FEC-Integrated Network Traffic Shaping Using the NIProxy

FEC-Integrated Network Traffic Shaping Using the NIProxy

Maarten Wijnants Wim Lamotte
Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2, BE-3590 Diepenbeek, Belgium
e-mail: {maarten.wijnants, wim.lamotte}@uhasselt.be

***Abstract*—Optimizing the satisfaction of users of distributed applications is an important yet non-trivial task. Data corruption incurred during network transmission plays a significant role in this context since it is likely to result in user frustration. As a countermeasure, Forward Error Correction (FEC) schemes complement source data with redundant information to enable data recovery at destination-side. This paper discusses the inclusion of adaptive parity-based FEC support in the NIProxy, a context-aware network intermediary which introduces network traffic shaping and multimedia service provision functionality in IP-based networks as tools to influence the user experience. Since FEC coding increases the load on the transportation network, an integrated approach with the NIProxy's network traffic shaping mechanism was adhered to enable deliberate FEC control and to guarantee the bandwidth overhead it introduces is justified. Representative experimental results confirm the validity of our approach and prove that, if applied in a well-considered manner, FEC coding is able to beneficially affect user experience and is hence a valuable addition to the NIProxy's feature list.**

***Keywords*-Multimedia networking, Forward Error Correction, bandwidth brokering, Quality of Experience**

I. Introduction

Exchanging data over packet-switched computer networks can lead to corruption, rendering the data (partly) unusable for the receiver. The type and amount of errors incurred during transmission depend on multiple factors. Broadly speaking however, data corruption can be caused by either the loss of entire packets or by the introduction of biterrors. Insufficiently capacitated network infrastructure (e.g. routers) is a frequent source for packet loss. Transmission errors on the other hand typically result from signal interference and noise on the communication channel, a common issue in wireless networks. Irrespective of its cause, data corruption is likely to have a detrimental effect on user experience and hence effort should be made to minimize it. Lost or contaminated data can for instance lead to hitches in voice streams or visual artifacts in video playback.

Solutions for dealing with data corruption can be categorized into two groups. The first category consists of retransmission-based techniques, where the receiver requests the source to retransmit missing or corrupted data [1]. This process can in theory be repeated indefinitely until all data has been received correctly. In practice however, timing constraints might limit the amount of retransmission attempts for a certain data item or might even render retransmission completely unfeasible. In the Forward Error Correction (FEC) approach on the other hand, the sender supplements the source data with redundant information which allows the receiver to repair, to a certain extent, errors introduced during transmission [2]. FEC schemes in other words enable the destination to recover lost or damaged data without incurring the round-trip time delay overhead introduced in retransmission-based solutions. This is an important advantage in case the network traffic has real-time characteristics like for instance a live video feed.

Despite their radically different methodology, retransmission- and FEC-based schemes share a common disadvantage. They both intrinsically introduce overhead in terms of the amount of data that needs to be transmitted over the communication network. Stated differently, they raise the bandwidth requirements of data streams and hence the load on the network. Because of this drawback, the surprising scenario might occur where the addition of error protection yields an increased instead of a decreased error rate. As a result, deliberate decision making regarding the amount of protection to add to network traffic is advocated.

The subject of this paper is the NIProxy, a network intermediary which aims to improve the experience of users of networked applications by performing network traffic shaping as well as multimedia service provisioning, hereby exploiting relevant contextual information [3]. Given its negative impact on user experience, techniques to counter lost or damaged data seemed like a meaningful extension of the NIProxy's feature list. The main contributions of this paper are hence a description of how FEC-based error protection functionality was introduced in the NIProxy and an investigation of its influence on user satisfaction. The decision to opt for a FEC solution instead of a retransmission scheme is motivated by the NIProxy's focus on real-time network traffic. FEC support was adaptively incorporated in the NIProxy to enable dynamic control of the error protection process on a per-flow basis. More importantly, an integrated approach with the NIProxy's network traffic shaping mechanism was adhered, this way guaranteeing that error protection coding reckons with contextual information (e.g. the currently prevailing channel conditions) and in addition will actually result in an optimization of the user's experience.

The outline for the remainder of this paper is as follows. Section II introduces the NIProxy network intermediary. The user experience optimization techniques currently supported by the NIProxy form the topic of the next two sections. Section V describes how FEC-based error protection was incorporated in the NIProxy. This inclusion and its impact on user experience is subsequently evaluated in Section VI. Section VII summarizes related work. Finally, Section VIII presents our conclusion and suggests possible future research directions.

II. Network Intelligence Proxy

The objective of the Network Intelligence Proxy (NIProxy) consists of optimizing the experience of users of networked applications. In research contexts, user satisfaction is often referred to as Quality of Experience (QoE, see for instance [4]). Guaranteeing a high QoE is particularly challenging for distributed applications because, compared to standalone systems, these involve an additional networking aspect which might introduce significant complications. As an example, streaming multimedia content imposes high and strict requirements on the networking substrate, for instance in terms of bandwidth guarantees and delay bounds. It is consequently not uncommon that trade-offs need to be made due to network resources being insufficiently available. Whenever such situations arise, the experience of the end-user should always be reckoned with, i.e. the detrimental impact of each trade-off decision on the user satisfaction should be minimized.

