BooSTER: Broadcast Stream Transmission Epidemic Repair

Spyros Voulgaris†‡, Aaron Harwood§¶
†*Dept. of Computer Science, VU University, Amsterdam, The Netherlands, spyros@cs.vu.nl*
‡*The Network Institute, Amsterdam, The Netherlands*
§*National ICT Australia (NICTA)*[1]
¶*Dept. of Computing and Information Systems, University of Melbourne, Australia, aharwood@unimelb.edu.au*

***Abstract*—Wireless broadcasting systems, such as Digital Video Broadcasting (DVB), are subject to signal degradation, having an effect on end users' reception quality. Reception quality can be improved by increasing signal strength, but this comes at a significantly increased energy use and still without guaranteeing error-free reception.**

In this paper we present BOOSTER, a fully decentralized epidemic-based system that boosts reception quality by cooperatively repairing lossy packet streams among the community of DVB viewers. To validate our system, we collected real data by deploying a set of DVB receivers geographically distributed in and around Amsterdam and Utrecht, The Netherlands. We implemented and tested our system in PeerSim, using our collected real trace information as input. We present in detail the crucial design decisions, the algorithms that underpin our system, the realistic experimental methodology, as well as extensive results that demonstrate the feasibility and efficiency of this approach. In particular we conclude that the upload bandwidth required by each node for significant recovery of a real DVB broadcast is in the order of 5KB/sec when nodes allow up to 2 second delay for repairing – a rather trivial bandwidth for today's typical ADSL connections with an acceptable introduced delay.

I. INTRODUCTION

Wireless broadcasting systems, such as Digital Video Broadcasting — Terrestrial [1] (DVB-T), are subject to data reception errors and an associated loss of quality for the end user. Such errors can be caused by signal attenuation (e.g., due to high distance from the transmitter in sparsely populated areas, due to densely built city centers, etc.), or by unpredictable interference with the environment,

Introducing data redundancy, such as forward error correction [2], [3], [4] (FEC), combined with interleaving-based schemes, constitutes an obvious solution. In fact, DVB *does* employ this technique, appending 16 error-correction bytes to every 188 data bytes, relieving the receivers from a significant fraction of (limited) errors. However, being able to repair more severe errors would require a substantially higher redundancy ratio. Notably, when the fraction of nodes experiencing an error on a given data packet is very small (which is typical in DVB-T), taxing *all* nodes with excessive redundancy on *all* packets is inefficient. In addition, such a technique would not guarantee to fix *all* errors.

[1] NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

The relatively small fraction of data packets affected by signal distortion hints at *reactive* solutions that aim at retrieving damaged or lost packets on demand from external sources, rather than *proactive* approaches that augment all packets with ample error correction codes. There are two potential sources to retrieve packets from: the *broadcaster* and *other receivers*.

In our proposed protocol, BOOSTER (**B**roadcast **S**tream **T**ransmission **E**pidemic **R**epair), we consider a reactive approach where receivers *detect* lost or damaged packets and *retrieve* them from *other receivers* in a peer-to-peer fashion. This has three main advantages over retrieving packets from a central server of the broadcaster. First, a central server that feeds packets to all other nodes is a potential bandwidth bottleneck, especially at peak times, such as football evening with bad weather producing lots of errors. Contrary to that, BOOSTER enjoys the inherent scalability of peer-to-peer architectures. Second, our system does not require cooperation from the broadcaster, thus allowing a TV manufacturer to build BOOSTER-enabled TVs, or an open-source community to provide BOOSTER software, without requiring individual agreements and arrangements with broadcasters all over the world. Third, as latency is crucial for the timely repair of lost packets, clustering peers based on a latency metric can improve the overall quality of service, contrary to using a central server. Note that the decision to design a cooperative repair mechanism was further motivated by our observation that packet loss is highly *uncorrelated* among receivers even in the same room. That is, a packet lost at some receiver is very likely to have been received correctly by another receiver nearby.

Although the concept of repairing lost packets on demand is in principle simple, practical repair in the case of DVB-T is severely hindered by the fact that packet identification is troublesome, as will be explained in the paper. BOOSTER solves this issue, and attempts to recover the broadcast at all receivers with minimal use of bandwidth. The solution involves a tradeoff between bandwidth used and the timeliness of error recovery – using more bandwidth can recover more errors over a given time interval. Of course we can only recover local errors, that is, errors that occur at a receiver or group of receivers, not global errors that occur at the wireless transmitter, unless the source of the transmission also participates as a peer.

BOOSTER introduces, essentially, a peer-assisted algorithm that belongs to the Cooperative Peer-to-peer Repair (CPR) protocol class. Such protocols generally involve a *primary channel* being repaired using a *secondary channel*. Here, the primary channel is a DVB-T wireless broadcast video stream, and the secondary is UDP over the Internet.

Note that our work focuses specifically on repairing errors on a *wireless broadcast system*. As such, it is not applicable to IPTV or related Internet streaming systems.

Finally, note that our proposed system is readily applicable, as, first, the vast majority of households have a DSL connection, and, second, set-top boxes decoding DVB-T signals are generally programmable, and can implement our protocol. Furthermore most new televisions today are so called "smart" TVs, meaning Internet connected with configurable application services, and are therefore perfect platforms to deploy our technology.

In this paper we provide a detailed discussion of our design choices and explicit fundamental peer-to-peer algorithms underpinning BOOSTER. Our results have sufficient detail to be applied to real systems, unlike other published work in this area. We extracted a variety of DVB-T streams, ranging from little to excessive error at multiple locations, and used this to substantiate the effectiveness of BOOSTER in terms of key measurements, including introduced playback delay, bandwidth usage and repair quality. We have also exported an application from our emulation platform and tested BOOSTER over a real network.

II. BACKGROUND ON DVB

Digital Video Broadcasting (DVB) is a standard for broadcasting of digital television. It adopts the ISO/IEC 13818-1 standard for packetizing streaming data, such as video and audio streams, including program stream multiplexing and mapping, conditional access management, timing references for playback synchronization, and metadata such as electronic program guides.

A video stream is first compressed by the MPEG algorithm, with a quality parameter that ultimately leads to an average bits per pixel of the encoding. This bit stream undergoes two levels of encapsulation, as depicted in Fig. 1. Bits are first encapsulated into variable length *Packetized Elementary Stream* (*PES*) packets. These PES packets are, in turn, encapsulated into 188-byte fixed length *Transport Stream* (*TS*) packets. Combined, this leads to a total data rate for the stream.

