Peer-to-Peer Error Recovery for Hybrid Satellite-Terrestrial Networks

Eric Weigle[†‡], Matti Hiltunen[‡], Rick Schlichting[‡], Vinay Vaishampayan[‡], and Andrew A. Chien[†*]
eweigle@ucsd.edu, {hiltunen, rick, vinay}@research.att.com, achien@ucsd.edu

[†]Computer Science and Engineering
and Center for Networked Systems
University of California, San Diego

[‡]AT&T Shannon Research Laboratory
Florham Park, NJ

Abstract

Media companies and other organizations with large amounts of digital content require transfer of extremely large files in a short time from a single source to a collection of geographically dispersed destinations. Due to the high cost of terrestrial networks of sufficient scope and bandwidth, satellite networks are the most common means for performing such transfers. However, satellite file transfer relies on expensive error correction based on a combination of forward error correction and whole-file retransmission.

This paper presents a hybrid solution that combines the advantages of satellite and terrestrial networks to provide cost-effective reliable file transfer. Specifically, we propose a new peer-to-peer (P2P) scheme that exploits fast terrestrial networks and the availability of multiple receivers to recover from high loss rates (5% or more) in near real-time (latency <400ms). This solution is efficient, robust under variable packet loss and connectivity, user tunable, scales to hundreds of nodes, and doubles bandwidth compared to existing approaches. The system has been validated via extensive simulations using a terrestrial network based on the AT&T common backbone core network.

Keywords: hybrid network, peer-to-peer, broadcast, content distribution

1. Introduction

The demand for timely transfer of extremely large data files between widely-dispersed geographic locations is increasing rapidly. Broadcast and media companies are looking to transfer HDTV files to affiliates for end-user broadcast or for collaborative editing, computational scientists need to distribute large amounts of scientific data across grids, and enterprises want faster and more complete offsite backups for disaster recovery.

The PlanetLab Grand Challenge with the Public Broadcasting Service (PBS) [1, 5] provides just one concrete example. In this scenario, the PBS requirements include the need to transfer up to 450 GB per day from PBS headquarters to approximately 180 affiliates scattered across North America. These affiliates have varying degrees of wired Internet connectivity; and in fact characteristics of this underlying network will dictate the feasible solutions for this type of hybrid broadcast problem.

Approaches using only a terrestrial network may be feasible when all hosts are connected end-to-end with high-speed links, but such a network is costly– many enterprise settings have only slow links connecting to their ISP's backbone network (in the range of T1 links at 1.5 Mbps; too slow to stream at the 20-40Mbps or faster rate required). Approaches using satellite transponders to broadcast data at higher speeds (e.g., via 40 Mbps/transponder hardware [5]) avoid terrestrial bottlenecks but suffer from packet corruption and loss.

To ensure reliable transfers, terrestrial networks can retransmit individual lost blocks. Satellites must rely on forward-error-correction (FEC) that can require utilizing 50% of the channel for redundant information, and fall back on whole file retransmission if errors are uncorrectable via FEC. The latter techniques consume unnecessary bandwidth and negatively impact the end-to-end transfer time.

This paper describes a new peer-to-peer (P2P) broadcast/recovery scheme for reliably transferring large files from a single source node to a number of geographically dispersed destination nodes. It takes advantage of *both* satellite and terrestrial networks. We divide our mechanism into two parts (broadcast and recovery) and optimize for each medium separately– exploiting their unique features while avoiding their problems.

*This work is part of the collaboration between AT&T Labs-Research and the UCSD Center for Networked Systems, and was done primarily while Eric Weigle was an intern at AT&T Labs-Research. Weigle and Chien were also supported in part by the National Science Foundation under awards NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from the UCSD Center for Networked Systems, BigBangwidth, and Fujitsu is also gratefully acknowledged.

Unsurprisingly, the broadcast part uses the satellite network for a one-to-many broadcast, while the recovery part detects corruption/loss and uses the terrestrial network for many-to-many transmission to recover. Part one may experience errors or omissions due to weather, insufficient power, crashes, etc, and error detection in part two is trivial by use of CRCs or cryptographic hashes.

Part two entails peer-to-peer error recovery in which receivers exchange data. However, the original sender ("source" node) may also be able to re-broadcast corrupted data, which provides a different optimization path than terrestrial-only networks and hence leads to different solutions than current tools. The two parts are pipelined and overlap for most of a transmission.

Our approach has the desirable properties that it: (1) corrects for large error rates/loss (5% or more), with arbitrary distribution; (2) scales in the file size (to gigabytes) and number of nodes (to 100s); (3) provides low block latency (<400ms); (4) is efficient (low network, memory overhead, globally minimizes cost function); (5) is fair (different peers share equally); (6) and it is user tunable (user may control peering behavior).

We validate claims 1-6 using "real-world" performance metrics on realistic networks, including scenarios in which the terrestrial network is based on the AT&T common backbone core network [3]. These tests show efficient, reliable, fair, low-latency transport of large files even under high loss.

This paper is structured as follows. First, we discuss the problem and target environment in more detail. Then we present our solution, including the algorithms involved and their implementation. We then evaluate the implementation in depth using the ns-2 simulator, before concluding with a brief discussion and related work.

2. The Problem

Here are the defining characteristics that specify the hybrid broadcast problem, as found in our target applications:

1. A single *source node*— the node with data to send in the beginning— is connected to a large number of *destination nodes* via a satellite link as well as a terrestrial network.

2. High data rate— on the order of 100-450 GB/day.

3. Continuous data transmission— little or no *inter*-transmission "free" time to recover from problems.

4. Relatively low *intra*-transmission delay tolerance— data must be received on the order of a few seconds after it is initially sent.

5. Loss varies between uplink and downlinks— loss will tend to be higher on the downlinks than the uplink.

One motivating application is the distribution of a high-resolution digital television signal to affiliate TV stations. These data files are large (4-8GB), must be played out by the destination nodes with low latency (frames displayed within a few seconds of receipt, inter-frame jitter < 40ms), and uplink loss is smaller than downlink loss (by as much as a factor of 10). In this case, the received uplink power is larger than the received downlink power at an antenna due to higher transmit power and antenna gain on the uplink.

It is worth noting that error recovery from uplink losses and downlink losses can be viewed as distinct problems. Uplink losses mean no node receives that block— in other words, the set of blocks lost on the uplink are only available from the source node and must be obtained via the terrestrial link. Downlink losses differ in that most peers will be able to provide such a missing block. Historically each has been solved using a separate mechanism. We show that one simple, unified solution can solve both with excellent performance.

2.1. Target Environment

Our target environment has high core bandwidth (10-40Gbps), moderate satellite bandwidth (20-40Mbps), and low access link bandwidth (1-5Mbps). As we will see, the exact values are less important than the ratios between them: the ratio of satellite bandwidth to access link bandwidth and the ratio of their costs define the effectiveness of this approach. We assume all nodes are covered by the satellite and well connected via the terrestrial network; if not, the hybrid broadcast is either equivalent to the terrestrial-only problem or simply impossible.

