Flexible Transport Services for Emerging Opportunities in Internet Content Delivery

John Byers *
Dept. of Computer Science, Boston University

ABSTRACT

The emergence of new technologies for Internet content delivery including CDNs, end system multicast, peer-to-peer content exchange (e.g. Napster) and application-level anycast are fundamentally changing the relationships between consumers and producers of digital content. These new technologies enable delivery mechanisms which go far beyond point-to-point connections between a client and a single origin server. For example, CDNs redirect clients to a number of equivalent replica servers, while end system multicast blurs the distinction between client and server by requiring that every participant who wishes to access content serve content as well. In this position paper, we motivate and describe scalable strategies for addressing the challenges and opportunities that distributed content distribution mechanisms will place on reliable transport protocols and distributed scheduling algorithms. Central to our discussion are lightweight reliable transport services designed for maximum flexibility which can accomodate connection preemption, connection migration and connection aggregation built in part on fast forward error correcting (FFEC) codes.

Keywords: forward error correction, end-system multicast, digital fountain, peer-to-peer networks, reliable transport, CDN, FEC, TCP

1. MOTIVATION

Reductionist approaches to content delivery in which consumers and producers of digital content are paired in one-to-one relationships are quickly becoming obsolete. For example, attempting to disseminate very popular content from a single origin server† using point-to-point connections is an ultimately unscalable strategy which has led to embarrassing and widely documented service outages. To remedy the problems associated with such a centralized approach, and as a result of the economic significance and research interest placed on developing scalable strategies for content delivery, a variety of innovative and effective mechanisms for content delivery have emerged in just the last few years. These include:

- Content distribution networks (CDNs)[1,16]: Services which distribute content and the request load – typically by caching popular content within the network or by replicating content and providing infrastructure services to redirect users to an appropriate replica.
- Peer-to-peer content exchange[28]: An appealing mechanism for information dissemination is epitomized by Napster, in which end systems (typically PC's) informally register and host the popular content that happens to be present on their hard drive. Napster indexes the content and provides a service which directs clients to a set of remote peers from which they can download the content they desire.
- Native IP multicast[14]: Use of IP multicast moves much of the burden of transport from the origin server to the routers implementing multicast functionality. While IP multicast can enable the audience size to scale without a commensurate increase in server-side provisioning; it is unfortunately still not widely deployed, and may never be.
- End system multicast[13,18,41]: A recently proposed strategy which places the burden of implementing multicast functionality at end hosts rather than at routers by connecting groups with end-to-end unicast tunnels. While end system multicast does not achieve all of the benefits of IP multicast, it can achieve many of them and no router deployment hurdles stand in its way.

*byers@cs.bu.edu; phone 1 617 353-8925; fax 1 617 353-6457; http://www.cs.bu.edu/fac/byers; Boston University, Computer Science Department, 111 Cummington St., Boston, MA 02215. This work is supported in part by NSF CAREER grant ANI-0093296 and by NSF grant ANI-9986397.

†By origin server, we mean a geographically proximate set of machines which could comprise a server farm.

- Application-layer anycasting[42,17]: A general request resolution service in which an anycast resolver maps an anycast domain name and a selection criteria to an appropriate IP address from which the request can be serviced. This approach has been naturally applied to server selection for content delivery.

While study of the performance of these emerging services[20,23] is an interesting research problem in its own right, in this position paper, we point to reasons why the complex, distributed many-to-one and many-to-many relationships between providers and consumers of digital content will place increasing strain on a standard TCP transport abstraction. To date, the services which have gained the most traction and have seen the widest adoption are those which require minimal changes to the network infrastructure and operate from the application layer; i.e. by running on top of established transport protocols such as TCP. However, we argue that these applications will soon need and demand more flexibility. In particular, these service models will lead to the need for ephemeral connections which can be *preempted*, *suspended*, *migrated* and *aggregated*, while still facilitating reliable (and congestion-controlled) delivery. Although support for connection suspension in TCP is currently feasible, socket migration[4] is at best an extremely cumbersome process, and seamless, scalable connection aggregation currently appears unrealizable.

1.1. Overview

To address the considerations above, we motivate the development of a flexible point-to-point transport service which relaxes the need for sequenced packet delivery and is optimized for transfer of large files, which still providing reliable congestion-controlled service. We then explore the integration of these flexible mechanisms in the context of the distributed service models defined above. An essential ingredient in our proposal is the use of forward error correction. Connections transmitting encoded content using the powerful abstractions we have developed and applied in the context of reliable multicast[8] and downloading content from multiple mirror sites in parallel,[6] realize a number of extremely useful properties (enumerated in Section 2.3), many of which cannot be realized when codes are not employed. In order to realize these abstractions, we cannot employ textbook codes but must instead leverage new codes developed in recent years. These codes serve as the core building block for the *fountain encoding streams* which we propose as a mechanism to provide this service.