The DVB broadcaster has preselected a desired total bitrate for the wireless channel from a table of possible choices. Clearly this total bitrate must equal or exceed the bitrate of the Transmission Stream. This leads to a required carrier-to-noise ratio that must be achieved for quasi-error free transmission at that rate, and an associated bit error rate. Fluctuations in the carrier-to-noise ratio lead to fluctuations in the bit error rate.

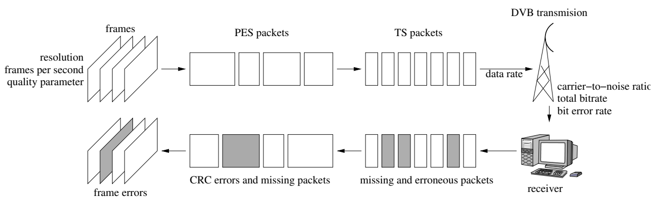


Figure 1. Relevant highlevel aspects of the system in our work. Frames of a stream are produced at a given resolution and frame rate and encoded using a quality parameter, with the final data ultimately encapsulated in a Transport Stream. The carrier-to-noise ratio determines the total bitrate available to the wireless channel, and the bit error rate associated with it. Reed-Solomon codes are used to ensure that the packet error probability at the receiver is lower than a target value.

The DVB system uses Reed-Solomon codes (appending 16 bytes of check symbols to every 188-byte TS packet) and two levels of interleaving to disperse packet loss to non-consecutive packets. At the receiver, each TS packet may be either *received* (i.e., received correctly, or with few errors that were corrected), or *missing* (i.e., not received at all, or received with irrecoverable errors).

Higher up the stack, PES packets use CRC error checking to ensure that none of their associated TS packets is missing. A single missing TS packet will lead to a CRC error for the PES packet that contains it.

Whether PES packets have CRC errors or not, they may be given to the viewer for the (possibly erroneous) frames to be rendered on the display. Error concealment can take place at the point of rendering (such as linear interpolation of missing pixel blocks) as a further strategy to reduce the effect of errors.

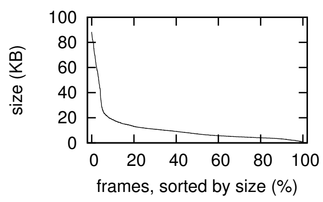
III. DESIGN DECISIONS AND CHALLENGES

A. *Scope of repairing*

A fundamental decision in our work has been the selection of the level at which repair should be applied. Specifically, there are three possibilities. Repairing *frames*, repairing *PES packets*, or repairing *TS packets*. Due to diverse pros and cons, no option constitutes a win-win tradeoff.

We decided to repair TS packets. While it is tempting to consider working at the PES or frame level because these levels provide more semantic information (notably, unique sequence numbers), there are some drawbacks that we identified:

- There is less total reliable information at the higher levels, as a single TS packet error can invalidate an entire PES packet.
- Semantic information tends to be optional and making use of optional information leads to a less portable solution.
- TS packets (188-byte long) conveniently fit into a UDP packet, while transmitting a PES packet or a frame may require fragmentation.



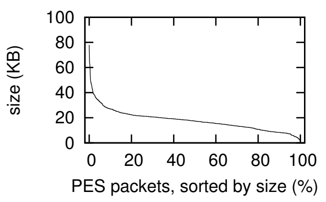


Figure 2. Size distribution of frames (left) and PES packets (right).

- More processing is required until an error is detected at higher levels, such as decoding PES packets and, in turn, frame information.

Fig. 2 shows the distribution of frame and PES packet sizes from a DVB video sample using MPEG2 704x576. Even at this relatively low resolution (HD television usually allows up to 1920x1080), frame sizes exceed a single UDP packet payload. The variability of frame and PES packet sizes and their inability to always fit in a single UDP packet would add complexity to our overall system. We mitigate that by working at the TS level.

On the down side, TS packets do not contain sequence numbers, which complicates the *detection* of missing packets. Even worse, they do not contain unique identifiers either, which perplexes the *identification* of a packet known (or suspected) to be missing. These obstacles can be overcome by the use of some convenient, mandatory, yet *sporadic*, semantic information contained in some TS packets (explained in the following section).

Furthermore, Transport Streams for the same broadcast may also differ across transmitters (e.g., the same TV channel in different cities), meaning that in general a sequence of TS packets from transmitter A cannot always be used to repair a sequence from transmitter B. We therefore need to group the nodes in our system on a per transmitter basis.

B. Transport Stream Packets

A DVB receiver, when tuned and locked to a carrier frequency, produces a stream of TS packets, called the raw TS stream. Among other information, *every* TS packet in the raw stream contains the following key fields:

- a 13-bit *Program Identifier* (*PID*) that maps the TS packet to an elementary stream, such as a video or audio stream for a given TV channel;
- a 4-bit *Continuity Counter* (*CC*) that is incremented by 1 modulo 16 for each subsequent TS packet in the elementary stream (certain flagged conditions may arise where the CC is not incremented); and
- a 1-bit *Transport Error Indicator* (*TEI*) that is set true by the receiver if the TS packet is erroneous.

Other information in the raw stream, including Program Association Tables (PATs), found in TS packets with *PID*=0, and Program Mapping Tables (PMTs), found in TS packets with a *PID* value as specified in the PAT, allows us to discover the *PID* values for elementary video and audio program streams. We can then, e.g., pull a specific video stream from the raw stream for presentation to the user or for cooperative repair prior to presentation.

term	*full term*	*description*
PES	Packetized Elementary Stream	Encapsulates fragments of MPEG frames. Variable packets length.
TS	Transport Stream	Encapsulates fragments of PES packets. Fixed length of 188 bytes.
CC	Continuity Counter	4-bit field of TS packets. Incremented by 1 modulo 16 in each subsequent packet.
PCR	Program Clock Reference	48-bit field contained in a few TS packets. Required to be inserted at least every 100ms, but not at fixed intervals.

Table I
LIST OF ACRONYMS AND TERMS.

The fields listed above are insufficient to synchronize two streams for the sake of cooperative repair. There is no uniquely identifying information in each TS packet, and indeed duplicate TS packets may arise in the stream. However the standard allows a TS packet to contain additional information in an optional Adaption Field, and this optional information contains a 48-bit *Program Clock Reference* (PCR).

In fact, the standard *requires* this optional information, containing the 48-bit Program Clock Reference, to appear at least every 100 milliseconds. PCR values are not sequential. They are, however, monotonically increasing, therefore *unique* within a stream. The presence of the PCR provides semi-regular, unique stamps on selected TS packets. Thus, we make use of the PCR to synchronize two nodes so that cooperative repair can take place.

