Low Delay Random Linear Coding and Scheduling Over Multiple Interfaces

Andres Garcia-Saavedra, Mohammad Karzand, and Douglas J. Leith, *Senior Member, IEEE*

Abstract— High-performance real-time applications, expected to be of importance in the upcoming 5G era, such as virtual and augmented reality or tele-presence, have stringent requirements on throughput and *per-packet in-order delivery delay*. Use of multipath transport is gaining momentum for supporting these applications. However, building an efficient, low latency multipath transfer mechanism remains highly challenging. The primary reason for this is that the delivery delay along each path is typically uncertain and time-varying. When the transmitter ignores the stochastic nature of the path delays, then packets sent along different paths frequently arrive out of order and need to be buffered at the receiver to allow in-order delivery to the application. In this paper we propose S-EDPF (Stochastic Earliest Delivery Path First), a generalization of EDPF which takes into account uncertainty and time-variation in path delays yet has low-complexity suited to practical implementation. Moreover, we integrate a novel low-delay Forward Error Correction (FEC) scheme into S-EDPF in a principled manner by deriving the optimal schedule for coded packets across multiple paths. Finally, we demonstrate, both analytically and empirically, that S-EDPF is effective at mitigating the delay impact of reordering and loss in multipath transport protocols, offering substantial performance gains over the state of the art.

Index Terms—Low-delay communications, stochastic scheduling, network coding, multipath transport, Tactile Internet.

1 Introduction

Some of the most interesting (and challenging) services envisioned for the next generation of mobile networks generate real-time streams of data packets [1]. Virtual and augmented reality require continuous low latency feedback to enable high-precision perception of objects and timely updates in response to user movements and changes in the surrounding environment. Tele-presence needs continuous real-time and accurate multi-sensory feedback. In summary: very low-delay, high-rate reliable streaming of information is expected to be key to enabling the next generation of applications of the Tactile Internet [2].

To illustrate the network footprint of the types of application we are interested in, we carry out the following simple experiment. The setup consists of a server running `openface` [3], a state of the art face recognition software, as a service and a laptop running a browser as the client. The browser sends images over a websocket to the server for pattern recognition processing and the server sends back a bounding box plus a tag for each face in the image. This setup mimics a simple augmented reality application if we imagine running an app which uses the video camera on a phone. The app, with the help of the server which does the actual pattern recognition legwork, adds an overlay in real-time which tags all of the people in the field of view with their names. In our tests the bandwidth of the link connecting laptop and server is large enough so that it is not a bottleneck. We plot in Fig. 1 an example of the packet rates measured over a 100-secs experiment. This data illustrates how the service requires high packet rates during the period in which the application is running. The transport protocol used in this experiment, TCP, guarantees in-order delivery and no losses at the cost of delay by employing a buffer at the receiver and Automatic Repeat reQuest (ARQ). Smooth rendering of the overlay requires that packets are transmitted to/from the `openface` server with less than around 150 msecs *per-packet in-order delivery delay* [4].

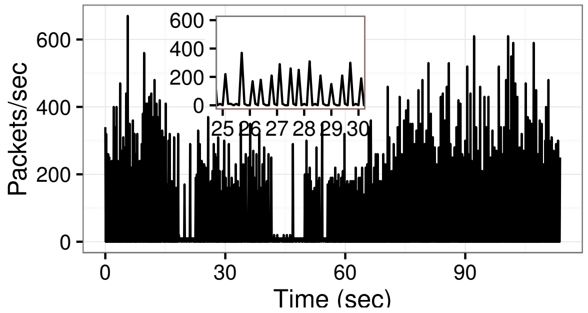


Fig. 1. Network footprint of `openface` in an augmented reality setup.

Transporting data between a source and destination in parallel along multiple paths is well-recognised as a potential means to increase throughput [5]–[7], improve resilience (if one path breaks, the connection can gracefully failover to the remaining paths) [8], and lower delay (by creating a "race" between different paths). Hence, there is much interest in its use for next generation services [2]. However, building an efficient, low latency multipath transfer mechanism remains highly challenging. The primary reason for this is that the delivery delay along each path is typically uncertain and time-varying [9]–[11] due to queueing, link layer retransmission, the action of congestion control, etc. When the transmitter ignores the stochastic nature of the

• *A. Garcia-Saavedra is with NEC Laboratories Europe, Germany. E-mail: andres.garcia.saavedra@neclab.eu*
• *M. Karzand and D. J. Leith are with Trinity College Dublin, Ireland. E-mail: {karzandm, doug.leith}@scss.tcd.ie*
• *Work supported by SFI grant 11/PI/1177 and 13/RC/2077.*

path delays when deciding which packets to send down which path, e.g. by making scheduling decisions based on only the average path delay, then packets sent along different paths frequently arrive out of order and need to be buffered at the receiver to allow in-order delivery to the application [12]. This so called *head-of-line blocking* (HOL) can cause substantial delay in packet delivery[1]. A second source of HOL, in addition to reordering, is packet loss due to queue overflow, wireless link losses, etc. When ARQ is used to recover from packet loss, as is often the case, retransmission incurs at least one round-trip time of additional delay (more if the retransmitted packet is itself lost). This delay is incurred not only by the packet that is lost but also by subsequent packets that remain buffered at the receiver until the lost packet is finally received.

In this paper we study achieving low latency multipath operation in the face of uncertain and time-varying path delays and packet loss. Our main contributions can be summarized as follows:

- We propose S-EDPF (Stochastic Earliest Delivery Path First), a generalization of EDPF which takes into account uncertainty and time-variation in path delays. To our knowledge, this is one of the first multipath schedulers that takes these into account (existing multipath schedulers mostly use EDPF, or a computationally cheap approximation to EDPF, with a measurement of average path delay and ignore time-variations). The major challenge is to contain the computational complexity since the stochastic problem may be intractably complex to solve in real time at high data-rates. In §2 we address this challenge and present a low-complexity scheduler suited to practical implementation.
- We extend our preliminary work in [42] on low delay code constructions for single path connections to the multipath setting and integrate it within S-EDPF in a principled manner by deriving the optimal schedule for coded packets across multiple paths. This is one of the first analytic results for scheduling of Forward Error Correction (FEC) packets in multipath. We demonstrate that coding is effective at mitigating the delay impact of reordering in multipath transport protocols due to variable path latencies—to our knowledge, first studied in this paper—and also the delay cost due to packet losses.
- We present a prototype implementation and perform a thorough performance evaluation of S-EDPF. This includes an analytical evaluation, simulations, and an extensive experimental campaign that includes controlled, semi-controlled, and "wild" scenarios. For benchmarking, we compare against a number of approaches which are representative of the main categories of multipath scheduler present in the related literature: EDPF (optimal for paths with constant delay and no losses) combined with ARQ (optimal for low-latency links with losses, used in most of the literature) and with block FEC codes (which maximize throughput performance over lossy links when the block size is sufficiently large).

1. Reordering is a major contributor to the latency performance of multipath transports, e.g. [13] measured the performance of Multipath TCP (MPTCP) with real US mobile operators and found that roughly 20% of packets suffer an additional delay larger than 150 ms *caused only by reordering and the resulting buffering at the receiver*.

1.1 Related Work

Two extreme cases are well understood. Namely, when the delay on each path is known and constant and paths are loss-free, then Earliest Delivery Path First (EDPF) [14] ensures in-order packet arrivals and is optimal in the sense that it minimises in-order delivery delay and maximises throughput. When the delay on each path is very low (so only one packet at a time is ever in flight) and the paths are lossy, then ARQ is optimal in the sense that it minimises in-order delivery delay and maximises throughput. Multipath transport over the Internet does not fall into either case since path delays are uncertain and time-varying, relatively large (over a 25Mbps link with 50ms RTT there may be >100 packets in flight) and paths are often lossy. A third case is also well understood, namely when paths are lossy then use of block code FEC (which includes rateless codes) maximises throughput provided the block size is sufficiently large, but this comes at the cost of high delay since delay scales with the block size. Note that, despite their poor fit with the characteristics of multipath over the Internet, EDPF (or approximations thereof), ARQ and/or block code FEC are used in almost all previous work on this topic.

Perhaps the most well-known example of a multipath transport protocol that guarantees in-order delivery is Multipath TCP (MPTCP) [15]. The Linux implementation of MPTCP supports two schedulers [9]: a *round-robin* scheduler that iterates over each path regardless of their properties, and the *lowest RTT First* (lowRTT) scheduler, a practical approximation to EDPF that gives priority to paths with lower round-trip times. CMT-SCTP [16] is another relevant protocol which has some similarities with MPTCP. It implements a selective ARQ scheme similar to SACK TCP, flow control and congestion control based on a congestion window (cwnd). A key difference with MPTCP is that SCTP's congestion window indicates how much data can be sent, rather than *which* data to send. In this way, when cwnd space is available in two or more paths, data can be assigned arbitrarily to each of these paths. Actual implementations of SCTP today use simple round robin or "First Come, First Served" scheduling algorithms [8], [17]. In [18] an improved scheduler is proposed which is similar to the lowRTT scheduler in MPTCP. In summary, both MPTCP and SCTP are frameworks where any scheduling algorithm can be integrated; this includes all of the algorithms we review next and also the scheduler that we propose in our paper.

EDPF has served as the baseline for much work over the years. As explained above, EDPF (and variants, e.g. [18]–[21]) has the fundamental disadvantage of not considering the stochastic time-varying nature of path delays. Moreover, the seminal work of [14] (and most others) does not consider losses. Other work (e.g. [21]), does consider loss, but combines EDPF with use of ARQ for loss recovery. See [12] for a survey of other schedulers with similar limitations.

Regarding loss recovery, a wealth of approaches based on ARQ (e.g. used in MPTCP and CMT-SCTP) and FEC have been proposed in the past. Network coding has been proposed for mesh and multicast networks, notable examples being CORE [22] (a fixed-rate code), SlideOR [23] and

CoMP [24] (non-systematic[2] random linear codes). Network coding, using a non-systematic random linear code, was first introduced in TCP by Sundararajan *et alia* [5]. Maelstrom [25] and LT-TCP [26] are also worth highlighting but they use simple LT codes and are blind to spare capacity. In the context of multipath, a fixed-rate block code is employed in MPLOT [27], a rateless LT code in HMTP [28] and PluriBus [11], a rateless Raptor code in FMTCP [6], a Reed-Solomon code in [29], and a systematic random linear code in SC-MPTCP [30]. All this work using FEC is well suited to maximising goodput (provided the block size is sufficiently large) but, as we will show later, *they all pay a high price in per-packet in-order delivery delay because they are based on block codes and delay scales with the block size*. Perhaps the closest work to ours in its focus on low latency is ReMP TCP [31] which, like us, can trade off bandwidth for delay gains. However ReMP TCP uses a primitive repetition code and, as in all of the above literature, does not take into account uncertainty and time-variation in the path delays.

There exists an extensive literature related specifically to video applications, including live video streaming. For example, [32] drops selected video frames when bandwidth is constrained. MSPlayer [33] adjusts the size of video chunks depending on the bandwidth available on the paths with the aim that chunk transfers scheduled across different paths complete at roughly the same time. The work reported in [34] uses a similar scheme though considering energy consumption. Both these approaches implement a bandwidth predictor (as do many others, e.g. [35]) to try to match the video chunk size to the available bandwidth. These differ fundamentally from our work because they focus on loss-tolerant video coding techniques, i.e. operate at the application layer rather than the transport layer.

1.2 The Cost Of Ignoring Stochasticity of Path Delay

We argue that to achieve low-delay high-rate support for real-time applications (e.g. see Fig. 1) over unreliable—lossy and variable—wireless links, spare network capacity should be used to transmit redundant packets (coded data) to achieve better protection against losses and fluctuations in path delay. Use of multipath protocols is a key enabler for this approach since it *creates* the necessary capacity. Importantly, however, for this strategy to work we must avoid creating *new* latency issues, such as excessive head of line blocking, caused by the use of multipath.

