ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections

Gu-In Kwon John W. Byers
guin@cs.bu.edu byers@cs.bu.edu
Computer Science Department
Boston University
Boston, MA

Abstract— **We consider the problem of architecting a reliable content delivery system across an overlay network using TCP connections as the transport primitive. We first argue that natural designs based on store-and-forward principles that tightly couple TCP connections at intermediate end-systems impose fundamental performance limitations, such as dragging down all transfer rates in the system to the rate of the slowest receiver. In contrast, the ROMA architecture we propose incorporates the use of loosely coupled TCP connections together with fast forward error correction techniques to deliver a scalable solution that better accommodates a set of heterogeneous receivers. The methods we develop establish chains of TCP connections, whose expected performance we analyze through equation-based methods. We validate our analytical findings and evaluate the performance of our ROMA architecture using a prototype implementation via extensive Internet experimentation across the PlanetLab distributed testbed.**

I. Introduction

For high-concurrency applications ranging from live streaming to reliable delivery of popular content, a recent research trend has proposed to serve these applications using *end-system*, or application-level, multicast [10], [15], [12], [21], [3], [6]. There is ample motivation for such an approach: multicast-based delivery provides excellent scalability in terms of bandwidth consumption and server load, while an end-system approach avoids the considerable deployment hurdles associated with providing multicast functionality at the network layer. Typically, an end-system architecture constructs an *overlay* topology, comprising collections of unicast connections between end-systems, in which each connection in the overlay is mapped onto a path in the underlying physical network by IP routing. Additional transport-level functionality such as congestion control and reliability can then be realized by employing standard unicast transport protocols. This methodology has been successfully applied to develop best-effort, UDP-based methods for streaming applications, augmented with congestion control. At first glance, it seems that a similar approach can be applied to high-bandwidth applications requiring *reliable* delivery, merely by employing separate TCP connections at each application-level hop. Use of TCP is clearly desirable, as it is universally implemented, provides built-in congestion control and reliability, and does not raise any questions of fairness. However, as we demonstrate next, naively architecting the overlay in this fashion leads to substantial performance degradation.

Consider a high-bandwidth upstream TCP flow relaying content through an end-system to a low-bandwidth downstream TCP flow (as depicted in Figure 1). As the transfer progresses, the intermediate end-system is forced to buffer a growing number of packets delivered by the upstream flow, but not yet sent to the downstream flow. This unwieldy set of in-flight packets will soon exceed the finite application level buffers available for relaying data at the intermediate end-system, and then there is a problem to solve. One solution, as proposed in [28], is to use push-back flow control to rate-limit the TCP connection of the upstream sender. But it is easy to see that push-back flow control will recursively propagate all the way back to the source, and thus this devolves into a scenario in which *all* TCP connections in the delivery tree must slow to a rate comparable to that of the slowest connection in the tree. Using this method, even if there is no bottleneck on a given source-to-receiver path, that receiver will nevertheless be forced to slow to the rate of the slowest receiver. In this sense, this method has performance which closely resembles TCP-friendly single-rate multicast congestion control [29], [23]. On the other hand, it is not clear how to devise a TCP-based solution which provides an effective, multiple-rate remedy.

Our main contribution in this paper is the design and evaluation of ROMA (Reliable Overlay Multicast Architecture), a TCP-based content delivery architecture. The primary set of target applications are applications requiring reliability and high bandwidth, such as delivery of large files. ROMA enables multiple-rate reception, with individual rates that match the end-to-end available bandwidth along the path, while using small buffers at application-level relays, and the standard TCP protocol. The key to our methods is to make a departure from the straightforward approach in which each intermediate host forwards *all* received packets to the downstream hosts to achieve reliability. Instead of using this *store-and-forward* approach, we apply a *forward-when-feasible* approach, whereby each intermediary forwards only those received packets to downstream hosts that can immediately be written into the downstream TCP socket. We then handle reliability at the application layer using erasure resilient codes, also known as

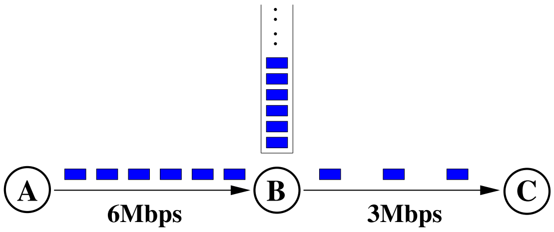

Fig. 1. Buffering is inadequate for handling rate mismatches.

FEC codes, applying well-known techniques developed for reliable (IP) multicast. The central component that enables our methods is the use of the digital fountain approach [7], a paradigm which is ideally capable of encoding n packets of original content into an unbounded set of encoding packets; and where receiving *any* n distinct encoding packets allows the complete, efficient reconstruction of the source data. Using the best codes currently available [17], a very close approximation to an idealized digital fountain can now be realized. This method has been widely used to enable receivers to recover from packet losses in the network; we apply it here to enable us to drop packets at TCP socket buffers which are full.

Our second contribution is performance evaluation of the chains of TCP connections that arise using our approach. We refer to these chains of TCP connections from the sender to end-hosts on a ROMA overlay as *loosely coupled*, since an upstream TCP connection may or may not affect the performance of downstream TCP connections, but a downstream connection *never* affects the performance of upstream connections. Applying standard equation-based methods [20], we examine the expected throughput across a chain of TCPs given per-hop RTTs and per-hop loss rates, where hop refers to a hop in the overlay. Conventional wisdom indicates that overlay multicast typically incurs a performance penalty over IP multicast, due to factors such as link stress, suboptimal routes, increased latency, and end-host packet processing. However, TCP chains offer us an opportunity to *increase* performance by finding an alternative overlay path whose narrowest hop in the chain gives better expected TCP throughput than the default IP path. This performance improvement is much in the spirit of alternative detour routes described in [25], [1]; these papers observe that IP does not provide the "best" path, measured in terms of delay or loss rates. We find that the best ROMA path is often a multi-hop path in which the minimum expected TCP throughput along any overlay hop is maximized.

Our third contribution is extensive PlanetLab [22] experimentation and insights gained from preliminary deployment of our system. We use a prototype Internet implementation that we built to validate our analysis for chains of TCP connections and to deploy our reliable multiple rate content delivery scheme. One interesting finding is that for many pairs of PlanetLab endhosts, we can often optimize the ROMA layout to provide considerably better end-to-end measured throughput using a chain of loosely coupled TCPs than we could using a single, direct TCP connection.

