Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access

Anita Sobe, Laszlo Böszörmenyi, Mario Taschwer
Institute of Information Technology
Klagenfurt University
Austria
{anita, laszlo, mt}@itec.uni-klu.ac.at

***Abstract*—The contributions of this paper are threefold: (1) the extensive introduction of a formal Video Notation (ViNo) that allows for describing different multimedia transport techniques for specifying required QoS; (2) the application of this formal notation to analyzing different transport mechanisms without the need of detailed simulations; (3) further application of ViNo to caching techniques, leading to the introduction of two cache admission policies and one replacement policy supporting non-sequential multimedia access.**
The applicability of ViNo is shown by example and by analysis of an existing CDN simulation. We find that a pure LRU replacement yields significantly lower hit rates than our suggested popularity-based replacement. The evaluation of caches was done by simulation and by usage of ViNo.

***Keywords*-Multimedia Formalism; QoS; Caching; CDN; Non-sequential Multimedia**

I. Introduction

A. Motivation

In the end of 1895 the Lumière brothers presented the first moving pictures in France (Lyon and Paris). They stored the movie as a sequence of images on a perforated celluloid tape. They were able to record and play this back at such a speed that the viewers had the impression of - more or less - continuous movement. This was the birth of the movie. People were so much fascinated from moving pictures that on the very first posters for movie performances we cannot find a title, author or the like - people just went to see moving pictures; whatever was showed.

Since then we have got used to the idea that a movie is a long sequence of images recorded and played back at a more or less constant speed. Even though this basic principle is still valid, the uncritical usage of this paradigm causes a lot of unnecessary difficulties in modern video access. Usage scenarios are rapidly changing. We have reported [2] how arthroscopic videos are used. The camera plays a central role in this kind of surgery and the recorded videos are highly interesting for research and education. The users - medical doctors - are interested to find special situations in a large archive of visually very similar recordings, e.g., the usage of surgery equipment of a given type, in a special pathological situation. They might be interested in comparing similar scenes, watch them in parallel and create sub-sequences or even single images from them. Several persons may do this in cooperation, in a distributed manner. Such usage patterns are obviously very different from that of the first viewers of Lumières' movies. In the following non-exhaustive list, we summarize the main aspects of the current situation:

1) Virtually everybody can create and consume videos.
2) The length of movies available on the Internet varies from a few seconds up to several hours.
3) Besides entertainment, professional use is gaining importance, e.g., in medicine, news production, traffic control and so on.
4) Both in entertainment and in professional usage, people are often only interested in a small fraction of long video sequences.

We conclude that we could gain a lot if we regarded videos as non-sequential, direct access media. To put it in another way: It is time to switch from the tape to the disk paradigm. Or again in other words: Instead of enforcing users to be passive viewers of movies it would be desirable to enable them to become active *composers* of video presentations.

B. Compositions

Instead of offering prefabricated long, sequential videos, we propose to offer a set of elementary *video units*, which can be composed with the help of sequential and parallel composition operators to arbitrary Video Notation (ViNo) *compositions*. A unit itself is regarded as an atomic composition. The result of a composition operation is also a composition. Thus, compositions can be constructed from video units by repeated application of composition operators.

We do not constrain the exact semantics of a video unit. It could be a single bit or byte, a video frame, or a semantically meaningful, short clip. By short we mean that the download time is short enough not to justify streaming. Streaming should rather be expressed as a composition (see below).

A given video can be logically described by several different compositions. It can also be physically decomposed, in order to get physically materialized units. We assume that for a given video usually several logical compositions exist, but only one physical decomposition exists. The problem of finding a physical decomposition of maximal unit size that is compatible with a given set of logical compositions is the subject of related research, but out of scope of this paper. In the following we assume that compositions rely on a given suitable physical decomposition.

C. Quality of Service (QoS) constraints on compositions

Compositions may be subject to certain QoS constraints to describe video processing requirements and properties using ViNo. For example, we may require that the processing delay for video units must not exceed a certain maximum. Or a given network bandwidth must be available when transmitting units according to a given ViNo composition. It is thus possible to describe a video streaming scenario as a sequential composition of bits with a bandwidth requirement of 1 Mbps, or as a sequential composition of video frames with an average throughput of 25 frames/sec. A video playback scenario could be expressed as a parallel composition of some clips with a maximal start-up delay of 500 ms. We can describe both required and provided QoS using the same formalism.

ViNo compositions may also describe pipelined video processing by an appropriate combination of sequential and parallel operators according to a certain number of stages. Each stage in the pipeline represents a buffering element. Classical video streaming can be described as a pipelined sequential composition, constrained by bandwidth and start-up delay.

More generally, ViNo can be used to express temporal relations between video units for the purpose of: (1) video presentation requests, (2) video delivery execution plans, and (3) video access methods.