High bandwidth is realistic for the ISP core networks and upcoming user-controlled optical networks, and essentially allows us to ignore locality of hosts (terrestrial delay is insignificant compared to satellite delay). We use the AT&T core network for our models. Extensive studies show that in the AT&T Core and similar networks, there is minimal loss due to congestion [10].

Satellite links are the primary source of loss in this network. This hybrid broadcast mechanism works on either the simple "reflector" type or more powerful store-and-forward satellites. In the former case, loss may occur on either the uplink or downlink transmission, while in the latter most losses will occur on the downlink. We assume burst errors occurring uniformly and independently across links using a standard Gilbert-Elliot model per link. Satellite broadcast packets arrive in order, allowing immediate detection of such loss for most cases.

One common configuration for affiliates is to have T1 links (1.5 Mbps) to a fast core network (gigabits per sec-

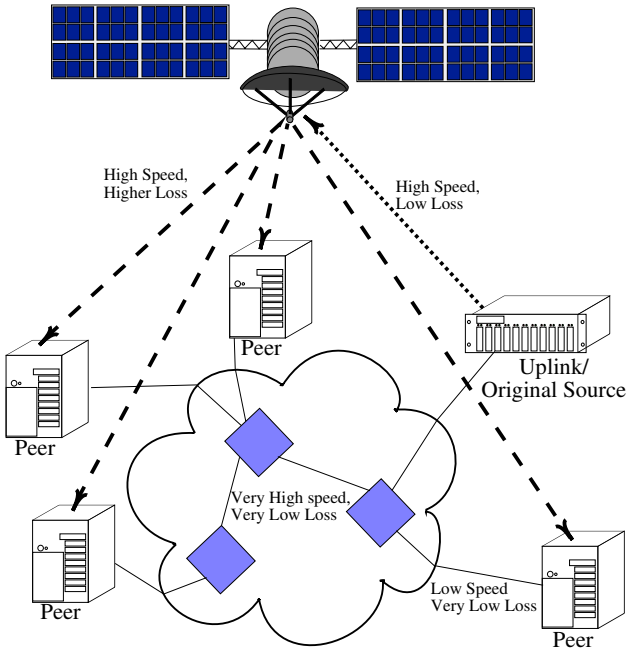

Figure 1. Satellite/Terrestrial Configuration

ond), and 40 Mbps satellite transponders. Of that 40 Mb, 17 Mb is for forward error correction and 23 Mb is for user data. Error rates on satellite links will vary between 0.05% to 5% depending on FEC and weather. In such an environment, the 15:1 ratio between satellite and terrestrial link speeds is sufficiently dramatic that one must use the satellite for any large data broadcast.

2.2. Design Decisions

Given the prior discussion, our design resolves to the questions:

1. *metadata* (state) management— location of blocks within a file, host load, etc, and
2. *data* (transmission) management— how to schedule transfers, when, from which node, and to which node.

Solutions to the metadata management problem vary between fully centralized metadata servers to fully distributed mechanisms based on e.g., distributed hash tables. Solutions to the data transmission problem range from centralized scheduling to distributed transfers based on some heuristic, or simply randomized transfers.

3. Transmission/Recovery Mechanism

In this section, we explain our solution to the hybrid broadcast problem. First we give a high level design overview, providing the answers to the prior questions, then cover the algorithms involved. In all discussion broadcast is done with minimal link layer FEC (no additional reliability) – one source uplink transmits blocks of data to the satellite which then broadcasts it to all nodes. Recovery is initiated as soon as nodes detect losses.

3.1. Design

The error recovery mechanism is built using three types of nodes: a *scheduler* which collects and maintains metadata information, an *uplink* which is the source node with the original copy of the data, and *peers* which are the destination nodes to receive the data.

First, the scheduler is a single node, ideally located in the high-bandwidth core of the network. The uplink may also act as the scheduler (as in Figure 1). Nodes send negative acknowledgments to the scheduler, which replies with peers that can provide the missing blocks. Periodic timeouts handle terrestrial losses by causing the node to send a cumulative ack and re-request information on missing blocks.

A single scheduler node has the benefits of allowing decisions based on global knowledge, ensuring fairness, and implementation simplicity. The obvious disadvantages include a single point of failure and limited scalability. This scaling limit depends on the total loss in the network: for high loss to a few hundred nodes, for low loss to a few thousand, for zero loss there is obviously no limit.

We can construct a high availability scheduler with standard fault tolerance and clustering techniques. Similarly, distributing the scheduler can be done via use of a distributed hash table such as Chord [15] or Pastry [8] to maintain its state. While this is more scalable and has a certain elegance, it induces higher overhead and latency, makes it harder to optimize globally, and produces a more complex system. In short, it is overkill for our motivating application which uses below 200 nodes: the current design more than suffices.

Second, there is a single uplink– the original source of date in the system. This node broadcasts its data over the satellite and concurrently acts as a peer node in the recovery algorithm, one which has completed its download.

Third, the peers can receive data in three ways: over the satellite link from a source broadcast (1), or over the terrestrial link from the source node (2) or another peer (3). Nodes should receive most blocks from the satellite. For the remainder we wish to, as much as possible, retrieve blocks from peers rather than the source node. We use a simple message-based mechanism that globally balances load to schedule retransmits of lost blocks among peers.

Another approach is to use block level Forward Error Correction (FEC) such as erasure codes on the satellite channel; this is infeasible due to high-latency encoding or

decoding inherent in e.g. Tornado or Luby codes [6, 7] or other more computationally intensive algorithms. Another drawback is that FEC consumes satellite bandwidth– reducing user-available bandwidth. For example, if 50% of the transmitted data is redundant, it takes twice as long to transmit a file. Other approaches which simply download recovery blocks from the source node directly are also infeasible; this does not even scale to tens of nodes.

Lastly, we use a message-based transport protocol. Use of a reliable transport such as TCP is inappropriate for high-latency environments with low delay requirements. First, TCP handshaking takes multiple RTTs, which is unacceptable. Second, TCP is in-order and 100% reliable, so when low-value packets are lost, high value packets may be blocked waiting for it to be retransmitted. Third, data ages quickly; if a packet is lost, it is better to get a fresh copy than wait for retransmission of an old, useless packet.

3.2. Error Recovery Algorithm

The error recovery algorithm addresses both the transfer of metadata (step 1, 2) and the transfer of data (step 3, 4). Our system uses a "soft" in-order mechanism (Other algorithms are discussed in Section 6.1) that maximizes fairness, and minimizes both block latency and global termination time, limited only by metadata accuracy:

1. Node detects data block loss via hole in the broadcast stream and sends NACK to scheduler
2. Scheduler replies with best (lowest cost) peer that can provide the missing block, with ties broken randomly
3. Node directly requests block from that peer
4. Peer replies with that block
5. Periodically, nodes provide metadata (load information, cumulative acknowledgements), and perform retransmission as required.