As alluded to by its name, the NIProxy's methodology to obtain its objective is to introduce additional "intelligence" in the networking infrastructure [3]. This intelligence is accumulated by collecting different types of contextual information. The NIProxy at the moment queries two distinct sources to fill up its context repository. The first source is the transportation network and the contextual knowledge in this case takes the form of network-related measurements and statistics like for instance the current bandwidth capacity of communication links. This type of context is obtained through performing active network probing. The second context source is the end-user application. Applications can provide the NIProxy with any application-related knowledge they deem relevant. An example could for instance be the relative importance of the different (categories of) network flows incorporated by the application. To streamline this process, a support library has been developed which enables applications to relay application-specific information to the NIProxy with minimal effort and without requiring drastic modifications to the application software [3].

Gathering contextual information by itself of course does not enhance the user experience, it is merely the means which enables it. The NIProxy exerts its contextual awareness to outfit IP-based networks with two traffic engineering techniques which improve their multimedia traffic handling capabilities and which hence enable them to influence user satisfaction. These techniques will be discussed in the next two sections.

III. Network Traffic Shaping

The first QoE optimization mechanism provided by the NIProxy is network traffic shaping. The accumulated contextual knowledge is in this case hence exploited to orchestrate the bandwidth consumption of distributed applications. How this is achieved exactly will be described in this section.

A. NI Stream Hierarchy

The NIProxy implements network traffic shaping by arranging network flows in a *NI stream hierarchy* [5]. The goal of this tree-like structure is to express the relationships that exist between network traffic introduced by a distributed application. Actual network streams are represented by leaf nodes in the NI stream hierarchy. Internal nodes on the other hand steer the traffic engineering as each implements a certain bandwidth distribution strategy.

The network traffic shaping process is controlled entirely by the choice of types of internal nodes to use and the way these nodes are composed to model the general layout of the NI stream hierarchy. Once this layout has been designed and assuming the NI stream hierarchy is kept up-to-date (e.g. newly initiated network flows are adequately incorporated), performing network traffic shaping becomes as simple as appointing the correct bandwidth amount to the hierarchy root node. The internal nodes, commencing with the root node, will automatically start distributing the bandwidth value they have been assigned among their children according to the bandwidth distribution approach they implement. Eventually, portions of the total bandwidth capacity will reach one or more leaf nodes, at which point this bandwidth amount is reserved for the transmission of the network stream that is associated with the leaf node.

B. Internal NI Hierarchy Node Types

The internal NI stream hierarchy nodes dictate the bandwidth brokering process by allocating bandwidth to their children in a specific manner. Different classes of internal nodes are available to structure the NI stream hierarchy [5]. Table I summarizes the behavior and mode of operation of the internal node types which played a role in the experimental evaluation performed for this paper.

C. Leaf NI Hierarchy Node Types

As is the case for internal NI hierarchy nodes, there also exist multiple leaf node types [5]. The classification of leaf nodes is based on their capabilities to control the bandwidth consumption of the network stream they represent. Leaf nodes belonging to the *discrete* category are limited to switching the bandwidth usage of their associated network flow between a discrete number of levels. In its most

Table I
BANDWIDTH DISTRIBUTION BEHAVIOR OF INTERNAL NI HIERARCHY NODE TYPES

Node Type	Behavior
`Mutex`	Implements mutex behavior in that the available bandwidth BW is allotted in its entirety to the child node with the largest still satisfiable bandwidth requirement; none of the other children receive any bandwidth. In case no child node with reconcilable bandwidth needs exists, the `Mutex` will consume no bandwidth at all.
`Percentage`	Implements a two-phase bandwidth distribution process. In the initial phase, each child c is granted its corresponding *percentage value* $p_c \in [0, 1]$ of the distributable bandwidth BW, i.e. child c is apportioned a bandwidth amount $BW_c = p_c \times BW$. Child percentage values are scaled so that they sum up to *1*. After executing phase 1, a portion of BW might be left unallocated due to children not fully consuming their assigned bandwidth percentage. If so, the second phase attempts to distribute this excess bandwidth by assigning it to child nodes on a one-by-one basis, in order of decreasing percentage value. The second phase in other words allows children to upgrade their bandwidth consumption based on unused bandwidth inherited from phase 1, hereby favoring children with a higher percentage value.

rudimentary form, a discrete leaf node supports only two such levels, corresponding to respectively a zero and maximal stream bandwidth consumption. Such nodes are hence confined to turning their associated network flow off and on and can consequently only effectively represent single-quality network flows in the NI stream hierarchy. Contrary to discrete leaf nodes, the class of *continuous* leaves is able to modify their associated stream's bandwidth consumption in a continuous manner. Stated differently, these nodes define a bandwidth consumption range and provide functionality to set their corresponding flow's bandwidth usage to an arbitrary value within this range. Continuous leaf nodes are more powerful than discrete leaves as they enable additional flexibility and dynamism to be introduced in the network traffic shaping process. This however comes at the expense of increased resource consumption and processing requirements.