D. Putting it together

Let us consider a simple example to put the elements together. Assume that a ski-jumping video has already been decomposed into six meaningful, short clips. The clips show two essential moments (jump-off and touch-down) for three athletes. The user would like to see the two clips belonging to the same jumper sequentially, but the three clip pairs in parallel. We assume that a video player capable of such presentation modes is available. The user creates a ViNo composition expressing her video presentation request (see formal examples below) with the help of some appropriate GUI. Now, the video transportation system transforms the request to an execution plan, which is again represented by a ViNo composition. For instance, if all clips are stored on the same network node then the clips must be downloaded sequentially (as we assume that clips are video units, which are handled atomically by the video transportation system). However, their order should be interleaved: first the jump-off clips of all three athletes and then the touch-down clips). On the other hand, if the clips happen to be distributed in the delivery network in an optimal way, i.e. the clips belonging to the same jumper are stored on the same network node and clips belonging to different athletes are stored on different nodes, with equal and minimal distance to the client, then the execution plan is actually represented by the same ViNo composition as the request. A good video transportation system obviously strives for finding such optimal placement of clips for popular requests. In any case the execution plan must, of course, fulfill the requested QoS constraints. If this is not possible, a good implementation is supposed to take certain adaptation actions such as replicating video units. These actions can again be expressed as ViNo composition transformations.

E. What are ViNo compositions good for?

We see two main advantages:

1) Flexibility: If we get rid of the dictatorship of the long, sequential, and continuous streams, then we gain a lot of freedom in the handling of video systems. By using ViNo we make any video delivery system programmable in a certain sense. This apparently rather theoretical point should not be underestimated. The success of digital computers relies exactly on this kind of flexibility. Analog computers had a number of advantages over the digital ones in solving differential equations. They were faster and more precise – but less flexible. No need to say who won the race between analog and digital computers. Note that in the early 1960s, this was not yet obvious.

2) Simplification: This is the actual topic of this paper. Making explanations and descriptions simpler had always been a driving power in science. It is not only a matter of costs – a simple solution is usually cheaper than a complex one, but a simple description is also easier to understand and therefore less error-prone. On the other hand, if something is getting very complicated then this is usually a sign of missing understanding.

F. Using ViNo for deriving delay bounds

In the first part of this paper we introduce syntax and semantics of ViNo and the associated QoS description language in detail. In describing QoS constraints we rely on QL, as defined by Blair and Stefani [3]. In the second part, we show how to model Content Delivery Systems (CDNs) with the help of ViNo. We can perform delay estimations of arbitrary complex compositions in a recursive manner. We use the CDNSim [4] simulator as a reference for evaluating our results. We already obtain good estimations using a rough model, which can easily be refined. Thus, we are able to estimate transport delay bounds of complex, distributed video delivery systems using a small set of ViNo expressions. The results can be sufficient to support system design decisions, thereby eliminating the need of sophisticated simulations. To the best of our knowledge, this is a unique achievement. ViNo expressions can easily be modified and extended. When creating the examples, we experienced indeed that we could not test all required situations using CDNSim. Modifying the simulation would have needed days – if not weeks – of work. Extending ViNo expressions, however, is a matter of hours or minutes (for an experienced user).

In the third part of the paper we use ViNo to experiment with some simple but novel video caching methods [1] based on units. An own prototype implementation serves as a reference. Also in this case ViNo yields suitable delay estimations (under the assumption, of course, that QoS parameter estimations are correct). The prototype implementation is a first step towards a novel, self-organizing video delivery system, where ViNo compositions and decompositions play a central role. However, this system will not be discussed in this paper.

II. Related Work

QoS languages have been defined to help a user or application to specify requirements and to formally define actions for recovery if the given requirements are not met. In [5] the authors give an overview and classification of QoS languages, which are categorized into user-layer QoS, application-layer QoS and resource-layer QoS. Examples are INDEX [6] as an expressive user-layer QoS language that helps translating the user's preferences to more specific network-related QoS. Application-layer QoS regards parameters such as frame rate and frame resolution. The authors of HQML [7] took advantage of XML for allowing developers to specify their own multimedia-related tags. Another example is QML [8], which is an object-oriented CORBA-based QoS language that allows for QoS hierarchies and reusability. Resource-layer QoS languages such as RSL [9] concentrate on resource management and allocation.

However, ViNo's aim is not to define a new QoS paradigm. ViNo uses QoS languages, in particular QL [3], in order to clarify QoS requirements. MMC# (see [10], [11]), a QL based QoS extension of C#, provides automatic QoS requirement formulation checking. A ViNo-compliant application might take advantage of that by being implemented in MMC#. However, calculations done with ViNo cannot be performed with any of the examples given in [5] nor with MMC#.