Table I lists the acronyms that are important for understanding the BOOSTER peer-to-peer algorithm.

IV. BOOSTER: THE ALGORITHM

A peer-to-peer repair algorithm relies on two main components: a mechanism to *detect* missing information, and a naming scheme to uniquely *identify* this information in requests to external sources.

A. Detecting Missing Packets

We use the Continuity Counter (CC) field, described in Section III-B, to detect missing TS packets. This has a clear limitation, as any sequence of k consecutive missing packets is reflected by a CC gap of k modulo 16. For instance, it is impossible to distinguish between missing a single packet, or 17, 33, and generally $16i + 1$ consecutive packets. Even worse, missing sequences of exactly a multiple of 16 packets will go undetected.

To assess the accuracy of the continuity counter CC in measuring the number of missed TS packets, we recorded and analyzed a number of DVB streams at locations with diverse reception qualities. For each *sample recording*, we

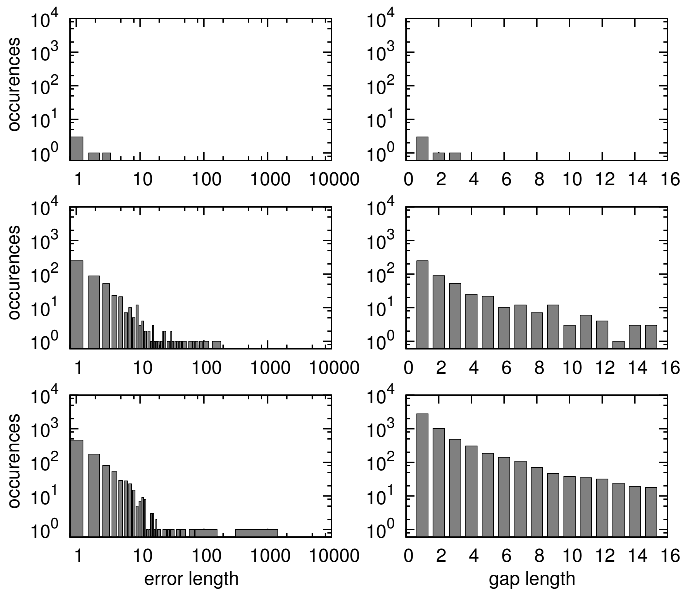
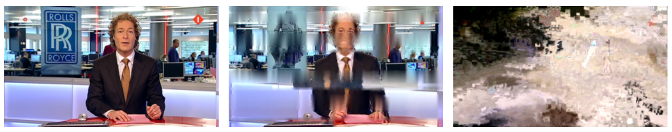


Figure 3. Distribution of actual consecutive error lengths (left) and the respective Continuity Counter gap lengths (right).



(d) *good*: infrequent errors

(e) *bad*: frequent errors

(f) *ugly*: unviewable

Figure 4. Sample streams demonstrating different reception quality.

also made a *reference recording*, by a DVB-T antenna fixed at a location with superb reception. Bitwise comparison between the TS packets in the sample and reference recordings allowed us to detect the exact sequences of lost TS packets in the former.

Fig. 3 shows the count of CC gaps, as well as the real error sequence lengths for three recordings of the same duration (circa 1 minute each), yet of very different viewing qualities: infrequent errors, frequent errors and unviewable. For streams of high quality, errors appear to be infrequent and short, mostly a handful of consecutive TS packets. However, worse reception qualities, that are quite common in real use cases per our experience, demonstrate error sequences of up to several thousands of consecutive TS packets. Clearly, the mighty 16-state continuity counter is not able to express the length of these errors.

Nevertheless, as demonstrated in Fig. 3, even when the reception quality is bad, short errors of up to a dozen TS packets are several orders of magnitude more frequent than long sequences of errors. This is reflected on the observed CC gaps, where gaps of length 1 (i.e., 1, 17, 33, etc., consecutive missed packets) are two orders of magnitude higher than those of length 15 (15, 31, 47 packets, etc.).

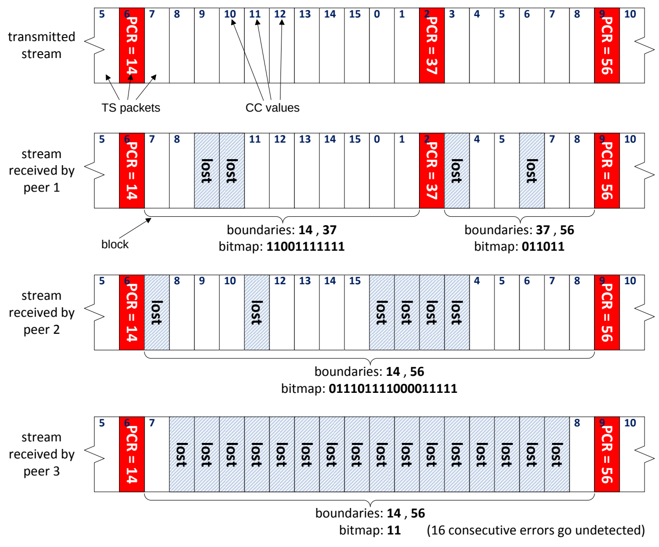


Figure 5. Blocks perceived by three receivers for a sample stream.

The respective snapshots in Fig. 4 illustrate characteristically the quality of each stream.

B. Naming Scheme for Identifying Packets

In order to let peers identify specific units for repair to other peers, we introduce the notion of blocks. A *block* (see Fig. 5) is a sequence of TS packets, consisting of all packets between two consecutive PCRs (exclusive). The starting and ending PCR values of a block are called its *boundaries*, and uniquely identify this block across all nodes.

TS packets within a block are identified by means of their offset relatively to the starting PCR boundary. This offset is computed by merely counting the number of TS packets received, as well as by adjusting this counter in the face of CC discontinuities to account for lost packets.

More specifically, when a node receives a PCR packet, it marks the beginning of a new block. By observing the CC in subsequent TS packets, it keeps track of which packets it received, and which it missed. In doing so, it follows an optimistic approach: a CC gap of $k \in [1, 15]$ is interpreted as a loss of exactly k packets, rather than $16i + k$. A zero gap is interpreted as no lost packet, rather than as $16i$ lost packets. With high probability, the assessment of what has been received is accurate, unless 16 or more *consecutive* packets were lost at some point. The next PCR received marks the end of this block and the beginning of a new one.

