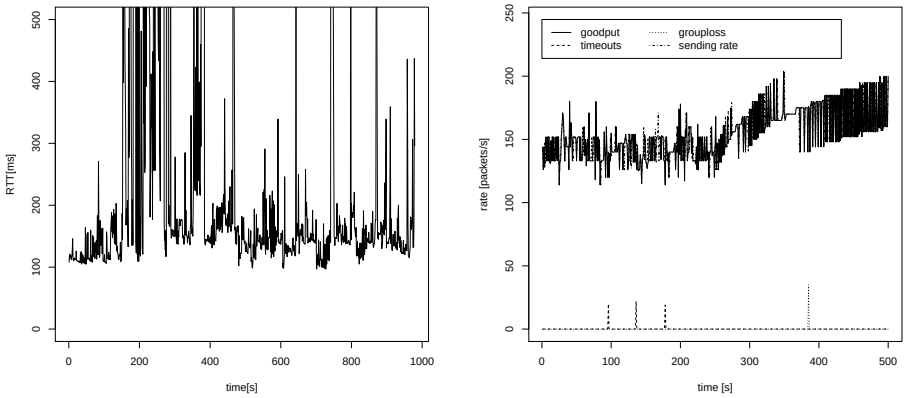
It might seem that using per-interface FEC instead of per-access-server is a more straightforward way to design the system, but this forces the access servers to exchange further data amongst each other to get all data sent from one interface in order to be able to reconstruct the original packets. Not only does this complicate the system, it also introduces more delay and complicates feedback. Hence, in our architecture, we assume that the access servers are able to deliver the decoded data stream reliably to the end user.

2.2 Forward error correction protects against small scale variation of the channel.

In [2], the authors propose EMS, a realtime streaming system on multiple paths. In EMS, redundancy adaptation is done by monitoring information loss and delay-induced losses due to missed deadlines at the receiver and feeding the information back to the sender using a linear control loop with 5% step size. Group size is adjusted by monitoring minimum slack time for all packets over a 30 second time window. The group size is then changed depending on the amount of slack that the latest packet had in the monitoring window.

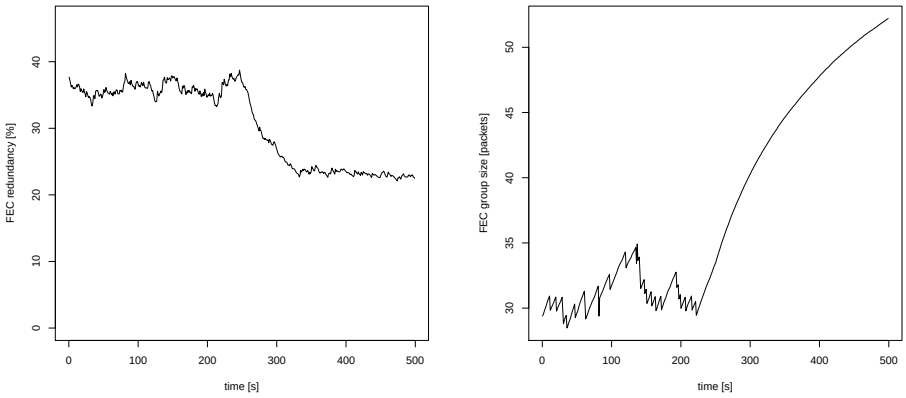
In contrast to this approach, we propose group size and FEC overhead adaptation with per-packet feedback and smooth step sizes based on probabilistic estimation. This enables the system to react to rapidly changing conditions as they are expected in wireless systems. Each multipath flow keeps track of mean and variance of data that cannot be reconstructed by the receiving FEC decoder the so-called information loss. Hence, we have to make sure that the probability of losing more packets than the amount of redundancy allows is smaller than a preset error probability. We achieve this by fitting a normal random variable to the observed information loss using its observed mean and variance in smoothed estimations and then deriving an approximation for the needed redundancy.

Group size adaptation is done on a per-packet basis based on observed slack time. Initially, a slow-start phase increases the group size by 1 packet every time a packet has been successfully received without missed deadlines. This quickly increases the group size to roughly its steady-state value. As soon as the first loss is observed, the group size is set back to the last value before the loss and is updated from then on with every successfully reconstructed packet that is reported by the feedback channel. FEC groups that remain incomplete at the receiver due to too much packet loss time out and send feedback after a fixed timeout deadline. On receiving feedback, the



(a) RTT measurement trace of ping packets

(b) Receiver goodput and loss



(c) FEC redundancy

(d) FEC group size adaptation

Figure 2: A measurement trace of RTT variability (a) and a simulation run starting with the standard MPTCP rate controller and switching to an optimised controller at t=250s (b-d).

group size is increased linearily if slack time is positive and reduced multiplicatively if slack time is negative.