In contrary to QoS languages, an XML-based language exists that handles the presentation of autonomous media objects, namely SMIL [12]. SMIL is a description language for synchronizing different media channels like sound, video and text in a SMIL player. Although ViNo might also be used to describe multimedia presentations without the XML overhead of SMIL, ViNo's main strength is its general applicability to the analysis of flexible transport mechanisms and related calculations.

ViNo was designed to be able to compare existing multimedia transport technologies, such as Client/Server, Content Delivery Networks (CDNs) or P2P download and streaming, to more flexible approaches. In this context, non-sequential multimedia access patterns open new possibilities of video services and require new ways of transport.

As described in [1], a first step in the direction of non-sequential media was investigated by Zhao et al. [13]. The authors define "non-linear" media as video consisting of several parallel branches. The streaming system maintains a channel per branch. The authors observed as major problem that there is no possibility to explore bandwidth reduction by sharing connections, because it is not known if the client will choose the branch just transmitted in advance. Nevertheless, the authors showed that some hints regarding the client branch selection lead to remarkable server bandwidth and client data overhead reduction. However, the possible paths are predefined and limited compared to the possibilities offered by our video unit model.

Videos are considered as too large with respect to size to be cached completely. A lot of research has been done on partial caching. Generally, the idea of caching only parts of a video supports our non-sequential media model.

In [14], a detailed overview of different caching strategies is given. Prefix caching and segment-based caching are most closely related to our work. A prefix may be fixed (e.g., the first 10 minutes of a video) or dynamic (for every video a proper prefix size is defined), see also [15]. Segment-based caching increases the number of cached segments of a video based on popularity measurements. Segments may be uniformly sized or grow exponentially [16].

The authors of [17] propose a caching algorithm for streaming media based on a measured popularity distribution of segments. Considering fixed-sized segments of one second, they observed that the popularity of segments of a single video (*internal popularity*) follows a k-transformed Zipf-like distribution (for $k_x = 10$ and $k_y > 200$). Hence the first segments of a video are most popular, so the proposed caching policy prioritizes prefix caching. However, we cannot expect that non-sequential media access exhibits the same internal popularity pattern. A user may not be aware of which unit is the "beginning" of a video.

Another segment-based caching mechanism aims to support interactive "jumps" in a video stream [18]. The authors introduce a basic interleaved segment caching (BISC) policy, which disperses prefetched segments uniformly over the video length to reduce response time for jump requests at the cost of a reduced hit rate for sequential access. When the client jumps to an uncached segment the cache delivers the closest cached segment. Since segments are likely to be accessed sequentially after a jump, BISC was extended to a dynamic interleaved segment caching (DISC) policy, which dynamically selects an interleaved or continuous segment caching strategy based on observed client access patterns. However, the authors assume, based on their analysis of a real RTSP workload in the year 2004, that video segments will be accessed sequentially in most cases. In our video unit approach we refrain from this restriction.

III. The ViNo formalism

As described in the Introduction ViNo is based on so called *compositions*. Its general syntax is given by the following definition (the EBNF specification appears in the appendix).

Definition 1: A *composition* is an expression defined inductively by these rules:

1) A single video unit is a composition.
2) Let $c_1, c_2, ..., c_n$ with $n \geq 2$ be compositions, which have already been defined. Then, the following expressions are compositions, too:
   a) $[c_1 \,||\, c_2 \,||\, \ldots \,||\, c_n]$ is called a *parallel* composition.
   b) $(c_1 \;\leftarrow_{Q_1}\; c_2 \;\leftarrow_{Q_2}\; \cdots \;\leftarrow_{Q_{n-1}}\; c_n)$ is called a *sequential* composition. A symbol Q_i, where

$i = 1, \ldots, n-1$, represents an optional QoS parameter and may be omitted.

Throughout this paper, u_i $(i \geq 1)$ always denotes a single video unit. The brackets or parentheses of a parallel or a sequential composition c, respectively, may be omitted if c does not appear as proper subexpression of a composition. So both $[u_1 \,||\, u_2]$ and $u_1 \,||\, u_2$ are valid compositions on their own, but $u_1 \,||\, u_2 \leftarrow u_3$ is not.

We define the semantics of ViNo in the context of video transmission, but analogous interpretations apply in other contexts as explained in the Introduction.

Definition 2: Semantics.

1) If $c = c_1 \,||\, c_2$ for some compositions c_1 and c_2, then the transport of c starts as soon as c_1 or c_2 starts, whatever is earlier; and it is finished when the transport of both c_1 and c_2 is completed.
2) If $c = c_1 \leftarrow_Q c_2$ then the transmission of c_2 must not start before the completion of c_1; the QoS predicate Q applies to the time period between completion of c_1 and completion of c_2.
3) The semantics of $c = c_1 \leftarrow_{Q_1} c_2 \leftarrow_{Q_2} c_3$ is defined as that of $(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3$.