In the first section of this position paper, we consider two different methodologies for integrating fountain encoding streams with existing transport protocols: (1) an application-level manager acting in concert with TCP flows transporting encoded content and (2) a protocol for transporting encoded content over congestion-controlled UDP flows, where congestion control is provided by the Congestion Manager (CM) architecture.[2] While we view flexible, reliable transport as fundamental in its own right; we perceive much of the power of the service as lying in its applications. As we shall demonstrate, this flexibility makes them ideally suited to operation in dynamic environments in which clients and servers seek to exploit the complex many-to-many relationships implicit in distributed mechanisms for content delivery. Thus in the second section of the paper, we focus on applying our approach to two general strategies for content delivery: (1) accessing content from replicated servers, an approach which encompasses the CDN and application-level anycast approach; and (2) content delivery via peer-to-peer communication, which encompasses the end-system multicast approach and the approach used by Napster. We conclude with a mention of additional applications of fountain encoding streams for facilitating content distribution.

The rest of this paper is organized as follows. In Section 2, we provide a technical overview of the major building block of our approach, fountain encoding streams via fast forward error correcting (FFEC) codes, and summarize the significant properties that fountain encoding streams provide. In Section 3, we sketch our proposed strategies for providing the flexible transport-level mechanisms we have discussed in this introduction and demonstrate how they directly provide flexible mechanisms for accessing content from replicated servers. Then in Section 4.1, we demonstrate the potential applications of our techniques in the realm of end-system multicast, where dynamics of network conditions demand the fluid transport services we have defined. In Section 4.2, we describe applications to streamlining content delivery from a server design standpoint before presenting our conclusions.

2. ENCODING PARADIGMS, PROPERTIES AND APPLICATIONS

Rabin[32] defined an Information Dispersal Algorithm (IDA) which disperses a file F of length ℓ into n pieces F_i, $1 \leq i \leq n$ each of length ℓ/m such that the original file F can be reconstructed from *any* m pieces, where $n > m$. Rabin cited many applications for this powerful procedure, ranging from dispersity routing, to storing files on disks which are prone to failure, to fault-tolerant broadcast. The coding techniques employed to realize IDA are in the family of error-correcting codes which add redundant bits to a message and can recover from the occurrence of any

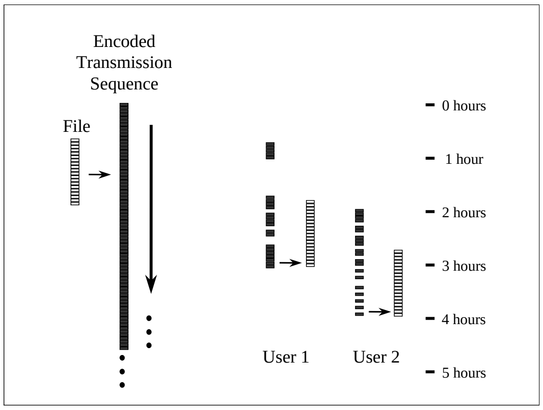
k bit errors within the block. For the applications which Rabin considered, pieces either arrive intact, or are lost entirely. For this reason, such codes are now often referred to as *erasure codes*, or, for certain networking applications, as *forward error correcting* (FEC) codes.

While the general approach proposed by Rabin is very powerful, a limitation of the approach is the computational complexity of the encoding and decoding operations he used. Most authors applying IDA approaches to networking applications have used variants of Reed-Solomon codes[26] as their forward error correcting primitive.[3,33,31,5] These codes rely on cumbersome finite field operations and have $O(n^2)$ encoding and decoding times. Thus, while IDA works well for reconstructing a file stored as n pieces distributed over n remote sites when n is 14 and m is 10, its decoding inefficiency is prohibitive when multicasting an encoded file spanning n packets when n is 10,000 and m is 5,000. Another potential limitation in the definition of IDA is that n, the number of pieces generated, must be known in advance. For many networking applications, the number of pieces needed may not be known in advance, and precomputing a certain quantity can either be wasteful or insufficient.

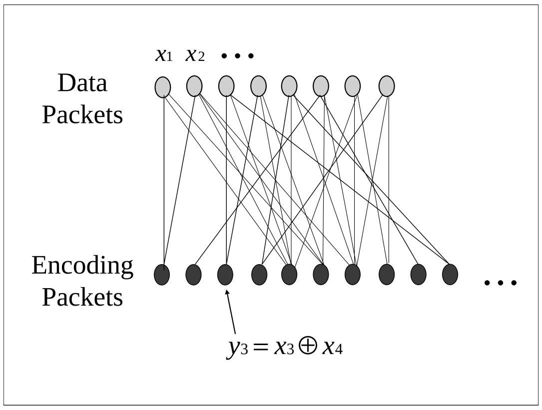
2.1. Applying FEC Codes to Reliable Multicast

As an example where these limitations pose a hindrance, consider the application of multicasting a large file from a single distribution server to a vast set of clients. Many authors have considered the strategy of applying forward error correction to this application,[33,34,31,30,37,22] since each forward error correcting packet enables different receivers to recover from different packet losses, which provides excellent scalability. For this application the protocol must be *reliable*, must allow *asynchronous* client arrival, must be *time and space efficient* at clients and must *tolerate client heterogeneity* in terms of end-to-end packet loss rates and data rates. These considerations motivated us in previous work to define the concept of a *digital fountain*,[8] which disperses a file F of length ℓ into an infinite sequence of encoding packets F_i, $1 \leq i \leq \infty$ each of length $p = \ell/m$ such that the original file F can be reconstructed from *any* m packets in $O(m)$ time and space.