The remainder of this paper is organized as follows. In Section II, we discuss other overlay multicast protocols and related work on constructing alternatives to the end-to-end path that IP provides. In Section III, we further motivate our work by describing some candidate architectures and the limitations of those proposed solutions. Then, in Section IV, we present the details of the ROMA architecture, followed by an analysis of chains of TCP connections in Section V. Extensive experimental results conducted on PlanetLab validate our analytical findings and conclude our paper in Section VI.

II. Related Work in Overlay Design

A large body of work has recently been proposed to support multicast functionality at the application layer, including [3], [4], [5], [8], [9], [12], [15], [21], [27]. The design of overlay network layout has also been impacted by work initiated in the measurement community. We review and critique work in these two areas that are relevant to our proposed methods.

PRM [5], ALMI [21], Overcast [15], and RMX [8] all address the issue of reliability in distributing content to end hosts. PRM was designed for applications which do not require perfect reliability and focuses on improving the rate of data delivery while maintaining low end-to-end latencies. ALMI and Overcast employ TCP to provide reliable file transfers between any set of hosts. However, like the methods of [28], ALMI uses a back-pressure mechanism to rate-limit the sender, resulting in a single rate control. Overcast was explicitly designed with the goal of building distribution trees that maximize each node's throughput from the source. However, the technical focus of Overcast was exclusively on topology optimization, and they did not consider issues associated with the transport protocol. Other works have also focused on the problem of efficient tree construction and on the challenges of optimizing the tree layout so as to minimize network costs such as average latency; or to minimize overlay costs, such as link stress; or to perform load balancing, such as by bounding the maximum fanout [3], [4], [5], [9], [12], [27].

Results from the measurement community have also been used in designing and optimizing overlay layouts. Savage et al [25] showed that the default IP path between two hosts often is quantitatively inferior to a "detour" route taken through an intermediate end-system. Using a large set of Internet path measurements taken between geographically diverse hosts, they identified detour paths which have superior round-trip time, loss rate, or available bandwidth compared to the default path with a surprising degree of regularity (at least 30 percent of measured paths had a detour path with shorter round-trip time, and over 75 percent had a detour path with lower aggregate loss rate). These results enabled the authors to identify detour paths over which the expected TCP throughput was higher than the default path (validated with actual TCP transfers). The designers of RON [1] employed the idea of alternative paths in an overlay context, and used paths similar to detour paths both to improve performance and to route around faults in their overlay. In our work, we leverage a

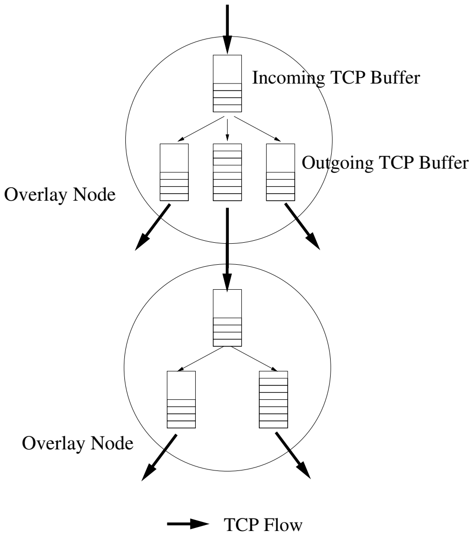

Fig. 2. Overview of TCP-based Content Delivery in an Overlay.

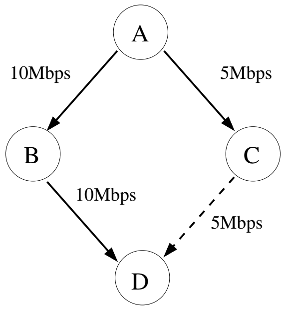

Fig. 3. Adaptive reconfiguration of the overlay.

similar measurement-driven strategy to identify the best routes in our overlay so as to optimize the layout of the set of TCP connections in our delivery tree. Our analysis goes beyond the conservative model used to estimate TCP performance common to both [25] and [1] — we find that their methods underestimate the actual throughput that a chain of TCPs is likely to see.

III. Candidate Architectures

We first develop a basic model for an overlay network and motivate our approach by describing the challenges that reliable content delivery imposes and the limitations of current TCP-based solutions. Figure 2 depicts two intermediate systems using a TCP-based overlay architecture. We refer to the node's *incoming buffer* as its TCP receive buffer for its upstream link. Similarly, the *outgoing buffers* of a node refer to its TCP send buffers for its downstream links. In the introduction, we described a simple store-and-forward method which *tightly couples* the TCP connections in the delivery tree:

- *Store-and-Forward*: For every packet arriving on an incoming buffer, buffer the packet, then forward it to all outgoing buffers.

As we saw in Figure 1, when a downstream link is slower than an upstream link, as the transfer progresses, the intermediate host is forced to buffer a growing number of packets using the store-and-forward approach, Working within the store-and-forward paradigm, there are two solutions, but both lead to performance problems of their own. We describe these alternatives next, then move beyond the store-and-forward paradigm in the next section.

A. Limited buffer space solution

If the host has finite buffer space in application layer, the push-back flow control or back-pressure mechanism [28], [21] can be used to avoid buffer overflow. The basic operation of this approach is to dequeue the packet from the incoming buffer only after it has been relayed in all of the outgoing buffers. In addition, coupling the flow control and congestion control avoids any buffer overflow in the face of different speed of downstream link. The intermediate host sends back an acknowledgment to its parent only if the arriving packet can be copied into all outgoing buffers. If there is insufficient space on any outgoing buffer, the host stalls. This results in queue buildup at the incoming buffer and subsequent decrease of the advertised window. The effects of this decreased advertised window will ultimately propagate all the way back to the source. This approach therefore results in performance which translates to single rate congestion control, where all nodes in the tree are sending packets to downstream links at approximately the speed of the slowest link.

B. Unlimited buffer space solution

Another alternative is to generalize the notion of what constitutes an application layer buffer for each downstream node. Since each intermediate node is also participating in downloading the content, it must store all received packets for its own use. When the content is large, this storage will take place on disk, instead of in a system buffer. Therefore, the application can implement store-and-forward by dequeuing each reliably received packet from the incoming TCP buffer and storing that packet on disk. Concurrently and independently, each outgoing buffer can be filled from disk, using appropriate prefetching methods to hide the substantial costs of I/O where possible. This approach enables multiple rate transmission, but with the following limitations:

- A separate application buffer for each downstream node is required.
- Complexity to support I/O accesses to fill each outgoing buffer is needed.
- The overlay cannot be adaptively reconfigured.