The record of which TS packets were received and which are missing constitutes the block's *bitmap*: a series of bits equal to the (perceived) number of TS packets in the block, where 1 stands for received and 0 for missing packet. Note that missing one or more packets containing PCR values leads to concatenated blocks that may later be repaired and split up to smaller blocks.

Algorithm 1: The Cooperative Repair Algorithm

```
   // used to gradually assemble a block from many TS packets
1  Variables
2      TsPacket[] tspArray
3      long startPCR
4      long endPCR
5
   // Called by the radio on new TS packet arrival
6  function DELIVER(tsp)
7      tspArray $\xleftarrow{append}$ tsp
8      if tsp contains PCR then
           // block boundary reached
9          endPCR ← tsp.PCR
10         MEMORY.STORE (startPCR, endPCR, tspArray)
11         bitmap ← COMPUTEBITMAP(tspArray)
12         if bitmap contains 0 then
               // at least one TS packet is known to be missing
13             REPAIR (startPCR, endPCR, bitmap)
14         tspArray ← {} // reset tspArray for next block
15         startPCR ← tsp.PCR // first PCR of next block
16
17 function REPAIR(startPCR, endPCR, bitmap)
18     peer ← SELECTRANDOMPEER()
19     invoke PULL(myAddress, startPCR, endPCR, bitmap) on peer
20     SETTIMEOUT (pullTimeout, startPCR, endPCR, bitmap)
21
22 function PULLTIMEOUT(startPCR, endPCR, bitmap)
23     if no response has been received for this repair request then
24         if GETTIME()-MEMORY.BCASTTIME(startPCR)<viewerTimeout then
25             REPAIR(startPCR, endPCR, bitmap)
26
27 function COMPUTEBITMAP($TS$[] tspArray)
28     bitmap ← {1}
29     for i ←1 to ||tspArray|| − 1 do
30         gap ← (tspArray[$i$].$CC$ − tspArray[$i-1$].$CC$ − 1)%16 // Continuity
             Counter gap, mod 16
31         bitmap $\xleftarrow{append}$ 0 × gap times // for missing TS packets
32         bitmap $\xleftarrow{append}$ 1 // for the packet at tspArray [i]
33     return bitmap
34
```

Algorithm 2: Network Operation

```
   // Called when the node receives a PULL request from another node
1  function PULL(sender, startPCR, endPCR, bitmap)
2      tspArray ← MEMORY.RETRIEVE (startPCR, endPCR)
3      if tspArray ==NULL then
           // Unknown block → send an empty reply
4          invoke PUSH("CAN'T HELP", startPCR, endPCR, ∅) on sender
5      else
6          myBitmap ← COMPUTEBITMAP(tspArray)
7          if len(myBitmap)==len(bitmap) then
               // Send selectively the TS packets he is missing
8              repairBitmap ← myBitmap AND NOT bitmap
9              tspArrayPatch ← Select the TS packets of tspArray with a set bit in
                 repairBitmap
10             invoke PUSH("MERGE",startPCR, endPCR, tspArrayPatch) on sender
11         else if len(myBitmap) > len(bitmap) then
               // Send all my TS packets
12             invoke PUSH("COPY", startPCR, endPCR, tspArray) on sender
13         else
               // I had more errors → send an empty reply
14             invoke PUSH("CAN'T HELP", startPCR, endPCR, ∅) on sender
15
   // Called when the node receives a PUSH response from another node
16 function PUSH(operation, startPCR, endPCR, bitmap, tspArrayPatch)
17     if operation =="MERGE" then
18         tspArray ← MEMORY.RETRIEVE(startPCR, endPCR)
19         for tsp ∈ tspArrayPatch do
20             tspArray $\xleftarrow{insert}$ tsp at appropriate position based on bitmap
21         MEMORY.CHECKFORNEWBOUNDARIES(startPCR, endPCR)
22     else if operation =="COPY" then
23         MEMORY.REMOVE(startPCR, endPCR)
24         MEMORY.STORE(startPCR, endPCR, tspArrayPatch)
25     forall blocks [startPCR′, endPCR′] ∈ [startPCR, endPCR] do
26         if TS packets still missing, and viewerTimeout not expired then
27             bitmap ← COMPUTEBITMAP(MEMORY.RETRIEVE(startPCR,
                 endPCR))
28             REPAIR(startPCR,endPCR, bitmap)
29
```

C. Regular Operation

Algorithms 1 and 2 show the pseudocode of the BOOSTER protocol.

When tuned to a channel, a node keeps receiving TS packets by means of its radio receiver calling the DELIVER() function (Alg. 1, line 6). When a new TS packet is delivered, it is appended to an array (line 7), and it is checked whether it contains a PCR entry (line 8). If yes, this signifies the completion of a block, and the beginning of a new one.

Upon completion of a block the node stores that block in memory (line 10), indexed by its boundary PCRs. Then the node checks whether all packets (are believed to) have been received (line 12). If not (line 13), it invokes a PULL operation on a random other node that is currently tuned to the same TV channel (see Sec. IV-E on how this is obtained), requesting the missing TS packets (lines 17-20). The request is composed by including the block boundaries and the block bitmap (line 20).

A node receiving a PULL request looks up its memory for a block with the specified boundaries (Alg. 2, line 2). If it has not registered such a block in its memory (e.g., it may have missed one of the PCR boundaries), it simply replies with a "Can't Help" message (line 4). Otherwise, it may have it complete, or it may be missing some TS packets too. In either case, it performs a bitwise operation on the two block bitmaps to figure out which of the requested TS packets it has (line 8). It then sends a PUSH response to the requester, piggybacking zero or more of the requested TS packets (lines 9-10).

Upon receiving a PUSH response (Alg. 2, line 16), a node adds the received TS packets to the block in question (lines 18-20), updates the block's bitmap, and checks if the block is now complete (line 25). If it is still missing packets, it issues a new PULL request on another random node (lines 26-27). This is repeated until either the block is complete, or a time threshold, VIEWERTIMEOUT, has been reached (not shown in pseudocode). At that point, the TS packets of the block are handed to the higher layers for decoding and viewing. A second timer, PULLTIMEOUT, is associated with each PULL request. If no response is received for that time, a new PULL request is sent to another random node (Alg. 1, lines 23-25).