With this in mind, we note that ignoring the stochastic time-varying nature of path delays can greatly increase latency when multiple paths are used. Let us present some simple examples to illustrate this. We begin by considering a scenario where we have to send two packets and we can use up to two paths to do this (possibly, but not necessarily, in parallel). Assume now that the transmission delay of each path (time taken between a packet reaching the head of the sender output queue and the last bit is actually put into the wire/air) is an independent and identically distributed (i.i.d.) Gaussian random variable with parameters (μ_1, σ_1) and (μ_2, σ_2) (neglecting propagation delays for the moment—time it takes for one bit to *fly* from source to destination across the Internet). As mentioned earlier, EDPF (an optimal scheduler when the path delay is constant and there are no losses [18]–[21]) bases its decisions on the average arrival time of each packet. It hence schedules the first packet on the path with the lower mean (without loss of generality, assume this is μ_1), and the second packet on the other path when $\mu_2 < 2\mu_1$ or on the same path otherwise (in this case parallelizing would not bring any gain). Now consider the case where $\mu_2 < 2\mu_1$. In this case, the actual in-order delivery delay (after buffering caused by possible reordering) of the second packet is a random variable which is the maximum of two Gaussian random variables and its mean is: $\mu_1\Phi(\frac{\mu_1-\mu_2}{\Theta}) + \mu_2\Phi(\frac{\mu_2-\mu_1}{\Theta}) + \Theta\phi(\frac{\mu_1-\mu_2}{\Theta})$, where $\Phi(\cdot)$ and $\phi(\cdot)$ are, respectively, the probability density function (pdf) and the cumulative distribution function (cdf) of the standard normal distribution and $\Theta = \sqrt{\sigma_1^2 + \sigma_2^2}$ [36]. This value is larger than either μ_1 or μ_2 but, depending on σ_1 and σ_2, it might be larger than $2\mu_1$, which means that the optimal decision, contrary to what EDPF suggests, may be to send both packets on path 1. This example illustrates that, depending on the relative *variances* of the delays on the different paths, the optimal decision which minimises per-packet delay can differ significantly from that taken when only considering the average path delays.

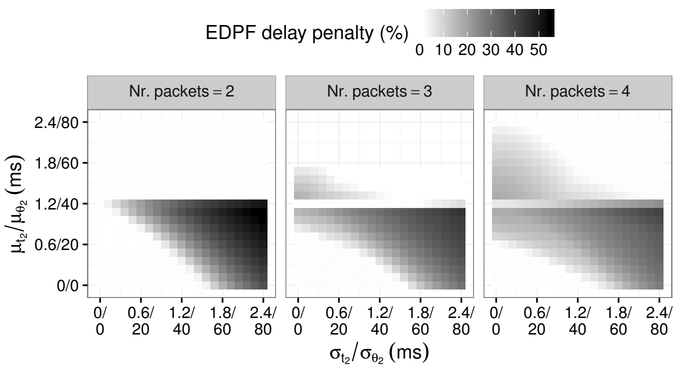


Fig. 2. Example illustrating the sub-optimality of EDPF when path delays are uncertain and time-varying. Two paths. Both transmission delay t_i and propagation delay θ_i on path i follow a normal distribution with parameters $t_i \sim N(\mu_{t_i}, \sigma_{t_i})$ and $\theta_i \sim N(\mu_{\theta_i}, \sigma_{\theta_i})$, respectively. On path 1, $t_1 \sim N(1.2, 0)$ ms and $\theta_1 \sim N(40, 0)$ ms. We vary the delay parameters on path 2. The shading indicates the delay penalty of using EDPF rather than the optimal choice found by exhaustive search.

In order to further illustrate this point for a wider range of scenarios, we now consider a situation where both the transmission delay t_i and the propagation delay θ_i on path i follow normal distributions, $t_i \sim N(\mu_{t_i}, \sigma_{t_i})$ and $\theta_i \sim N(\mu_{\theta_i}, \sigma_{\theta_i})$, respectively. In Fig. 2, we fix the parameters in path 1 to $t_1 \sim N(1.2\text{ms}, 0\text{ms})$ (transmitting a 1500-byte packet on a 10-Mb/s link takes 1.2ms) and $\theta_1 \sim N(40\text{ms}, 0\text{ms})$, and show the in-order delivery delay achieved with EDPF relative to that with an optimal packet schedule (found via exhaustive search) for a wide variety delay parameters on path 2[3]. Considering again the case where only two packets are to be sent, it can be seen that

2. Systematic codes include the input data bits in the output stream unmodified, followed by error-correcting coded information. Conversely, non-systematic codes do not include unmodified bytes into the output stream.

3. Random samples are left-censored at 1 ms (the lowest possible delay).

EDPF makes a good choice when the mean delay on the second path is larger than that of the first path. The reason is that in this case the optimal choice is to send both packets back-to-back through path 1 because the extra delay of the second path does not compensate the waiting time of the second packet (1.2 ms in this example). In contrast, when the delay of path 2 is lower than path 1's, EDPF chooses to send both packets through path 2. However, as the figure illustrates, given the variability in that path, this choice often leads to high in-order delivery delay. Fig. 2 also shows the corresponding data when sending 3 and 4 packets. We do not show data for larger numbers of packets as the exhaustive search approach quickly becomes too time-consuming, but it can be seen that as the number of packets grows the use EDPF incurs a penalty in an increasingly large number of configurations. Finally, we note that though we have made simplifying assumptions in the above analysis (Gaussian i.i.d. random variables), we make similar observations in our experimental evaluation in §5.4.

From the above analysis, we conclude that a careful exploitation of stochastic information—other than averages—can lead to a substantial reduction of out of order packet arrivals. However, stochastic problems are generally much harder to solve and hence particularly challenging for applications like the one shown in Fig. 1. In the next section, we propose a low-complex stochastic scheduler to this aim.

2 Multipath Stochastic Scheduler

In this section we assume lossless channels and address the problem of scheduling packets such that the impact of out of order arrivals in the presence of random path delays is minimised. We put a particular focus on achieving a low complex solution suitable for high-rate applications like the one shown in Fig. 1. The extension to lossy paths is considered in §3.

2.1 Model Description

We have to schedule $\mathcal{K} = \{1, 2, \dots\}$ packets across $\mathcal{P} = \{1, \dots, P\}$ paths with the goal of minimising per-packet in-order delivery delay. Assume that on each path the time at the sender is slotted, indexed by $\mathcal{S} = \{1, 2, \dots\}$, such that one packet can be transmitted in a slot. This abstraction allows us to operate independently of any rate or congestion control algorithm. Let $t_{p,s}$ denote the start time in seconds of slot s on path p. We do not assume that the slots on different paths are aligned or that slots have fixed duration, and so the link rate of each path may change over time due to e.g. the action of congestion control. The time at which a packet sent in slot s on path p arrives at the destination is a random variable $a_{p,s}$, with $a_{p,s} \geq t_{p,s}$ to respect causality.

While packet reordering can occur within a single path (e.g. due to sudden routing changes), it is relatively infrequent compared to the multipath case [37], and so for design purposes we adopt the mild assumption[4] that there is no reordering of arrivals within the same path. Formally, consider two slots with indices a and b on path p, then when $a < b$, $a_{p,a} < a_{p,b}$. It is important to stress that this assumption applies only to packets sent on the *same* path. Packets on different paths with different (random) delays may still arrive out of order. The above implies that the delays experienced by packets on the same path are correlated and not i.i.d. To make this explicit, let random variable $\Delta_{p,s,s+1}{=}a_{p,s+1}{-}a_{p,s} \geq 0$, $s = 1, 2, \dots$ and $\Delta_{p,0,1} := a_{p,1}$. Then,

$$a_{p,s} = \sum_{r=1}^{s} \Delta_{p,r-1,r} \tag{1}$$

where $a_{p,1}$ can be computed as $a_{p,1} = \theta_{p,1} + L_{p,1}/B_{p,1}$, where $\theta_{p,1}$, $L_{p,1}$ and $B_{p,1}$ are the *propagation* delay experienced, the *size* in bits of the scheduled packet, and the access link *bitrate* on slot 1 and path p, respectively.

To facilitate scheduling, for design purposes we make the regularity assumption that $\Delta_{p,s,s+1}$ is i.i.d. ($\Delta_{p,s,s+1} \sim \Delta_p$).[5] We also assume that a packet is transmitted in every slot. Although a lower in-order delivery delay might strictly be possible e.g. by reverting to single path operation where packets are sent only along the path with lowest delay, we argue that the associated loss in throughput is generally too great and that a better approach is to transmit redundant coded information to trade off capacity granted by congestion control (possibly spare capacity, but not necessarily) for delay gains. As we show in §3, this approach has the additional advantage of reducing delay even in the presence of packet losses.

2.2 Minimum Delay Packet Scheduling

Let $p_k \in \mathcal{P}$ denote the path on which packet $k \in \mathcal{K}$ is transmitted, and let $s_k \in \mathcal{S}$ denote the slot on path p_k in which packet k is transmitted. Packet k is thus sent at time t_{p_k,s_k} and the time at which packet k arrives is random variable a_{p_k,s_k}. At the receiver we require in-order delivery of packets arriving from multiple paths. To achieve this, the receiver maintains a reassembly buffer where out of order packets are held until they can be delivered to the application in order. The delivery time of packet k is therefore the random variable,

$$Y_k = \max\{a_{p_1,s_1}, a_{p_2,s_2}, \dots, a_{p_k,s_k}\} \tag{2}$$

It will prove useful later to rewrite this expression equivalently as follows. Let $K_{p,k} := \{q \in \{1, \dots, k-1\}, p_q = p\}$ denote the set of packets sent on path p with indices lower than that of packet k and

$$Y_{p,k} := \max\{a_{p,s_q} : q \in K_{p,k}\} \tag{3}$$

We then have that

$$Y_k = \max\{Y_{1,k}, \dots, Y_{P,k}, a_{p_k,s_k}\} \tag{4}$$

4. Note that in §5.4 we evaluate performance in real-life environments where there may be within-path re-ordering, which helps to validate this design approach.

5. Although this assumption may be relaxed (e.g. by keeping track of correlation and using this information to compute the expectation of the maximum in (8)), this would entail significant extra computational burden in the scheduler. Note that while we make this simplifying assumption in the design of our scheduler, in §5 we evaluate the scheduler performance in environments where the inter-arrival times may not be i.i.d. and demonstrate significant performance gains, which helps validate the design approach taken.

Our aim is to schedule packet transmissions (i.e. to select path-slot pairs (p_k, s_k), $k = 1, 2, \dots$) so as to minimise the mean per-packet in-order delivery delay

$$D := \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}[Y_k] - t_{p_k,s_k}. \tag{5}$$

Remark: Since the path delays are stochastic and can vary substantially a min-max approach (scheduling packets with the aim of minimising the maximum in-order delivery delay) with a hard maximum is generally not feasible: there will usually be some probability of exceedence of a given in-order delivery delay "maximum" and so some jitter. A probabilistic approach might be adopted, whereby we define the "maximum" as the delay which has some target probability of exceedence, e.g. 5%, and then try to minimise this "maximum". However, we argue that this type of approach is also not well suited to our purposes since (i) the solutions typically increase the average cost in order to improve the worst case and (ii) our context is essentially a best effort one since we have no control over the wireless link layer plus the augmented reality applications that we target are themselves best effort. We therefore adopt minimising the expected per-packet in-order delivery delay D as a reasonable objective for practical purposes.

2.2.1 Low-complexity Scheduling

Finding the optimal schedule that minimises eq. (5) is a complex combinatorial problem. Our goal is to design an effective scheme that solves the problem with low computational complexity such that scheduling choices are taken in a timely manner, i.e. faster than the lowest packet inter-arrival time in the sender's transport queue.

We first observe that if there is no reordering of packets sent within the same path then we can remove all choices where packets within the same path are *not* scheduled in order since these will have higher in-order delivery delay. This intuitive observation is formalised by Lemma 1 in the appendix. It follows that the optimal choice is for packets to be transmitted in ascending order on a path, in which case

$$Y_{p,k} = \max\{a_{p,s}, s = 1, 2, \dots, s_{p,k}\} \tag{6}$$

where $s_{p,k}{=}\max\{s_q\colon q{\in}K_{p,k}\}$. Observe now that $Y_{p,k}$ does not depend on the indices of the packets in $K_{p,k}$ but only on the slots these fly with.