The first two limitations are clear, but the third (and arguably the most serious), requires more careful discussion.

A robust overlay network should have the ability to adaptively reconfigure itself when congestion or failures of intermediate nodes occur. Therefore, a host must be able to switch its parent to maximize its performance. But in many situations, this design does not facilitate such a transition. Consider the case of host D in the example in Figure 3, in which B, C and D are performing a reliable download from A. The average reception rate for host D is 10Mbps, that of host C is 5Mbps. Due to the different transfer rates, the data received by D will be a prefix of the content that is twice the length of the prefix received by C at any point in time. Now suppose that an hour into the transfer, the B to D link becomes congested, degrading performance to 1Mbps. Host D would now prefer to use the route through C, but since C is thirty minutes behind (in terms of received data), this alternative route is useless to D. (Note that this problem is specific to multi-rate reliable transfers; it does not apply to the single rate back-pressure solution or to live streaming applications). A similar synchronization problem also arises in asynchronous transfers when hosts initiate the downloads at different times.

This significant limitation seems to be difficult to find a workaround for, but in fact, the use of erasure-resilient codes in the ROMA architecture that we describe next provides a very satisfactory solution that does not encounter any of the limitations presented in this section.

IV. Reliable Overlay Multicast Architecture

We now describe ROMA, a simple reliable multi-rate overlay multicast architecture for reliable content delivery. The two central novelties leveraged in our design are the use of erasure resilient codes, as we describe in more detail in Section IV-A, together with the use of a *forward-when-feasible* paradigm, rather than the standard store-and-forward paradigm:

- *Forward-when-feasible*: For every packet arriving on an incoming socket, for each outgoing buffer, determine whether it can immediately accept the newly arrived packet. Copy the packet to those buffers which can accept it, then deliver the packet to the application. (Those outgoing buffers which are full will never receive or transmit a copy of this packet).

Together with the encoding methods we employ, an intermediate host using the forward-when-feasible approach does not have to store all received data in an application-level buffer and as a result, managing buffer overflow is not a problem. In practice, we use one additional level of indirection to implement the forward-when-feasible paradigm, a point we touch upon in the following more detailed overview of the ROMA architecture:

- Each node runs TCP between the upstream and downstream link nodes and itself.
- While there are interested participants, the sender transmits a continuous erasure-resilient encoding of the content of size n along its downstream links.
- Each host dequeues the arrival packet from the incoming TCP buffer and copies the packet to a small application layer buffer managed as a circular queue. If the buffer is full, then the host overwrites the buffer in a circular fashion.
- Each intermediate host copies data to all outgoing buffers that have available space.
- Each host completes its reception after receiving a set of encoding packets of size approximately $1.03n$ (explanation of this small 3% overhead to follow).
- Upon completing the reception of the original content, the node may leave the ROMA group by closing its TCP connections. In the event it elects to continue servicing downstream connections, it may do so either by continuing to relay encoded content generated by the source, or by generating encoding symbols of its own from the full content, and closing its upstream connection.

In the next section, we provide more details of erasure-resilient codes, and the node architecture in the ROMA system. We also describe how to transmit the encoded data on a byte-stream transport protocol like TCP.

A. Erasure-resilient Codes

We now review the basics of erasure-resilient codes[1], a close relative of error-correcting codes: While error-correcting codes provide resilience to bit errors, erasure-resilient codes provide resilience to packet-level losses. We use the following terminology. The content being sent by the encoder is a sequence of symbols $\{x_1, \ldots, x_\ell\}$, where each x_i is called an *input* symbol. An encoder produces a sequence of *encoding symbols* $y_1, y_2, \ldots$ from the set of input symbols. For our application, we will set the input and encoding symbol size both to be equal to a packet payload. For the family of erasure-resilient codes we use, parity-check codes, each encoding symbol is simply the bitwise XOR of a specific subset of the input symbols. A decoder attempts to recover the original content from the encoding symbols. For a given symbol, we refer to the number of input symbols used to produce the symbol as its *degree*, i.e. $y_3 = x_3 \oplus x_4$ has degree 2. Using the methods described in [18], the time to produce an encoding symbol from a set of input symbols is proportional to the degree of the encoding symbol, while decoding from a sequence of symbols takes time proportional to the total degree of the symbols in the sequence. Encoding and decoding times are a function of the *average* degree; when the average degree is constant, we say the code is sparse. Well-designed sparse parity check codes require recovery of a few percent (less than 5%) of symbols beyond ℓ, the minimum needed for decoding. The *decoding overhead* of a code is defined to be ϵ_d if $(1 + \epsilon_d)\ell$ encoding symbols are needed on average to recover the original content. (There is also a small amount of overhead for the space needed in each packet to identify which input symbols were combined, which is typically represented by a 64-bit random seed.)

Provably good degree distributions for sparse parity check codes were first developed and analyzed in [18]. However, these codes are fixed-rate, meaning that only a pre-determined number of encoding symbols are generated, for example only $c\ell$, where c is a small constant > 1. In our application, this can lead to inefficiencies as the origin server will eventually be forced to retransmit symbols. Newer codes, called *rateless* codes, avoid this limitation and allow unbounded numbers of encoding symbols to be generated on demand. Two examples of rateless codes, along with further discussion of the merits

[1] Often referred to as forward error-correcting (FEC) codes.

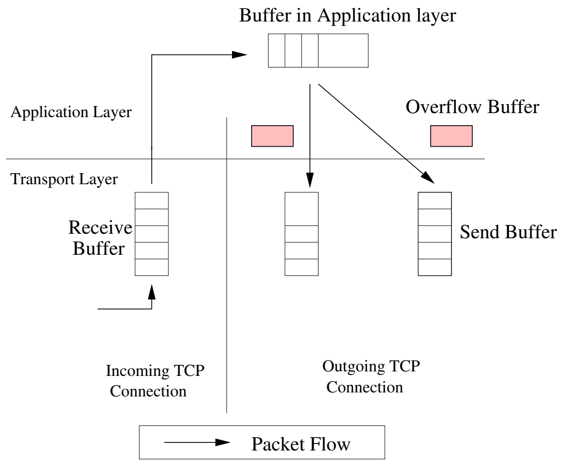

Fig. 4. Overlay Node Implementation.

of ratelessness, may be found in [17], [19]. Both of these codes also have have strong probabilistic decoding guarantees, along with low decoding overheads and average degrees. In our experiments, we simulate use of LT codes [17], and assume a fixed decoding overhead of 3%.