D. Special Cases

A node receiving a PULL request may realize that the requested block can be located in its memory, but its length does not match the length reported in the request. This implies that at least one of the two nodes has missed a sequence of 16 TS packets or more, out of this block. Clearly, it is impossible to identify the missing packets, as bitmap offsets cannot be matched anymore. Nevertheless, we do know that the version with the longest bitmap is the best pick among the two. As such, the node replies with a complete *copy* of its entire block if its version is the longer one (Alg. 2, line 12), or with a "Can't Help" response otherwise (line 14). Upon reception of a *copy* response, the requester will replace its current version of the block with the one received (lines 22-24).

In another occasion, a node receiving a PULL request may discover that startPCR and endPCR can be located in its memory, but they do not belong to the same block. This means that the requester missed one or more TS packets containing a PCR value, that is, it missed one or more block boundaries between startPCR and endPCR, consequently perceiving two or more blocks as one. In that case, the receiving node concatenates (internally done by MEMORY.RETRIEVE() in Alg. 2, line 2) to a temporary buffer all consecutive blocks, from the one starting with startPCR to the one ending in endPCR, and responds to the request as if it were a usual single block (lines 6-14). When the requester receives the PUSH response, it will populate its block with the offered TS packets, and it will notice that some of them contain PCR values. It will then split the block in two or more blocks accordingly (line 21), and it will continue trying to repair them *individually*, if some or all of them are still missing TS packets and the respective VIEWERTIMEOUT has not yet expired (lines 25-28).

Finally, a PULL request may refer to a block for which both nodes have missed an *equally sized*, yet *different*, set of 16 or more TS packets. The two nodes have the same (wrong) perception of the block's length, so the receiver will trust the offsets found in the request bitmap. The PUSH response, however, will most probably contain packets of wrong offsets. Unfortunately, there is no way for our mechanism to detect this rare anomaly. The error will propagate higher and it will trigger a CRC error at the PES level.

E. Supporting Mechanisms

Like many epidemic peer-to-peer protocols, the BOOSTER algorithm relies on communication between peers selected *uniformly at random*. To that end, we rely on the family of Peer Sampling Service protocols [5], and specifically CYCLON [6], which provides each node with a regularly refreshed list of links to random other peers, in a fully decentralized manner and at negligible bandwidth cost.

In CYCLON each node maintains a (very short) *partial view* of the network, that is, a handful of links (IP addresses and ports) to other nodes. Each node periodically gossips with one of its neighbors, mixing their views. As a result, views are *periodically refreshed* with new links to random other nodes. When the right policies are followed (see [5] and [6] for details), this method has shown to produce overlays that strongly resemble random graphs, that is, at any given moment each node's view contains links to nodes selected uniformly at random among all alive nodes. Then, selecting one of these neighbors at random (e.g., to send a PULL request), is essentially equivalent to selecting one node at random out of the whole node population. Further, when the node's CYCLON view is changing over time, the node has essentially access to an endless stream of random peers to communicate with. Notably, the emerged overlays are remarkably robust to node churn and failures.

In BOOSTER we additionally need to group together nodes that are tuned to the same transmitter and are additionally watching the same TV channel. This is fairly straightforward using a distributed topology construction framework, such as T-MAN [7], which clusters nodes based on a proximity function. In short, each node maintains a short view of ℓ neighbors, where ℓ is fixed. Each node periodically picks one of its neighbors, they send each other a few of their neighbors, and each one maintains the closest ℓ (based on the proximity function) out of its previous neighbors and the ones received. Each node converges at exponential number of iterations at having the closest neighbors out of the whole network.

In our application, we define the proximity between two nodes to be the numeric difference of their concatenated *Transmitter Identifier* and *Program Identifier*. This way, a newly joined node gradually selects neighbors of "concatenated ID" closer to its own, and in a few rounds it reaches its target cluster and keeps gossiping within it. Both CYCLON and T-MAN have shown to operate with remarkable reliability and robustness in (even highly) dynamic conditions.

Finally, their bandwidth cost is minimal. In our setup, CYCLON and T-MAN were configured to maintain 100 links in memory (each), and to exchange 10 links when gossiping (each). That is, in each gossip exchange it initiates to a neighbor, a node sends 10 links. Given that on average a node also has to respond to a gossip exchange initiated by another node once per round, it has to send another 10 links. Given that CYCLON and T-MAN gossip independently, this gives a total of 40 uploaded links per round. Finally, given that a link can be as short as 10 bytes (IP address: 4B, port: 2B, Transmitter and Program IDs: 4B), and that nodes were configured to gossip every 5 seconds, this gives 80B/sec upload bandwidth total, for both protocols. The download bandwidth is symmetric.

V. EVALUATION

For the evaluation of our system we deployed a real-world testbed consisting of 12 DVB-T receivers to collect real

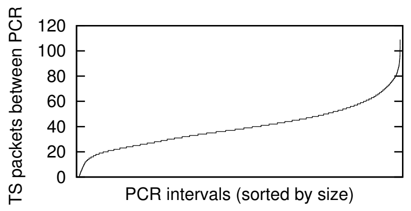


Figure 6. Intervals of TS packets between consecutive PCRs.

traces. These traces were fed to our local simulation and emulation engine (based on PeerSim [8]) that allowed us to scale to orders of magnitude larger networks.

A. Methodology

We deployed 12 USB DVB-T receivers at distinct and diverse locations across Amsterdam and Utrecht, The Netherlands. We developed and deployed a data collection framework that enabled the centrally controlled, synchronized collection of DVB-T input data at all collaborating sites, and we collected snippets of DVB stream recordings at various times during a period of one week.

One of the receivers was kept in a location with excellent reception quality (high floor, unobstructed view). The majority of its recordings were absolutely error-free, and they served as our *reference recordings*. The remaining receivers were distributed to volunteer students, that were instructed to operate them in different parts of their apartments or even their neighbors' apartments every other day.

We extracted two types of traces from the collected recordings:

1) PCR traces: These traces register the occurences of TS packets carrying PCR values in a stream of TS packets. Recall that PCR values are included in a (small) subset of TS packets, and are required to appear at least every 100ms, yet not at fixed intervals. Indeed, we observed a high interval variation, ranging from one to 109 TS packets between two consecutive PCR occurences. Fig. 6 shows the distribution of interval lengths.

As PCRs serve as block boundaries, determinining block lengths, we used these traces in our simulations to replay the exact block length patterns observed in reality.

2) TS error traces: These traces register, for each recording separately, the exact pattern of received and missed TS packets. This is obtained by comparing a recording against the corresponding reference (error-free) recording. Some of the weakest recordings suffered close to 30% packet loss.