Data block loss can be detected immediately via a hole in the incoming packet stream. Handling loss of control messages is more complicated, so we focus on minimizing their loss. One way to do so is via prioritizing packets with more value (those further along in the algorithm, above) and we also use a token-bucket mechanism to avoid overloading links.

Again, we use a packet-based rather than reliable bytestream approach because the value of metadata drops sharply with time; on a timeout we do not want to use stale information to request a data packet, because that node may currently be experiencing high load (which could in fact be the cause of the original loss). Instead, we start over by contacting the scheduler.

"Soft" in-order refers to two things: the fact that these control messages may be lost, and that there is no synchronization. Due to loss or latency in the network, recovery data packets may arrive out-of-order. Without total control over the network we can provide only probabilistic guarantees on this behavior, but with sufficiently low loss these guarantees are quite strong.

More rigorously, let

N	= Number of terrestrial nodes
BS	= Block size
$Pr(L)$	= Probability of loss on a satellite link
BW_s	= Bandwidth of satellite bottleneck
BW_t	= Bandwidth of terrestrial bottleneck
T_{send}	= Time that a block is sent (absolute)
D_{sat}	= Max Delay on satellite (uplink↔nodes)
D_{ter}	= Max Delay between terrestrial nodes
D_{s_xmit}	= Max Delay to send a satellite block (BS/BW_s)
D_{t_xmit}	= Max Delay to send a terrestrial block (BS/BW_t)
D_{tick}	= Delay between clock ticks (timeouts)

Using this notation we can calculate aspects of the system behavior: maximal block delay, termination time, and so forth.

With uniform independent losses, the chance that a node immediately gets any particular packet is simply $(1 - Pr(L))^2$ since packets may be lost on the uplink or downlink. This is the probability of "ideal" reception at time $T_{ideal} = T_{send} + D_{s\_xmit} + D_{sat}$. If only one copy of the packet is lost, it will be detected in at worst one timeout, then metadata requested from the server, then the packet requested from a peer and received at time at most $T_{ideal} + D_{tick} + 4 \cdot D_{ter} + D_{t\_xmit}$. When multiple copies of a block are lost, at most $k = \lfloor (D_{tick} * BW_t)/BS \rfloor$ losses can be recovered per clock tick by each node that *has* received the block. Then, after another (at most) D_{ter} those nodes can provide the block to others.

In the worst case, (1) a block is lost on the uplink so only the source has it and (2) the source is on the terrestrial bottleneck. In this case, after one clock tick (plus network delay) $k+1$ nodes will have the block, after two ticks approximately k^2 have it, and so on. At worst, the last node will receive the block at time $T_{ideal} + (D_{tick} + D_{ter}) \cdot \lceil log_k N \rceil + 4 \cdot D_{ter} + D_{t\_xmit}$. This is a desirable bound as it grows very slowly, but it is subject to a few constraints we discuss below.

3.3. Algorithm Constraints

The first constraint is that D_{tick} is large compared to D_{t_xmit} and D_{ter}. If not, the recovery algorithm breaks down – to make progress, replies to requests must arrive before the next timeout and its subsequent re-request.

The second is that the satellite loss rate is "streaming-recoverable", that is, the terrestrial links are fast enough to fix the losses on the satellite links within a few RTTs

of when they occur. This is only true when $(1-(1-Pr(L))^2) \cdot BW_s < BW_t$ or equivalently when $Pr(L) < 1-\sqrt{1-(BW_t/BW_s)}$. This is why it is the ratio between satellite and terrestrial speeds, rather than their absolute values, is most important. In practice, the maximum loss rate recoverable at streaming rates is slightly lower due primarily to congestion loss on terrestrial links and stale data at the scheduler.

To make this concrete, when losses are streaming-recoverable, the maximum latency to receive a block in our simulations is less than half a second. If the satellite loss is *not* streaming-recoverable, then losses will accrue and delay to replace a lost packet will grow without bound. Eventually, if the satellite transmission completes and there is no subsequent transmission, the errors can be recovered at the speed of the terrestrial network.

3.4. Peer Selection Algorithm

The peer selection algorithm is the core of the recovery mechanism; it selects from whom a given block should be recovered. The scheduler selects the best peer as follows:

> Select a random peer from
> the minimal cost peers from
> all peers which have the block.

This is, in effect, a specialized database operation which can be expressed compactly using relational algebra: $\sigma_{\text{random}}(\sigma_{\text{minimal cost}}(\sigma_{\text{has block k}}(\{\text{all nodes}\})))$.

The first selection is easy; any reasonable pseudo-random number generator suffices. Similarly, the third selection is easy when information is centralized or other infrastructure exists to maintain it.

The second selection's difficulty depends on a user's definition of "cost." Cost can be any formula, but in these simulations it is just a measure of load: lower load, lower cost. We treat the uplink as having infinite load. Initially it has the only complete copy of the data, and will be the most loaded node so should be avoided whenever possible.

We have claimed this is an optimal algorithm. It is in fact a trivial instance of the job scheduling or bin packing problem where all jobs are the same size (transferring one block). In such a case, minimizing global cost resolves to globally balancing cost, which is exactly what this algorithm would do if it had perfectly accurate information on host load and experienced loss as it occurs. In practice, this information will be slightly stale at any given time due to network delay.

The delay in metadata propagation means the scheduler may err by up to the number of requests to arrive in the time it takes a packet to traverse the network[1]. The expected difference is only a few blocks (below D_{tick}/D_{t_xmit}) and will be self-correcting over time: the percent difference from optimal falls as the transmission size increases. Our empirical results show this behavior. Randomization avoids overloading individual hosts during the update period.

3.5. Scheduling and Inaccurate Data

Note how load status information is used in the core of the scheduling algorithm (the peer selection step) in Section 3.4. To make good decisions, it is critical that this information be both correct and up-to-date. Experimental versions of our system using only "pessimistic" load information performed poorly. This information is more accurate, but also slightly stale – it is from peers indicating the actual nodes which *did* provide each data block, rather than "optimistic" information, which is the node the scheduler has decided *should* provide each data block. While the current optimistic scheme may overestimate load on certain nodes, it avoids overloading any one in the short term.

Although not currently used for scheduling, this pessimistic load checking mechanism is still be required to detect underperforming nodes (which may require administrator attention) and to avoid undesirable user behavior. "Leeching" is the most prevalent, in which some users download but do not upload blocks. To detect this in more malicious environments with colluding users, the scheduler needs to cross-reference between the information it has provided and load information returned.