D. Sibling Dependencies Framework

A secondary contribution of this paper is the incorporation of a framework in the NIProxy's network traffic shaping mechanism that enables dependencies to be enforced between sibling nodes in the NI stream hierarchy. Currently only the `SD_BW_ALLOC_CONSTRAINED` dependency has been defined but the set of supported sibling dependency types could readily be extended, for example based on future network traffic shaping requirements. The `SD_BW_ALLOC_CONSTRAINED` dependency introduces a bandwidth allocation constraint between siblings in the NI stream hierarchy. In particular, the existence of such a dependency between sibling nodes A and B specifies that B is allowed to consume bandwidth if and only if A's bandwidth consumption is non-zero. In case this constraint is violated, node A is allowed to alter the current bandwidth distribution by "borrowing" the bandwidth amount assigned to node B. If the combined bandwidth reserved for nodes A and B remains insufficient to satisfy node A's minimal bandwidth requirements, both nodes will be turned off and a new network traffic shaping iteration will be performed from which the subtrees that are rooted at nodes A and B are excluded. As a result, the bandwidth that was originally allocated to node B will be distributed over the other nodes in the NI stream hierarchy (recall that node A's bandwidth consumption was zero in the initial iteration). If on the other hand the additional bandwidth enables node A to switch to its minimal non-zero bandwidth consumption, node A is allowed to do so; any bandwidth that remains will subsequently be made available to node B. Note that this latter amount will inherently be smaller than the bandwidth that was at first reserved for node B, meaning node B might be forced to lower its bandwidth consumption or might even get completely disabled.

IV. MULTIMEDIA SERVICE PROVISION

As a second technique to boost their multimedia handling capabilities, the NIProxy introduces the possibility to perform processing on multimedia traffic inside communication networks. This is achieved by acting as a service provision platform, meaning the NIProxy enables the in-network execution of *services* on transported data. The range of services that can be provided is theoretically limitless, spanning from lightweight data filtering to flow aggregation and even adaptation. In previous work, we for example presented a service which enables the NIProxy to modify the bitrate of video streams by on-the-fly transcoding them to a lower quality [3]. Like the network traffic shaping mechanism, provided services have access to the NIProxy's context repository and can exploit it to make sure the processing they implement will actually lead to an improvement of the end-user experience. The contextual knowledge could however also be exploited for secondary purposes, for instance to optimize the efficiency of resource-intensive services.

A. Services as NIProxy Plug-Ins

To guarantee maximal flexibility and to achieve run-time extensibility of the NIProxy's functionality, each service is

implemented as a plug-in that can be (un)loaded dynamically during NIProxy execution. By opting for a plug-in based design, services become standalone entities that are conceptually separated not only from each other but also from the NIProxy's general software architecture. This separation confers several advantages like for instance third-party service development [3]. Despite their isolation from each other, services are still able to cooperate. Multiple services, each implementing a well-delineated function, can namely be combined to form a service chain. Services belonging to a chain are executed consecutively so that the output produced by each service serves as input for the next service in the chain, this way achieving collaboration.

B. Network Traffic Shaping Interoperation

Besides inter-service collaboration through service chaining, the NIProxy also enables services to interoperate with its network traffic shaping mechanism. To be more precise, services are able to query and even influence the network traffic shaping strategy devised for NIProxy clients. An exemplification hereof is again provided by the previously mentioned video transcoding service [3]. This service introduces a new type of network traffic, i.e. the transcoded version of original video flows. To inform the NIProxy of the existence of this new flow type and to ensure it is taken into consideration during bandwidth brokering, the video transcoding service extends the NI stream hierarchy by incorporating nodes representing transcoded video flows. Once incorporated, the NIProxy's network traffic shaping mechanism can start making deliberate decisions about which video version to assign bandwidth to. These decisions in turn influence the operation of the video transcoding service, since the service will only transcode incoming video flows in case consultation of the NI stream hierarchy indicates the transcoded video flow is currently enabled. As a result, unnecessary transcoding operations are avoided.

Practical experience and empirical evaluation has shown supporting service and network traffic shaping interfacing to be a powerful feature since it enables the implementation of highly effective and resource-efficient services. More importantly however, we put forward the proposition that it unlocks end-user QoE optimization possibilities and results that could not be attained by applying both mechanisms independently. Illustrations of this argument were already provided in our previous work (e.g. [3]) and it will again be confirmed in the evaluation section of this paper.

V. FEC Integration in NIProxy

A. XOR-Based Parity Coding

A large variety of schemes to enable receiver-side correction of transmission errors exists [2]. Each has its particular characteristics in terms of error correction capabilities, computational complexity, information rate (the ratio between media data and redundancy), etcetera. For this paper, we opted for a parity-based technique that relies on bitwise XOR (eXclusive OR) encoding [6]. As input, this technique accepts a group of n media packets and produces as output a single parity packet. This parity packet is constructed by applying the XOR operator on the bits stored at identical locations in the n media packets and subsequently saving the outcome at the corresponding location in the parity packet. Each bit in the parity packet in other words represents the parity of the equivalent bits in the input packets. At decoding side, the parity packet can be used to recover a singly lost or corrupted media packet that contributed to its construction. This is achieved by XOR-ing the $(n-1)$ correctly received media packets with the (also perfectly received) parity packet. In case the packet is corrupted instead of completely lost, the number of introduced biterrors is irrelevant to the recovery process; as long as all biterrors are confined to a single packet per input group, recovery will be perfect.

Our choice for an XOR-based parity scheme is motivated by the fact that it exhibits a number of characteristics that make it very suitable for inclusion in a middleware system aiming to improve the end-user experience. First of all, the scheme has maximal usability as it is a generic technique that is not bound to a particular media type. Adaptability is another very important asset as protection can at runtime be traded off for bandwidth consumption by varying the size of the input packet group used to constitute a single parity packet. A certain level of scalability is also guaranteed due to the scheme being lightweight in terms of computational requirements. Final advantages include ease of implementation, the existence of an efficient and standardized transport mechanism for the redundant data [6] and the backward compatibility of the scheme with FEC-agnostic receivers [6]. The main disadvantage of the parity approach is that it is a relatively bandwidth consuming technique as it generates additional packets that need to be transmitted over the transportation network. This introduces significant overhead since each packet requires its own protocol headers and, as a result, schemes with higher information rates exist. This effect is further aggravated in case the sizes of the to-be-protected media packets exhibit high variability, as it imposes the need for extensive padding [2].