Assume now that the delay of each path has an upper bound[6] $\bar{T}_p$ and so older slots $\{s : t_{p,s} + \bar{T}_p \le t_{p,s_{p,k}}\}$ do not affect $Y_{p,k}$. Then, it is sufficient to calculate the max over a finite window of slots,

$$Y_{p,k} = \max\{a_{p,s}, s = \underline{s}_{p,k}, \dots, s_{p,k}\} \tag{7}$$

with $\underline{s}_{p,k} := s_{p,k} - \delta_{p,k}$ and $\delta_{p,k} = s_{p,k} - \max\{s : t_{p,s} + \bar{T}_p \le t_{p,s_{p,k}}\}$. If we approximated the slot duration as being a constant T_p, then $\delta_{p,k} = \delta_p := \lceil \bar{T}_p / T_p \rceil$. That is, $\delta_{p,k}$ is constant and it is sufficient to calculate $Y_{p,k}$ over a fixed window. This assumption lets us contain the computational complexity of the scheduling algorithm and it is mild.[7]

Recalling random variables $\Delta_{p,\underline{s}_{p,k},\underline{s}_{p,k}+j} = a_{p,\underline{s}_{p,k}+j} - a_{p,\underline{s}_{p,k}}$, $j = 0, 1, \dots, \delta_p$, we can rewrite $Y_{p,k}$ as

$$Y_{p,k} = a_{p,\underline{s}_{p,k}} + \max\{\Delta_{p,\underline{s}_{p,k},\underline{s}_{p,k}+1}, \dots, \Delta_{p,\underline{s}_{p,k},\underline{s}_{p,k}+\delta_p}\}$$

and thus sequence

$$\boldsymbol{\Delta}(\underline{s}_{p,k}) := \{\Delta_{p,\underline{s}_{p,k},\underline{s}_{p,k}+1}, \dots, \Delta_{p,\underline{s}_{p,k},\underline{s}_{p,k}+\delta_p}\}$$

is i.i.d. ($\boldsymbol{\Delta}(\underline{s}_{p,k}) \sim \boldsymbol{\Delta}_p$). This leads to Theorem 1.

Theorem 1 (Invariance of $Y_{p,k} - a_{p,s_{p,k}-\delta_p}$). *The distribution of $Y_{p,k} - a_{p,s_{p,k}-\delta_p}$ is invariant on path p for packets k scheduled in slots $s_{p,k} > \delta_p = \lceil \bar{T}_p/T_p \rceil$.*

That is, we can let Z_p describe the distribution of $Y_{p,k} - a_{p,s_{p,k}-\delta_p} \sim Z_p \ \forall k$ such that $s_{p,k} > \delta_p = \lceil \bar{T}_p/T_p \rceil$. Theorem 1 helps us to greatly reduce the complexity of our scheduler as we don't have to recompute Z_p for every packet schedule. Moreover, note that the distribution of $a_{p,s_{p,k}}$ and Z_p can usually be readily estimated in an online fashion. For example, the realisation of $a_{p,s_{p,k}-\delta_p}$ may already be known via ACK feedback from the receiver. Otherwise, we can use past observations of packet delivery times on path p to estimate the distribution of $a_{p,s_{p,k}-\delta_p}$. We can also use past observations to estimate the distribution of $\boldsymbol{\Delta}_p$. To reduce the number of observations required to estimate the distribution and to reduce computational/memory cost, we can also use a parametric model and estimate the parameters using past observations, e.g. when using a Gaussian model we can find a reasonably accurate approximation of Z_p using the approach in [36]. We describe our actual system design in §4.

2.2.2 Stochastic Earliest Delivery Path First

Using Theorem 1, we can rewrite eq. (5) as

$$D = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}[\max\{Z_1 + a_{1,s_{1,k}-\delta_1} - t_{p_k,s_k}, \dots, Z_P + a_{P,s_{P,k}-\delta_P} - t_{p_k,s_k}, D_{p_k,s_k}\}]$$

where $D_{p,s} := a_{p,s} - t_{p,s}$. Then, minimising

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}[\max\{Z_1 + a_{1,s_{1,k}-\delta_1}, \dots, Z_P + a_{P,s_{P,k}-\delta_P}, a_{p_k,s_k}\}] \tag{8}$$

minimises D. We thus propose a lightweight algorithm to address the minimisation of (8) whereby packets are scheduled in the first available slot of the path with lowest expected delivery time, i.e.,

$$p_k = \arg\min_{q\in\mathcal{P}} \mathbb{E}[\max\{Z_1 + a_{1,s_{1,k}-\delta_1}, \dots, Z_P + a_{P,s_{P,k}-\delta_P}, a_{q,s_k}\}], \quad \forall k \tag{9}$$

The algorithm described by (9) contrasts with EDPF which merely schedules packets on the path that minimises the expected arrival time, i.e. $p_k = \arg\min_{q\in\mathcal{P}} \mathbb{E}[a_{q,s_k}], \forall k$, and thus it *does not account for the expected reordering delay in the buffer due to uncertainty in the path delays.*

Observe that (9) requires taking the max over a set of random variables and then its expectation. This is generally

6. We can easily keep track of the distribution of RTTs with feedback and thus compute a reasonable upper bound.

7. Using a Chernoff bound it can be shown that the probability that $Y_{p,k}$ depends on events occurring prior to the fixed window is small for an appropriate choice of window size.

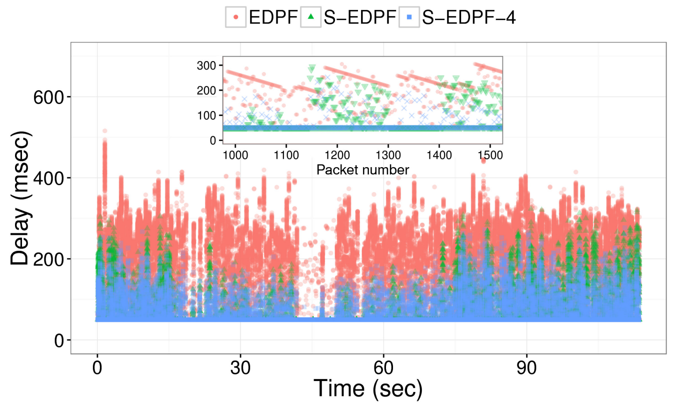
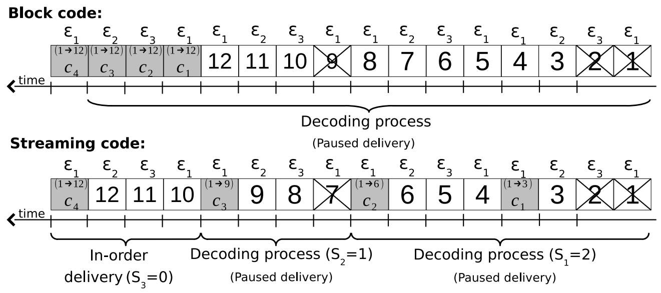


Fig. 3. Simulation results with two paths, equal mean delay (50 ms) and rate (10 Mb/s). No variance on one path; high delay variance on the other. Delay performance of `openface` traces used in Fig. 1.



Block code:
time
Decoding process
(Paused delivery)
Streaming code:
In-order delivery (S_3=0)
Decoding process (S_2=1)
Decoding process (S_1=2)

Fig. 4. Block (top) and streaming (bottom) code with rate $\frac{\tau-1}{\tau} = 3/4$. $(a \rightarrow b)$ indicates range of data packet indices c_k encodes. Packets are received from 3 paths with erasure probability ϵ_1, ϵ_2, and ϵ_3.

complex but if we use a Gaussian model we can use the low-complex algorithm proposed in [36]. Alternatively, if we model delay as being discrete valued and we have estimates of the probability of each value, then the expectation of the max in (9) can be calculated directly provided the number of values considered is not too large.

The intuition behind the algorithm described by (9) is the following. $Y_{k-1} = \max\{Z_1 + a_{1,s_{1,k}-\delta_1}, \ldots, Z_P + a_{P,s_{P,k}-\delta_P}\}$ essentially models the distribution of the time packet $k-1$ is delivered (after reordering). As we explain above, this can be nicely kept track of with low complexity by estimating the distribution of Z_p at reasonable time-scales, e.g. only after observing drastic network changes (see Th. 1). The arrival time (before reordering) of the first packet within the chosen window of slots of each path $(a_{1,s_{1,k}-\delta_1}, \cdots, a_{P,s_{P,k}-\delta_P})$ can be easily learnt from feedback. Since k will be delivered at the application at time $Y_k = \max\{Y_{k-1}, a_{q,s_k}\}$ if path q is used, (9) assigns packet k to the path q with shortest expected delivery delay taking into account the expected penalty (time spent in the receiver reordering buffer) caused by delay variability *with low complexity*. This is similar to a two-stage stochastic program where a scheduling choice must be taken before instantaneous delays are known (first stage) and a recourse action, reordering, is used to compensate for unfortunate choices when packets arrive (second stage) [38].

Let us illustrate the quantitative and qualitative gains of path selection according to (9) via simulations. Fig. 3 presents simulation results using the `openface` traces presented in §1 for a setup with two paths having equal average propagation delay (50 ms), fixed rate (10 Mb/s) and no loss. In this example, the delay of one of the paths is random and follows a normal distribution with 100 ms standard deviation whereas the other link has a constant delay (no variance). It can be seen that scheduling packets based on averages only (EDPF) often results in high in-order delivery delays at the receiver because of frequent head-of-line blocking (see also the inset in Fig. 3, where we show detail of the delay of 500 consecutive packets). In particular the mean delay is 201 ms and its std. deviation (a measure of jitter) is 80 ms. In contrast, with the S-EDPF scheduler (full details in §4) the delay is substantially lower, having a mean of 61 ms and std. deviation 33 ms. It can also be seen that use of FEC to add redundancy, labelled S-EDPF-4 in the figure and explained in detail in the next section, tends to further mitigate the impact of out of order arrivals.

3 Low Delay Streaming Code

In the event of a loss, which may be frequent in paths containing wireless links or congested queues, the usual recovery method, ARQ, incurs extra delay because of the round-trip time that it takes for the repeat request to be delivered to the sender and the packet to be retransmitted to the receiver. Our approach is to precode enough information in redundant packets so that with high probability loss recovery can take place without the need to wait for retransmissions.

As already noted, block codes (used e.g. in [6], [11], [22]–[24], [26]–[30]) introduce a new source of head-of-line blocking. This is unimportant when the aim is to maximise throughput, but is problematic when the aim is to minimise per-packet in-order delivery delay. Namely, consider a rate k/n systematic block code where k data packets are transmitted followed by $n-k$ coded packets. Suppose that the i'th information packet is lost. Then recovery cannot be attempted until a coded packet is received, e.g. when the first packet is lost then the receiver must wait at least k slots until the first coded packet is received. From the per-packet delay perspective, it is generally better to interleave coded packets within the data stream such that the decoder can release data from the buffer to the receiver application more quickly. Using this type of approach, for a given coding rate we can always achieve lower mean per-packet in-order delivery delay than when using a block code, see [39]. Moreover, since a non-systematic code will always have higher mean in-order delivery delay than a systematic code, we consider only systematic codes. We thus adapt the single-path streaming code from [39] to the multipath setting, analyse its delay performance and, importantly, establish the optimal schedule (path and slot) of coded packets. To our knowledge, this is one of the first analytic results for scheduling of FEC packets in a multipath setting.