A naïve checking implementation requires $O(n^2)$ space to store a matrix for $\langle$ sender, receiver, count$\rangle$ tuples, but we can achieve similar results in less space. One $O(n)$ method is to use a hash table keyed on node ID, whose value is incremented each time the scheduler sends an information packet and decremented each time a load packet is received. A single large positive value means that node is not providing data (is leeching), a a single large negative value means one or more peers are providing false load information, and a high variance means some nodes are consistently underperforming or colluding in their reporting (groups can be seen by displaying a sorted histogram of values). When these are detected it is trivial to track all references to the offending node IDs and resolve the undesirable behavior.

3.6. Design Implications

This section explains a couple of the results that follow immediately from the design. First, one benefit of the concept of maximal streaming-recoverable loss rate is that when loss is dramatically lower, we can exploit that fact to send less data over the satellite link. These would then

[1] up to $D_{t\_xmit} + 4 * D_{ter}$: D_{ter} to return metadata, D_{ter} to request the block, D_{t_xmit} to send it, D_{ter} to traverse the network, and D_{ter} to return an ACK to the scheduler.

be treated as losses, and recovered on the terrestrial network. This makes sense when the cost structure is such that cost(satellite)$> \Sigma_N$cost(terrestrial$_i$), since for every satellite broadcast we must send N terrestrial blocks. This will reduce the amount of satellite data to be sent by a few percent when loss is low.

For satellites whose level of redundancy can be adjusted, we should therefore set the FEC level to the minimum tolerable by the terrestrial error correction. This value is known from the analysis in the prior section, and with some knowledge of the satellite error rate we can set the level of redundancy appropriately.

Second, one particularly useful cost function for scheduling in practice is as follows: a simple (linear) weighted sum of load, link speed, link expense (i.e. cost in dollars), and distance between peers. Link speed is required when links differ significantly in speed; however in such a case global termination will always be determined by the slowest node in the network given uniform loss. If losses were more common on faster nodes, we could still perform well. Link expense ties cost to real-world price. Distance allows us to minimize latency; in particular for soft real time systems we can increase the cost of nodes farther away as deadlines approach.

This approach can also help engineer the minimum cost hybrid networks that satisfy user requirements. Specifically, when we know the parameters to the cost function and the expected/experienced error rates at different receivers (e.g., due to climate or being on the boundary of satellite coverage), we can determine the terrestrial bandwidth each node will require. For example, some nodes might only need DSL lines, while others may require T-3. However, this must be globally optimized since we must ensure receivers have sufficient bandwidth the peer with others– techniques such as linear programming can be used with the cost functions to determine an appropriate global cost minimum.

4. Evaluation

The system was implemented using the `ns-2` network simulator. Due to the expense of satellite transponders, it is quite difficult to perform this type of large scale exploratory work in any other way. This section discusses the design and validation of the simulations, the loss model we use, and the performance metrics with which we test the system.

4.1. Simulation Design and Validation

Our implementation includes the algorithms discussed in Section 3 as well as a fair amount of infrastructure including code for peer connection handling/message passing, block tracking, message passing, and so forth[2]. It comprises about 1600 lines of C++ code adding to the simulator itself, 1000 lines of TCL code to set up and test various configurations, and some shell programming to assist in analyzing results.

One of the most important and yet unexpected requirement of the simulation code is the addition of the ability to generate randomized noise. This comes into play when selecting among equal cost hosts (randomly select one), when setting timers (randomly add a few percent jitter), and when dropping packets while overloaded (randomly select among low-priority packets). Without it, some hosts are deterministically overloaded; with it performance improves dramatically (by a factor of 2 or more in some cases). This is usually not a problem in physical systems, which by nature include some nondeterminism.

The correctness of the implementation was validated in three ways beyond the standard `ns` validation suite. These measures make us confident that the simulations accurately capture the salient features of this problem. First, by comparing results to the theoretical predictions from the earlier analysis. Second, by running against a set of small contrived configurations and manually reproducing all output. Third, by visualizing results in the network animation tool `nam` and looking for unexpected behavior. Over the course of development, we also repeated prior tests to verify consistency in results.

We use `ns` to perform tests varying different aspects of the algorithm, error rates, error distribution (correlation across hosts), link speeds, link costs, network topology, block size, and transfer sizes. Some of these parameters turn out to be insignificant, and thus we present only the interesting data. Our results validate the six claims made in Section 1.

Most tests are performed on a small relatively simple topology meant to provide the equivalent of microbenchmarks. This topology connects access links to a single backbone router, creating a terrestrial star topology. The satellite links are 23Mbps and terrestrial links are 1.5Mbps T1 links.

Other tests are on a larger, realistic topology using the AT&T core US network [3] (covering a significant portion of the core Internet in the United States) and the information from the PBS/Planetlab Grand Challenge [5] correlated with station/location information from the PBS web site [14]. This is a concrete, realistic topology and shows that our approach provides an alternative solution to many issues in the grand challenge problem.

[2]Ironically, the lack of some desired features in the simulator means that that much of the work to create a "real-world" implementation is complete.

4.2. Loss Modeling

The discussion in Section 3.3 assumed loss in terms of packets, which we use in our simulations. This is a terrestrial-centric view where loss is due to congestion. Loss in a satellite network is usually due to bit errors, which is somewhat more difficult to model accurately with `ns`. In particular, it has the additional problem that loss rate becomes a function of block size. In a hybrid network we must account for both features.

In fact, loss rate, block size, and overheads of packet headers, connection set-up and tear down (in TCP based systems), and memory for book-keeping are all related and represent a particular design point amongst a set of tradeoffs. The larger the block size, the higher the chance of corruption but the lower the overhead. Terrestrial-only peer-to-peer systems use blocks of size 64KB to 8MB– which reduces book keeping and TCP connection startup/shutdown overhead, but would be infeasible with high bit error rate. We use a block size of 1KB, which is particularly convenient because blocks and packets are roughly interchangeable.

Figure 2 captures a few of these trade-offs. It shows the "Effective Goodput" as a function of block size; the probability that a packet will be received uncorrupted multiplied by the proportion of useful data in the packet. That is, for a small block the probability it will be received uncorrupted is high, but most of the data sent is overhead, while for a very large block the overhead is low but the probability of corruption is high. This is under the assumption that the block size of the block-code does not scale with the size of the data block. When the block length of the code is allowed to scale with the data block (using concatenated codes for example) it will be possible to use larger data blocks than indicated in Figure 2.