B. NI Stream Hierarchy Incorporation

Like any other FEC technique, parity coding introduces overhead in terms of redundant data that needs to be transmitted over the communication network. Since this FEC-generated network traffic might consume significant amounts of bandwidth, it should be reckoned with by the NIProxy's network traffic shaping mechanism. This in turn necessitates its integration in the NI stream hierarchy. Redundant parity information is therefore represented as a discrete NI stream hierarchy leaf node which provides a discrete bandwidth consumption level for each supported input packet grouping size (plus an additional level corresponding to a zero

bandwidth usage to indicate that parity coding should be disabled). Merely representing the FEC data does not suffice however, it also needs to be adequately related to the media stream it protects as any bandwidth allocated to the FEC information can no longer be consumed by the media flow itself. The objective is hence to amortize the bandwidth that has been reserved for FEC-protected traffic among the media data and its FEC overhead in such a manner that the reception at the destination is optimized. This process is often referred to as Joint Source-Channel Coding (JSCC) and its impact on user QoE should be apparent. As will be demonstrated in Section VI, in this paper we rely on the `Percentage` node type to group together a media flow with its FEC protection in the NI stream hierarchy. By adjusting the percentage values assigned to both nodes, the JSCC process can be controlled. It should be noted however that our current approach might prove to be too coarse and rudimentary to be practically usable in realistic environments. The results that will be presented later on in this paper can consequently be considered to be preliminary as one of the objectives of our future work is to investigate the integration of more fine-grained JSCC support in the NIProxy. Finally, as the FEC redundancy is rendered useless in case the media data it protects is not received by the destination, a sibling dependency of type `SD_BW_ALLOC_CONSTRAINED` is defined between the NI stream hierarchy nodes representing the media and its FEC protection. As described in Section III-D, doing so installs the condition that FEC traffic can consume bandwidth if and only if its associated media flow is currently enabled.

C. Implementation

FEC support was not incorporated in the NIProxy as an integral part of its general software architecture but instead as a NIProxy service. As explained previously in Section IV, the advantage of this design decision is that FEC-related functionality will only be loaded in case it is effectively needed and that any issues it might introduce will not contaminate the NIProxy's basic implementation. The service accepts as input original media data and supplements it with redundant parity information. The plug-in's exact mode of operation is as follows. First of all, it registers interest for the class of network streams carrying data that is eligible for FEC protection[1]. On the discovery of each such network stream, the service performs two initialization tasks. The first task consists of instantiating a FEC encoder for the newly discovered flow, in this case an XOR-based parity coder. Secondly, as explained in Section V-B, the service informs the bandwidth brokering process of the possibility to FEC protect the media stream and the thereby associated bandwidth requirements. On completion of this initialization phase, the network flow is added to the service's main processing loop. Each iteration of this loop starts with the service exploiting its interface with the network traffic shaping mechanism to determine the discrete level to which the FEC data for the media flow that is being processed is currently set. The corresponding FEC encoder is subsequently switched to the input grouping size that is associated with this level, after which it is fed with the input media packet. In case this packet completed a protection group, the media packet is outputted together with the resulting parity packet. If not, only the media packet is returned as output of the service. As an exception, if the FEC stream's current discrete level corresponds with a zero bandwidth consumption, FEC processing is bypassed by simply not handing over the input packet to the parity encoder.

[1]NIProxy services are able to control the data they are handed over by registering interest for certain network stream types or even individual network streams [3]. Only data transported on these streams will be provided to the service as input.

D. Supporting Other FEC Techniques

It is important to note that the NIProxy's FEC support does not need to be limited to parity-based coding. Other FEC schemes such as the popular ReedSolomon (RS) code [2] could just as well be incorporated. Analogous to the way parity coding was integrated, this would involve the introduction of a new FEC encoder in the NIProxy as well as a representation of the network traffic it generates in the NI stream hierarchy. Being able to fall back on a catalog of FEC schemes would likely improve the NIProxy's performance as the effectiveness of individual FEC approaches tend to vary considerably depending on environmental factors like for instance the current characteristics of the data corruption process. In case multiple FEC codes would be supported, the NIProxy could exploit its contextual awareness to estimate the beneficial impact of each code on the user QoE under the present conditions. Based on these estimations, the optimal FEC technique could subsequently be selected. It would even be possible to enable collaboration between different FEC techniques through concatenation in case this would benefit the end-user experience.

VI. Evaluation

This section harbors experimental results which comprehensively demonstrate the added value of integrating FEC-based error protection functionality in the NIProxy. In particular, the advantageous influence on the NIProxy's user QoE optimization capabilities will be evaluated by analyzing the outcome of a practical experiment that was conducted multiple times under varying circumstances. The presented experiment, which will be described in detail in Section VI-A, was deliberately kept simple to ensure the reader's attention is focused on the specific contributions of this paper and to enable intelligible distillation of their impact from the produced results. Furthermore, despite the experiment being minimalist, the presented results are representative as they

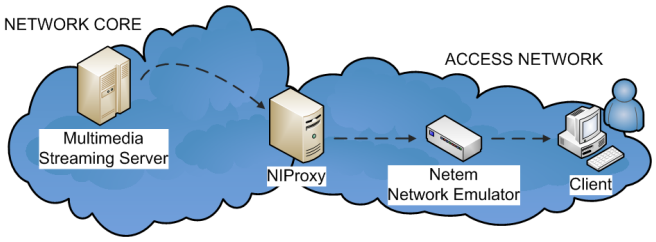

Figure 1. Experimental setup

can be extrapolated to realistic contexts in a straightforward manner.