2.3 Load balancing adapts to long term variations of the channel

Long term changes in mean drop rate and RTT cannot be efficiently handled by FEC since this would involve too much overhead in terms of redundancy. Instead, load balancing traffic away from high RTT and high loss rate interfaces can better address long term changes. However, a load balancer cannot just assign any rate to any flow, it has to take a notion of fairness into account to prevent hurting competing flows of other users too much. The fairness properties of TCP traffic are usually used as a guideline for congestion control of streaming protocols. TCP traffic adapts its sending rate according to the path characteristics. This feedback enables multiple TCP flows to share bottleneck links fairly without direct knowledge about if and where bottlenecks are present. Since most of Internet traffic is TCP, some kind of rate control is desirable for UDP traffic as well since overly aggressive UDP flows can send with much higher rate than TCP causing competing TCP flows to overly reduce their rates. EMS [2] does not consider TCP-friendliness, but we see it as one of the main features that enable deployability. In general, TCP adapts its sending rate, x^{TCP}, depending on roundtrip time, RTT, and loss rate, p, according to the following formula: $x^{TCP} = \frac{\sqrt{2}}{RTT\sqrt{p}}$. The standard way to make a UDP flow TCP friendly is to measure loss rate and RTT of the path and then adapt the sending rate to the rate TCP would get. This approach is called equation-based rate control and it has been intensely studied for single path connections in [3].

A TCP controller for multiple paths, multipath TCP (MPTCP), has been introduced in [6]. It accomplishes

cooperation between separate flows in the network such that if all flows are MPTCP, the global resource usage of the network is optimal. In addition, it addresses the problem of unfairness if all subflows of one end-to-end multipath flow share common bottleneck links along the way. Therefore, the coupled rate controller of MPTCP makes sure that an end-to-end flow is not more aggressive than standard TCP even in case of shared bottleneck links. Applying the standard approach for achieving TCP-friendliness in this setting again means to measure roundtrip time, RTT_r, and loss rate, p_r for path r, and adapt the subflow rates to the rate, x_r, that MPTCP would get: $x_r = \sqrt{2a}\frac{1/p_r}{RTT_r\sqrt{\sum_s \frac{1}{p_s}}}$ with $a = \sum_r x_r RTT_r \frac{max_r x_r/RTT_r}{(\sum_r x_r)^2}$.

We now argue that using the standard approach for TCP-friendliness with the MPTCP controller can lead to suboptimal results in terms of goodput. In general, there are two problems: Firstly, the MPTCP controller does not take RTTs into account when choosing its rates, so the FEC controller might have to use high redundancy to protect against timeouts: As an example for this behaviour, let's assume that the source uses three interfaces and there is only one access server in the network. The deadline constraint is set to 300ms. RTTs are assumed to be i.i.d. exponentially distributed with a given mean per subflow and losses are independent. Let the mean RTTs and loss rates be as follows: $(p_1 = 10\%, RTT_1 = 10ms), (p_2 = 2\%, RTT_2 = 90ms), (p_3 = 1\%, RTT_3 = 290ms)$. The resulting rates that the MPTCP controller gives are: $x_1 = 88.2$ [packets/s], $x_2 = 88.2$ [packets/s], $x_3 = 47.1$ [packets/s]. Running a simulation of the entire system with these rates leads to a receiver goodput of 133 [packets/s] using FEC redundancy of 40%.

On the other hand, sending only on path 1 is also TCP-friendly. TCP-friendliness demands that path 1 sends with at most $x_1 = 223.6$ [packets/s]. With this choice of rates, the receiver goodput is 173 [packets/s] while FEC redundancy stays at 22%. This is an increase in goodput of 30% in comparison to the MPTCP controller. Figure 2 shows traces of a simulation run started with the MPTCP controller. At t=250s, we switched to the the alternative controller that sends everything on the first path to make the difference clear. The obvious solution to treat timeouts as loss and let the MPTCP controller handle the load balancing also does not work well. The main problem is that if many packets time out, the MPTCP controller is overly friendly and reduces its rate too much.

Our solution is based on solving an optimisation problem to optimise the subflow rates. The objective function is maximise total goodput subject to constraints on the permissible rates of the subflows to guarantee TCP friendliness. We are currently still exploring this solution and further alternatives.

3 Conclusion

We introduced a new architecture for realtime live-streaming on multiple paths that is friendly to competing TCP traffic. We identified the main problems that inspired our choices. Statistical estimation enabled us to derive a simple algorithm for update of FEC redundancy and basic control is used to update group sizes. We showed that using standard equation-based rate control to adapt subflow rates might lead to high throughput but suboptimal receiver goodput and we proposed that modifying the rate controller while staying TCP friendly can lead to higher goodput.

References

[1] www.liveu.tv.

[2] A. L. Chow, H. Yang, C. H. Xia, M. Kim, Z. Liu, and H. Lei. EMS: Encoded Multipath Streaming for Real-time Live Streaming Applications. In *Proceedings of ICNP*, 2009.

[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based Congestion Control for Unicast Applications. In *Proceedings of SIGCOMM*, pages 43–56, New York, NY, USA, 2000. ACM.

[4] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. Internet-Draft draft-ietf-mptcp-architecture-05, Internet Engineering Task Force, Jan. 2011. Work in Progress.

[5] C. Raiciu, M. Handley, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols. Internet-Draft draft-ietf-mptcp-congestion-01, Internet Engineering Task Force, Jan. 2011. Work in Progress.

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