A depiction of how an (idealized) digital fountain encoding can be used in the context of reliable multicast is depicted in panel (a) of Figure 1. On the left-hand side of this panel, the source pre-encodes or dynamically generates a packetized encoding of the large file to be transmitted. Time elapses from top to bottom and the source transmits packets across a multicast channel, which in this simplified depiction is assumed to be a constant bit rate transmission. On the right-hand side of the panel, two clients arrive asynchronously, receive different encoding packets and are still able to recover the file efficiently once they receive a sufficient quantity of encoding packets. Issues posed by client 1's temporary midstream suspension of the download, client 2's substantial packet loss rate, and issues in scaling to a vast number of clients pose no difficulties.



(a)



(b)

Figure 1. (a) Reliable multicast using a digital fountain encoding and (b) The encoding process used by sparse parity-check codes

File Size	Tornado Z	Reed-Solomon
250 KB	0.18 seconds	2.06 seconds
500 KB	0.24 seconds	8.4 seconds
1 MB	0.31 seconds	40.5 seconds
2 MB	0.44 seconds	199 seconds
4 MB	0.74 seconds	800 seconds
8 MB	1.28 seconds	3,166 seconds
16 MB	2.27 seconds	13,629 seconds

Table 1. Comparison between decoding times for Tornado codes and Reed-Solomon codes

2.2. Practical Issues in Fast Forward Error Correction

In practice, information-theoretic considerations make the generation of such an infinite sequence impossible, but by *relaxing the decoding guarantee* and by the use of randomization, we can achieve an approximation which is nearly as powerful[8] using recently developed sparse, parity-check codes such as Tornado codes.[25] By relaxing the decoding guarantee, we mean that we require slightly more than m encoding packets to recover the original file. This extra quantity, which we refer to as the decoding *overhead*, is a random variable which depends on the random subsequence of packets received. For large files, i.e. when $m > 1,000$, the codes we employ have an average overhead of 5% or less and low overhead variance.[8]

We overview the production of encoding packets as depicted in the right panel of Figure 1. A document is divided into a collection of ℓ fixed-length input symbols, or data packets, $x_1, \ldots, x_\ell$. An *encoder* produces a potentially unbounded sequence of output symbols, or encoding packets, $y_1, y_2, \ldots$ from the set of data packets. With parity-check codes, each output symbol is simply the bitwise XOR of a specific subset of the input packets. To optimize decoding, the distribution of the size of the subsets chosen is irregular; a heavy-tailed distribution was proven to be a good choice.[25] We say that a code is capable of on *on-the-fly* or *memoryless* encoding if each encoding packet can be generated identically and independently from the same distribution according to the same randomized procedure. While Tornado codes are not of this form; Reed-Solomon codes are compatible with on-the-fly production, as are more refined codes derived from Tornado principles used in the asynchronous layered coding (ALC) draft[24] and by Digital Fountain, Inc.[15] A *decoder* attempts to recover the entire set of input symbols from a subsequence of the output symbols. For a given encoding packet encoded with parity-check codes, we refer to the cardinality of its neighborhood as its *degree*, i.e. $y_3 = x_3 \oplus x_4$ has degree 2. The time to produce an encoding packet from a set of data packets is proportional to the degree of an output symbol, while decoding from a sequence of output packets takes time proportional to the total degree of the output packets in the sequence. Thus, while heavy-tailed distributions introduce some packets of large degree, it is important to note that encoding and decoding times are a function of the *average* degree, not the maximum.

For the purpose of providing a rough performance comparison, relative decoding times of Tornado codes against standard Reed-Solomon codes on identical hardware is given in the right hand side of Table 2.2. This table has been reproduced from[8]; please refer to that reference for the full details of the two implementations and the experimental setup. In both cases, a packetized encoding of a source file using 1KB packets was randomly generated and transmitted to a receiver across a lossy channel. The decoding comparisons measure the times for Tornado codes and Reed-Solomon codes to recover a file of a given length k from a randomly chosen, sufficiently large collection of encoded packets. Sparse-on-average coding, the use of the XOR operation, randomization and relaxation of the decoding guarantee provide the very fast coding times.

To summarize the coding technique, we can now achieve an approximation which is nearly as powerful as an ideal digital fountain, an algorithm which disperses a sufficiently large file F of length ℓ into packets F_i, $1 \le i \le \infty$ each of length $p = \ell/m$ such that the original file F can be reconstructed from a randomly chosen subsequence of $1.05m$ packets in $\mathrm{O}(m)$ time and space with very high probability.

2.3. Properties

We now enumerate some of the key properties that a content stream protected by forward error correction provides. While the differences between an IDA approach and a digital fountain approach are subtle, they make for a marked difference in terms of the properties that can be realized. In the spirit of Rabin's work, we refer to a block of data protected by a pre-specified amount of forward error correction as a *dispersed encoding*. Likewise, we refer to transmission of a dispersed encoding as a *dispersed encoding stream*. Similarly, we refer to the infinite sequence of codewords produced by a memoryless encoding process as a *fountain encoding* stream. Dispersed encoding streams have the following key properties:

+ **Dispersed encoding streams are resilient to packet loss.** The foremost benefit of transforming the content into a dispersed encoding is fault tolerance to packet losses. In multicast settings, this is especially powerful, as multicast transmission of an encoding packet can facilitate error recovery for a set of different receivers who are attempting to recover from different packet losses. Without encoding, retransmissions have to be carefully scoped to avoid exposing other clients to redundant data.