The code construction works as follows. We divide time into slots $\mathcal{S} = \{1, 2, \ldots\}$ each corresponding to the transmission of one packet. S-EDPF generates a *coded packet* c_k and schedules it every τ slots (which we refer to as the *coding interval*). A coded packet c_k is a random linear combination of information packets 1 to $N_\tau k$, where $N_\tau = \tau - 1$

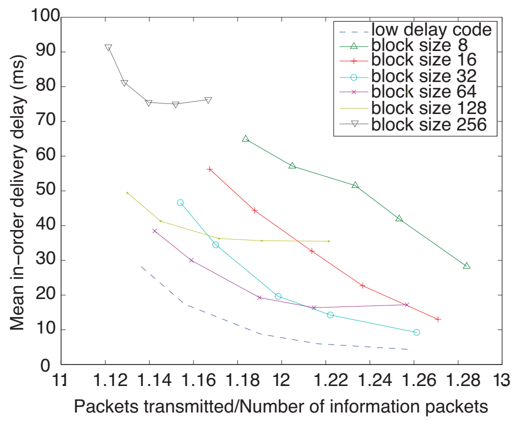


Fig. 5. Mean per-packet in-order delivery delay of our streaming code and block codes for different coding rates. A single link with 25Mb/s rate, RTT of 60ms, and loss rate of 10%. Reproduced from [39].

is the number of uncoded packets u_i sent between c_k and c_{k-1} across all paths, i.e.,

$$c_k = f_\tau(u_1, u_2, \dots, u_{N_\tau \cdot k}) := \sum_{j=1}^{N_\tau \cdot k} w_{kj} \cdot u_j \qquad (10)$$

with coefficients w_{kj}, selected identically and independently, uniformly at random from a finite field of size Q. At the receiver, upon reception of c_k, a matrix G_k is constructed with rows formed from the coefficients of the received packets (normal uncoded packets adding a row with a 1 in the corresponding diagonal, all other entries being 0). Decoding can then be carried out on-the-fly using Gaussian elimination (i.e. maximum likelihood decoding). In our analysis we will make the standing assumption that the field size Q is sufficiently large that with probability one each coded packet helps the receiver recover from one information packet erasure. That is, each coded packet row added to generator matrix G_k increases the rank of G_k by one. Note however that we evaluate non-ideal decoding (with a non-zero decoding failure probability) in §5. Also, we can (and do) adjust the starting index for j in (10) so that it tracks the unacknowledged information packet with oldest/smallest index and in this way manage the encoding/decoding complexity.

This streaming technique aims to quickly unblock the receiver's buffer after loss so to reduce in-order delivery delay. In contrast, block codes only allow loss recovery after receiving a full block of packets which, as already noted, causes a type of HOL (see Fig. 4). Other approaches, such as PluriBus [11], only send coded packets when transmission queues are idle, and therefore suffer from a similar issue. For comparison purposes, we show in Fig. 5 the delay performance of our streaming code and conventional systematic random linear block codes for different coding rates and block sizes. As expected, based on [39], the in-order delivery delay performance of the block code constructions is a function of the coding rate and block size. On the other hand, the streaming code achieves better performance over the entire range of coding rates, i.e. for the same throughput, the delay of the streaming code is always smaller. Similar trends can be observed for other packet loss rates.

3.1 Buffering Delay Analysis

In this section we characterise the in-order delivery delay performance of the low-delay code construction in a multipath setting. Remarkably, we are able to obtain accurate closed-form expressions for the in-order delivery delay and it is these which enable us to derive an optimal scheduling approach for the coded packets.

Our setup is that received data packets are delivered to the application in order as they arrive (without buffering delay), until there is a loss. When there is a loss, a "decoding process" starts: packets are buffered until the decoder has enough coded packets to reconstruct the missing packets, at which point in-order delivery resumes. We index each "in-order delivery"/"decoding process" period with $j = 1, 2, \dots$, and let the random variable S_j count the coded packets required in j to resume an in-order delivery state ($S_j = 0$ if stage j is already in the in-order delivery state). This is illustrated in Fig. 4 (bottom figure) for $\tau = 4$. This sequence of periods forms a renewal process with $S_j \sim S$. The next theorem characterises the distribution of S.

Theorem 2 (S process). *Suppose we transmit information packets across $\mathcal{P} = \{1, \dots, P\}$ independent paths with erasure probability $\{\epsilon_1, \dots, \epsilon_P\}$ and fixed delays*[8]*. Suppose we have $N_{\tau,p}$ slots in path p in each interval where we transport information packets such that they arrive in order, except the last slot in $p_c \in \mathcal{P}$ (with erasure probability ϵ_{p_c}) where we transport a coded packet; then $\sum_{p\in\mathcal{P}} N_{\tau,p} = \tau$. Assume that each coded packet can help us to recover from one erasure, irrespective of the path where the erasure has occurred. Then, if we let $\bar{\epsilon} = \frac{\sum_{p\in\mathcal{P}} N_{\tau,p}\epsilon_p}{\sum_{p\in\mathcal{P}} N_{\tau,p}}$, we have:*

1. *For all ϵ_p and $N_{\tau,p}$ s.t. $\tau\bar{\epsilon}<1$, the mean of the S probability distribution exists and is finite.*
2. *The distribution of S is characterised by:*

$$P(S{=}0) = (1{-}\epsilon_{p_c})^{N_{\tau,p_c}-1} \prod_{i\neq p_c} (1{-}\epsilon_i)^{N_{\tau,i}} \qquad (11)$$

$$P(S{=}1) = (N_{\tau,p_c}{-}1)\epsilon_{p_c}(1{-}\epsilon_{p_c})^{N_{\tau,p_c}-1} \prod_{j\neq p_c} (1{-}\epsilon_j)^{N_{\tau,j}} + \\ + \sum_{i\neq p_c} N_{\tau,i}\epsilon_i(1-\epsilon_i)^{N_{\tau,i}-1} \prod_{j\neq i}(1-\epsilon_j)^{N_{\tau,j}} \qquad (12)$$

$$P(S{=}k) \approx \frac{\tau - 1}{k}\bar{\epsilon}^k(1{-}\bar{\epsilon})^{k(\tau-1)}\binom{(k-1)\tau}{k-1}, \ \forall k>1 \qquad (13)$$

3. *The first and second moments of S can be approximated by the following closed-form expressions:*

$$\mathbb{E}[S] \approx P(S{=}1) + \frac{\tau(\tau-1)\bar{\epsilon}^2(1{-}\bar{\epsilon})^{\tau-1}}{1{-}\tau\bar{\epsilon}}$$

$$\mathbb{E}[S^2] \approx P(S{=}1) + (1{-}\bar{\epsilon}{+}(1{-}\tau\bar{\epsilon})^2)\frac{\tau(\tau-1)\bar{\epsilon}^2(1{-}\bar{\epsilon})^{\tau-1}}{(1{-}\tau\bar{\epsilon})^3}$$

Proof. See appendix. □

Observe that Theorem 2 requires $\tau\bar{\epsilon} < 1$ for $\mathbb{E}[S]$ (and so in-order delivery delay) to be finite, i.e. $\sum_{p\in\mathcal{P}} N_{\tau,p}\epsilon_p < 1$.

8. Fig. 3 shows simulation results in the presence of variable path delays and we further assess performance in the presence of random delays experimentally in §5. However, we leave the mathematical analysis with variable path delays for future work.

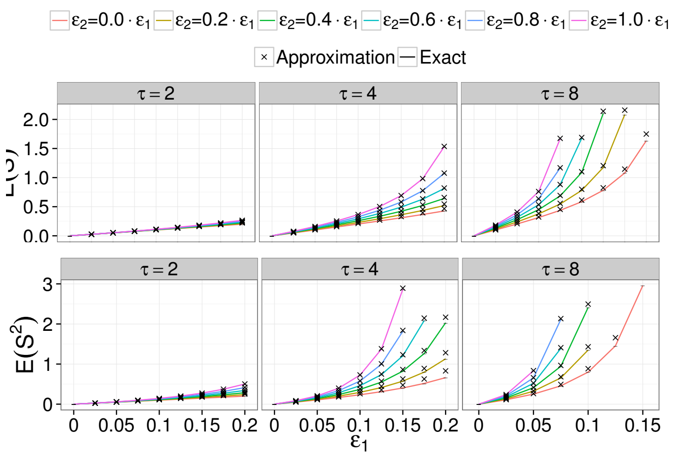


Fig. 6. $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ for different coding intervals τ and packet loss probabilities. Exact values *vs.* approximated values from Theorem 2.

The aggregate rate R at which coded packets are sent across all paths is $1/\tau$, and for $\bar{\epsilon}\tau < 1$ we require $R > \bar{\epsilon}$, where $\bar{\epsilon}$ is the aggregate loss rate across the paths, as expected.

In this theorem we approximate *the tail* of the distribution of S (see (13)). In more detail, we approximate a poisson-binomial distribution with a binomial distribution (see the proof of Theorem 2.2) which is more tractable. This provides a good approximation of the complete distribution of S because we only apply it to its tail (i.e. in $P(S = k)$, $\forall k > 1$), which does not contribute much to the whole distribution, i.e., $\sum_{k=2}^{\infty} P(S = k) \ll \sum_{k=0}^{1} P(S = k)$, in most of the cases of interest.

To illustrate the accuracy of this approximation, we set up two paths of equal rate and packet loss rates ϵ_1 and ϵ_2, respectively, and compute the 1st and 2nd moments for different coding intervals with (i) the expressions in Theorem 2, and (ii) the exact solution using [40] to compute the poisson binomial. The coded packets are sent on the path with lowest loss rate[9]. Fig. 6 shows $\mathbb{E}[S]$ and $\mathbb{E}[S^2]$ as a function of ϵ_1 and it can be seen that the expressions in Theorem 2 are particularly accurate for large coding intervals and small $|\epsilon_1 - \epsilon_2|$, but are also good in the extreme cases, e.g. with highly heterogeneous links, the reason being that we only approximate the tail of the distribution of S.

We now can use Theorem 2 to characterise the in-order delivery delay when scheduling coded packets across multiple paths. Let us define a *frame* as the transmission of $\tau - 1$ information packets plus a coded packet. The scheduler in each frame assigns $N_{\tau,p}$ packets to each path p such that $\tau - 1 = \sum_{p\in\mathcal{P}} N_{\tau,p}$. The relationship between the S process and the buffering delay due to losses is as follows.

Theorem 3 (Buffering delay)**.** *At the receiver, the asymptotic mean buffering delay per transmitted packet is upper bounded by*

$$\frac{\mathbb{E}[S^2]}{2\tau(\mathbb{E}[S] + P(S = 0))} \sum_{p\in\mathcal{P}} \max\{N_{\tau,p}(N_{\tau,p} - 1), 1\}\,\Delta_p$$

Proof. See appendix. □

To illustrate the above, consider a scenario with 2 paths of equal bit rate and packet loss rate ϵ_1 and $\epsilon_2 =$

9. As we will see later on, this is actually the optimum choice.

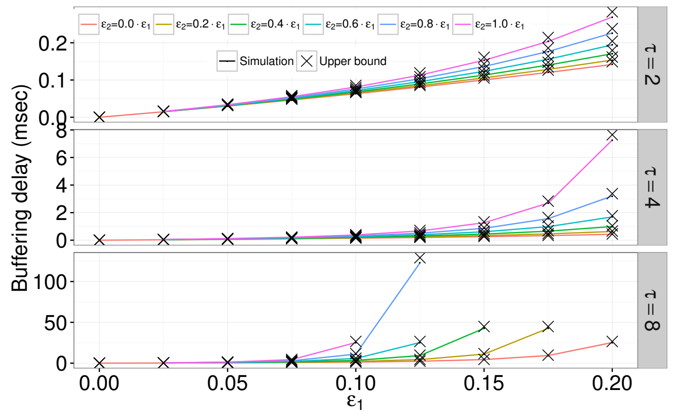


Fig. 7. Buffering delay. Simulation results *vs.* theoretical upper bound (Theorem 3) for two paths with a range of path loss rates ϵ_1, ϵ_2.

$\{0, 0.2\epsilon_1, 0.4\epsilon_1, 0.6\epsilon_1, 0.8\epsilon_1, \epsilon_1\}$, respectively. First, we compute the bound on packet buffering delay given by Theorem 3 for different coding intervals; second, we simulate these scenarios with an event-driven simulator and measure the mean buffering delay; and finally we compare both results in Fig. 7. From the figure[10], we conclude that Theorem 3 predicts buffering delay rather accurately, which also helps to further validate the approximations used in Theorem 2. Observe also from Fig. 7 that the buffering delay decreases rapidly with decreasing τ i.e. as we send more coded packets. This behaviour can also be seen in Fig. 6, where the mean time $\mathbb{E}[S]$ between in-order packet delivery events decreases as τ decreases.

3.2 Path selection for coded packets

The next result tells us how best to exploit multiple paths when transmitting coded packets constructed using our low-delay coding scheme.