The most important observation from these traces is that there is no correlation between the lost packets of different receivers, despite their proximity. In fact, this was true even when we tested multiple receivers in the same room. This is fundamental for the efficiency of BOOSTER, as when a node misses a packet, there are high chances that some other node has received it.

These traces were used in our simulations to replay the exact packet loss pattern observed in reality. Each simulated node was associated with one such trace. Due to the scale of our experiments, with up to 1000 simulated nodes, each error trace had to be shared by multiple nodes. Each node started processing its trace at a different random point, to avoid systematic errors.

Finally, we modeled Internet latency between nodes using the MIT King dataset, containing a matrix of pairwise latencies among 1740 DNS servers collected by the King method [9]. Note that this stress tests our system at *worse* latency conditions than we would expect in real operation, for two reasons. First, the MIT King dataset was collected in 2004, and latency has improved since. Second, the dataset captures latencies at a global scale, while Internet communication in BOOSTER is inherently taking place among nodes in the same city, receiving a DVB-T stream from the same transmitter. Therefore latencies would be among the low end of this dataset. Nevertheless, we used the complete MIT King dataset due to its wide acceptance.

B. Fixed error experiments

The principal metrics of interest in our system are the percentage of TS packets that can be completely repaired and the bandwidth required. In the first batch of experiments we investigate how the two parameters of the system, PULLTIMEOUT and VIEWERTIMEOUT, affect these metrics.

We assign the same fixed error pattern to all nodes in an experiment, ranging from experiments with nearly zero errors to experiments with nearly 30% TS packet loss ratio for each node. This uniform error behavior across nodes allows us to concentrate on experiment-wide statistical properties, rather than diving into the individual behavior of different nodes (which is left for the following section).

Fig. 7 contains four plots, for PULLTIMEOUT 100ms, 200ms, 400ms, and 800ms, respectively. Each plot shows, as a function of the experiment-wide packet error rate, the percentage of missing TS packets right after broadcasting (solid line – top), and how this percentage drops with an increasing VIEWERTIMEOUT, namely after 500ms, 1000ms, 1500ms, 2000ms, 2500ms, 3000ms, and 5000ms of collaborative repairing.

Clearly, as the VIEWERTIMEOUT increases, the ability to know and repair more packets increases as well. In fact, VIEWERTIMEOUT has a dominant effect in this respect, more so than PULLTIMEOUT. E.g., for a mere delay of 3sec or 5sec, users that would otherwise hardly be able to watch a program, can now enjoy a crystal clear stream, irrespectively of the PULLTIMEOUT used.

C. Mixed error experiments

We now investigate the interaction of nodes in the presence of diverse error patterns. In that respect, nodes are associated with error patterns captured through our real

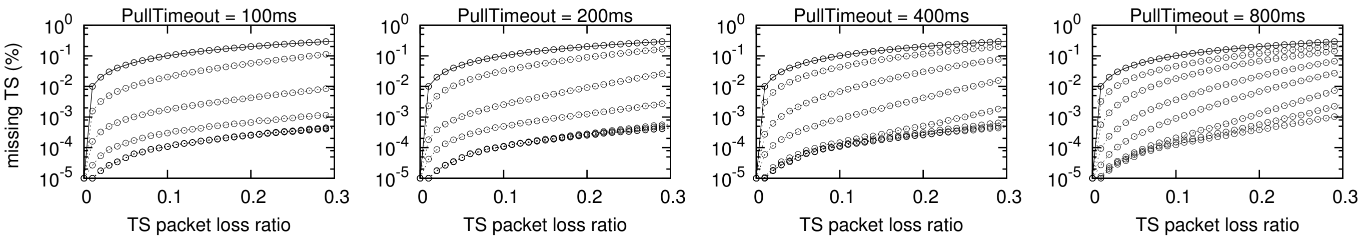


Figure 7. Repair effectiveness vs. node error rates, for different ViewerTimeout and PullTimeout values. In each plot, from top down — Missing TS packets after 0, 500, 1000, 1500, 2000, 2500, 3000 and 5000 milliseconds of repairing. Note that each circle corresponds to the average of a single experiment, where all nodes experience the same TS packet loss ratio.

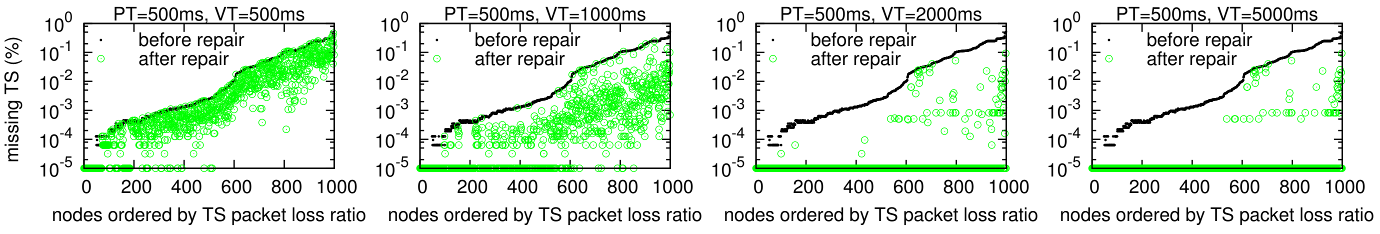


Figure 8. Repair effectiveness by means of missing TS packets *before* and *after* repair, for different ViewerTimeout values. Note that 0 values (zero missing packets) have been mapped to the value 10^{-5} to show up in the log-scale plots.

world testbed. Thus, the experiments we carry out this way "replay" a very close approximation of our system in the real world.

First, we are interested to explore the collaborative repair efficiency on nodes of different error patterns. As we previously identified ViewerTimeout to be the dominant parameter for repair efficiency, Fig. 8 presents four plots for different ViewerTimeout values, namely 500ms, 1000ms, 2000ms, and 5000ms, respectively. Each plot corresponds to a single experiment, and plots the number of TS packets a node is missing *before* and *after* repairing, in log-scale. Note that we explicitly mapped 0 missing packets to the value 10^{-5}, so that they appear in the log-scale.

As expected, with a short ViewerTimeout of 500ms, or even 1sec, nodes manage to repair several, but not an overwhelming fraction of their missing TS packets. However, by allowing the collaborative repair algorithm a bit more time, such as 2 or 5 seconds, it can turn a rather weak signal to very good, and in most cases crystal clear. Note the dense horizontal line of points at $y = 10^{-5}$, denoting zero errors.