A. Experiment Description and Setup

The experiment simulated a video streaming scenario between a multimedia server and a receiving client. As depicted in Figure 1, the server was conceptually deployed somewhere inside a high-capacity and relatively error-free network backbone. The client on the other hand was located at the periphery of the network and was connected to the backbone through a resource constrained access network (the so-called last mile). In between these two end-hosts, a NIProxy component was interposed which was responsible for engineering the network traffic destined for the receiving client. The objective was hence to determine whether the user benefitted from the inclusion of the NIProxy in the experiment. The NIProxy instance was conceptually deployed on the boundary between the backbone and the access network. This location was chosen as it represents a crucial junction point in the simulated network topology where data emitted by the multimedia server needed to transfer from the resource-abundant network core to the much less capacitated and possibly error-prone access network. Finally, to be able to emulate packet loss on the client's access link, the experimental setup also included an instance of the netem network emulator [7].

Within this experimental setup, the multimedia server maintained two simultaneous RTP video sessions VS1 and VS2 with the receiving client. The streaming server emitted video data in unprotected form (i.e. without FEC information) as the network core was assumed to be nearly error-free. The NIProxy instance however had its FEC service loaded and was hence able to add XOR-based parity protection to transiting network traffic. Parity coding could be performed per 3 or per 6 input packets. Recall that resorting to a smaller input grouping size increases the error recovery possibilities at the expense of elevating the bandwidth required for transporting the parity information. To unambiguously demonstrate the effect of the FEC processing and the bandwidth requirements it imposed, in the experiment only video session VS2 was marked as being eligible for receiving FEC protection. For the comparison with unprotected flow VS1 to be meaningful, an identical video fragment was streamed over both sessions. Besides the FEC service, the NIProxy also made use of its video transcoding service and was consequently able to address bandwidth shortage on the access network by reducing the bitrate of incoming video streams. The quality parameters of the employed video fragment as well as the output settings of the video transcoding process are listed in Table II.

Table II
VIDEO ENCODING PARAMETERS

	Resolution	**Framerate**	**Bitrate**	**Codec**
Original	320×240	20	120.000	H.263+
Transcoded	176×144	15	60.000	H.263+

Figure 2 depicts the NI stream hierarchy which steered the transmission of the two video flows over the receiving client's access connection during the experiment. The root node was of type `Percentage` and had as children two subtrees which each represented one of the video flows emitted by the streaming server. Both subtrees were assigned a percentage value of 0.5 to specify that the bandwidth capacity of the access link should at all times be split perfectly fair over both video connections. This was again decided to enable meaningful comparison of the way both video streams were treated by the NIProxy as well as of their reception at client-side. The leftmost subtree corresponded with video session VS1 that was not qualified for receiving FEC protection from the NIProxy. This subtree comprised two discrete leaf nodes which respectively represented the original and transcoded version (OV and TV) of the transported video fragment and which were differentiated from each other using an internal node of type `Mutex`. Both leaf nodes supported two discrete bandwidth consumption levels representing the extremes of respectively obstructing their associated network stream and forwarding it at its maximal rate. An analogous construction can be found in the rightmost subtree, which however also included a number of additional nodes as it corresponded with video flow VS2 that was considered for being FEC protected by the NIProxy. The XOR-based parity FEC data was incorporated as a discrete leaf node providing three discrete bandwidth levels, one for each supported input grouping size plus an additional level associated with a zero bandwidth consumption. As previously discussed in Section V-B, JSCC was implemented by making this FEC node a sibling of the quality grouping `Mutex` node using a `Percentage` node as parent. In this experiment, a static JSCC approach was employed where 90 percent of the available bandwidth was always assigned to the media stream itself, while the remaining 10 percent was reserved for the parity data. Also observe the sibling dependency of type `SD_BW_ALLOC_CONSTRAINED` that was defined between the quality grouping `Mutex` and the

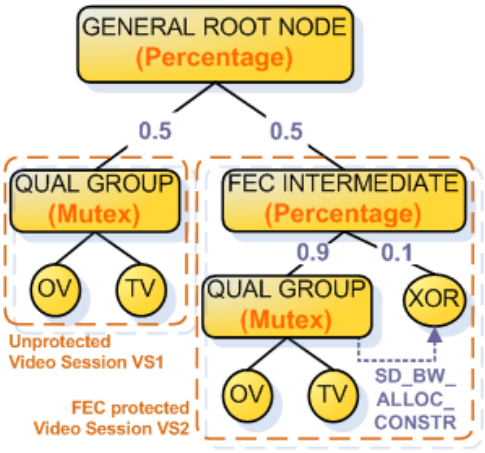

Figure 2. NI stream hierarchy which steered the shaping of the network traffic destined for the receiving client during the experiment

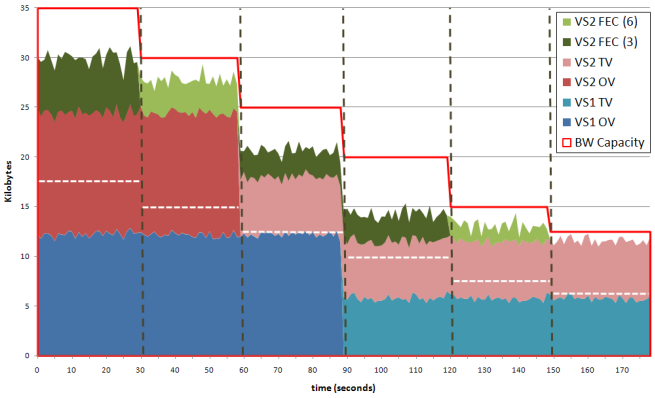

Figure 3. Stacked graph illustrating the network traffic received by the destined client during the error-free execution of the experiment

FEC leaf node.