Theorem 4 (Path choice for coded packets)**.** *Under the conditions of Th. 2, the buffering delay is minimised when coded packets are sent in the path with highest erasure probability.*

Proof. See appendix. □

The importance of Theorem 4 is not only that it tells us how to optimally schedule coded packets in a multipath setting, and that this optimal schedule is simple to implement, but also that it says we can exploit lower-quality links to improve delay performance. This contrasts with previous approaches where multipath transport need not provide any gain in performance (even a loss in some cases) over its single-path counterpart when paths have different packet loss patterns and path delay [13].

3.3 FEC-aware Packet Selector Algorithm

Based on our analysis above, we propose in Algorithm 1 a simple, yet very effective, low-delay FEC-aware scheduling algorithm. The algorithm builds a coded packet using eq. (10) every time τ information packets are scheduled. Parameter τ determines the trade-off between throughput

10. Note that some points are not depicted in Fig. 7 because delay goes to infinity when $\tau\bar{\epsilon} > 1$.

Algorithm 1 S-EDPF streaming FEC

```
1:  function S-EDPF_GET_PACKET(p, $\tau$)
2:      if backoff_ctr== 0 then                         ▷ Queue coded pkt
3:          c_k ← generate_codedpkt()                   ▷ Eq. (10)
4:          enqueue_pkt(coding_q, c_k)
5:          backoff_ctr← $\tau$                          ▷ Reset backoff counter
6:      k←NULL
7:      if lossy_path==p & size(coding_q)> 0 then
8:          k ← dequeue_pkt(coding_q)
9:      if k==NULL then
10:         k ← dequeue_pkt(data_q)
11:     if k==NULL & size (unacked)> 0 then
12:         k ← generate_codedpkt()
13:     backoff_ctr← backoff_ctr−1
14:     return k
```

Algorithm 2 S-EDPF scheduling algorithm

```
1:  function S-EDPF
2:      if Ztimer then
3:          for $p \in \mathcal{P}$ do
4:              $Z[p]$ ←compute_max($\vec{a}$) − $a_{p,s_{p,k}-\delta_p}$
5:      /*Loop until there are no empty queues*/
6:      while empty_qs() $\neq \varnothing$ do
7:          /*1. Get best path as shown in §2 (eq. 9)*/
8:          p ← s-edpf_get_path($\vec{Z}$)
9:          /*2. Get a pkt to schedule as shown in §3*/
10:         k ← s-edpf_get_packet(p)
11:         /*3. Do actual schedule*/
12:         enqueue_pkt(p->sched_q, k)
```

and delay (the capacity used for redundancy being proportional to $1/\tau$, delay given by Theorem 3) that depends on the application constraints and network capacity. These coded packets are enqueued in `coding_q` until they can be assigned to the lossiest path.[11] To maximise resiliency, an additional coded packet is scheduled (regardless of the path) whenever there is no data ready to send yet there is unacknowledged data awaiting feedback of reception.[12]

4 S-EDPF System

4.1 Overall S-EDPF algorithm

Pseudocode for the main logic of the overall S-EDPF system is given in Algorithm 2. A scheduling job is triggered every time there is an empty transmission queue (`p->sched_q` for path p). This scheduling job, which is repeated until all transmission queues have at least one packet, consists of three tasks that are performed sequentially, namely (i) S-EDPF selects a path p using (9), (ii) S-EDPF selects a packet k using Algorithm 1, (iii) S-EDPF schedules packet k in path p. This algorithm ensures a packet is always available to be sent in all transmission queues unless all data sent has been acknowledged. A rate/congestion control algorithm then takes care of granting transmission opportunities to each of these queues (i.e. we do not rely on any specific rate control scheme). A timer (`Ztimer`) triggers re-computation of $\vec{Z} = \{Z_1, \cdots, Z_P\}$. Note that $\vec{Z}$ is recomputed only when the distribution of rates and delays changes. To do so, we need to compute $Z_p = \max(a_{p,k-\delta_p}, \cdots, a_{p,k}) - a_{p,s_{p,k}-\delta_p}$ and use the approach discussed in §2.

11. Note that, if this path's bitrate is overly low, `coding_q` could saturate. In this case, packets from `coding_q` can be dequeued to the next lossiest path. This is not shown in Algorithm 1 due to space constraints.

12. We explain our feedback mechanism in detail in §4.

4.2 Prototype Implementation

We used the framework of Coded TCP (CTCP) [41] to implement S-EDPF. CTCP is composed of a pair of SOCKS5 proxies that route data from/to TCP applications into/from (multiple) UDP standard sockets. This gives us the ability to easily implement and evaluate congestion/rate control, coding/decoding and scheduling algorithms *in userspace*. This not only facilitates research and development but it also maximises its deployability: S-EDPF/CTCP runs in Linux, OS X, Android and *BSD without root privileges. A schematic overview of our prototype is shown in Fig. 8.

The encoder/decoder is implemented using *finite field arithmetic* in $GF(256)$ which lets us encode and decode information efficiently, using XORs for addition/subtraction and quick table lookups using SIMD programming for multiplication, without compromising performance. We use 2 bytes of each header to transport the seed used to generate the random coefficients (these are needed for decoding at the receiver), and 2 bytes to indicate to the decoder which packets are encoded together. Note that this overhead is the same as that for a random linear block code. Decoding is based on on-the-fly Gaussian elimination, and decoded packets are released to the receiver as soon as they can be handed off in order.

Received packets trigger the transmission of cumulative selective ACKs that, similarly to MPLOT [27], provide the transmitter with information regarding delivery delays (used by the packet scheduler), packet loss rate (used by the coding scheme), which packets have been lost (used for retransmitting packets when decoding fails), and congestion information (used for rate control if needed).

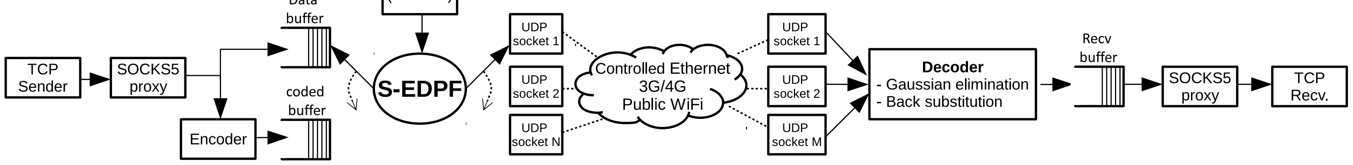


Fig. 8. Prototype architecture.

S-EDPF does not do any rate control. Its role is to select which packet is sent down which path when a transmission opportunity is granted. Unless otherwise stated, we use CTCP's default rate control algorithm, which is particularly suitable for the sort of networks we target (lossy/variable) and which is fair to other TCP flows (see [42]). Note however that S-EDPF is compatible with *any* rate controller.

To provide a baseline for evaluation, we also implemented ARQ-based EDPF (optimal when path delays are constant and there is no loss, used in most previous work) and an ARQ-based LowRTT scheduler (used in MTCP).

5 Performance Evaluation

We next summarise our experimental evaluation. First, we validate our prototype by comparing its performance with EDPF/ARQ and the default Linux MPTCP implementation; secondly, we run a set of experiments in controlled environments where a range of network conditions are emulated; and finally, we run an experimental campaign "in the wild". The environmental conditions (delay, loss patterns, bandwidth) used in our controlled experiments are similar to those measured in [9]–[11]. For convenience, in most of these tests we install a *local proxy* in a laptop[13] with Ethernet, WiFi and 3G/4G network interfaces, and a *remote proxy* in a campus server that acts as a gateway to Internet. We keep the local and remote proxies synchronized with PTP (controlled environments) or NTP to measure delays. S-EDPF/CTCP operates between the local and remote proxy. Note that S-EDPF has no impact on other competing flows (we rely on external rate control to guarantee fairness), and thus we do not explore this in our evaluation. Finally, we show results for $\tau = \{4, 8, 16, 32\}$ in most of our evaluations to span a reasonable range of τ values[14].

5.1 Complexity

We have profiled our prototype with `(v/c)allgrind` and a `Monsoon` power meter which shows that the encoder/decoder is the most costly task (as expected because it uses CPU-intensive operations). We installed our prototype in an Android LG G5 smartphone and measured its power consumption in a series of experiments. To simplify the setup, we connect it to a local WiFi AP (only one path) with fixed rate (54 Mb/s), fixed CPU frequency, and backlight turned off. We stream backlogged TCP data to the AP for 30 s, while randomly dropping packets. We repeat each experiment 5 times and collect the mean throughput and power for S-EDPF and S-EDPF without FEC (only relying on ARQ) and Linux TCP, see Fig. 9. With S-EDPF, we send as many coded packets as needed to compensate for losses. Note that, when there are no losses, the throughput provided by both S-EDPF and TCP is the same, though at an extra energy cost of 5% (barely 1% with the backlight on). When there are losses (not due to congestion), CTCP's default congestion control achieves higher throughput than the TCP congestion control and TCP therefore uses less energy simply because its send rate is lower.

13. We have also tested it in Android and MAC OS X.
14. We observe diminishing sensitivity to changes in τ as τ gets larger because the coding rate is $\frac{\tau-1}{\tau}$ and so the marginal change in coding rate decreases as τ increases.

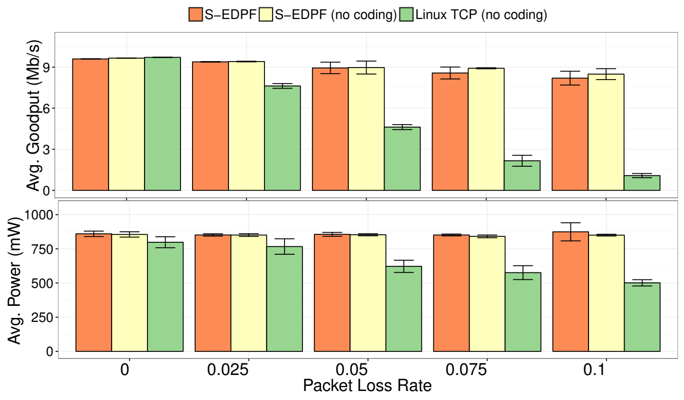
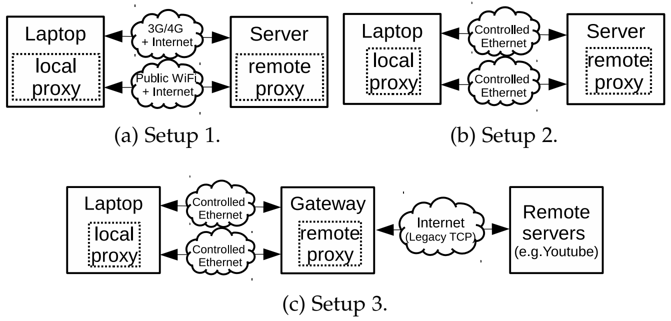


Fig. 9. Energy usage of S-EDPF *vs.* legacy TCP.

5.2 Comparison with MPTCP

Fig. 11 shows a comparison between MPTCP, EDPF and S-EDPF-τ (τ denotes the coding interval used). The upper plot shows the mean goodput with error bars indicated. The lower plot shows a box and whiskers plot of the distribution of the measured delay: the bottom and top of the box indicate the first and third quartiles, the band inside the box is the median, and the ends of the whiskers represent the lowest sample within 1.5 IQR of the lower quartile, and the highest sample within 1.5 IQR of the upper quartile. We will use this type of representation for most of our experimental results hereafter. This experiment was carried out in a home environment, streaming backlogged data for 30 s from a laptop attached to a 2.4-Ghz WiFi Access Point and a Meteor 3G/4G dongle to our remote server using `mgen` (See Fig. 10a). For "Default MPTCP", we used the Linux implementation of MPTCP v0.89 with OLIA [7] for congestion control and LowRTT as scheduler. We repeat the experiment 5 times for every scheme. The first observation from Fig. 11 is that MPTCP shows the worst performance in terms of both goodput and delay. This may be explained by MPTCP's use of a different congestion control scheme (less friendly to wireless losses as shown in [42]), a packet scheduler (LowRTT) that does not account for delay variability (neither does EDPF), and the lack of FEC for packet loss control. The second observation is that S-EDPF both increases goodput and decreases delay, providing the ability to trade goodput for lower delay by adjusting the coding rate (it can be seen in Fig. 11 that decreasing the coding interval τ in S-EDPF-τ reduces delay at the cost of reduced goodput, as expected from the analysis in §3.1).