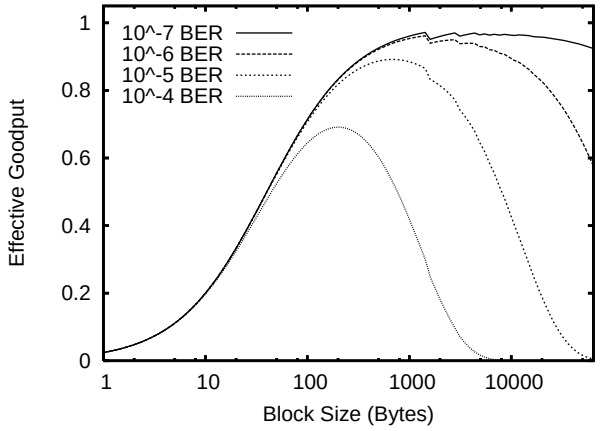

Figure 2. Block Reception and Overhead

With uniform losses the probability of receiving a block correctly is proportional to $(1-\text{loss rate})^{\text{block size}}$, and we use a maximal packet size of 1500 bytes for this graph. This explains the drop at multiples of 1500 as blocks fragment into multiple packets. This graph shows peaks at the ideal block size for various loss rates— note how our 1024 byte block size is ideal for bit error-rates on the order of 10^{-5}.

Lastly, in practice, bit errors tend to occur in non-uniform bursts. This means fewer packets will actually be corrupted, and hence this graph slightly underestimates the ideal packet size. This is a reasonable tradeoff to build a robust system.

4.3. Performance Metrics

Our algorithm builds in an explicit performance metric for peer selection (Section 3.4), which is unsurprisingly related to our overall performance model. We are primarily interested in:

- global transfer termination time– simulated seconds; when the last peer receives the last block required
- fairness– Jain's metric; how evenly distributed cost is among peers
- streaming latency– percent completion (in aggregate, total blocks) as a function of time; a measure of how good the system is at streaming
- raw bandwidth– megabits/second; a simple measure of system speed
- link utilization– percent of aggregate or individual links; a simple measure of efficiency.

The only metric worth further discussion is fairness, using Jain's measure from [11]. In this case, x is the cost incurred on the uplink by counting the number of blocks they have provided, or equivalently the amount of work done on behalf of other peers which is of no direct benefit to a given node:

$$\text{fairness} = \frac{(\sum x_i)^2}{(n \cdot \sum x_i^2)}$$

We want this value to be close to 1 (completely fair) to both provide incentive for users to participate and because fairness is a good measure of global performance: spreading load evenly leads to the fastest termination. We exclude the initial source node from this calculation, as it will by definition be transmitting an order of magnitude more data.

Together, our system attempts to maximize broadcast link utilization, minimize access link utilization, minimize latency to receive each block, and maximize global aggregate goodput in the system.

5. Results

We present the results for our performance experiments for the micro-benchmark topology and the AT&T core network topology. First, we demonstrate the system performance and efficiency as a function of the satellite link loss. We also demonstrate the robustness of the protocol under correlated losses due to loss on the satellite uplink. Second, we demonstrate the scalability of the approach both in terms of file size and number of receiver nodes. Third, we demonstrate that the approach meets the goal of low latency by measuring the end-to-end latency for each block. Fourth we demonstrate the efficiency of the approach in utilizing the access links, both in the case where the transmission is streaming-recoverable and when it is not. Finally, we conclude the tests with results showing that our metadata update mechanism is very efficient and the protocol as a whole is very fair.

5.1. Loss Recovery

The first question concerns how much loss we can recover using this system. Figure 3 shows a graph of loss rate over the satellite links and completion time for a 100MB broadcast to 10 nodes.

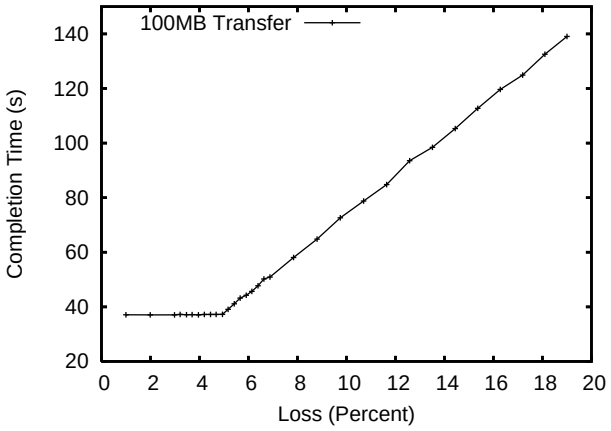

Figure 3. Loss Rate vs. Completion Time

As we expected there is a sharp knee showing the point at which streaming recovery is no longer possible. According to our calculations in Section 3.2 the maximal theoretical streaming-recoverable loss rate is 6.5%, but we see the actual maximum at 5%. This is about 75% efficiency, with the difference primarily due to in-order request of blocks; those lost on the uplink exist on only one node and require multiple iterations to propagate throughout the network while other lost blocks may be available to download.

The above graph shows the aggregate loss when loss rates are equal on uplink and downlink. That is, a unidirectional loss rate of $k\%$ leads to a loss rate above of $1-(1-k/100)^2$. There is no reason to expect the loss rates to equal; in fact, uplink losses will tend to be lower as described above.

Consider Figure 4, which for a fixed loss (10%) varies the proportion of the loss incurred on the uplink versus the downlinks. In all cases, the total number of blocks to be recovered is the same; all that varies is the correlation of the errors. This is equivalent to varying the location of the original blocks from which to be recovered or the proportion of losses in the "must globally retransmit" case.

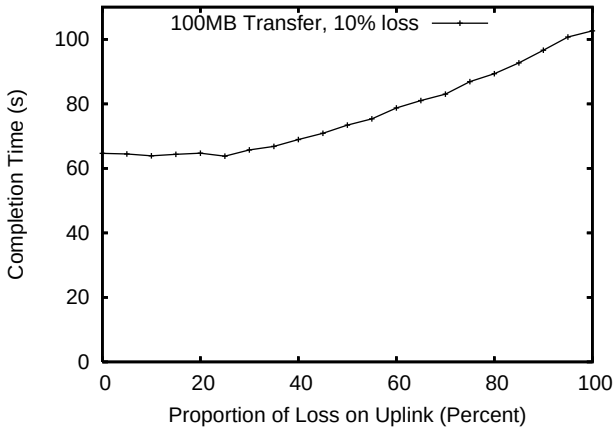

Figure 4. Effect of Uplink/Downlink Loss

As more losses occur on the uplink (loss correlation among peers grows), more must be globally rebroadcast on the terrestrial network. This leads to overloading of the source node, and requires multiple iterations of the peer-to-peer recovery mechanism to get the block to all nodes.

Put another way, the set of losses which occur on the uplink must be rebroadcast on the terrestrial network, as in the PlanetLab grand challenge. The difference in performance between none and all of the loss occurring on the uplink is the improvement in performance by exploiting the hybrid network. These differences would be magnified ten-fold using only the terrestrial network (10% satellite loss $\rightarrow$ 100% satellite loss).

The deeper cause of this performance difference is more subtle. With a uniform loss distribution on the downlink, as the number of nodes grows the probability that no node has a given block (the probability that it must be retrieved from the source) grows proportional to $loss\ rate^n$. On the other hand, the number of losses and the capacity to recover losses grows proportional to n. This means that as the number of nodes grows, we have an excellent chance of being able to recover using a peer-to-peer mechanism. Thus we

should minimize uplink losses if at all possible for good performance. In the limit (no uplink losses), the best approach to recover a lost block is to (1) randomly select a peer (2) weight the selection away from local nodes (since losses will be locally correlated due to weather), and (3) add a simple load-balancing mechanism to ensure global fairness.