+ **Content transmitted across a dispersed encoding stream is not client-specific.** Thus, an arbitrary number of clients can attach to a stream without a performance penalty and content can be shared across clients seamlessly. In particular, client-specific retransmissions are not employed. Moreover, sources of dispersed encoding streams need not retain connection-specific state for the purpose of providing reliability.

In addition to the properties listed above, a fountain encoding stream also has the following properties:

+ **Packet sequences produced by distinct fountain encoding streams are uncorrelated.** From a client perspective, the sequence of encoding packets produced by distinct servers are independent.

+ **Fountain encoding streams are memoryless.** At the server, the production of each encoding packet is independent of past events and requires only a source of randomness. From a client perspective, the content produced by a server is time-invariant.

+ **Fountain encoding streams are additive.** As a corollary to the spatial and temporal invariance described by the two properties above, a client may attach to multiple or additional sources of encoded content seamlessly with a commensurate impact on the transport rate.

+ **Fountain encoding streams are self-checking.** A small percentage of received encoding packets will be redundant and of no value in reconstructing the message. These packets are typically discarded. However, in Section 4.3, we sketch methods whereby these packets can be used to validate correctness of received content. Also, content received in part from separate encoding streams can be "cross-checked" to authenticate untrusted sources.

We now describe how the properties here can be advantageously applied to deliver flexible unicast transport, then move on to discuss applications to content distribution.

3. FLEXIBLE UNICAST TRANSPORT

One of the limitations of current point-to-point transport services is that after a TCP connection is established between a client and a server, that connection cannot be prematurely suspended, preempted or redirected. For clients performing a large file transfer over a modem line, breaking such a connection can result in hours of time lost. Thus from a client standpoint, the capability to checkpoint a transfer or to temporarily suspend a transfer and resume it later is a valuable option that is now being met by commercial products.[29] But there are other natural situations which require more than simple checkpointing and demand connections to be *redirected*. For example, from a content delivery standpoint, server load imbalance, or end-to-end quality of service considerations may result in situations where it is desirable to *preempt* an established connection and redirect it to another server equally well equipped to handle the request. Similarly, when network conditions become degraded on the path from the server to the client, it can be in the client's best interest to redirect the connection. Fountain encoding streams are ideally suited to either client-initiated and server-initiated connection redirection because of the first property stated in the previous section:

Packet sequences produced by distinct fountain streams are uncorrelated.

The chronology of an example of server redirection appears in Figure 2. In this example, a client is downloading a large file from one of a number of equivalent replica servers. For the sake of this example, we assume that the client

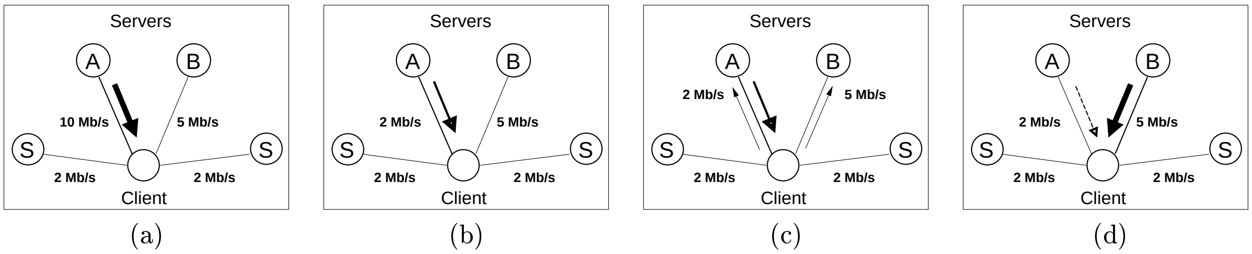


Figure 2. The chronology of connection migration (edge labels correspond to estimates of bottleneck bandwidth on the paths): (a) Client downloading normally from server A, (b) Network conditions now make server B a more attractive choice, (c) Client notifies servers of transition, (d) Transition completing.

has the capability to initiate server redirection and measures performance by maintaining an estimate of bottleneck bandwidth (or any other selection criterion) to other potential servers even after the original server selection has taken place. One example of a lightweight probing techniques capable of tracking bottleneck bandwidth in the context of server selection is the bprobe tool[10,11]; more general architectures for network performance discovery include the SPAND[39] architecture. As resource utilization changes on the paths from servers A and B to the client over the course of a large transfer, the client identifies a more promising path to a server, and invokes a redirection.[‡] Without any inter-server synchronization or communication, the client succeeds in redirecting its request. In fact, the final frame depicts the fact that tight synchronization is not necessary – encoding packets can continue to arrive from the original source, with full benefit for the client.