We consider two compositions c_1 and c_2 as *equivalent* if they lead to the same semantics according to Definition 2. We then write $c_1 = c_2$. It is easy to check that the following equations hold:

$$[c_1 \,||\, c_2] \,||\, c_3 = c_1 \,||\, [c_2 \,||\, c_3] \tag{1}$$

$$[c_1 \,||\, c_2] = [c_2 \,||\, c_1] \tag{2}$$

$$(c_1 \leftarrow c_2) \leftarrow c_3 = c_1 \leftarrow (c_2 \leftarrow c_3) \tag{3}$$

$$(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3 = c_1 \leftarrow_{Q_1 \circ Q_2} (c_2 \leftarrow_{Q_2} c_3) \tag{4}$$

where c_1, c_2, c_3 are arbitrary compositions and $Q_1 \circ Q_2$ means a suitable combination of both QoS predicates Q_1 and Q_2, e.g. the sum if Q_1 and Q_2 refer to maximal delay. Note that according to (1) a parallel composition c is an associative binary operation, so the semantics of c is well defined. The same applies to sequential composition without QoS predicate.

Definition 3: The null unit u_0 is a video unit of length 0 (empty).

The null unit u_0 serves as a "dummy" composition (in a similar way as dummy targets are used to express side-effects in functional languages). The following properties apply:

- $u_0 \,||\, c_1 = c_1$
- $u_0 \leftarrow c_1 = c_1$
- $c_1 \leftarrow u_0 = c_1$

A. Simple examples

In order to show how the before mentioned definitions work, three artificial examples are created. All examples are based on the same video delivery system architecture. It consists of one origin server, four interconnected proxies and one client. We show how the transmission of six video units can be described using ViNo for different configurations with respect to unit placement. For sake of simplicity, QoS is postponed to the next section.

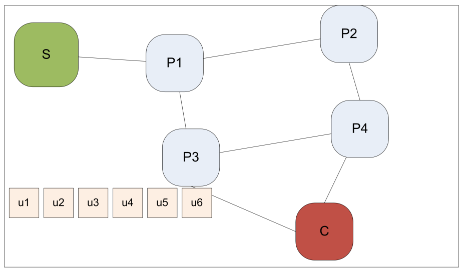

Fig. 1. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where all video units $u1 - u6$ are available at proxy $P3$.

As described in the Introduction a presentation request may be created using some appropriate GUI. The request of displaying video units 1–6 sequentially can be expressed as:

$$r = u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6$$

The actual transport of video units may differ from the presentation request. To keep it simple, we assume that all video units are downloaded to the client completely before play back starts. So the video delivery process can be decomposed into one or more *download stages* corresponding to a ViNo *transport description* s, followed by a *play back stage* equivalent to the presentation request r: $s \leftarrow r$.

Example 1. All video units are located at proxy $P3$ as shown in Fig. 1. The units will be downloaded sequentially to the client, corresponding to the ViNo transport description:

$$s_1 = u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6$$

By adding the play back stage $s_2 = r$ we obtain the complete video delivery description:

$$\begin{aligned} c = s_1 \leftarrow s_2 = s_1 \leftarrow r = \\ u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \\ \leftarrow u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \end{aligned}$$

Example 2. Three of the units are located on proxy $P3$ and the other three are located on proxy $P4$. Both proxies are direct neighbors of the client (Fig. 2). The download from $P3$ is described as $s_1 = u_1 \leftarrow u_2 \leftarrow u_3$ and the download from $P4$ is described as $s_2 = u_4 \leftarrow u_5 \leftarrow u_6$. There are two possibilities to combine these download stages to describe the overall video transport:

(1) The client downloads everything from $P3$ and afterwards everything from $P4$, resulting in the ViNo expression: $s_1 \leftarrow s_2 = (u_1 \leftarrow u_2 \leftarrow u_3) \leftarrow (u_4 \leftarrow u_5 \leftarrow u_6)$.

(2) While downloading everything from $P3$ the units are downloaded from $P4$ in parallel: $s_1 \,||\, s_2 = (u_1 \leftarrow u_2 \leftarrow u_3) \,||\, (u_4 \leftarrow u_5 \leftarrow u_6)$. The video transport expressed by this composition is finished when all video units have been transmitted. Note that there is no temporal relation between downloading units of s_1 and s_2, respectively. That is, u_2 can be received before or after u_5 by the client. However, a system designer may decide to synchronize the transport of s_1 and s_2;

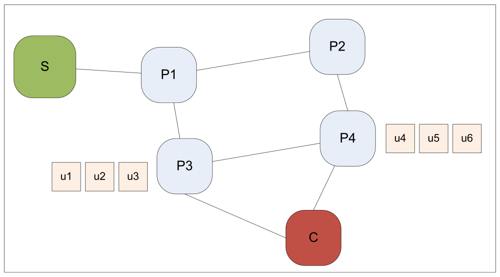

Fig. 2. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where the video units $u1 - u6$ are available at proxies $P3$ and $P4$ near the client.