(a) Setup 1. (b) Setup 2. (c) Setup 3.

Fig. 10. Experimental setups.

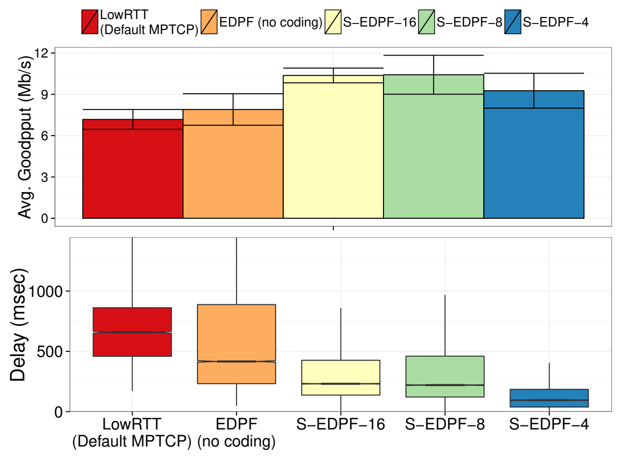


Fig. 11. Uploading over WiFi+LTE in a home scenario.

5.3 Controlled Environments

For these experiments, we use `tc/netem` to generate non-correlated normally distributed delays in a controlled Ethernet network between laptop and server (Fig. 10b), and `dummynet` to emulate link bitrates and drop packets randomly when required. To avoid effects associated with congestion control, in this subsection we fix the packet rate on each subflow to match the bandwidth-delay product of each path (i.e. the highest rate without causing congestion). Unless otherwise stated, we stream backlogged data using `mgen` for 30 s from a laptop attached to two emulated networks, and repeat each experiment 5 times per mechanism. In the sequel, we first evaluate one variable at a time (randomness, then losses) and then we emulate live streaming of video with a real-life service.

5.3.1 Random propagation delays

We set up two links, each of 10 Mb/s, connected to a router. The router introduces 50 ms of fixed propagation delay in link 2 and a random propagation delay with mean 50 ms and variable standard deviation in link 1. We do not drop any packets in these experiments. We compare MPTCP's LowRTT scheduling scheme, EDPF and S-EDPF-τ (i.e. S-EDPF with coding interval τ). Results are shown in Fig. 12. The top plot shows the average goodput performance and the bottom plot the measured buffering delay at the receiver (i.e. not including the path propagation delay). Note that we make full use of the aggregate capacity of the network, i.e., the receiver always receives 20Mb/s of raw traffic. It can be seen that our FEC mechanism uses part of this capacity to send redundant information; for instance, S-EDPF-4 trades 5 Mb/s to reduce delay by half (roughly) when the standard deviation of link 1's propagation delay is 100ms. Observe that FEC helps delay even when there is no loss. The reason is that, depending on the behaviour of the paths, coded packets might arrive sooner than preceding data packets and thus help the receiver to decode still-on-the-air packets sooner than would be the case when without FEC. We have also evaluated the performance for a range of path delays and find that S-EDPF offers similar gains in performance (not shown here due to space constraints). Note also that when τ is larger, the coding rate $\frac{\tau-1}{\tau}$ decays and so it does the protection to reordering.

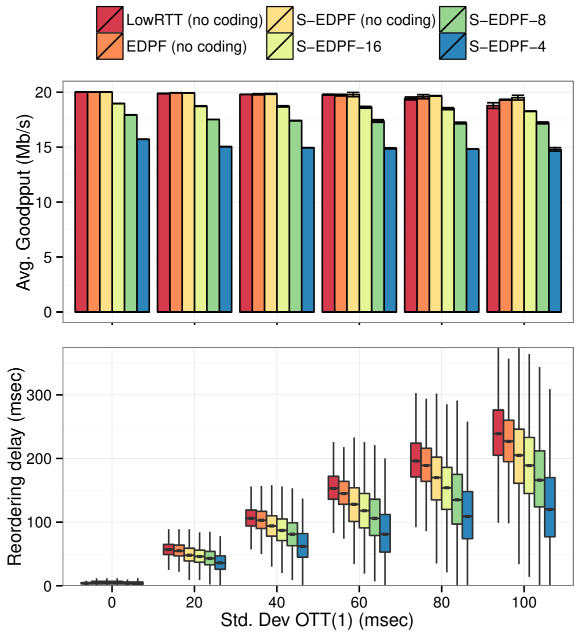


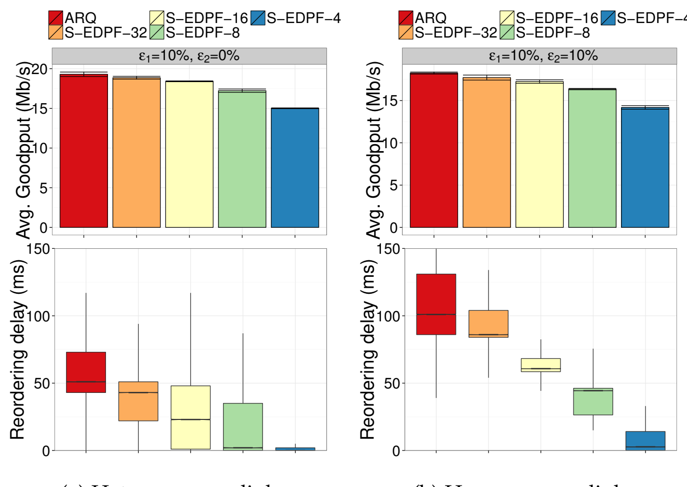
Fig. 12. Buffering delay and goodput. Paths with random delay, no loss.

5.3.2 Lossy Paths

We now assess S-EPDF with lossy paths. To this end we fix the propagation delay of two links to be 50 ms, the rate to be 10 Mb/s and randomly drop 10% of packets in link 1. In link 2, we do not drop packets in Fig. 13a, and drop 10% of packets in Fig. 13b. Note that we only drop data packets and not ACKs, to benchmark against ideal ARQ. In the figures we evaluate S-EDPF for a range of coding intervals and compare against an EDPF scheduler that relies only upon retransmissions (ARQ) to recover from losses (as does MPTCP). It can be seen that use of FEC practically eliminates buffering delay with the most aggressive configuration (S-EDPF-4) at the cost of 15% of throughput. When τ is larger, the coding rate $\frac{\tau-1}{\tau}$ decays and so it does the protection to losses but the throughput loss moderates. It is worth mentioning that higher RTTs yield larger performance gains (not shown here for space reasons) because ARQ only retransmits packets upon the reception of feedback and this takes longer the higher the propagation delay.

5.3.3 Real-time Streaming

In the following, we evaluate the performance of EDPF and S-EDPF for a live streaming application. The setup is shown in Fig. 10c. A `vlc` video player located in a laptop streams a video from a `Youtube` server through the Internet. A remote proxy acts as the Internet gateway and relays `Youtube` data to the local proxy across two independent paths where we control the packet loss and delay. In order to emulate real-time (live) streaming, we configure `vlc` with 200 ms of caching and decoded pictures experiencing a delay greater than 150 ms are discarded. This lets us test a live streaming application while having repeatability in our experiments. We selected an HD video (with resolution 1280x534), *Star Wars VII teaser trailer 2* (https://www.youtube.com/watch?v=wCc2v7izk8w accessed on 01/05/2016) encoded with H264/MPEG-4 AVC,



(a) Heterogeneous links. (b) Homogeneous links.

Fig. 13. Buffering delay and goodput over lossy links.

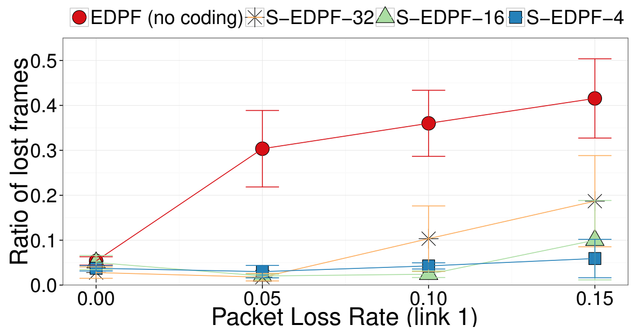


Fig. 14. Live video streaming.

with duration of 1:49 mins, and a rate of 23.97 fps. We configure the two controlled links with 10 Mb/s, 100 ms of RTT, and a variable random packet loss rate (for both data and ACKs) on one of the links; we do not drop any on the other link. Fig. 14 shows the ratio of video frames that have been discarded (e.g. due to excess buffering delays), over 5 runs, when using EDPF (using ARQ for loss recovery) and S-EDPF-τ for different coding intervals τ. It can be seen that S-EDPF achieves a dramatic improvement in the streaming experience. In particular, when the access link experiences losses the video delivered using EDPF stutters and skips significant sections whereas using S-EDPF essentially no video frames are skipped and playout is consistently smooth.

5.4 "Into The Wild"

Finally, we evaluate the performance of S-EDPF over production networks. We use a laptop attached to a 3G/4G dongle and connect to a range of public WiFi hotspots around Dublin, Ireland (Fig 10a). Our experimental campaign covers measurements in the campus of Trinity College Dublin, a home environment surrounded by many apartments, a public pub during busy hours, St. Stephen's Green Mall in Dublin on a Saturday evening, and Dublin airport at peak hour. At each location, we stream backlogged data for 30 s from our server and repeat the measurement 5 times for each configuration. We remark that these scenarios generally offered unreliable wireless links; we have observed delay deviations of several milliseconds and bursts of packet losses of a few hundred packets in many of our experiments. Fig. 15 shows the measured goodput in the top plot, the distribution of packet end-to-end delay in the lower plot.

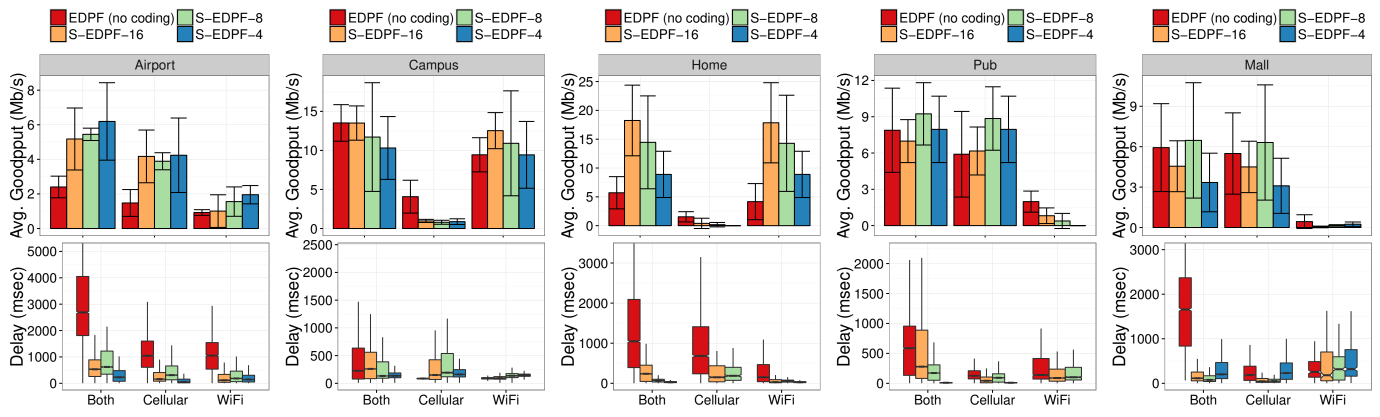
First consider the "Airport" scenario. When using the individual links (labeled "Cellular" and "WiFi") we observe very poor performance when using only ARQ (labelled "EDPF (no coding)"). Adding low-delay FEC (labelled "S-EDPF-4" etc) alone improves delay by an order of magnitude when using a single path (we observed similar gains in our controlled environments). When bonding both links together, the delay performance of EDPF worsens even further because of the high level of packet reordering induced across paths. S-EDPF, on the other hand, continues to achieve much lower delay when both links are bonded due to its superior scheduling decisions in the face of uncertainty and time-variations in the path delays. In this case, with S-EDPF-4 we can increase goodput by 50% with respect to using the cellular link only and triple it with respect to using WiFi link only at a negligible cost in delay .