Next, we want to assess the effect of PullTimeout on bandwidth. In Fig. 9 we fix the ViewerTimeout to 2sec, and we present the upload and download bandwidth, per node, for four values of the PullTimeout, namely 100ms, 200ms, 400ms, and 800ms. Note that we do not plot the bandwidth for the maintenance of random links, as in Sec. IV-E we calculated it to be constant and negligible: 80Bytes/sec upload + 80Bytes/sec download.

We make two observations from these plots. First, as expected, the lower the PullTimeout, the more aggressive nodes are with repairing their missing TS packets fast, at the cost of higher bandwidth (both download and upload).

The second—and most important—observation is that, although *download bandwidth* is proportional to the number of TS packets a node needs to repair, the contribution of all nodes in *upload bandwidth* is roughly the same, and largely independent of the node's original packet loss ratio. This is attributed, first, to the random selection of nodes for Pull requests, and second, to the fact that errors are *not* correlated across different nodes, a property we have confirmed in thorough trace analysis. Additionally, although a node may be missing a lot of TS packets due to weak reception, repairing them through our system makes it eligible to serve, in turn, Pull requests of other nodes.

It should be emphasized that the upload bandwidth required by each node is in the order of 5KB/sec for reasonably selected settings, a rather trivial bandwidth for today's typical ADSL connections.

Finally, we introduced packet loss by randomly dropping Pull requests and Push responses. In Fig. 10 we show the cases for 1%, 2%, 5% and 10% packet loss. The plots show that increased Internet packet loss has relatively little impact on the percentage of repaired TS packets, with most affected nodes still managing to substantially recover their errors. Note that the ViewerTimeout is set at 2000ms and PullTimeout set to 500ms which allows nodes enough time to recover from lost Internet packets on top of recovering from lost TS packets.

VI. Related Work

Cooperative repair represents a middleground between wireless broadcasting and live video streaming, such as

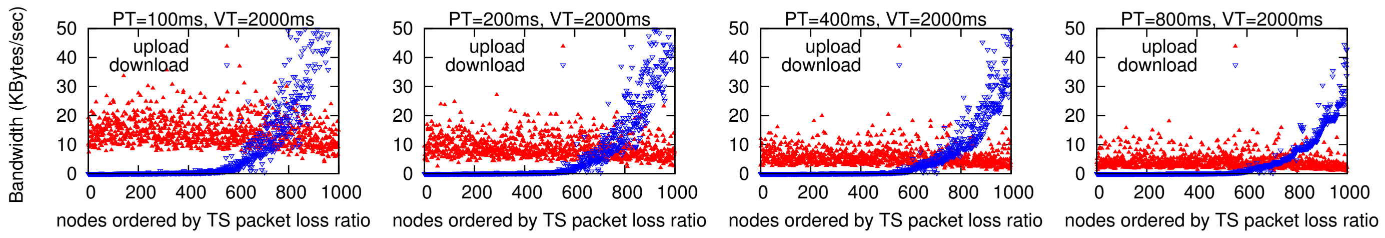


Figure 9. Upload and download bandwidth for nodes of diverse packet loss ratios, for different values of PULLTIMEOUT. Upload bandwidth is kept very low, and nearly uniformly balanced across all nodes.

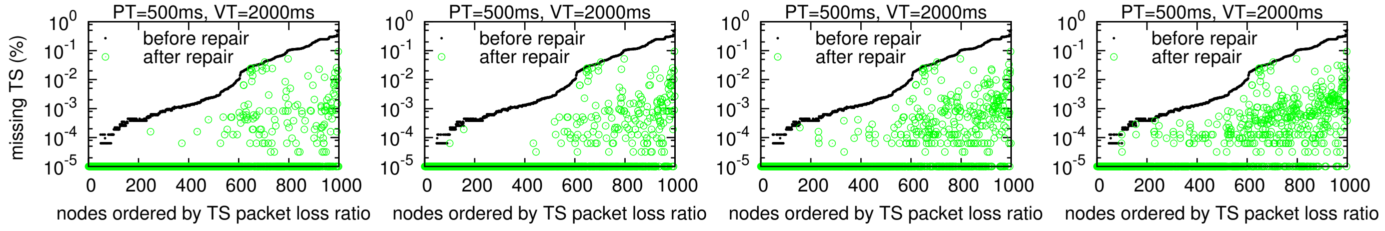


Figure 10. Repair effectiveness by means of missing TS packets *before* and *after* repair, for different Internet packet loss rates of (from left to right) 1%, 2%, 5% and 10%. Note that 0 values (zero missing TS packets) have been mapped to the value 10^{-5} to show up in the log-scale plots.

IPTV. Unlike IPTV, which delivers the entire video over the Internet, BOOSTER uses the Internet only when necessary to repair a received wireless broadcast. In this way, BOOSTER places significantly less burden on the Internet than IPTV does. Furthermore, a given repair packet is typically needed by a small fraction of the receivers, so off-the-shelf use of systems like BitTorrent would introduce unnecessary complexity and overhead.

The work in [10] is closely related to ours. The authors consider a satellite broadcast and repair it via a terrestrial network. They consider the cases where both a centralized scheduler is available to coordinate the repair between peers, and where no such scheduler exists. If a centralized scheduler is available, each peer sends a negative acknowledgement (NAK) to the scheduler for each block that it receives in error. The scheduler then responds with a list of peers that may be contacted for the block. When no central scheduler is available, a peer randomly selects k other peers to recover from a lost block. In both cases it is assumed that peers can reliably detect the loss of each and every lost block – which is something that in practice we could not assume in our system due to TS packets not being uniquely identifiable. Their work also has a slightly different goal since they allow a signal to the source for resending blocks which were not received by any peer – and their experiments consider transfer of files of various sizes.

Also addressing satellite transmission as the primary channel, the work in [11] maintains trees within the terrestrial peer-to-peer network to facilitate the exchange of repair information and retrieval of lost packets.

A number of researchers have concentrated on wireless broadcasts in 3G cellular networks with cooperative repair using 802.11 or Bluetooth as the secondary channel. The work in [12] proposes Fair Load Point Coordinated Error Resilience, which uses an approach similar to a centralized scheduler, but where each peer is a scheduler for a subset of peers in the network. The primary channel is GSM and the secondary channel is WLAN. After a node detects a packet loss, it indicates the loss to the leader in the form of a NAK. Then the leader selects an appropriate peer from its group to respond with the requested packet.

The work in [13] allows a peer to broadcast over the secondary channel that it requires assistance to recover from errors. A detailed protocol to establish and maintain partnerships between peers is proposed. In [14], the BOPPER scheme is proposed, where a peer with a correctly received packet rebroadcasts its packet to its neighborhood, with a certain predetermined probability in order to reduce flooding in the network.