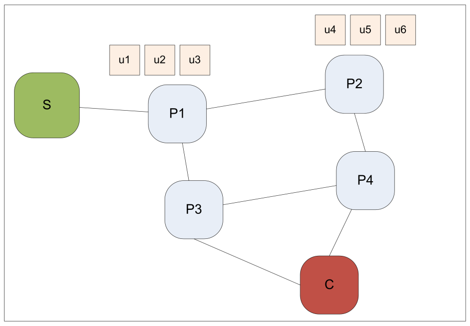

Fig. 3. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where the video units $u1 - u6$ are available at proxies $P1$ and $P2$, which are not directly connected to the client.

then the description specializes to $[u_1 \,||\, u_4] \leftarrow [u_2 \,||\, u_5] \leftarrow [u_3 \,||\, u_6]$.

As in example 1 the transport description involving the download stages s_1 and s_2 is followed by a play back stage $s_3 = r$.

Example 3. Three of the video units are located on proxy $P1$ and the other three units are located on proxy $P2$. None of these proxies is directly connected to the client (Fig 3), so units have to be replicated to proxies $P3$ or $P4$, respectively, before they are downloaded to the client. This results in 5 stages: the transport of 3 units from $P1$ to $P3$ (stage s_1), from $P2$ to $P4$ (stage s_2), from $P3$ to the client (stage s_3), and from $P4$ to the client (stage s_4); and finally, the play back of all 6 units at the client (stage $s_5 = r$).

Let us assume a pipelined transport where proxies $P3$ and $P4$ forward units immediately after receiving them from $P1$ or $P2$, respectively. For the sake of simplicity, let us further assume that the transmission times of all video units between adjacent network nodes and the play back duration of a single video unit are roughly equal to a certain time period t. Then the temporal evolution of the video delivery process can be represented by TABLE I. Consequently, the user has to wait 4 time slots until play back can start, and the entire video delivery process takes 10 time slots.

The table also helps creating the appropriate ViNo expression that describes the given video delivery scenario. Units within the same time slot are transmitted in parallel, units in different time slots are processed sequentially. The resulting

t	1	2	3	4	5	6	7	8	9	10
s_1	u_1	u_2	u_3							
s_2	u_4	u_5	u_6							
s_3		u_1	u_2	u_3						
s_4		u_4	u_5	u_6						
s_5					u_1	u_2	u_3	u_4	u_5	u_6

TABLE I
TEMPORAL EVOLUTION OF VIDEO DELIVERY SCENARIO OF EXAMPLE 3.

ViNo expression is:

$$[u_1 \,||\, u_4] \leftarrow [u_2 \,||\, u_5 \,||\, u_1 \,||\, u_4] \leftarrow [u_3 \,||\, u_6 \,||\, u_2 \,||\, u_5] \\ \leftarrow [u_3 \,||\, u_6] \leftarrow u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \quad (5)$$

B. Introducing QoS

To specify a request a client has only to provide information about the required video units and whether these units have to arrive in order (e.g. at the player). For example, a user may express "I want to download units x and y, the order does not matter" as $u_x \,||\, u_y$. Note that this does not mean that the units have to be delivered in parallel. A user who wants to watch the units using a video player would be more specific: "I want to watch unit x and then unit y, and unit x has to arrive within the next 30 seconds". This request can be expressed as $u_0 \leftarrow_{D=30sec} u_x \leftarrow u_y$, where u_0 is the null unit needed only to express the required delay for unit x.

In the sequel the usage of QoS parameters is demonstrated for expressing video unit transport. However, if ViNo is used in a different context, the semantics of QoS annotations may differ and need to be clarified prior to any calculations based on ViNo expressions. We provide examples for the well-known transport-related QoS parameters bandwidth, delay, and jitter.

We derive the notation and semantics of QoS parameters from the QoS language QL [3]. QL is based on events like reception and sending of messages. It uses a function τ mapping events to points in time. Since ViNo is based on compositions, we restrict ourselves to the event of *receiving a composition* c at a given network node or video display. This event occurs as soon as all video units referenced by c have been received completely. We denote the corresponding point in time as $\tau(c)$. In this paper we focus on QoS parameters that can be used for delay calculations of video transport processes described by ViNo expressions.

Definition 4: We define a recursive function *delay* to calculate a delay bound for a QoS-annotated ViNo transport description c:

1) The null unit causes no delay: $delay(u_0) = 0$.
2) If $c = c_1 \leftarrow_Q u$ for some composition c_1 and a video unit u, then

$$delay(c) = delay(c_1) + delay(u, Q)$$

where $delay(u, Q)$ is defined to be the delay $\tau(u) - \tau(c_1)$ assuming a provided QoS parameter Q (trivial case of recursion).