If we consider the "Campus" and "Home" scenarios now, observe that the goodput on the WiFi link is substantially higher than on the cellular link. In this case we can expect there to be little benefit from bonding both links, and indeed we see this in the measurements. When bonding these links and using S-EDPF the goodput and delay are similar to those when using the WiFi link alone. Conversely, in the "Pub" and "Mall" scenarios the WiFi link is poor and the cellular link has higher goodput. When bonding these links and using S-EDPF the goodput and delay are similar to those when using the cellular link alone.

These results demonstrate that multipath S-EDPF capitalises on both paths when there is a benefit to be gained (as in the 'Airport" scenario) and otherwise provides goodput and delay performance similar to that of the best single path (as in the "Home" scenario). Importantly, observe that when using EDPF the multipath delay performance is consistently poorer than with S-EDPF, often by an order of magnitude.

6 Summary & Conclusions

Building an efficient, low latency multipath transfer mechanism is highly challenging. The primary reason for this is that the transmission delay along each path is typically uncertain and time-varying. When the transmitter ignores the stochastic nature of the path delays, then packets sent along different paths frequently arrive out of order and need to be buffered at the receiver to allow in-order delivery to the application. In this paper we propose S-EDPF (Stochastic Earliest Delivery Path First), a generalization of EDPF which takes into account uncertainty and time-variation in path delays yet has low-complexity suited to practical implementation. We integrate FEC into S-EDPF in a principled manner by deriving the optimal schedule for coded packets across multiple paths. We demonstrate that S-EDPF is effective at mitigating the delay impact of reordering and loss in multipath transport protocols, offering substantial performance gains over the state of the art.



(a) Airport. (b) Campus. (c) Home. (d) Pub. (e) Mall.

Fig. 15. Experiments *into the wild*.

Appendix
Proofs

Lemma 1. *Supposing there is no reordering in packets sent within the same path, it is sufficient to consider situations where packets on the same path are sent in order to minimise delay.*

Proof. Let $\alpha_{p,1}, \alpha_{p,2}, \ldots, \alpha_{p,n}$ denote realisations of random variables $a_{p,1}, a_{p,2}, \ldots, a_{p,n}$ i.e. the arrival times of packets sent in slot n of path p. Suppose packets on the same path are sent in ascending index order, i.e. for any $k \in \{1, s, \ldots\}$ and $q_1, q_2 \in K_{p,k}$ such that $q_1 < q_2$ then $s_{p,q_1} < s_{p,q_2}$. Since we assume there is no reordering within a single path, it follows that $\alpha_{p,q_1} < \alpha_{p,q_2}$. Let $\psi_{p,k} := \max\{\alpha_{p,s_q} : q \in K_{p,k}\}$ and $\psi_k := \max\{\psi_{1,k}, \ldots, \psi_{P,k}, \alpha_{p_k,s_k}\}$. Then we have $\psi_{p,k} \le \psi_{p,k+1}$, $k \in \{1, 2, \ldots\}$, with equality only when $K_{p,k+1} = K_{p,k}$. To see this, note that when $K_{p,k+1} = K_{p,k}$ equality is trivial, and when $K_{p,k+1} \ne K_{p,k}$ then $q^*_{k+1} := \max\{q \in K_{p,k+1}\} > q^*_k := \max\{q \in K_{p,k}\}$ due to the ascending order and so $\alpha_{p,q^*_{k+1}} > \alpha_{p,q^*_k}$ and thus $\psi_{p,k+1} > \psi_{p,k}$. Hence, $\psi_k < \psi_{k+1}$, $k \in \{1, s, \ldots\}$ i.e. the in-order delivery time is strictly increasing. Suppose now that the slots of two packets q_1 and q_2 with $q_1 < q_2$ on path p are swapped, so that the packets are now sent in non-ascending order $s_{q_1} > s_{q_2}$. Let $\psi^{non}_{p,k} := \max\{\alpha_{p,s_q} : q \in K_{p,k}\}$ and $\psi^{non}_k := \max\{\psi^{non}_{1,k}, \ldots, \psi^{non}_{P,k}, \alpha_{p_k,s_k}\}$. Then $\psi^{non}_{p,k} \le \psi^{non}_{p,k+1}$ but now for at least one $k \in \{1, 2, \ldots\}$, namely $k = q_2$, there will be equality when $K_{p,k+1} \ne K_{p,k}$ and so $\psi^{non}_k = \psi^{non}_{k+1}$. Further, for $k \ge q_2$ then $\psi^{non}_k = \psi_k$ and also for $k < q_1$. Hence, $\psi^{non}_k > \psi_k$ for $k \in \{q_1, \ldots, q_2 - 1\}$ and so the sum-delay is increased relative to sending packets in ascending order. We can proceed by induction to show that swapping further packets cannot decrease the sum-delay below that when packets are sent in ascending order (there are two cases to consider, (i) when the further set of swapped packets is disjoint from those already swapped, in which case the above argument can be re-applied directly, and (ii) when the further set of swapped packets intersects with those already swapped, in which case we can recover an ascending packet order). Since the above holds for *any* realisation $\alpha_{p,1}, \ldots, \alpha_{p,n}$, we are done. □

Lemma 2 (Lyapunov CLT)**.** *Suppose $\{X_1, X_2, \cdots\}$ is a sequence of independent random variables, each with finite expected value μ_i and variance σ_i^2. Let us define $s_n^2 = \sum_{i=1}^n \sigma_i^2$. If for some $\delta > 0$, the Lyapunov's condition $\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{i=1}^n \mathbb{E}\big[|X_i - \mu_i|^{2+\delta}\big] = 0$ is satisfied, then a sum of $(X_i \mu_i)/s_n$ converges in distribution to a standard normal random variable, as n goes to infinity: $\frac{1}{s_n} \sum_{i=1}^n (X_i - \mu_i) \xrightarrow{d} \mathcal{N}(0,\ 1)$.*
Proof. See [43]. □

Proof of Theorem 2: S process.
1. We know that for the decoding process to go beyond k we need at least k erasures in the first k frames. This means that there are cases with more than k erasures in the first k intervals that the decoding process stops before k but for it to be greater than k we should have at least k erasures. Let E_k denote the number of losses in k frames. We have $P(S > k | E_k \le k) = 0$. Formally:

$$
\begin{aligned}
P(S > k) &= P(E_k > k) P(S > k | E_k > k) + \\
&+ P(E_k < k) P(S > k | E_k < k) = \\
&= P(E_k > k) P(S > k | E_k > k) < P(E_k > k)
\end{aligned}
$$

E_k is the sum of τ independent Bernoulli variables with parameter ϵ_i, i depending on the path. By Lemma 2, we know Lyapunov's condition is satisfied and

$$
\frac{1}{\sqrt{\sum_{p\in\mathcal{P}} k N_{\tau,p} \epsilon_p (1 - \epsilon_p)}} \Big(E_k - \sum_{p\in\mathcal{P}} k N_{\tau,p} \epsilon_p\Big)
$$

converges in distribution to $\mathcal{N}(0,\ 1)$ for large values of k. $P(E_k > k)$ goes to zero only if k is larger than $\sum_{p\in\mathcal{P}} k N_{\tau,p} \epsilon_p$. This means $\sum_{p\in\mathcal{P}} N_{\tau,p} \epsilon_p < 1$.

2. It is trivial to compute $P(S{=}\,0)$ and $P(S{=}\,1)$ exactly. Now, following the proof of Theorem 1.2 in [39], we can compute $P(S = k)$, $k > 1$ as:

$$
P(S{=}k) = \sum_{r=2}^{\min(k,\tau)} P(\mathcal{L}(\vec{\epsilon}) {=} r) \frac{r{-}1}{k{-}1} P(\mathcal{L}([\vec{\epsilon}]_{\times(k-1)} {=} k{-}r)), \forall k > 1 \tag{14}
$$

where $\vec{\epsilon} = \{\epsilon_{p_1}, \ldots, \epsilon_{p_i}, \ldots, \epsilon_{p_\tau}\}$, ϵ_{p_i} is the erasure probability in the path assigned to packet i, and $[\vec{\epsilon}]_{\times(k-1)}$ appends the $\vec{\epsilon}$ vector $k - 1$ times. However, the computation of

$P(\mathcal{L}(\vec{\epsilon}) = r)$ requires a complex iterative algorithm [44], which is not helpful for us. We thus approximate this part of the S distribution with:

$$P(S = k) \approx P(S_1 = k) = \tag{15}$$
$$= \sum_{r=2}^{min(k,\tau)} P(\mathcal{B}(\bar{\epsilon}, \tau{=}r)) \frac{r{-}1}{k{-}1} P(\mathcal{B}(\bar{\epsilon}, \tau(k{-}1) = k{-}r))$$
$$= \frac{N_\tau}{k} \bar{\epsilon}^k (1{-}\bar{\epsilon})^{kN_\tau} \binom{(k-1)\tau}{k-1}, \forall k > 1$$

where $\mathcal{B}$ is the binomial distribution with parameters $\bar{\epsilon} = \frac{\sum_{p\in\mathcal{P}} N_{\tau,p}\epsilon_p}{\tau}$ and $\tau = \sum_{p\in\mathcal{P}} N_{\tau,p}$.

3. Let us define S_1 as a random variable with the distribution of S but with erasure probability $\bar{\epsilon} = \frac{\sum_{p\in\mathcal{P}} N_{\tau,p}\epsilon_p}{\tau}$ for all paths. From [39], the first and second moments of S_1 are

$$\mathbb{E}[S_1] = \frac{(\tau-1)\bar{\epsilon}(1-\bar{\epsilon})^{\tau-1}}{1-\tau\bar{\epsilon}} \tag{16}$$
$$\mathbb{E}[S_1^2] = \mathbb{E}[S] + \frac{\tau(\tau-1)\bar{\epsilon}^2(1-\bar{\epsilon})^{\tau}}{(1-\tau\bar{\epsilon})^3} \tag{17}$$

Now, $\mathbb{E}[S]$ can be approximated as follows:

$$\mathbb{E}[S] \approx P(S = 1) + \sum_{k=2}^{\infty} kP(S_1 = k). \tag{18}$$

Note that $\sum_{k=2}^{\infty} kP(S_1 = k) = \mathbb{E}[S_1] - P(S_1 = 1)$, Thus,

$$\mathbb{E}[S] \approx P(S = 1) + \mathbb{E}[S_1] - P(S_1 = 1) =$$
$$= P(S{=}1) + \frac{\tau(\tau{-}1)\bar{\epsilon}^2(1{-}\bar{\epsilon})^{\tau-1}}{1{-}\tau\bar{\epsilon}}, \tag{19}$$

given that $P(S_1 = 1) = (\tau-1)\bar{\epsilon}(1-\bar{\epsilon})^{\tau-1}$. $\mathbb{E}[S^2]$ can be approximated similarly. □

Proof of Theorem 3: In-order delivery delay. Let us assume that a transmission of a stream of N_t packets is approximately a multiple of τ at time t. Then we have that, at time t, we have sent $\frac{N_t}{\tau}$ frames. Let us assume now that the decoding process consists of decoding periods of length $\{s_1, \ldots, s_n, \ldots\}$. Note that $\{S_1, \ldots, S_n\}$ is a sequence of positive i.i.d. random variables. Assume $S^+ = \min\{S, 1\}$, and define J_n as follows: $J_n = \sum_{i=1}^{n} S_i^+, n > 0$; then the renewal interval $[J_n, J_{n+1}]$ is a decoding period. Let $(X_t)_{t\geq 0}$ count the τ-intervals that have occurred by time t, which is given by $X_t := \sum_{n=1}^{\infty} \mathbb{I}_{\{J_n \leq t\}} = \sup\{\, n : J_n \leq t \,\}$ and is a renewal process ($\mathbb{I}$ is the indicator function).