5.2. Scalability

The second question concerns scalability; in terms of file sizes and in terms of system size (number of nodes). Ideally, completion time should be linear in the file size. Similarly, as the number of nodes grows the completion time should stay constant (below the streaming point) and grow slowly above it (as nodes become overloaded).

To test the former, we broadcast files of size 1MB to 1GB under various amounts of loss (above and below the knee in the prior graph), and present the results normalized to the percent of ideal time (for no loss on the satellite link).

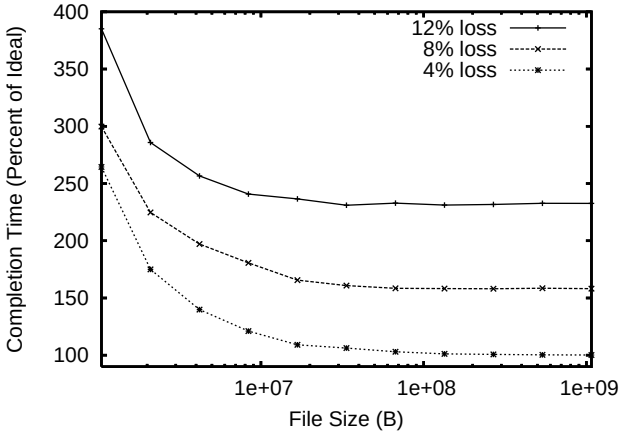

Figure 5. Completion Time vs. File Size

For 1-8MB transfers network latency and other overhead dominates, since the ideal is on the order of 1-2 seconds. Above that size, overhead is insignificant. For low loss we can effectively meet the ideal transfer time and for higher loss we differ by a constant factor.

Larger files were not simulated due to time constraints; even were the system to not scale beyond 1GB such files could be transferred in 1GB chunks to give this desirable performance behavior. Similarly, since results are linear above 8MB, use of 100MB transfers for other results we present is a reasonable choice.

To test scalability in terms of nodes, we test a 100MB broadcast to varied number of nodes. Figure 6 shows completion time for increasing numbers of nodes under the same loss conditions as the prior figure.

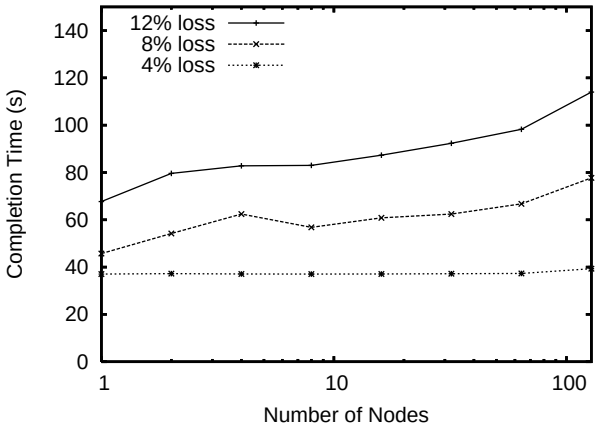

Figure 6. Completion Time vs Node Count

The first thing to note is that the curves grow very slowly. The streaming-recoverable curve's increase is negligible. The others grow approximately with the log of the number of nodes, as expected– this is due to block propagation requiring an additional iteration with each doubling of the number of nodes.

There is a small drop in the curve around 8 nodes. This is the point at which peer nodes begin to be useful; for 2-4 nodes the peer-to-peer transfer is ineffective as there are too few peers to adequately share blocks.

As this graph shows, the system is insensitive to the number of nodes, so our use of 10 nodes in the results we present is a reasonable choice. Larger numbers of nodes were not simulated due to time constraints. We reiterate that it was never our goal in this proof-of-concept to scale beyond the order of hundreds of nodes, and within this range we perform quite well. For larger numbers of nodes, one would have to move to a slightly different design; giving up slightly on performance to gain the extra scalability.

5.3. Intra-Transfer Performance

The next logical question is how the system is performing within a transfer; how the different parts of the transfer progress and whether we are meeting our latency goals.

Figure 7 shows the blocks received via the satellite and terrestrial networks over time, for 10 nodes under high-loss (10%) conditions. Data is aggregated over all nodes, and thus the graph is very smooth.

The ideal curve transfers all data over the satellite link with no loss. Our actual performance is effectively a linear combination of the degraded satellite signal and terrestrial recovery transmissions. For low-loss cases, both terminate at same time. For higher loss cases such as this, we

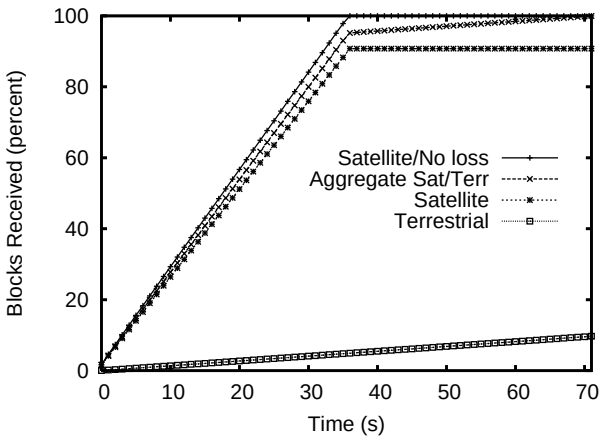

Figure 7. Initial Transfer and Recovery Phases

spend time after the satellite transmission has completed to recover the errors.

This graph shows that while the system is able to recover higher losses at low cost, the satellite link may be underutilized. In such a case we need to increase the forward error correction slightly to bring the net satellite errors down; otherwise the per-block latency grows unacceptably (up to 32s for the last block lost in this test).

The normal latency for blocks is captured in Figure 8, which shows the difference from the expected time of arrival for the blocks in a 100MB broadcast to 10 hosts with 2% satellite loss.

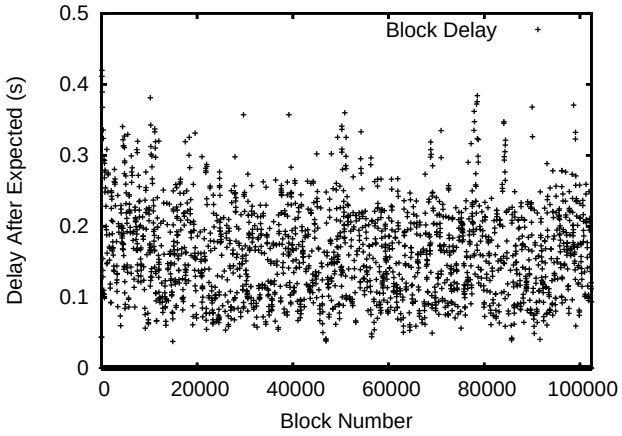

Figure 8. Block Delay Due to Loss

The vast majority of the blocks (98%) are received when expected from the satellite. The 2% lost are recovered via the peer-to-peer mechanisms, with latency up to 450ms but on average about 175ms. Each doubling of the number of hosts increases the worst-case latency we observe by approximately 50ms. The total latency incurred is of the same order as the baseline latency to reach and return to a geosynchronous satellite.