The most significant technical challenges associated with this basic approach are protocol design and seamless client-side merging of encoded content from different sources. In our recent ongoing work, we have considered two separate methodologies for doing so: (1) an application-level manager acting in concert with TCP flows transporting fountain encoded content and (2) merging fountain encoded content from congestion-controlled UDP flows, where congestion control is provided by the Congestion Manager (CM)[2] architecture. To study the first approach, we have built a simple application-level manager capable of merging and decoding content from multiple fountain streams running over TCP. We suspect that the second approach, which we are currently building, may be conceptually simpler and more streamlined, and we are in the process of developing a detailed performance comparison of the costs and benefits of these two approaches. A related note is that in evaluating these designs, one must consider the practical consequences of employing an underlying transport protocol. For example, many firewalls are currently configured to block UDP traffic, thus special administrative steps need to be taken to support the delivery of content over UDP.

This development relates closely to one of our previous works, in which we demonstrate that fountain encoding streams could in theory enable clients to speed up downloads by downloading from multiple mirror sites in parallel.[6] This ongoing work also requires client-side *aggregation* and merging of encoded content from multiple incoming streams, but has not yet addressed the technical aspects of how to accomplish this objective from a pragmatic, protocol-oriented standpoint. We anticipate that the application-level techniques to support flexible transport described here can be employed directly in this context as well.

4. APPLICATIONS TO INTERNET CONTENT DELIVERY

The flexible transport-level service mechanisms using fountain encoding streams which we motivate in the previous section are of fundamental importance; we anticipate that their value will become most apparent when leveraged by higher-level applications. In this section, we detail some of the specific content delivery applications and mechanisms which we foresee as able to benefit from the services we will enable.

‡We expect that redirection would be a relatively infrequent occurrence; application of hysteresis to avoid oscillations should naturally be employed.

4.1. End-System Multicast

End system multicast[13] and similar infrastructure service architectures[18,12,41] have recently been proposed as feasible content delivery alternatives to native IP multicast. The basic idea behind end system multicast, which is primarily designed for relatively small, sparse multicast sessions, is to have end systems, rather than the IP layer, provide all multicast services including group management, packet forwarding and flow and error control. Other architectures, such as Scattercast,[12] employ similar principles, but also rely on proxies within the network to scale to large group sizes and to streamline resource discovery protocols. These architectures achieve their objective by establishing and dynamically maintaining an overlay network consisting of unicast tunnels spanning all end systems in the session. A multicast group containing N members would require establishment of at least $N-1$ such tunnels to connect the group; it is therefore natural to think of the collection of tunnels as a tree, even though tree edges (point-to-point connections between end systems) may share physical links, as depicted in Figure 3. For an end system multicast session, the number of tree edges which share a physical link is defined to be the *stress* on that link, which corresponds to the number of copies of each packet in the session that traverse the link. The stress on a link is a measure of the cost of end system multicast (or a true point-to-point approach) relative to IP multicast, as all physical links in an IP multicast tree have stress 1. The strategy of end system multicast sidesteps many of the basic deployment issues associated with IP multicast, such as the need for routers to maintain per-group state; the need for changes in the network infrastructure; and the challenge of arriving at consensus on new scalable congestion control and reliability mechanisms for native multicast.

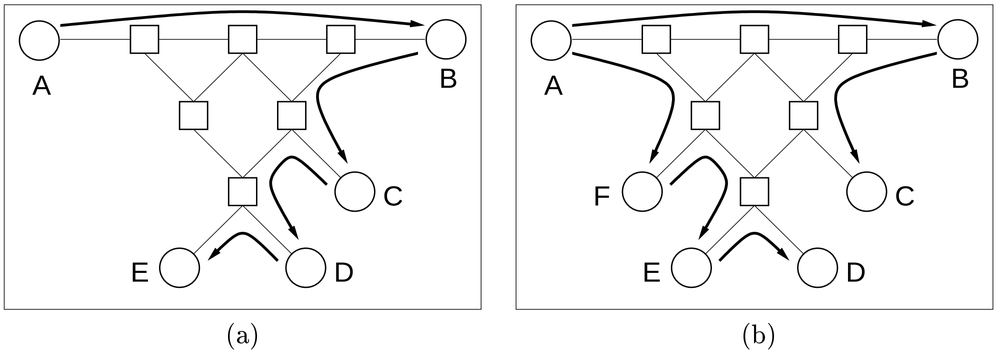


Figure 3. Dynamic adjustment of an end system multicast tree after arrival of F. Routers are denoted by squares, end systems by circles, network links by thin lines and unicast tunnels by darker arcs directed away from the source (node A).

An intriguing aspect of end system multicast is that routing protocols no longer completely dictate the route packets take from a source to a given destination. Instead, routes are substantially influenced by the manner in which distributed end systems self-organize. Moreover, one would expect end systems to take advantage of this flexibility, thus routes would change dynamically as sessions evolve. In Figure 3, we depict a situation in which arrival of a new client induces updates to the delivery tree. We refer to an end system which does not forward content as a *leaf* in the distribution topology, and nodes which do as *relay points*. In this particular example, host E, a former leaf, becomes a relay point responsible for transmitting content to its former parent. Other changes in the topology include host C transforming from a relay point to a leaf and host E inheriting a new parent in the distribution tree.