The main benefit of erasure codes in our architecture is that it makes it possible to design the control mechanisms independently of reliability. Intuitively, using an erasure-resilient encoding, packets can flow through ROMA intermediaries (all with small buffers) toward a set of destinations, and can be dropped whenever they reach a bottleneck (in the form of a full buffer). With this intuition, one can see that this provides for a multiple rate solution. The use of codes also enables a number of additional benefits, including the ability to tolerate asynchronous joins, the ability to adaptively reconfigure the topology, and the ability to speed up downloads with collaborative peer-to-peer transfers as described in [6].

B. Transmitting Encoding Symbols with TCP

One nuance of using codes is that the encoding symbols must be treated atomically, i.e. receipt of a fraction of an encoding symbol is not useful. For this reason, some care must be taken to send encoded symbols as logical segments, or datagrams, across TCP. The main difficulty is to ensure that each encoding symbol is written in its entirety into the TCP socket. But, using only application-level, system-independent calls, it is not simple to determine whether a given packet will fit into the TCP send buffer without performing the write explicitly. Our solution (depicted in Figure 4) is to maintain a one-packet overflow buffer per socket to store those bytes which could not be successfully written into the socket. Before performing a subsequent write to the socket, the contents of the overflow buffer are written first. Using this strategy, encoding symbols are always written contiguously and in their entirety to the buffer. For more details of the implementation, please refer to the full version of this paper [16].

C. Intermediate and Sender node Architecture

In our overlay multicast architecture, we assume that each host is also participating in downloading the content and therefore must read data from the upstream socket into an application layer buffer before writing into disk. In our implementation, we use an application buffer of 1MB to overcome the limitation of small default socket buffer sizes on many systems. Most implementations have an upper limit for the sizes of the socket send buffer and the upper limit is only 256 KB in many systems. Use of the application buffer for additional buffering at intermediate hosts avoids known pitfalls associated with bursty packet arrivals when high bandwidth connections with large window sizes use small socket buffers [26].

As described earlier, each intermediate host dequeues arriving packets and copies them to an application buffer. If the buffer is full, then the host writes the packets in the buffer into disk and overwrites the buffer in a circular fashion. The downstream socket buffer is filled from this application buffer, with each downstream socket making sure not to wrap around the tail end of the circular queue.

The sender architecture is virtually identical to that of the intermediate node except that the application buffer is filled with fresh encoding symbols (typically precomputed and stored on disk) at a speed that is sufficient to satisfy the fastest downstream connection. As with intermediate nodes, the sender also maintains a one-packet overflow buffer for each downstream node to avoid sending a fraction of an encoded packet. The functionality of the sender is as follows:

- Files are encoded into encoding symbols that are stored on disk prior to their delivery.
- A single, fixed-length memory buffer is used for all receivers.
- If the fastest receiver exhausts all data in the buffer, the buffer is filled with new data from the disk.

The sender's functionality is similar to the Cyclone web-server architecture [24], which is optimized for delivery of content in situations in which a group of clients is concurrently downloading a small set of large, popular files. In particular, the sender can be optimized to employ the sliding cache buffer mechanism in the Cyclone design to minimize the waiting time to fill the buffer from the disk.

V. CHAINS OF TCP CONNECTIONS

We now provide a simple analysis of the chains of TCP connections that arise in the design of our system.

A. Modeling Chains of TCP Connections

For simplicity, we begin with the simple case of an overlay host with just one upstream and one downstream TCP connection, depicted as host B in Figure 5. In this example, B is just relaying received packets to its one downstream host, i.e. it sends TCP ACKs for received packets back to host A and transmits data segments to host C. We assume that the overlay host has sufficient memory space in the application layer to store received packets in the case of a slower downstream link.

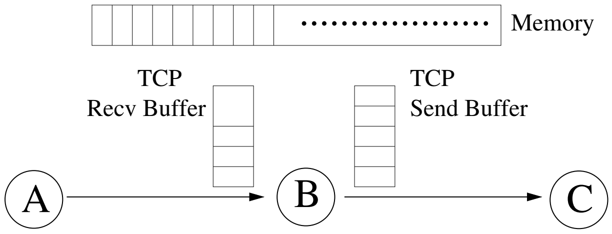

Fig. 5. Basic Model for Chains of TCP Connections

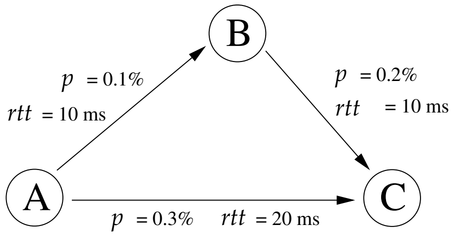

Fig. 6. Chains of TCP Connections

(In the ROMA design, this assumption is realized provided the application layer buffer is sufficiently large that it never drains). In this simple model, we assume that the intermediate host B dequeues the packet from its TCP receive buffer fast enough to prevent flow control algorithms from impacting its upstream transmission rate. Finally, we assume that the relevant network conditions (loss rate, RTT) along the chain of connections are stationary over time. These assumptions (unlimited buffer and fast dequeuing) make this chain of TCP connections *loosely coupled*, which we define as follows:

Definition 1: A chain of TCP connections is *loosely coupled* if an upstream TCP connection may or may not affect the performance of a downstream TCP connection, but a downstream connection never affects the performance of an upstream connection.

If the downstream transfer rate is slower than the upstream transfer rate, then the application layer buffer will grow without bound. In this case, the downstream TCP will behave like a TCP driven by an application that always has data to send, and thus the performance of the downstream TCP is independent of the upstream TCP.

Alternatively, consider the case in which the downstream transfer rate is larger than the upstream transfer rate. In this case, host B will periodically drain the application level buffer filled by the upstream connection when sending packets to C, and thus the downstream TCP connection has to wait for incoming packets to send. Therefore, in this case, the downstream throughput to C is limited to that of the upstream rate into B.