5.4. Network Efficiency

To show system efficiency we must have high access link utilization over the duration of the transmission (the core network is lightly loaded relative to its total capacity). Figure 9 shows the percent of access link capacity utilized by data; that is, not including packets corrupted, metadata, or packet header overhead on a 100MB transfer with 10% satellite packet loss.

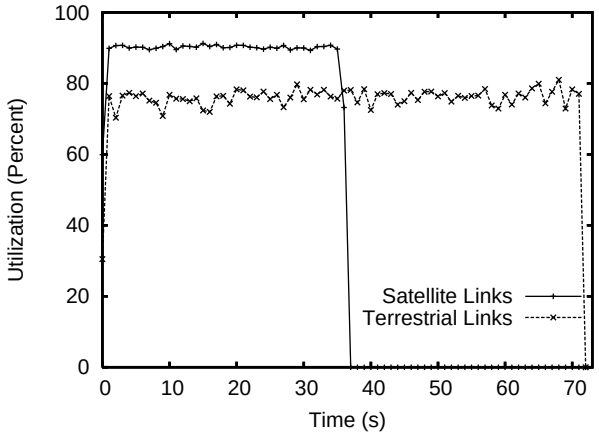

Figure 9. Percent Utilization of Access Links with 10% Satellite Loss

Satellite utilization is very high. Nearly 100% is initially used to transmit data packets, but we have 10% loss for this test. Terrestrial utilization is also high, between 70-80%. About 25% of this is idle due to in-order requests of blocks which can not be immediately satisfied, the remainder for control and metadata packets. Also note how utilization is more random during the satellite transmission phase; this is because losses on the satellite links create high but transient demand for some blocks, while afterward system state is known and behavior is more uniform.

Note that the data to be sent by each node, and hence the utilization of links, is controlled by the token-bucket mechanism discussed in section 3.2. Currently, it allows 100% of the link to be used for recovery, but the protocol is user-tunable. Users can trivially limit this to any desired proportion to avoid competing with other higher priority traffic. This effect on performance is exactly as though the node had a slower link.

How do results on this small topology compare with the larger AT&T (Internet) configuration? Figure 10 gives link utilization for a larger 1GB transfer and 2% loss rate on this network, parameters which represent a realistic scenario (which is streaming-recoverable).

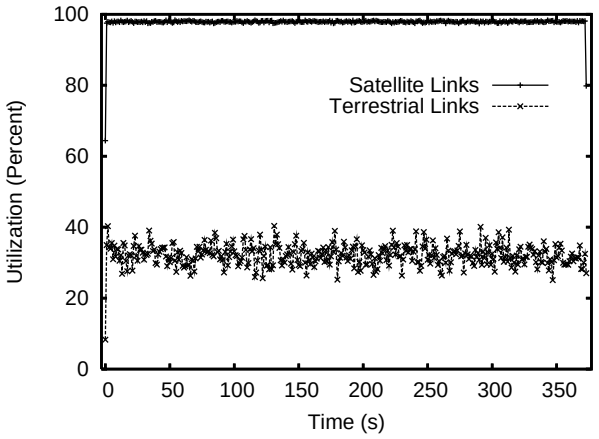

Figure 10. Percent Utilization of AT&T Access Links with 2% Satellite Loss

The first thing to notice is that since the losses are streaming-recoverable there is no phase where the satellite is idle. both networks are utilized to the end of the transfer. Second, the losses only require about 30% of the data capacity of the terrestrial network– this is in fact what makes the transfer streaming-recoverable. Third, the results are not qualitatively different than those on the simpler configurations. This has held true for all our tests. The primary difference is that the backbone structure creates a higher variation in the data; visible here as a 30Mb$\pm$10Mb terrestrial utilization, compared to the prior 75Mb$\pm$5Mb terrestrial utilization.

In general, demand for access link bandwidth is directly proportional to satellite loss, and satellite utilization (goodput) is inversely proportional to loss. The 10% loss rate for Figure 9 implies high demand for recovery bandwidth and that figure shows efficient use of resources to satisfy the demand. The lower loss rate of 2% in Figure 10 correspondingly shows higher satellite utilization and lower access link utilization. The point at which transient access link utilization reaches 100% is the point at which streaming recoverability becomes impossible.

5.5. Metadata Update Efficiency

There are two reasons a node may be unable to satisfy a data request. First is load limitation- if it is heavily loaded and has no tokens available (as in the prior section). Second is lack of up-to-date metadata information. Metadata must be fresh for effective transfer scheduling— the Scheduler needs to know which blocks have been lost at which nodes.

Figure 11 shows the unacknowledged blocks (blocks whose status is known to the receivers but not yet to the Scheduler) as a function of time. Recall that blocks are implicitly marked 'acknowledged' at the Scheduler when a NACK comes in for a lost block after them, or when a cumulative acknowledgment arrives (one or the other is sent on every timeout). This figure shows how efficient the metadata update mechanism is, and hence the freshness of Scheduler data.

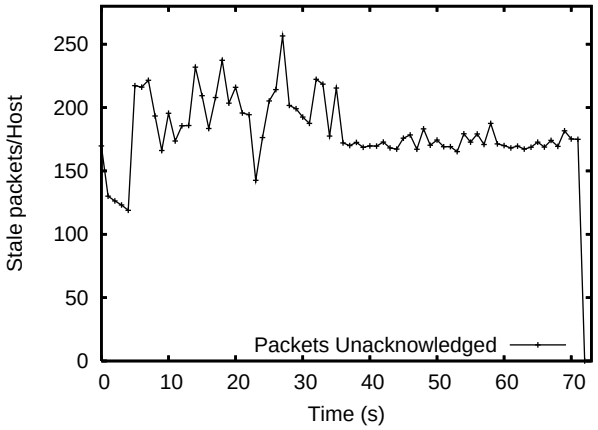

Figure 11. Number of Unacknowledged Blocks

This graph shows that the Scheduler information lags about 180 blocks behind actual state of node receptions. This resolves to about 60 milliseconds of simulated time, of which block timeouts make up 10-20ms, network latency about 30ms, and the remainder is due to lost metadata packets and queuing delay. We conclude that our metadata update mechanism is efficient; with most of the delay due to unavoidable physical constraints.