Alternatively [15] explores the optimal communication required between a set of peers to recover the correct transmission. The algorithm is NP-hard and they consider various optimizations and a distributed implementation.

Finally, the work in [16] uses network coding to exchange data between peers for the purpose of cooperative repair. They consider reducing the effect of lost data in terms of the distortions at the visual level, by appropriate coding at the recovery level. The system proceeds in two logical phases, where in the first phase peers receive a group of packets from the broadcast, then in the second phase the peers broadcast network coded information among themselves. The phases are overlapped with subsequent groups of packets. The information exhanged is such that peers can reconstruct lost packets with high probability.

VII. Conclusion

We presented BOOSTER, an epidemic cooperative repair system that repairs DVB transmissions among DVB receivers over the Internet. Our system works at the Transport Stream level, where packets are conveniently fixed to 188 bytes in length. The system operates between the receiver and the viewer, recovering lost TS packets from other peers before handing them to the viewer. We provide a method for peers to identify identical intervals of TS packets, using the Program Clock Reference (PCR) that occurs in intermittent packets, and to merge or copy intervals for the purpose of converging to a completely repaired interval.

To validate our work we gathered trace data of real DVB transmissions, decoded the packets to gather error statistics, PCR statistics and other data, and simulated the repair algorithm over a large number of peers on a cluster. Our simulations show the feasibility of repairing a DVB transmission via standard ADSL Internet connections where about 5KB/sec upload bandwidth was acceptable for adequate repair in reasonable circumstances. We showed detailed tradeoffs between repair quality and Internet bandwidth used, with respect to DVB errors and the number of peers in the network.

Our system thus far assumes well behaving peers. We have not considered solutions to other aspects such as "free riders". Peer-to-Peer systems are known to be vulnerable to exploitation by free riders, selfish users making use of a P2P service's benefits but refusing to play by the rules and to contribute to it.

In BOOSTER, free riding can manifest itself in two ways. The first involves users who instrument their own client (or network settings) to ignore incoming PULL requests. This falls in the classic domain of free riding, and should be addressed by appropriate incentive mechanisms, such as BitTorrent's tit-for-tat. The second free riding scenario involves users that do not own a DVB-T receiver, but participate in a collaborative repair network to download all packets from other, legitimate users. The implication of this is more severe than free riding on bandwidth resources: as several of these channels are subscription based, one could get to watch them for free.

Improvements such as the tit-for-tat algorithm are orthogonal to our existing work. However we believe that some properties of the underlying system may be useful to incorporate into solutions to the free riders problem. E.g. since pulling a block of TS packets requires to explicitly state the block's boundary PCR values, a node with no access to the original stream will have no way to initiate a pull with two valid 33-bit PCR values. Further work is required to establish how such properties could be applied.

Acknowledgments

We would like to thank the students that volunteered to help us with data collection.

References

[1] U. Ladebusch and C. Liss, "Terrestrial DVB (DVB-T): A broadcast technology for stationary portable and mobile use," *Proc. IEEE*, vol. 94, pp. 183–193, jan. 2006.

[2] M. Claypool and Y. Zhu, "Using interleaving to ameliorate the effects of packet loss in a video stream," in *Proc. 23rd Int. Conf. on Distributed Computing Systems Workshops*, pp. 508–513, may 2003.

[3] J. Cai, X. Li, and C. W. Chen, "Layered unequal loss protection with pre-interleaving for fast progressive image transmission over packet-loss channels," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, no. 4, pp. 338–353, 2005.

[4] S.-r. Kang and D. Loguinov, "Impact of fec overhead on scalable video streaming," in *Proc. Int. workshop on Network and operating systems support for digital audio and video*, (New York, NY, USA), pp. 123–128, ACM, 2005.

[5] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, p. 8, 2007.

[6] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, pp. 197–217, June 2005.

[7] M. Jelasity and O. Babaoglu, "T-Man: Gossip-based overlay topology management," in *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers* (S. A. Brueckner, G. Di Marzo Serugendo, D. Hales, and F. Zambonelli, eds.), vol. 3910 of *Lecture Notes in Computer Science*, pp. 1–15, Springer-Verlag, 2006.

[8] PeerSim. http://peersim.sourceforge.net.

[9] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: estimating latency between arbitrary internet end hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 32, pp. 11–11, July 2002.

[10] E. Weigle, M. Hiltunen, R. Schlichting, V. A. Vaishampayan, and A. A. Chien, "Peer-to-peer error recovery for hybrid satellite-terrestrial networks," in *Proc. IEEE Int. Con. on Peer-to-Peer Computing*, (Los Alamitos, CA, USA), pp. 153–160, IEEE Computer Society, 2006.

[11] F. D. Belleville, L. Dairaine, M. Gineste, and C. Fraboul, "Reducing satellite communication cost using terrestrial peer-to-peer for lost recovery," in *Proc. 23rd AIAA Int. Communication Satellite Systems Conf.*, pp. 1–12, 2005.

[12] P. Sanigepalli, H. Kalva, and B. Furht, "Using p2p networks for error recovery in mbms applications," in *Proc. IEEE Int. Conf. on Multimedia and Expo*, (Los Alamitos, CA, USA), pp. 1685–1688, IEEE Computer Society, 2006.

[13] K. Sinkar, A. Jagirdar, T. Korakis, H. Liu, S. Mathur, and S. Panwar, "Cooperative recovery in heterogeneous mobile networks," in *Proc. 5th Ann. IEEE Communications Society Conf. on Sensor, Mesh and Ad Hoc Communications and Networks*, pp. 395–403, june 2008.

[14] S. Li and S.-H. G. Chan, "Bopper: Wireless video broadcasting with peer-to-peer error recovery," in *Proc. IEEE Int. Conf. on Multimedia and Expo*, pp. 392–395, IEEE, 2007.

[15] S. Raza, D. Li, C.-N. Chuah, and G. Cheung, "Cooperative peer-to-peer repair for wireless multimedia broadcast," in *Proc. IEEE Int. Conf. on Multimedia and Expo*, pp. 1075–1078, IEEE, 2007.

[16] X. Liu, G. Cheung, and C.-N. Chuah, "Deterministic structured network coding for wwan video broadcast with cooperative peer-to-peer repair," in *Proceedings of the International Conference on Image Processing (ICIP)*, pp. 4473–4476, IEEE, Sep. 2010.