3) If $c = c_1 \leftarrow c_2$ for compositions c_1 and c_2, then the delay bound is computed recursively as:

$$delay(c) = delay(c_1) + delay(c_2)$$

For delay calculations, we assume that the transmission of c_2 occurs as soon as possible after the transmission of c_1, which is expressed by omitting the QoS parameter.

4) If $c = c_1 \,||\, c_2$ for compositions c_1 and c_2, then the delay bound is computed recursively as:

$$delay(c) = max(delay(c_1), delay(c_2))$$

Note that the $delay$ function is defined only on a subset of all possible ViNo expressions. The following two expression types will occur frequently in the subsequent examples, so we introduce a separate notation for the corresponding delay bounds:

- If $c = u_0 \leftarrow_{Q_1} u_1 \leftarrow_{Q_2} \cdots \leftarrow_{Q_n} u_n$ for video units u_i (u_0 is the null unit), then
$$\begin{aligned} delay(c) &= \sum_{i=0}^{n} delay(u_i, Q_i) \qquad (6)\\ &= delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, seq) \end{aligned}$$
where the last term introduces a new notation.
- If $c = (u_0 \leftarrow_{Q_1} u_1) \,||\, \ldots \,||\, (u_n \leftarrow_{Q_n} u_n$ for video units u_i (u_0 is the null unit), then
$$\begin{aligned} delay(c) &= \max_{1 \le i \le n} (delay(u_i, Q_i)) \qquad (7)\\ &= delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, par) \end{aligned}$$
where the last term introduces a new notation.

Whether the $delay$ function represents a lower or upper delay bound depends on the definition of the $delay(u, Q)$ values. We now demonstrate how to define these values if the QoS parameters are given in terms of bandwidth, delay, or jitter, respectively.

1) Bandwidth: By $Q = BW$ we express that a given bandwidth BW is available for transmission. The delay of transmitting a video unit u is defined as:

$$delay(u, BW) = \frac{size(u)}{BW}$$

The $delay$ function therefore computes a *lower bound* of the end-to-end delay corresponding to a given ViNo expression. We assume that video units transmitted in parallel according to some parallel ViNo composition use separate links, so that the available bandwidth is not reduced by parallel transmissions.

Calculation of the lower delay bounds of sequential and parallel compositions of video units according to equations (6) and (7) results in:

$$\begin{aligned} &delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, seq)\\ &\qquad = \sum_{i=1}^{n} \frac{size(u_i)}{BW_i} \qquad (8) \end{aligned}$$

$$\begin{aligned} &delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, par)\\ &\qquad = \max_{1 \le i \le n} \left(\frac{size(u_i)}{BW_i} \right) \qquad (9) \end{aligned}$$

2) Delay: By $Q = D$ we express that transmission yields a given delay D. The delay of transmitting a video unit u is defined as:

$$delay(u, D) = D$$

If all delays occurring in a video delivery system are expressed as provided QoS parameters of a corresponding ViNo composition and if the composition is an appropriate model of the system, the calculated end-to-end delay value should be accurate. However, for practical purposes, the ViNo composition is constructed to provide an *upper delay bound* only, which may lead to a simpler ViNo expression.

3) Jitter: According to the QoS language QL, jitter can be defined by specifying *lower and upper delay bounds* (D_{min}, D_{max}). To calculate the jitter of a given video transport system described by an appropriate ViNo expression, we therefore just need to apply the $delay$ function to both bounds separately. We obtain two functions $delay_{min}$ and $delay_{max}$ with appropriate definitions of delay bounds for transmitting a video unit u:

$$\begin{aligned} delay_{min}(u, D_{min}) &= D_{min}\\ delay_{max}(u, D_{max}) &= D_{max} \end{aligned}$$

The jitter of a ViNo composition c is then computed as $(delay_{min}(c), delay_{max}(c)$.

To illustrate delay calculations, we now apply the $delay$ function to example 3 of section III-A. We restrict the discussion to delay as QoS parameter. Let the delay D_1 for one unit delivered from $P1$ to $P3$ be 300 ms, and the delay D_2 from $P2$ to $P4$ be 350 ms. The delay D_3 from both proxies $P3$ and $P4$ to the client shall be 200 ms each. We need to extend the ViNo transport description (see (5) and TABLE I) to introduce delay parameters:

$$\begin{aligned} c = &[(u_0 \leftarrow_{D_1} u_1) \,||\, (u_0 \leftarrow_{D_2} u_4)]\\ &\leftarrow [(u_0 \leftarrow_{D_1} u_2) \,||\, (u_0 \leftarrow_{D_2} u_5)\\ &\qquad || \,(u_0 \leftarrow_{D_3} u_1) \,||\, (u_0 \leftarrow_{D_3} u_4)]\\ &\leftarrow [(u_0 \leftarrow_{D_1} u_3) \,||\, (u_0 \leftarrow_{D_2} u_6)\\ &\qquad || \,(u_0 \leftarrow_{D_3} u_2) \,||\, (u_0 \leftarrow_{D_3} u_5)]\\ &\leftarrow [(u_0 \leftarrow_{D_3} u_3) \,||\, (u_0 \leftarrow_{D_3} u_6)] \end{aligned}$$