To develop formulas for the expected TCP throughput as a function of the per-hop loss rates and RTTs, we employ the following equation derived in [20]:

$$\mathrm{T} = \frac{s}{rtt\left(\sqrt{\frac{2p}{3}} + (12\sqrt{\frac{3p}{8}})p(1+32p^2)\right)}. \tag{1}$$

This provides an estimate of the expected throughput T of a TCP connection in bytes/sec as a function of the packet size s, the measured round-trip time rtt, and the steady state loss event rate p. For simplicity in the remainder of the exposition, we use the following simpler formula as a shorthand for the equation above:

$$T \simeq \frac{\sqrt{1.5}}{rtt\sqrt{p}}.$$

To extend this result to a chain of loosely coupled TCP connections, our observations above demonstrate that a given hop in the chain either has local network conditions that limit its rate to a value below that of the upstream connections or is already limited by the rate of the upstream connections. Also recall that by the definition of loosely coupled connections, events downstream have no bearing on upstream throughput. Letting rtt_i and p_i respectively denote the round-trip time and loss rate experienced by a TCP connection traversing overlay hop i, the expected throughput to a ROMA host below hop j is:

$$\mathrm{T} \simeq min_{i<j}\left(\frac{\sqrt{1.5}}{rtt_i\sqrt{p_i}}, \frac{\sqrt{1.5}}{rtt_j\sqrt{p_j}}\right). \tag{2}$$

In an overlay setting, one factor which is not captured by this simple equation is the impact of link stress, which occurs when distinct overlay hops j and k share underlying physical links. Link stress further implies that measured values of p_j and p_k are not independent. We show the effect of link stress in our experimental results, but do not incorporate this effect into our simple model.

B. Examples and a Comparison with Other Models

To develop some intuition, consider the example in Figure 6, in which the propagation delay and the loss rate on each link are labeled. Using the direct route from A to C, and using the simple version of the throughput equation, the expected throughput of direct unicast from A is about 9.0Mbps. In contrast, the throughput from A to C via B using a chain of two TCP connections is about 22.2Mbps, which is also the expected throughput along the direct B to C connection. In other words, the loss and delay on the hop from A to B have no measurable impact on the performance along the detour path from A to B to C.

It is worth noting that in previous work, a different, and more conservative formula was used to estimate the throughput of a chain of TCP connections. Following the methodology used in [1], [25], the aggregate RTT is defined as the sum of rtt_i along the path and the aggregate loss rate is defined as $1 - \prod(1-p_i)$ (assuming uncorrelated losses). Instantiating these values into the simple version of the throughput equation gives:

$$T \simeq \frac{\sqrt{1.5}}{\sum rtt_i\sqrt{1-\prod(1-p_i)}}. \tag{3}$$

Receiver	Sender Host								
	BU	UCLA	UTK	Arizona	GT	Duke	Cornell	Berkeley	UW
BU	64.1Mbps	12.6Mbps	19.0Mbps	16.0Mbps	27.0Mbps	39.0Mbps	21.3Mbps	6.9Mbps	8.7Mbps
UCLA	17.9Mbps	88.0Mbps	18.8Mbps	20.3Mbps	20.5Mbps	14.5Mbps	14.3Mbps	38.6Mbps	39Mbps
UTK	47.0Mbps	18.8Mbps	98.0Mbps	21.0Mbps	46.3Mbps	39Mbps	29.1Mbps	18.7Mbps	12.8Mbps
Arizona	21.3Mbps	21.7Mbps	21.0Mbps	92.0Mbps	22.2Mbps	15.4Mbps	19.0Mbps	21.7Mbps	19.3Mbps
GT	53.9Mbps	18.8Mbps	74.0Mbps	22.8Mbps	92.0Mbps	45.0Mbps	34.5Mbps	18.8Mbps	17.1Mbps
Duke	40.9Mbps	10.2Mbps	26.8Mbps	10.3Mbps	31.8Mbps	92.0Mbps	15.4Mbps	9.5Mbps	9.2Mbps
Cornell	33.1Mbps	14.5Mbps	30.7Mbps	19Mbps	28.7Mbps	15.4Mbps	98.0Mbps	16.3Mbps	16.5Mbps
Berkeley	10.1Mbps	30.3Mbps	12.6Mbps	17.1Mbps	13.4Mbps	9.4Mbps	11.4Mbps	98.1Mbps	38.4Mbps

TABLE I
END-TO-END MEASURED THROUGHPUT

Plugging the values from the example in Figure 6 into this equation gives expected throughput across the detour route of 9.0 Mbps, or no different than the direct route. Indeed, it is easy to see that in general, this aggregation model treats a "split" TCP connection no differently than its aggregate. In practice, our experimental results demonstrate that this method of aggregation underestimates throughput, while the model embodied by Equation (2) provides a much more accurate estimate.

C. Discussion

Conventional wisdom indicates that overlay multicast typically incurs a performance penalty over IP multicast, due to factors such as link stress, stretch factor, and end host packet precessing. However as we have seen in the example in Figure 6, TCP chains also offer us an opportunity to *increase* performance compared to direct unicast. This performance improvement comes from finding an alternative overlay path whose narrowest hop in the chain (as perceived by TCP) is wider than the default path used by IP. In general, an improvement in throughput can be realized whenever one identifies a decomposition of a long TCP control loop into several smaller loops in which each member of the chain has expected throughput greater than that of the original loop. As we have argued, this gain applies even when the aggregate loss rate and the aggregate RTT across this chain are larger than the values of the original long loop. Breaking long TCP control loops in the context of overlay networks has a similar effect as *split TCP* [2], which shortens the TCP feedback loop and separates lossy components. Ideas from split TCP are commonly used in satellite communication and in various terrestrial wireless contexts to improve TCP performance.

At this point, we feel compelled to note that TCP is not actually a mandatory component of the ROMA architecture. In principle, any TCP-friendly protocol with forward error correcting codes can be used to achieve the same performance benefits and our analysis still applies as long as it follows Equation 1. In practice, however, we prefer to use TCP because it is already ubiquitous and well understood, and because we feel that alternative protocols would merely be imitating the TCP behavior. (The potential benefits of UDP over TCP, e.g. for streaming or other real-time applications, do not seem to apply to reliable transfer applications.)

In the next section, we show that there exist ample opportunities to exploit this advantage in constructing overlay topologies so as to maximize the total throughput to participant hosts across the Internet.

VI. EXPERIMENTS

We have implemented ROMA and conducted experiments on the PlanetLab distributed testbed [22]. PlanetLab consisted of 160 machines hosted by 65 sites in June 2003; we ran experiments on a subset of roughly 30 sites. All PlanetLab machines run a Linux-based operating system and they all meet certain hardware requirements (see details in [22]). Most of the hosts in PlanetLab are university hosts and those hosts in the U.S. are connected through Abilene, which has high capacity and is highly available. Therefore, while the experiments we conducted on PlanetLab are not intended to be representative of typical Internet performance, they nevertheless enable us to validate our models and performance of our architecture across a substantial set of Internet paths.