Returning to the main theme of our position paper, employing end system multicast in its dynamic generality will require the flexible, reliable transport services that we presented in Section 3. Below we enumerate some of the key challenges in end system multicast which we can now address with these services. We note that the development of such an end system multicast architecture with support for these facilities is one of the primary goals of our Shapeshifter[38] project.

Dynamically changing distribution trees: As depicted in Figure 3, arrivals and departures of end systems and dynamically changing network conditons can induce the distribution topology to change on the fly. This figure highlights many of the aspects of flexible transport which we touched on earlier: unicast connections can be deleted, established, or rerouted. The need for a flexible transport protocol which can tolerate changes of this form without a substantial performance penalty is clear. Our ongoing empirical studies in simulation and over the Internet study fountain encoding methods to minimize the costs associated with dynamic changes in the distribution tree.

Optimizing distribution tree layout: The proposed architectures for end system multicast propose a combination of bounded fanout and delay measurements as the metrics over which to optimize construction of the distribution tree. While delay measurements,[13,19] are useful and easily obtained, we propose a different optimization criterion: *minimizing stress on congested links*. For example, returning to Figure 3, if paths from A to C and B to C share congested links, then establishing those paths as part of the distribution tree is a poor choice, especially when the shared congested links are close to the source, as this could constrict the session rate. We have developed fast, robust unicast probing techniques for determining whether or not two paths sharing a common endpoint also share congested links[21]; a different unicast probing approach to the same end[36] was also recently proposed. These active unicast probing techniques, which currently run over UDP, fit in naturally with the existing framework for end system multicast.

Generalized topologies for distribution: Attention need not be restricted to routing content over multicast *trees*, which is an artifact of IP routing. In general, end system multicast can naturally support multipath or dispersity routing. We envision scenarios in which a remote receiver can choose from multiple bottleneck-disjoint routes through a set of end systems back to the source. The importance of bottleneck-disjoint routes[6] was recently provided, as were subsequent techniques for identifying them.[21,36] Conceivably, fountain encoding streams could enable a receiver to use these routes in parallel. We argue that the most natural topologies for single-source content distribution lies somewhere between a tree and an arbitrary DAG; however, there are significant technical obstacles to support effective dynamic multi-path content delivery algorithms. One such obstacle we foresee is to ensure that the content across parallel streams still maintains the memoryless condition, as this may not always be possible.

Optimizing outbound transmissions at relay points: For small or sparse multicast sessions, the out-degree of a relay point is not likely to be large. But in scalable proxy-based architectures,[12] proxies may have to manage a large number of concurrent, outbound flows serving identical content. In the next section, we describe a general tactic for optimizing concurrent delivery of fountain encoded content to a set of clients which applies to end system multicast.

Independent encoding at relay points: In architectures which propose to use end system multicast for content delivery,[41] it is assumed that participants may serve as relay points even after they have recovered the file. With fountain encoding, after file reconstruction, these participants can then generate encoding packets *independent from the original source*. This has significant implications, including the fact that the distribution topology can become a set of independent connected components, or can enable content delivery from distinct sources in parallel, as described above.

To summarize, we view end system multicast as an important paradigm for content delivery that provides a rich research agenda. In order to realize scalable, reliable end system multicast, we need to first address the fundamental challenges posed by the need for flexible transport services, then couple those services with appropriate resource discovery mechanisms and appropriate augmentation of end host architectures. This final point leads us in to the next application that fountain encoding streams enable.

4.2. Streamlining Content Delivery at the Server

In Section 3, we have already described the client-side strategies we can employ to access content from spatially or temporally disparate sources. In this section, we consider the challenges associated with server-side delivery of content. We focus on the following specific problem motivated by the discussion in the previous section: at an instant in time, a relay point or proxy is responsible for delivery of a single file across a set of unicast connections.[§] Arrivals of

[§]For several of the applications we consider, one would expect a server or proxy to serve a small number of popular files concurrently. In order to simplify the discussion we ignore this consideration here and restrict attention to the case of a single file.

new requests for service arrive asynchronously; moreover congestion-controlled transfer rates across these connections may have high variability. A generic TCP-based approach to serving these clients does not scale exceptionally well, as we derive no benefit from the fact that all clients are being served the same content. Therefore, the substantial amount of per-connection state associated with the shared data (i.e. retransmission buffers) is replicated unnecessarily. But referring back to one of the properties of both dispersed encodings:

Content transmitted across a fountain encoding is not client-specific.

Thus, if we were able to employ fountain encodings, retransmission buffers would become unnecessary and a *shared* encoding pool can be used for all transfers. This procedure, which can be thought of as "multicast within the operating system," enables substantial savings, especially when employed by resource-contrained end systems.