The described experiment was executed twice, once without and once with the netem component introducing packet loss on the last mile. During each experiment execution, all conditions remained constant, except for the bandwidth capacity of the access link. Artificially modifying the access network throughput at predefined points in time enabled us to investigate the effect of these bandwidth fluctuations on the way the NIProxy shaped the network traffic destined for the receiving client. Five such bandwidth modifications were performed throughout the experiment, causing it to be conceptually divided into six discrete intervals.

B. No Packet Loss

The experiment was first executed in an error-free environment to enable complete and perfect tracing of the way the video traffic transited the access network and arrived at the destined client. The outcome is plotted in Figure 3. In this network chart, the solid red line specifies the bandwidth capacity of the access connection, whereas the dashed vertical lines separate the consecutive experiment intervals. As can been seen, the interval transitions occurred at interspaces of approximately 30 seconds and always coincided with an (artificial) reduction in access bandwidth. Finally, the dashed horizontal lines indicate the bandwidth percentages reserved for both video connections during the different experiment intervals. As each connection was assigned an equal percentage value in the NI stream hierarchy, the bandwidth capacity of the access link was cut in halve and split perfectly fair over both. Notice however that in the case of video session VS2, the allocated bandwidth amount needed to be distributed over the video data itself and its FEC protection.

Except during the first experiment interval, the NIProxy was forced to apply network traffic engineering due to access bandwidth being insufficiently available to inject all involved network streams at their maximal rate into the last mile. In the third interval for example the client received VS1 at full quality and the parity coding for VS2 also ran at maximal bandwidth consumption; VS2 itself however was transcoded to a lower quality by the NIProxy before it was relayed to the access network. As the experiment progressed, access bandwidth became gradually more constrained and consequently increasing bandwidth reductions needed to be enforced for the involved network streams, up to the point where FEC coding was even completely disabled in the last experiment interval. Also observe from the network trace that reducing the bandwidth consumption of the media data yielded an equivalent reduction in the bandwidth requirements of its FEC protection. This is explained by the fact that as data packets decrease in size, so do the packets carrying parity information. Finally, the network graph also reveals that the available access bandwidth was not always fully exploited. This is most pronounced in experiment interval 4 where approximately a quarter of the bandwidth capacity remained unallocated. This behavior was nonetheless justified as in these situations none of the involved network flows could be switched to a higher bandwidth consumption level, either because doing so would violate the current bandwidth constraints or because they were already running at their maximal rate.

C. 10 Percent Packet Loss

In the second iteration of the experiment, the access connection suffered from 10 percent packet loss, randomly introduced by the netem component of our experimental setup. As illustrated in Table III, this caused video data as well as FEC protection packets to be lost. On the other hand, it also resulted in the FEC data being put to meaningful use, i.e. to reconstruct lost packets at receiver-side. Table III therefore also includes packet recovery statistics for video stream VS2. Looking at the table, we see that the redundant parity information enabled the receiving client

Table III
PACKET LOSS AND RECOVERY STATISTICS (10 PERCENT PACKET LOSS RANDOMLY INTRODUCED ON ACCESS LINK)

	Experiment Interval						Total
	1	**2**	**3**	**4**	**5**	**6**	
# Lost							
VS1	47	45	52	32	21	27	224
VS2	56	31	38	34	27	24	210
FEC	15	6	11	11	3	0	46
% Lost							
VS1	11.03	10.23	11.79	9.07	7.09	9.64	10.02
VS2	12.33	7.11	10.83	11.53	9.12	8.39	9.92
FEC	10.27	6.38	12.09	11.22	4.69	0	9.33
# Recovered	40	18	20	22	18	0	118
% Recovered	71.43	58.06	52.63	64.71	66.67	0	56.19

to recreate 56.19 percent of the packets that were lost on video session VS2, yielding a residual loss of 92 packets instead of the original 210. Besides collecting packet loss and recovery statistics, data reception at client-side was again also recorded. Unsurprisingly, the resulting network trace perfectly resembled the chart depicted in Figure 3, except for the bandwidth consumption of the monitored streams this time displaying occasional irregularities (i.e. drops) caused by the packet loss. As the network trace does not convey extra information nor provides any additional insight, it was omitted from this paper.

Executing the experiment in an error-free environment yielded perfect video playback at the destination. This was unfortunately no longer the case in this iteration of the experiment. As the destination did not always have all video packets at its disposal, decoding issues arose which in turn caused (sometimes severe) perceptual artifacts to be introduced in the decoded video. The availability of FEC information for VS2 however enabled the receiver to repair a substantial fraction of the packets that went missing on this session. Compared to video stream VS1, the playback of VS2 was hence significantly less distorted as decoding artifacts were much less pronounced. Space limitations unfortunately refrain us from providing screenshots to endorse this statement.