For our experiments, we considered 1 GB transfers using a packet size of 1 KB. As a baseline, we conducted end-to-end transfers of this size between pairs of hosts using TCP. We report on a representative subset of these baseline measurements across Abilene in Table I, where each entry represents the average measured throughput of ten independent measurements from source nodes to destination nodes conducted in June 2003. In addition, entries on the diagonal report measurements between two PlanetLab nodes at the same university. We will use the name of university as the host name throughout this section for simplicity. One important observation is the substantial bandwidth asymmetry we see in our measurements. In some cases, there are significant constant factor differences: for example, between BU and UTK, the path asymmetry is 47 Mbps vs. 19 Mbps.

This table is intended primarily to give a flavor of the data

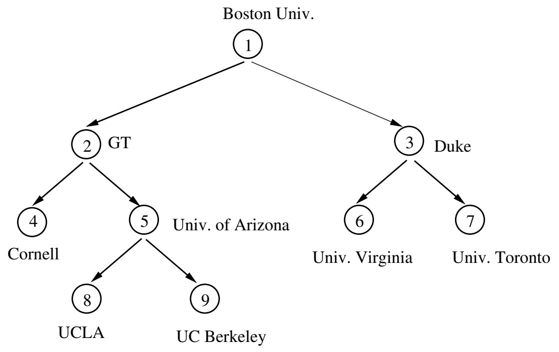

Fig. 7. Multirate Reliable Multicast with B.U. as the sender

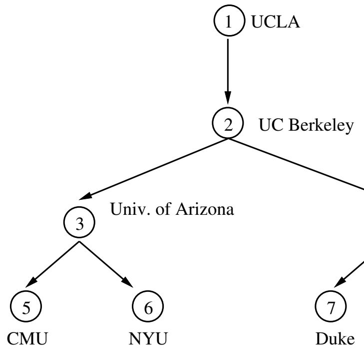

Fig. 9. Multirate Reliable Multicast with UCLA as the sender

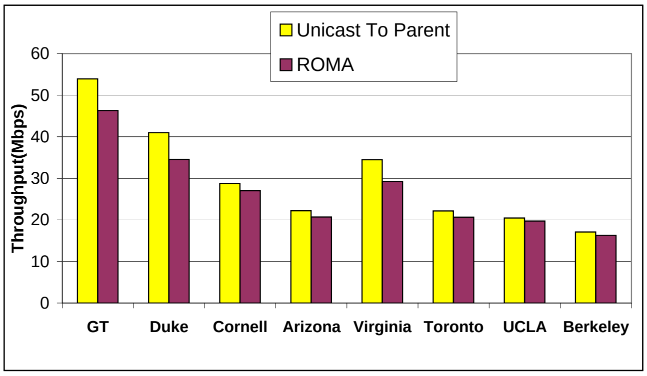

Fig. 8. Host throughput with B.U. as the sender

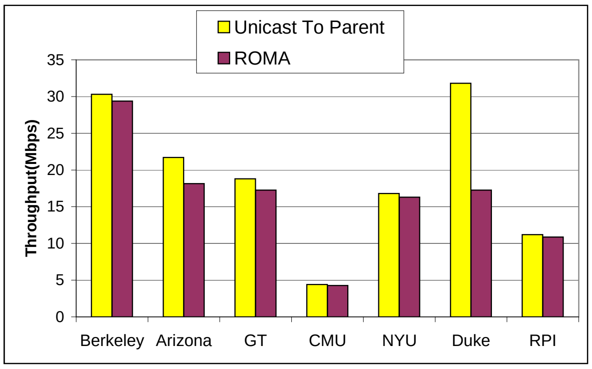

Fig. 10. Host throughput with UCLA as the sender

rates we are working with, and does not capture the variability of throughput measurements over time (which we found to be relatively small on the lightly loaded Abilene backbone), nor does it provide a highly accurate measure of available bandwidth. One could certainly adapt our methods to address dynamic changes in available bandwidth, for example, by using non-intrusive methods to monitor available bandwidth such as pathload [14] or PTR [13]. In the following sections, we use values from the table as input to our algorithms to construct overlay multicast trees.

Next, we describe additional details involving experimentation with ROMA. First, in some cases, we identified PlanetLab hosts whose throughput appeared to be constrained by their local network configuration (perhaps due to router capability, link capacity, or rate limiting). For example, several hosts were unable to concurrently send and receive at sustained rates above 10Mbps. We did not include measurements from these rate-limited hosts in Table I. Also, we disallowed these hosts from being intermediate hosts in our experiments, but did allow them to be leaf nodes.

At each of the hosts where we deployed ROMA, we established a 1 MB application buffer (as in Figure 4), primarily to facilitate copying between upstream and downstream sockets. Finally, as described in section IV-A, the erasure resilient codes we propose in ROMA induce decoding overhead; the codes we simulate have decoding overhead of approximately 3%. We include this decoding overhead into the ROMA throughput calculations whenever we compare to direct unicast throughput (which would not use codes).

A. Multiple Rate Reliable Multicast

Our first experiment uses the topology depicted in Figure 7 to validate that a single slow link does not impact the performance either at upstream nodes, or at nodes in other regions of the tree. Figure 8 compares the average throughput at each host when a 1GB transfer is performed using two different methods defined as follows:

- *ROMA*: An overlay multicast tree is established to all hosts, and the throughput is measured at each point in the multicast tree.
- *Unicast To Parent*: For each host, we measure the throughput of a TCP transfer directly from its parent to the host itself. (This corresponds to a single entry in Table I.)

The values reported are the average measurements across ten trials.

In this topology, every upstream link offers better unicast throughput than all of its downstream links, thus the throughput on any path from the sender to a receiver decreases monotonically. Here, the set of downstream links fanning out from every intermediate host also have different characteristics. Figure 8 shows that the ROMA throughputs measured by each host are diverse, but all are similar to the throughput of a single unicast connection to its parent node, as we desire. Clearly,

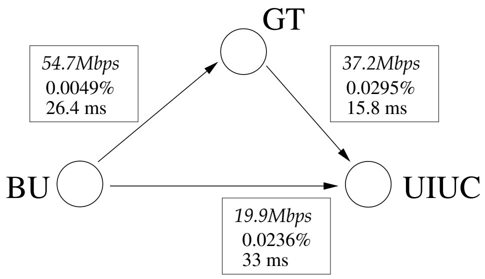

Fig. 11. Throughput Improvement on Chains of TCP

slow links do not degrade the performance of unrelated peer hosts or ancestors in the tree.