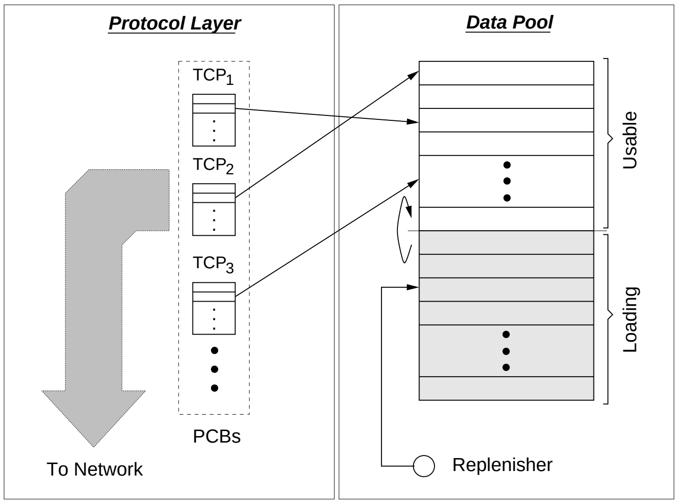


Figure 4. Resource management in Cyclone server

A depiction of some key points of the Cyclone architecture[35] we have designed to achieve this goal is presented in Figure 4. In this depiction, rather than maintain separate TCP buffers for each connection, each TCP connection can siphon packets from a shared pool of encoding packets. We have implemented in-kernel support for this flavor of shared TCP, which we call TCP-ERC, or TCP for erasure-resilient content. In TCP-ERC, the TCP congestion and flow control algorithms are unchanged, and together determine the TCP window size independently for each client. But in TCP-ERC, we have made a small server-side modification to TCP to finesse issues associated with packet retransmission, which in no way alters TCP timing semantics or impacts TCP clients. Specifically, in a situation in which TCP would normally retransmit a packet due to loss, TCP-ERC proceeds to transmit a *fresh* encoding packet from the shared pool (the original payload is not available since TCP retransmission buffers are no longer present). Since all encoding packets are equivalently likely to be of value to the client, this "spoofed" retransmission does not negatively impact client performance. The benefit is that cumbersome (and superfluous) retransmissions buffers no longer need be maintained in server memory, substantially streamlining a server's management of TCP connections.

While we have arrived at this application from the context of end system multicast, an equally valuable application of this technique is at a webserver responsible for transmitting very popular documents to a large volume of clients. When serving a static[¶] document that is very popular, such a webserver may have a vast number of concurrent TCP connections open, all of which are busy transmitting identical content. We argue that such a webserver is missing a golden opportunity to take advantage of high concurrency levels. While it would be feasible to address this opportunity with fountain encoding streams in conjunction with congestion-controlled, layered multicast,[7–9,27,31,40] the techniques we have outlined can still realize a potential gain without multicast support.

4.3. Content Authentication

A considerable risk in a scenario in which a client may either choose or be redirected to one of many information repositories is the issue of trust: the client must implicitly trust *all* of the sources that it employs to acquire the data.

[¶]By static, we mean a document that is static over timescales substantially longer than a single download.

In this final section devoted to applications, we outline a convenient authentication procedure built into the fountain encoding mechanisms that we have already defined when sparse parity-check codes are employed. This simple mechanism does not provide security in any strong sense, nor does it prevent collusion among dishonest content providers. However, it can provide an informal level of confidence that the content that has been downloaded is in fact consistent across a subset of the available repositories. Moreover, the mechanism we suggest comes at virtually no cost.

To provide this guarantee, recall that the codes we have defined have a small drawback in that they do not achieve the information theoretic lower bound on the decoding requirements, since they require decoding overhead of a few percent. We propose to provide other uses for that excess: either checking the correctness of the content, or cross-checking independently operating sources against one another. In the course of decoding, it may be the case that an encoding packet was produced from a set of data packets which have already been recovered. In standard decoding, this encoding packet is discarded, as it has no material benefit. However, when trustworthiness of the sources is an issue, this encoding packet can *validate* the result of earlier decoding steps. In the absence of bit errors which occur in the course of transmission (and which can be detected by packet checksums), a failed validation can only mean that the content disseminated by the sources are inconsistent. In this manner, exactly those packets which formerly contributed to the overhead of using these erasure codes can instead be used for content validation. While other mechanisms for authenticating content exist (i.e. acquiring a hash from a trusted source), the mechanism we sketch comes at virtually no cost, and provides some informal level of authenticity.

5. CONCLUSIONS

In this position paper, we have drawn from the diverse set of recently emerging and innovative set of methods used to facilitate Internet content delivery to motivate the need for reliable transport protocols which operate in highly dynamic environments. Applications such as end-system multicast in which the distribution topology changes over time will require congestion-controlled, eventually reliable, transport-level connections which can easily be suspended, migrated and re-routed. We argued that these services are not easily obtained by leveraging today's transport protocols.

However, we then demonstrated that by exploiting fast forward error correcting codes, we now have effective, practical mechanisms which can make content erasure-resilient. We described the properties of erasure-resilient content and argued that such encoded content is ideally suited to facilitating the flexible transport services we motivated. We outlined our ongoing work building these transport services on top of existing protocols and sketched the direct application of these techniques to a number of content delivery settings, which we expect to ultimately have a significant impact.

Acknowledgments

Many thanks to long-standing collaborators of mine whose viewpoints are reflected in this manuscript either directly or indirectly, especially Michael Luby, Michael Mitzenmacher, Stanislav Rost, Azer Bestavros, Gene Itkis and the Digital Fountain team.