Note that this ViNo composition is of the form $c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow c_4$, where each c_i denotes a parallel composition. By applying equation (7) and case 3 of Definition 4 we therefore obtain:

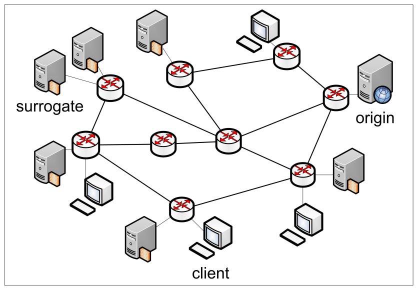

Fig. 4. Sample Architecture in CDNsim

$$
\begin{aligned}
delay(c_1) &= delay(u_1, u_4, D_1, D_2, par) \\
&= \max(D_1, D_2) = D_2 \\
delay(c_2) &= \max(D_1, D_2, D_3, D_3) = D_2 \\
delay(c_3) &= \max(D_1, D_2, D_3, D_3) = D_2 \\
delay(c_4) &= \max(D_3, D_3) = D_3 \\
delay(c) &= delay(c_1) + delay(c_2) + delay(c_3) + delay(c_4) \\
&= D_2 + D_2 + D_2 + D_3 \\
&= (350 + 350 + 350 + 200)\text{ ms } = 1250\text{ ms}
\end{aligned}
$$

IV. Analyzing Transport and Caching

In this section the potential of ViNo is shown as a tool for analyzing existing delivery systems, such as Content Delivery Networks (CDN). Additionally, we applied the same analysis to our non-sequential multimedia cache to compare its caching techniques to CDN.

A. Content Delivery Networks

CDNs consist of origin servers that are supported by strategically placed surrogate servers to which the content is replicated and/or cached (see [19], [20]). In most of the commercially available CDNs, the content is passively pulled by surrogate servers. Usually, commercial CDNs are not available for research purposes. Even academic CDNs, which are available on PlanetLab, are treated as black boxes. For this reason, Stamos et al. [4] developed a simulation environment, called CDNsim, for large scale CDN simulations. This simulation is the basis for our experiments with ViNo.
A GUI for configuring simulations is also part of CDNSim. CDNSim is an Omnet++ [21] simulation and uses the INET Framework Library [22]. It covers all typical CDN functionality, such as DNS request redirection and LRU replacement. CDNSim supports different cooperation policies such as closest surrogate or random surrogate cooperation, but also simple non-cooperative behavior can be configured. A very interesting point for our investigations is the fact that if the number of nodes and routers remains the same the same architecture is generated for each simulation run. Therefore, the clients connect always to the same surrogate servers.

TABLE II shows the configuration parameters used for our experiments. Sample request traces and router topologies are made available by the CDNSim developers at [23]. For sake of simplicity we decided to use the non-cooperative policy for our experiments, i.e, if a requested object is not available at the client's surrogate server the request is forwarded to the origin server. However, all delay calculations can also be applied to the cooperative policies as well.
The optimal unit size for a given application is an open research issue. For our simulation experiments we simply selected some reasonable size, namely 1500 bytes. This means that a web page of 4,500 bytes is divided into 3 units and is described as $u_1 \leftarrow u_2 \leftarrow u_3$. The link speed is specified to be 200 Mbits/sec, which results in a bandwidth BW of 16,666 units/sec.
We evaluate ViNo by calculating delay in miss and hit situations and compare the results to the simulated values.
In general a hit is represented as the distance from a client to its surrogate, which is 1 hop. On a miss the transport represents a sequential composition of two stages, i.e., from origin to surrogate and from surrogate to client (e.g., $c = (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3) \leftarrow (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3)$). Thus, all calculations can be done without the complete knowledge of the CDN's architecture. The calculations represent the time a transport takes at minimum, i.e., it is the best case transport delay. For the experiments these calculations are referred to as *ViNo generic*.
The delay function for the example above can be described as $delay(c)$ and can be calculated as follows:

$$
\begin{aligned}
delay(c) &= delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
&\quad + delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
&= \frac{6}{16,666} = 0.36\ ms
\end{aligned}
$$