D. Discussion

A number of important findings can be deduced from the network chart presented in Figure 3. First of all, it proves that the NIProxy shaped the network traffic destined for the client in such a manner that the capacity of the client's access connection was at all times respected. A second observation is that the bandwidth distribution strategy delineated for this experiment was successfully put into effect by the NIProxy, as the available access bandwidth was shared equitably among the involved video sessions. Either network connection was only allowed to raise its bandwidth consumption beyond its "fair share" in case spare bandwidth originally reserved for the other connection was available. This is a direct consequence of the two-phase bandwidth distribution approach implemented by the `Percentage` NI stream hierarchy node and it resulted in a more complete exploitation of the access bandwidth capacity. Notice that this observation is not limited to the way bandwidth was shared among the video connections themselves but that it also applied to the JSCC process for the FEC protected video stream. This is exemplified in experiment intervals 3 and 4, where the FEC data stream was initially entitled to a bandwidth percentage that barely sufficed to perform parity coding per 6 input packets. As the video data it protected however did not fully consume its reserved bandwidth amount, the FEC stream was able to claim this excess bandwidth and exploited it to switch to a grouping size of 3 packets. Third, the presented experimental results are an interesting exemplification of the potential of supporting interoperation between NIProxy services and its bandwidth brokering operations. In particular, the network trace demonstrates how the JSCC process (i.e. the amount of client bandwidth spent on FEC data) was directed entirely by the NIProxy's network traffic shaping mechanism[2]. Notice

[2]As stated previously, the current implementation of the JSCC control leaves room for considerable improvement as it should more accurately and flexibly exploit contextual knowledge.

that JSCC might result in the need to reduce the quality of the multimedia data to accommodate its FEC protection flow. In the presented experiment, this occurred in the third interval, where sufficient bandwidth was available to forward video stream VS2 at its maximal rate. However, as its associated FEC data was entitled to a fraction of this bandwidth capacity, VS2 itself needed to be transcoded to a lower quality. In contrast, since video stream VS1 was not subject to FEC protection, the client received VS1 in original quality during this experiment interval. Remark that all discussed findings also apply to the second iteration of the experiment as the NIProxy enforced an identical traffic shaping strategy in both experiment executions.

The discussion thus far has revealed that FEC protecting a multimedia flow does not come for free as it imposes an unnegligible bandwidth overhead. The bandwidth consumed by the redundant information however enables missing or corrupted data to be repaired at the destination. The beneficial impact hereof cannot be inferred from the presented network trace but instead is comprehensively highlighted by the loss and recovery statistics provided in Table III. In the third experiment interval for instance, recall that it was necessary to transcode video stream VS2 to a lower bitrate to accommodate its FEC protection. Table III however demonstrates that the availability of the FEC data in this interval enabled the receiving client to reconstruct 52.63 percent of the packets belonging to video stream VS2 that were corrupted during their passage through the noisy access network. In contrast, in the same experiment interval the client received video stream VS1 at original quality but as this stream lacked FEC protection, none of its 52 lost packets could be recovered. Comparing the playback of VS1 and VS2 on the end-user device, the latter was much smoother and less perceptually degraded. It is our belief that a lower quality but only mildly distorted version of a video fragment yields a more enjoyable viewing experience than a high-quality video fragment displaying grave visual deformations and/or temporal interruptions caused by unaccounted for transmission errors. Notice that this claim is highly subjective and has not yet been formally confirmed by subjecting the QoE optimization attempts executed by the NIProxy to a qualitative user study. In any case, if a user does not share our vision and actually prefers distorted high-quality video over a less distorted lower quality variant, it is possible to prevent the network traffic destined for this user from receiving FEC protection from the NIProxy. In fact, FEC support was incorporated in the NIProxy in a sufficiently flexible manner so that such decisions can even be made on a per-flow basis.

VII. Related Work

As means to improve user satisfaction, the NIProxy network intermediary provides network traffic shaping as well as multimedia service provision functionality. Both techniques are topics of active research. Interesting related work on network traffic shaping and bandwidth brokering includes the utility-based bandwidth partitioning scheme proposed in [8] and the mathematical study by Massoulié and Roberts [9]. The NIProxy's multimedia service provision mechanism on the other hand is largely related to the Service Oriented Architecture (SOA) paradigm [10] and the Active Networking (AN) design philosophy [11]. The Protocol Boosters approach [12] and the general purpose proxy filtering mechanism presented in [13] are concrete examples of platforms which, like the NIProxy, enable in-network processing of transported data. Finally, the NIProxy shows substantial interfaces with architectures that are concerned with Quality of Service (QoS) provision such as the Congestion Manager middleware [14] and the OverQoS framework [15]. The NIProxy consequently does not innovate in terms of the objectives it sets forth nor the techniques it employs to achieve them. What does distinguish the NIProxy from related research is that it integrates both techniques in a single system and in addition does so in an interoperable and collaboration-enabled manner. As is again validated in this paper, this integrated approach confers a number of important advantages and unlocks potent user QoE optimization possibilities. Also, the NIProxy's awareness includes application-related context, a type of knowledge that is often left unconsidered in other approaches.

This paper specifically addressed the incorporation of FEC support in the NIProxy. An excellent reference work on error correction coding is the book by Moon [2]. An important consequence of FEC-based solutions is the need to perform Joint Source-Channel Coding (JSCC). Examples of proposed JSCC strategies and implementations can be found in [16], [17] and [18]. It is important to note that every JSCC approach could theoretically be incorporated in the NIProxy by implementing its behavior and mode of operation in a new type of internal node for the NI stream hierarchy.