In Figure 8, note also that some hosts have slightly decreased throughput using ROMA as compared to a direct unicast connection to their parents (and beyond that of 3% decoding overhead). For example, consider the intermediaries at GT and Duke. The unicast throughput of connections to GT and Duke from BU were 53.9Mbps and 40.9Mbps respectively while the multicast throughput of GT and Duke while running in the ROMA experiment were 47.8Mbps and 35.6Mbps respectively. This is primarily because the ROMA experiment is running under the disadvantage of delivering data across all tree edges simultaneously. The throughput degradation comes from the effect of *link stress*, which is defined as the number of identical copies of a packet carried by a physical link in an overlay [10]. In our example, downstream and upstream links from a single node often share some physical links. When these shared links are a bottleneck, the contention at these resources negatively impact the performance of those overlay connections crossing the link.

These measurement results also provide agreement with our analytical argument that the expected throughput is the minimum of the throughputs along the path (and no worse). Indeed, the throughput of each host in this topology is determined only by the network conditions along the overlay hop to its parent host.

In Figure 10, we report on a similar experiment, showing the throughput at each host in the topology depicted in Figure 9. As before, we see the effect of stress on links at Univ. of Arizona and GT. Another interesting case is the throughput at Duke. Although Duke's upstream link from Georgia Tech has high throughput (31.8 Mbps), the measured throughput at Georgia Tech using ROMA is much lower (17.8 Mbps) and therefore the measured ROMA throughput to Duke is limited to this lower value. This experiment (and many similar experiments not presented due to space limitations) provide confirmation that the TCP throughput of the overlay host is bounded above by the minimum throughput across the upstream links.

B. Throughput Improvement from Chains of TCP

In the following two sections, we report on experiments in which use of ROMA can actually improve the throughput as compared to direct unicast (and even when the aggregate RTT and loss rate increase). Consider the simple example in Figure 11 derived from our Internet experiments. The throughput and RTT from the pairwise unicast measurements between the three hosts are as labeled. Since we were unable to directly derive the loss rate of the TCP connection without root access, we used equation (1) to compute the approximate loss rate based on the measured throughput and the measured RTT. To doublecheck our measurements, we also concurrently ran TFRC [11] on the same path and measured the average loss rate from TFRC. Details of our methodology and comparisons of measured loss rate and computed loss rate are in the full version of this paper [16].

Even though both the aggregate loss rate (0.0344%) and the aggregate round-trip time (42.2 ms) increase, the throughput to UIUC via GT along the detour path is consistently larger than that achieved by direct unicast from BU. Using ROMA, the measured throughput to UIUC was 37.2 Mbps, which is the minimum of the throughput across the overlay links, as our model predicts. This throughput improvement comes from the benefit of employing chains of TCP connections.

C. Maximizing Overall Throughput

The earlier analysis and the experiments in the previous sections point to a natural method for optimizing the layout of an application level multicast tree using ROMA: construct the single-source "widest path" tree, i.e. the tree that maximizes the minimum per-hop available bandwidth to every destination. In this section, we sketch a simple algorithm for building this widest path tree and construct the tree for a PlanetLab overlay rooted at the University of Washington (depicted in Figure 12) using end-to-end measurements from an extended version of Table I.

The algorithm to construct the widest path tree is a simple variant of Dijkstra's algorithm, which is typically used to construct single-source shortest path trees. In a standard invocation of Dijkstra's, links have associated weights representing propagation delay, and the algorithm repeatedly and greedily selects the unvisited node closest to the source, where proximity is measured by the sum of the weights on the path. To construct the widest path tree, links have associated weights representing available bandwidth (as per the entries in Table I), and the algorithm repeatedly and greedily selects the unvisited node with the widest path from the source, where path width is measured by the *minimum* of the weights on the path. The short proof that this greedy algorithm constructs the widest path tree follows the same reasoning as the shortest path tree argument.

The multicast tree depicted in Figure 12 is a widest path tree rooted at UW that we constructed using this algorithm from a set of measurements extending Table I. We note that the widest path tree is not typically unique, since decisions below an unavoidable bottleneck link are immaterial. To build the first level of the tree, we used the fact that the maximum available bandwidth from UW to other hosts is about 39 Mbps (to UCLA is 39 Mbps, to Berkeley is 38.6 Mbps). At the second level of the tree, we used the fact that the maximum

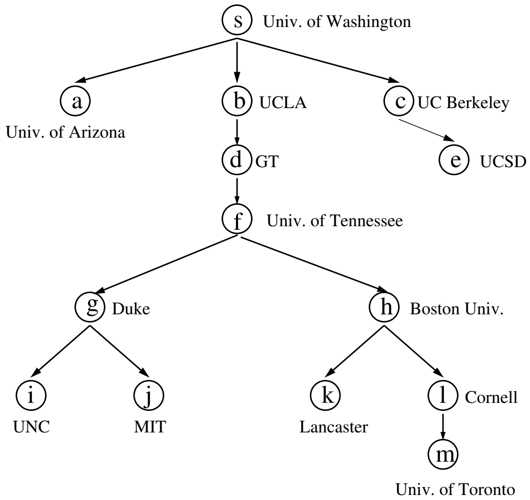

Fig. 12. Single Source Widest-Path Tree rooted at UW

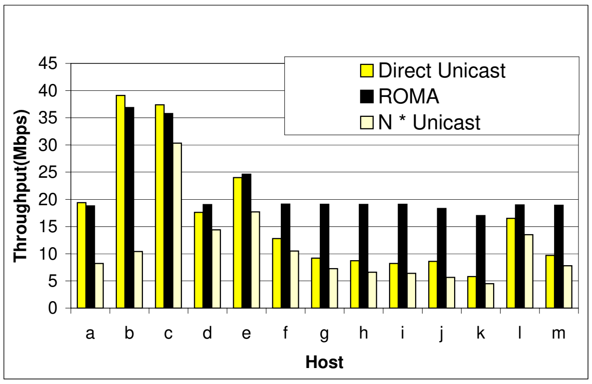

Fig. 13. Host throughput with UW as the sender

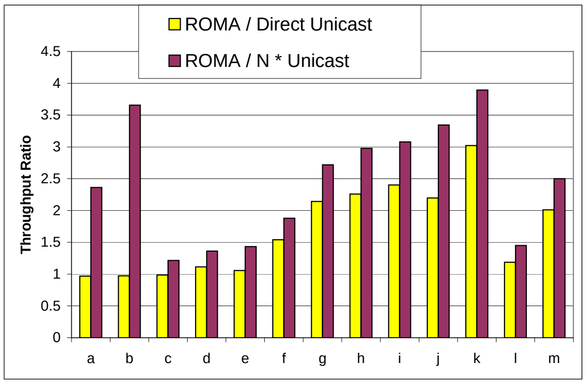

Fig. 14. Throughput Advantage From ROMA

available bandwidth from UCLA or Berkeley to other nodes is about 21 Mbps, which is higher than any other available bandwidth from UW. Below these upper levels, we broke most ties arbitrarily, since the available bandwidth between all pairs of hosts not on the west coast were mostly higher than 21 Mbps on Abilene.