REFERENCES

1. Akamai Technologies, Inc. Freeflow overview. `http://www.akamai.com/service/freeflow/html`.
2. H. Balakrishnan, H. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of ACM SIGCOMM '99*, September 1999.
3. A. Bestavros. AIDA-based real-time fault-tolerant broadcast disks. In *Proceedings of RTAS'96: The 1996 IEEE Real-Time Technology and Applications Symposium*, Boston, Massachusetts, May 1996.
4. A. Bestavros, M. Crovella, J. Liu, and D. Martin. Packet Rewriting and its Application to Scalable Web Server Architectures. In *Proceedings of ICNP'98: The 6th IEEE International Conference on Network Protocols*, Austin, TX, October 1998.
5. A. Bestavros and G. Kim. TCP Boston: A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of Infocom'97: The IEEE International Conference on Computer Communication*, Kobe, Japan, April 1997.

6. J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proceedings of IEEE INFOCOM '99*, pages 275–83, March 1999.
7. J. W. Byers, M. Luby, and M. Mitzenmacher. Fine-grained layered multicast. In *Proceedings of IEEE INFOCOM '01*, pages 275–83, April 2001.
8. J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM '98*, Vancouver, September 1998.
9. J.W. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, and W. Shaver. FLID-DL: Congestion Control for Layered Multicast. In *Proc. of the Second Int'l Workshop on Networked Group Communication (NGC)*, Palo Alto, CA, November 2000.
10. R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet switched networks. *Performance Evaluation*, 27&28:297–318, 1996.
11. R. L. Carter and M. E. Crovella. Dynamic server selection using bandwidth probing in wide area networks. In *Proceedings of Infocom '97, the Sixteenth Annual Joint Conference of the IEEE Computer and Communication Societies*, April 1997.
12. Y. Chawathe, S. McCanne, and E. Brewer. An architecture for Internet content distribution as an infrastructure service, February 2000. Unpublished manuscript.
13. Y.-H. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, Santa Clara, CA, June 2000.
14. S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.
15. Digital Fountain, Inc. Digital fountain technology overview. `http://www.digitalfountain.com/tech/presentation.html`.
16. Digital Island, Inc. Footprint content delivery. `http://www.digisle.net/services/cd`.
17. Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of INFOCOM '98*, April 1998.
18. P. Francis. Yallcast: Extending the Internet multicast architecture, September 1999. Unpublished manuscript available at `http://www.yallcast.com`.
19. P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An Architecture for a Global Host Distance Estimation Service. In *Proceedings of IEEE INFOCOM '99*, March 1999.
20. S. Gadde, J. Chase, and M. Rabinovich. Web caching and content distribution: A view from the interior. In *Proceedings of The 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.
21. K. Harfoush, A. Bestavros, and J. Byers. Robust identification of shared losses using end-to-end unicast probes. In 8^{th} *International Conference on Network Protocols (ICNP)*, Osaka, Japan, November 2000). Errata to this publication appears as BU-CS Technical Report 2001-001.
22. C. Huitema. The case for packet level FEC. In *Proc. of IFIP 5th Int'l Workshop on Protocols for High Speed Networks*, October 1996.
23. K. Johnson, J. Carr, M. Day, and M. F. Kaashoek. The measured performance of content distribution networks. In *Proceedings of The 5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May 2000.
24. M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, and B Lueckenhoff. Asynchronous layered coding: A scalable reliable multicast protocol, March 2000. Work in progress presented in the 47^{th} IETF in Adelaide, Australia.
25. M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the* 29^{th} *ACM Symposium on Theory of Computing*, April 1997.
26. F. J. Macwilliams and N. Sloane. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, 1977.
27. S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM '96*, pages 117–130, August 1996.
28. Napster, Inc. Napster tutorial. `http://www.napster.com/help/tutorial.html`.
29. Netscaler, Inc. Technology overview. `http://www.netscaler.com/technology.html`.
30. J. Nonnenmacher and E. Biersack. Asynchronous Multicast Push: AMP. In *Proceedings of Int'l Conference on Computer Communications*, November 1997.

31. J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM '97*, September 1997.
32. M. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 38:335–348, 1989.
33. L. Rizzo. Effective erasure codes for reliable computing. In *Computer Communication Review*, April 1997.
34. L. Rizzo and L. Vicisano. A reliable multicast data distribution protocol based on software FEC techniques. In *HPCS '97*, Athens, Greece, June 1997.
35. S. Rost, J. Byers, and A. Bestavros. The cyclone server architecture: Streamlining delivery of popular content. In *Proceedings of the 6th International Web Caching and Content Delivery Workshop (WCW)*, Boston, MA, June 2001.
36. D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. In *ACM SIGMETRICS '00*, Santa Clara, Ca, June 2000.
37. E. Schooler and J. Gemmell. Using multicast FEC to solve the midnight madness problem. Technical report, Microsoft Research, September 1997.
38. The Shapeshifter project. `http://www.cs.bu.edu/groups/shapeshifter`.
39. M. Stemm, S. Seshan, and R. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems (USITS '97)*, Monterey, CA, December 1997.
40. L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of IEEE INFOCOM '98*, April 1998.
41. The Yoid Project, April 2000. `http://www.aciri.org/yoid`.
42. E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *ACM/IEEE Transactions on Networking*, 8(4):455–466, 2000.