VIII. Conclusion and Future Work

Multimedia data destined for clients of distributed applications might arrive in corrupted form or could even be partially lost during its propagation through error-prone transportation networks. The typical outcome is a deteriorated media presentation at receiver-side which in turn is a likely source for user frustration. Forward Error Correction (FEC) schemes address this issue by adding redundancy to the transported data to enable receivers to repair compromised or lost information. Given its ability to negate or at least alleviate the detrimental effects of data corruption, it was decided to introduce FEC coding in the NIProxy, a network intermediary which strives to optimize the experience witnessed by users of distributed applications. FEC support was incorporated in the form of an adaptive XOR-based parity coder whose operation is directed by the NIProxy's network traffic shaping mechanism to ensure the bandwidth overhead

it introduces is justifiable and is adequately weighed against not only the multimedia stream it protects but also any other network traffic that is being exchanged as part of the distributed application. The FEC inclusion was practically evaluated using a video streaming use case. The resulting experimental results comprehensively corroborate that adaptive FEC support was successfully incorporated in the NIProxy and in addition demonstrate that it is a valuable addition to the NIProxy's toolset to improve the experience of users of distributed applications.

Our future research directions have already been mentioned throughout this paper. To recapitulate, these include extending the NIProxy's FEC support with techniques other than XOR-based parity protection (e.g. RS coding) and the introduction of more powerful and effective JSCC algorithms. Regarding the JSCC topic, a first important improvement would be to make the division of bandwidth among the media data and its FEC protection dynamic so that it at least takes the current loss characteristics into account. Secondly, it might turn out beneficial to design a new type of internal NI stream hierarchy node to direct the JSCC instead of relying on an existing type, as none of these might be capable of efficiently modeling the JSCC process. Finally, besides performing implementational adjustments, we also plan to organize user studies to obtain qualitative feedback regarding the user experience optimization attempts performed by the NIProxy.

ACKNOWLEDGMENTS

This research is part of the IBBT Gr@sp project, funded by the Flemish government. Part of this research is also funded by the EFRD.

REFERENCES

[1] A. S. Tanenbaum, *Computer Networks*, 4th ed. Prentice-Hall PTR Publishing, 2005.

[2] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[3] M. Wijnants and W. Lamotte, "The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision," in *Proc. IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, Helsinki, Finland, June 2007.

[4] T. Deryckere, W. Joseph, L. Martens, L. D. Marez, K. D. Moor, and K. Berte, "A Software Tool to Relate Technical Performance to User Experience in a Mobile Context," in *Proc. 3rd IEEE Workshop on Advanced Experimental Activities on Wireless Networks & Systems (EXPONWIRELESS 2008)*, Newport Beach, CA, USA, June 2008.

[5] M. Wijnants and W. Lamotte, "Managing Client Bandwidth in the Presence of Both Real-Time and non Real-Time Network Traffic," in *Proc. 3rd IEEE International Conference on COMmunication System softWAre and MiddlewaRE (COMSWARE 2008)*, Bangalore, India, January 2008.

[6] A. Li, "RTP Payload Format for Generic Forward Error Correction," Internet Engineering Task Force, RFC 5109, December 2007.

[7] Net:Netem - The Linux Foundation, http://www.linuxfoundation.org/en/Net:Netem.

[8] V. Rakocevic, J. Griffiths, and G. Cope, "Performance Analysis of Bandwidth Allocation Schemes in Multiservice IP Networks using Utility Functions," in *Proc. International Teletraffic Congress (ITC17)*, Salvador da Bahia, Brazil, December 2001.

[9] L. Massoulié and J. Roberts, "Bandwidth Sharing: Objectives and Algorithms," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, pp. 320–328, June 2002.

[10] Web Services and Service-Oriented Architectures, http://www.service-architecture.com/index.html.

[11] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, January 1997.

[12] W. S. Marcus, I. Hadzic, A. J. McAuley, and J. M. Smith, "Protocol Boosters: Applying Programmability to Network Infrastructures," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 79–83, October 1998.

[13] B. Zenel, "A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment," *Wireless Networks*, vol. 5, no. 5, pp. 391–409, September 1999.

[14] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, "System Support for Bandwidth Management and Content Adaptation in Internet Applications," in *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI 2000)*, San Diego, CA, USA, October 2000, pp. 213–226.

[15] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, "OverQoS: An Overlay based Architecture for Enhancing Internet QoS," in *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI 2004)*, San Francisco, CA, USA, March 2004, pp. 71–84.

[16] P. Frossard and O. Verscheure, "Joint Source/FEC Rate Selection for Quality-Optimal MPEG-2 Video Delivery," *IEEE Transactions on Image Processing*, vol. 10, no. 12, pp. 1815–1825, December 2001.

[17] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive FEC-Based Error Control for Internet Telephony," in *Proc. 18th IEEE Conference on Computer Communications (INFOCOM 1999)*, New York, NY, USA, March 1999, pp. 1453–1460.

[18] F. S. Filho, E. H. Watanabe, and E. de Souza e Silva, "Adaptive Forward Error Correction for Interactive Streaming Over the Internet," in *Proc. 49th IEEE Global Telecommunications Conference (GLOBECOM 2006)*, San Francisco, CA, USA, November 2006, pp. 1–6.