Using this same topology, we compare the throughput of each host in each of the following three scenarios.

- *ROMA*: An overlay multicast tree is established to all hosts, and the throughput is measured at each point in the multicast tree.
- *Direct Unicast*: The throughput is measured when the content is transferred across a single unicast connection to the individual host.
- *N * Unicast*: The throughput is measured when all N hosts simultaneously download the content via separate unicast connections from the sender.

Comparing ROMA against direct unicast jointly demonstrates the performance advantages derived by split TCP connections, and the disadvantages of using an overlay infrastructure. The comparison of ROMA against `N *` unicast demonstrates the benefit of multicast by reducing the transmissions of many copies of the same data on outgoing links from UW.

In Figure 13 we depict the head-to-head comparison of our three methods. The figure shows that even though the overlay multicast generates some link stress, it is still far superior to `N *` unicast at all nodes. We also see that in many cases, the throughput of ROMA is better than direct unicast case and that this throughput advantage of ROMA comes from finding the widest path to destinations. We also see the effect of link stress, especially at nodes with considerable fanout, which results in ROMA having slightly worse performance than direct unicast.

Figure 14 depict the relative performance of pairs of these three methods as ratios. A ratio of 1.0 indicates no difference in throughput, while a ratio of 2.0 indicates a two-fold speedup. The results show that ROMA provides excellent performance compared with the other unicast methods, and provides significantly improved performance over a single end-to-end TCP connection with surprising regularity.

VII. Conclusion

This paper presented ROMA, a new architecture for reliable distribution of large content across an overlay network using TCP. ROMA enables multiple-rate reception, with individual rates that match the end-to-end available bandwidth along the path, while using a minimal amount of resources at the application layer. A key component that our method employs is the use of erasure-resilient codes to provide reliability. The degree of freedom that the use of codes provides enables us to loosen the tight coupling of TCP connections that is needed in other designs to provide reliability, but also limits performance. The use of a digital fountain approach in our architecture also provides us with many additional benefits: small buffers, the ability to adaptively reconfigure the topology, and the ability to speed up downloads with collaborative peer-to-peer transfers.

Another contribution of our work is the analysis of chains of loosely coupled TCP connections that are established using our approach. We provide a simple model for the expected throughput across a chain of TCPs given per-hop RTTs and per-hop loss rates, along with validation using Internet experimentation. Our analysis and experimental results show that TCP chains offer an opportunity to increase performance by finding an alternative overlay path that is "wider" (as far as TCP is concerned) than the default path provided by IP. This observation also guides the construction of multicast trees that ROMA uses.

VIII. Acknowledgments

We are grateful to the numerous individuals who participated in the conception, design, build-out and support of the PlanetLab infrastructure. Without such a freely available wide-area distributed testbed, we could not have conducted the experiments reported on in this paper. Our work was supported in part by NSF grants ANI-9986397, ANI-0093296 and ANI-0205294.

References

[1] D. Andersen, H. Balakrishman, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. of ACM Symposium on Operation Systems Principles*, 2001.

[2] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. *15th International Conference on Distributed Computing Systems*, 1995.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of ACM SIGCOMM*, 2002.

[4] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-Time Applications. In *Proc. of IEEE INFOCOM*, 2003.

[5] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient Multicast using Overlays. In *Proc. of ACM Sigmetrics*, 2003.

[6] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proc. of ACM SIGCOMM*, Aug. 2002.

[7] J. Byers, M. Luby, and M. Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. *IEEE J-SAC Special Issue on Network Support for Multicast Communication*, 20(8):1528–1540, October 2002. A preliminary version appeared in ACM SIGCOMM '98.

[8] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *Proc. of IEEE INFOCOM*, 2000.

[9] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proc. of ACM SIGCOMM*, 2001.

[10] Y. Chu, S. G. Rao, and H. Zhang. A Case For End System Multicast. In *Proc. of ACM SIGMETRICS*, 2000.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM*, 2000.

[12] P. Francis. Yoid: Extending the Multicast Internet Architecture, 1999. White paper at `http://www.aciri.org/yoid/`.

[13] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE J-SAC Special Issue in Internet and WWW Measurement, Mapping, and Modeling*, 21(6), August 2003.

[14] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. of ACM SIGCOMM*, 2002.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proc. of USENIX Symp. on Operation Systems Design and Implementation*, 2000.

[16] G. Kwon and J. Byers. ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections. Technical Report BU-CS-TR-2003-015, Boston University, 2003.

[17] M. Luby. LT codes. In *Proc. of The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[18] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Efficient erasure correction codes. *IEEE Transactions on Information Theory*, 47(2), 2001.

[19] P. Maymounkov and D. Mazières. Rateless Codes and Big Downloads. In *Proc. of Int'l Workshop on Peer-to-peer Systems (IPTPS)*, 2003.

[20] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, April 2000.

[21] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, 2001.

[22] PlanetLab. `https://www.planet-lab.org/`.

[23] L. Rizzo. pgmcc: A TCP-friendly single-rate multicast congestion control scheme. In *Proc. of ACM SIGCOMM*, 2000.

[24] S. Rost, J. Byers, and A. Bestavros. The Cyclone Server Architecture: Streamlining Delivery of Popular Content. *Computer Communications Journal*, 25(4):403–412, March 2002.

[25] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *Proc. of ACM SIGCOMM*, 1999.

[26] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. In *Proc. of ACM SIGCOMM*, 1998.

[27] D. Tran, K. Hua, and T. Do. ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. In *Proc. of IEEE INFOCOM*, 2003.

[28] G. Urvoy-Keller and E. Biersack. A Congestion Control Model for Multicast Overlay Networks and its Performance. In *Proc. of Int'l Workshop on Networked Group Communication (NGC)*, 2002.

[29] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *Proc. of ACM SIGCOMM*, 2001.