Let $W_1, W_2, \ldots$ be a sequence of i.i.d. random variables denoting the sum of in order delivery delay in each decoding frame. We have two cases to consider. Case (i): suppose the j'th period is an idle period. Then $S_j = 0$ and the information packets are delivered in-order with delay Δ_p, where Δ_p is the transmission time of a packet sent in path p, here assumed as constant. Case (ii): suppose the j'th period is a busy period and the information packet erasure that initiated the busy period started in the first slot $t_{i(j)} + 1$. Then the first information packet in each path p is delayed by $S_j \Delta_p N_{\tau,p}$ slots, the second by $S_j \Delta_p N_{\tau,p} - 1$ slots and so on. The sum-delay over all of the information packets over all different paths in the busy period is therefore $\sum_{p\in\mathcal{P}}(\sum_{k=1}^{S_j\Delta_p N_{\tau,p}} k - \sum_{k=0}^{S_j-1} k\Delta_p N_{\tau,p}) < \sum_{p\in\mathcal{P}} \frac{S_j^2 \Delta_p N_{\tau,p}(\Delta_p N_{\tau,p}-1)}{2}$. The random variable $Y_t = \sum_{i=1}^{X_t} W_i$ is a renewal-reward process and its expectation is the sum of in order delivery delay over the time-span of t. Based on the renewal theorem for renewal-reward processes [45], we have: $\lim_{t\to\infty} \frac{1}{t}\mathbb{E}[Y_t] = \frac{\mathbb{E}(W_1)}{\mathbb{E}[S_1^+]}$. Based on the construction of W_i, we have

$$\mathbb{E}[W_1] = \sum_{i=1}^{\infty} \mathbb{E}[W_1 | S_1^+ = i] \Pr(S_1^+ = i) =$$
$$\sum_{i=0}^{\infty} \mathbb{E}[W_1 | S_1{=}i] \Pr(S_1{=}i) {=} \mathbb{E}[S^2] \sum_{p\in\mathcal{P}} \frac{\Delta_p N_{\tau,p}(\Delta_p N_{\tau,p}-1)}{2}. \tag{20}$$

We also have that $\mathbb{E}[S_1^+] = \mathbb{E}[S] + P(S = 0)$, and that $\lim_{t\to\infty} \frac{1}{t}\mathbb{E}[Y_t] \leq \frac{\mathbb{E}[S^2]}{\mathbb{E}[S]+P(S=0)} \sum_{p\in\mathcal{P}} \frac{\Delta_p N_{\tau,p}(\Delta_p N_{\tau,p}-1)}{2}$. Since we assumed that $N_t = t\tau$, we have:

$$\lim_{N_t\to\infty} \frac{1}{N_t} E[Y_t] \leq \frac{\sum_{p\in\mathcal{P}} \Delta_p N_{\tau,p}(\Delta_p N_{\tau,p}-1)}{2\tau(\mathbb{E}[S] + P(S = 0))} \mathbb{E}[S^2]$$

□

Proof of Theorem 4: Path choice for coded packets. Suppose we have a system with $\mathcal{P} = \{1, \ldots, p\}$ paths with erasure probabilities $\epsilon_1 \geq \epsilon_2 \geq \cdots \geq \epsilon_p$. Let us first evaluate two independent S processes, S_1 and S_2, and schedule coded packets on path 1 in the first case and on p_2 (any other) in the second case. Let $p_c = 1$ and $p_c = p_2$ describe this. We know from eq. (14) that the path selected for the coded packets *does not* affect the decoding process when $S > 1$:

$$P(S_1 = k) = P(S_2 = k), \ \forall k > 1. \tag{21}$$

When $\sum_{p\in\mathcal{P}} N_{\tau,p}\epsilon_p < 1$ (i.e. when we operate below the capacity limit), the above yields:

$$P(S_1 = 0){+}P(S_1 = 1){=}P(S_2 = 0){+}P(S_2 = 1) \tag{22}$$

given that $\sum_k P(S = k) = 1$.

Note that the decoding delay is equal to 0 slots with probability $P(S = 0)$ (it is the in-order delivery state), and non-zero otherwise (packets are being buffered until all losses are recovered). This means that, given equations (21) and (22), *the S_i process that achieves lower delay is the one that maximises* $P(S = 0)$. Let us then compare $P(S_1 = 0)$ and $P(S_2 = 0)$:

$$\frac{P(S_1 = 0)}{P(S_2 = 0)} = \frac{(1-\epsilon_1)^{N_{\tau,1}-1} \prod_{i\neq 1}(1-\epsilon_i)^{N_{\tau,i}}}{(1-\epsilon_{p_2})^{N_{\tau,p_2}-1} \prod_{i\neq p_2}(1-\epsilon_i)^{N_{\tau,i}}} =$$
$$= \frac{(1-\epsilon_{p_2})}{(1-\epsilon_1)} \geq 1 \tag{23}$$

because $1 \geq \epsilon_1 \geq \epsilon_{p_2} \geq 0$. Thus $P(S_1 = 0) \geq P(S_2 = 0)$, i.e., scheduling coded packets on path 1 (the one with highest loss probability) minimises the decoding delay, and this is valid for any S_i process different than S_1. □

ACKNOWLEDGEMENTS

This work has received funding from the Science Foundation Ireland under grant agreement No 11/PI/1177 and the European Union's Horizon 2020 research and innovation programme under grant agreement No 671598.

References

[1] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. C. Zhang, "What Will 5G Be?" *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[2] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-Enabled Tactile Internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 460–473, March 2016.

[3] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "OpenFace: A general-purpose face recognition library with mobile applications," Technical report, CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[5] J. Sundararajan *et al.*, "Network Coding Meets TCP," in *INFOCOM 2009, IEEE*, April 2009, pp. 280–288.

[6] Y. Cui, X. Wang, H. Wang, G. Pan, and Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, June 2012, pp. 366–375.

[7] R. Khalili *et al.*, "MPTCP is not pareto-optimal: performance issues and a possible solution," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1651–1665, 2013.

[8] M. Zhang, J. Lai, and A. Krishnamurthy, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *In USENIX Annual Technical Conference*, 2004, pp. 99–112.

[9] C. Paasch *et al.*, "Experimental evaluation of multipath TCP schedulers," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM, 2014, pp. 27–32.

[10] M. Báguena, D. J. Leith, and P. Manzoni, "Measurement-Based Modelling of LTE Performance in Dublin City," http://arxiv.org/abs/1506.02804, 2015.

[11] R. Mahajan, J. Padhye, S. Agarwal, and B. Zill, "High performance vehicular connectivity with opportunistic erasure coding," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 22–22.

[12] S. A. Latif *et al.*, "An investigation of scheduling and packet reordering algorithms for bandwidth aggregation in heterogeneous wireless networks," *Middle-East Journal of Scientific Research*, vol. 16, no. 12, pp. 1613–1623, 2013.

[13] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A Measurement-based Study of Multipath TCP Performance over Wireless Networks," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, pp. 455–468.

[14] K. Chebrolu and R. Rao, "Communication using multiple wireless interfaces," in *Wireless Communications and Networking Conference, 2002. 2002 IEEE*, vol. 1, Mar, pp. 327–331 vol.1.

[15] A. Ford, C. Raiciu, M. Handley, O. Bonaventure *et al.*, "TCP extensions for multipath operation with multiple addresses," *RFC 6824 (Experimental)*, Jan. 2013.

[16] J. Iyengar, P. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 5, pp. 951–964, Oct 2006.

[17] T. Dreibholz, R. Seggelmann, M. Tüxen, and E. P. Rathgeb, "Transmission scheduling optimizations for concurrent multipath transfer," in *Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, vol. 8, 2010.

[18] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 1222–1227.

[19] 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.

[20] J. C. Fernandez *et al.*, "Multi-path scheduling algorithm for real-time video applications in next-generation wireless networks," in *Innovations in Information Technology, 2007. IIT'07. 4th International Conference on*. IEEE, 2007, pp. 73–77.

[21] G. Liu, X. Zhou, and G.-x. Zhu, "A scheduling algorithm for maximum throughput based on the link condition in heterogeneous network," *Journal Communication and Computer*, 2007.

[22] Y. Liao *et al.*, "Cooperative robust forwarding scheme in dtns using erasure coding," in *Military Communications Conference, 2007. MILCOM 2007. IEEE*. IEEE, 2007, pp. 1–7.

[23] Y. Lin, B. Liang, and B. Li, "SlideOR: Online Opportunistic Network Coding in Wireless Mesh Networks," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–5.

[24] S. Gheorghiu, A. L. Toledo, and P. Rodriguez, "Multipath TCP with network coding for wireless mesh networks," in *Proc. of ICC*, 2010, pp. 1–5.

[25] M. Balakrishnan *et al.*, "Maelstrom: Transparent error correction for lambda networks," in *Proc. of NSDI 2008*, April 2008.

[26] V. Subramanian, S. Kalyanaraman, and K. Ramakrishnan, "An end-to-end transport protocol for extreme wireless network environments," in *Military Communications Conference, 2006. MILCOM 2006. IEEE*. IEEE, 2006, pp. 1–7.

[27] V. Sharma and others., "A transport protocol to exploit multipath diversity in wireless networks," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 4, pp. 1024–1039, Aug 2012.

[28] Y. Hwang *et al.*, "HMTP: Multipath transport protocol for multihoming wireless erasure networks," *Transactions on Emerging Telecommunications Technologies*, 2014.

[29] J. Wu *et al.*, "Loss tolerant bandwidth aggregation for multihomed video streaming over heterogeneous wireless networks," *Wireless personal communications*, vol. 75, no. 2, pp. 1265–1282, 2014.

[30] M. Li, A. Lukyanenko, and Y. Cui, "Network coding based multipath tcp," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, March 2012, pp. 25–30.

[31] A. Frömmgen, T. Erbshäußer, T. Zimmermann, K. Wehrle, and A. Buchmann, "ReMP TCP: Low Latency Multipath TCP," 2016.

[32] D. Jurca and P. Frossard, "Video packet selection and scheduling for multipath streaming," *Multimedia, IEEE Transactions on*, vol. 9, no. 3, pp. 629–641, 2007.

[33] Y.-C. Chen, D. Towsley, and R. Khalili, "MSPlayer: Multi-Source and multi-Path LeverAged YoutubER," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: ACM, 2014, pp. 263–270. [Online]. Available: http://doi.acm.org/10.1145/2674005.2675007

[34] D. H. Bui, K. Lee, S. Oh, I. Shin, H. Shin, H. Woo, and D. Ban, "GreenBag: Energy-Efficient Bandwidth Aggregation for Real-Time Streaming in Heterogeneous Mobile Wireless Networks," in *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, Dec 2013, pp. 57–67.

[35] S. Gouache, G. Bichot, A. Bsila, and C. Howson, "Distributed & adaptive HTTP streaming," in *2011 IEEE International Conference on Multimedia and Expo*, July 2011, pp. 1–6.

[36] D. Sinha, H. Zhou, and N. V. Shenoy, "Advances in computation of the maximum of a set of Gaussian random variables," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Tran. on*, vol. 26, no. 8, pp. 1522–1533, 2007.

[37] S. Jaiswal *et al.*, "Measurement and Classification of Out-of-sequence Packets in a Tier-1 IP Backbone," *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 54–66, Feb. 2007.

[38] A. Shapiro and A. Philpott, "A tutorial on stochastic programming," www2.isye.gatech.edu/ashapiro/publications.html.

[39] M. Karzand and D. Leith, "Low delay random linear coding over a stream," in *52nd Annual Allerton Conference on Communication, Control, and Computing, 2014*, Sept 2014, pp. 521–528.

[40] M. Fernandez and S. Williams, "Closed-form expression for the Poisson-binomial probability density function," *Aerospace and Electronic Systems, IEEE Trans. on*, vol. 46, no. 2, pp. 803–817, 2010.

[41] M. Kim *et al.*, "CTCP: Coded TCP using multiple paths," *arXiv preprint arXiv:1212.1929*, 2012.

[42] M. Kim *et al.*, "Congestion control for coded transport layers," in *Proc. of ICC*, 2014, pp. 1228–1234.

[43] P. Billingsley, *Probability and Measure*, ser. Wiley Series in Probability and Statistics. Wiley, 1995.

[44] W. Ehm, "Binomial approximation to the Poisson binomial distribution," *Statistics & Probability Letters*, vol. 11, no. 1, pp. 7–16, 1991. [Online]. Available: http://EconPapers.repec.org/RePEc:eee:stapro:v:11:y:1991:i:1:p:7-16

[45] R. G. Gallager, *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2014.