5.6. Fairness

Finally, fairness under Jain's measure has been very high in all cases. For the simple topologies and blocks sent it was uniformly over 0.999, meaning all nodes sent out almost the same number of blocks. Similarly, for recovery blocks received it was over 0.999, in this case due to the global termination constraints and uniform loss behavior. Under nonuniform loss, by definition some nodes will unfairly load the system to recover their data, but the blocks sent will still be evenly allocated (i.e. no tit-for-tat behavior).

The more realistic AT&T topology had fairness values consistently around 0.987, also very high but somewhat lower than the simpler topology. The difference was again due to the structure in the core network; some nodes were closer to the Scheduler. This implies the Scheduler data was consistently more fresh for those nodes, and they would tend to have slightly higher load.

In sum, these results coincide with our prior analysis and support our claims as to the system's performance under a variety of conditions.

6. Discussion and Related Work

There is a large body of work on content distribution, multicast tree construction, data distribution over unreliable transports, and peer-to-peer file transfer. It falls into three categories: terrestrial only, based on a set of point-to-point transfers, satellite only, based on a global broadcast mechanism, and other hybrid solutions based on use of both satellite and terrestrial systems.

Content distribution networks such as Akamai [2] fall into the first category. They have many of the same goals as this work, but focus on putting smarts "in the network" and the use of careful engineering to get caches close to the expected user base. For example: in our tests, if one places the source node on a very fast link there is no need for peer-to-peer transfers as the source can provide all error-correction information.

Multicast trees or meshes (such as Bullet [12]) and current P2P solutions (such as BitTorrent [9]) also fit here. In the prior case they solve the problem of scalable one-to-many transfer, but induce problems such as ACK implosion, node churn, agreement, etc. which we avoid entirely. The latter, are pull-based mechanisms for relatively small files (<4GB) with high delay tolerance, due to the use of randomization, rarest-first, erasure coding, and other mechanisms. Neither can exploit hybrid networks, and assume little or no trust between peers. The algorithms used also differ; see Section 6.1.

Commercial media distribution networks (ClearChannel, CNN, etc) fall into the second category. They are proprietary systems which involve custom hardware and software at all levels, specialized and dedicated to whatever they transmit (e.g. AM or FM radio signals, PAL or NTSC TV signals, HDTV, etc).

Much work in this category has centered around mechanisms for error detection and correction, focusing on FEC and TCP's well-known performance problems with loss being treated as congestion. Similarly, error detection after-the-fact is trivial via checksums, CRCs, or stronger cryptographic hashes. Again, these approaches can not take advantage of a side channel (the terrestrial network) for error recovery information.

The third category has one closely related project; a peer-based recovery mechanism for satellite transmission by Awal et al [4]. Differences from our work include an assumption of no terrestrial link to the source; meaning there is no way to communicate with the original source and hence no guarantee that nodes will receive the complete file. Also, error recovery is only done when the satellite broadcast is completed, thus, no concept of block latency or streaming-recoverability. Finally, they do not address the optimizations of link utilization or system cost as we have.

6.1. Other Peer-to-Peer Recovery Algorithms

In contrast to the peer selection algorithm discussed in Section 3.4, most other systems use a thresholded random rarest first algorithm to select the peer and block to download. That is, for blocks that are "rare enough" (fewer known to be in the network than some arbitrary threshold) they are requested from a random peer which has the block. Various heuristics are used to weight the random selection toward faster/more local hosts. Above the threshold, blocks are selected by other mechanisms, either in-order or randomly from the fastest hosts available.

Abstractly, such approaches select a block first based on estimations of block distribution in the network and then a host based on their expected bandwidth. Our approach selects a block first based on the next block required in the stream, and then a host based on globally balanced cost.

The difference is in the tradeoffs being made between latency, bandwidth reliability, and transfer locality. Rarest block first works well for latency tolerant systems with large number of hosts arriving/departing, but poorly for stable, latency intolerant systems. Ours works well in systems with relatively stable networks and good short-term behavior (implying small blocks) but poorly in unreliable networks with large churn.

Similarly, some systems use erasure codes [6, 13] and special-case end-of-file behavior to avoid problems with block scheduling and "stragglers." In the prior case, the problem is again latency as it may take reception of the entire file to decode the first block. We also use slightly less bandwidth; we receive no redundant data but send more metadata. The second case is generally a problem only with large numbers of hosts. In both cases, it is hard to capture the true costs in simulation: CPU utilization, disk thrashing, and so forth.

7. Summary and Future Work

We have presented a simple, flexible, robust, fair, and minimal cost mechanism for low-latency data transmission over hybrid satellite/terrestrial networks. Analysis and simulations have shown excellent performance in both specially

contrived and realistic environments under a variety of conditions. One can easily include arbitrary link cost models and still achieve globally maximal performance.

The performance of this approach is sensitive to the ratio between satellite and terrestrial link speeds (fewer losses can be recovered at speed when the ratio is high) and requires good connectivity to the scheduler node.

On the other hand, performance is insensitive to (is robust under changes in) network topology, absolute link speeds, link latency, and error correlation. It is particularly useful when satellite errors are low; it requires no complicated infrastructure or large memory footprint (e.g. multicast trees, erasure coding), and recovers the original data stream in almost real time.

Future directions for this work include implementation of a fully distributed scheduler to more fully explore the tradeoffs we have discussed here, integration of an automatic congestion control mechanism based on feedback rather than the current token bucket system, and if possible experiments on real-world hybrid networks.

References

[1] PlanetLab Consortium. http://www.planet-lab.org/.

[2] Akamai Technologies Inc. Akamai content distribution system. http://www.akamai.com/.

[3] AT&T. Global IP network. http://ipnetwork.bgtmo.ip.att.net/pws/index.html.

[4] M. Awal, Y. Tsuchimoto, and K. Kanchanasut. An approach of peer-based packet recovery using edbit for unidirectional satellite environment. In *Proceedings of the Workshop on Asia-Pacific Networking Technology*, Jan 2005.

[5] M. Bowman, J. Sedayao, and R. McGeer. Bakeoffs. http://www.planet-lab.org/Talks/2005-05-01/-Bakeoffs_final.ppt, Sep 2005.

[6] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM*, pages 56–67, 1998.

[7] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proceedings of INFOCOM*, pages 275–283. IEEE, March 1999.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *Proceedings of the SIGOPS European Workshop*, Sep 2002.

[9] B. Cohen. The BitTorrent file sharing protocol. http://bittorrent.com/.

[10] S. Floyd. Measurement studies of end-to-end congestion control in the internet, 2002. http://www.icir.org/floyd/ccmeasure.html.

[11] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared systems. Technical Report TR-301, Digital Equipment Corporation, Littleton, MA, 1984.

[12] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of ACM SOSP*, 2003.

[13] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 150–159, 1997.

[14] Public Broadcasting Service (PBS). PBS — Station Finder. http://www.pbs.org/stationfinder/index.html.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001*, pages 149–160, Aug 2001.