If the architecture is known in more detail, which is the case for CDNSim, more precise calculations can be performed. As shown in Fig. 4 routers are placed on the path of clients and surrogates. The idea was to consider those routers as hops, e.g., a client is 3 hops away from its surrogate server. On a hit the delay can be calculated based on the distance (measured in hops) from client to surrogate. For the experiments these calculations are referred to as *ViNo routers*.
Two experiments were started, (1) all clients download web pages of small size; (2) one client downloads a number of different sized web pages.
Experiment 1. This experiment proves the general applicability of ViNo, its results are shown in Fig. 5. It is seen that both ViNo routers and ViNo generic estimate well the delay pattern of the measured values. The distance of ViNo routers to the simulated delay is smaller because its calculations are more precise, for the price that the transport paths have to be known in advance.
We show by the example of downloading one single object, how the corresponding delay is calculated. A randomly chosen client with the id $c1009$ connects to the surrogate server with id $s1199$ in 4 hops. A miss means a transport over 9 hops from the origin server. This client downloads the

Parameter	Value
Router Topology	Waxman for 1000 Routers
Link speed	200 Mbit/sec
Number of clients	100
Number of surrogate servers	100
Number of origin servers	1
Number of outgoing connections	1000
Websites	50000 Web objects, 100MB max object size, sizes' zipf=1, size vs popularity correlation = 0
Traffic	1000000 requests, popularity's zipf = 1.0, expo mean interarrival time = 1, 100 client groups

TABLE II
CDNSIM CONFIGURATION PARAMETERS

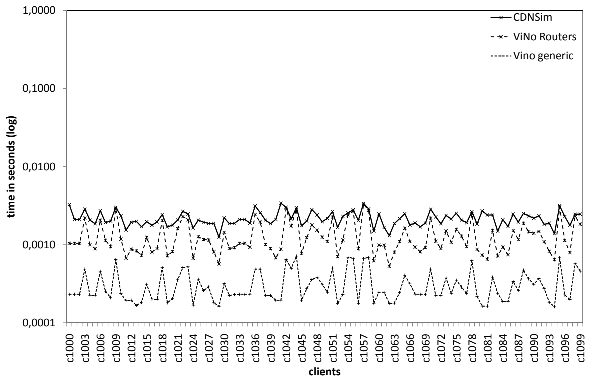

Fig. 5. Comparison of download time ViNo vs. CDNSim

object with id 13, which has a size of 5 units. In ViNo one stage consisting of these 5 units can be described as $c_i = u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3 \leftarrow_{BW} u_4 \leftarrow_{BW} u_5$. In the simulation object nr 13 was not present at the surrogate server. Thus, the overall composition c for ViNo routers is a sequential composition of stages 1-9 (one stage per hop).

$$\mathrm{h} = 9, \mathrm{BW} = 16,666u/sec$$
$$delay(c) = \sum_{i=1}^{h} delay(c_i)$$
$$= h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 2.7\ ms$$

The ViNo generic calculation has no detailed knowledge of the routers and reduces therefore the miss to two hops, such that the calculation changes to:

$$\mathrm{h} = 2, \mathrm{BW} = 16,666u/sec$$
$$delay(c) = \sum_{i=1}^{h} delay(c_i)$$
$$= h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 0.6\ ms$$

The measured value was 3.01 ms, which shows that ViNo routers calculation is a really good estimation.

Experiment 2. This experiment was done to investigate the impact of different file sizes, since video objects are in general larger than web objects. One client was picked out of all clients, which downloads a set of very different sized objects. The generic and the router based delay was calculated and then compared to the simulated results. The ViNo router calculated delay is shown in Fig. 7 and it can be seen that the calculations do not always represent the lower bound of the simulated duration. One extreme case is shown for the object with id 637 (the peak in Fig. 7), which has a size of 24 MBytes (16,666 units). At this point of the simulation the object was not present at the surrogate, thus it had to be downloaded from the origin with a distance of 8 hops. The measured value was 3 seconds. The calculations with ViNo routers are provided below:

$$delay(c) = \sum_{i=1}^{8} delay(c_i)$$
$$= 8 * delay(u_1, ..., u_{16666}, BW, ..., BW, seq) = 8sec$$

This effect appears for files that exceed the size of 10 KBytes, which are routed in a different way than smaller files (as in experiment 1). Larger files are split up and are routed in parallel over several paths. Therefore, the transport is a mixture of parallel and sequential compositions and not purely sequential as assumed before. Since the routing algorithm is part of the INET Framework and the paths are not predictable with reasonable effort, we cannot provide a more detailed calculation. However, the router based calculations might represent the worst case delay if the routing path is always the same.

The generic calculations are always representing the lower bound of the duration as shown in Fig. 6, since in any way the surrogate downloads the complete website before forwarding it to the client. In comparison to the ViNo routers result the ViNo generic result is 2 seconds for the object with id 637.

Thus, the generic case represents the larger file downloads better and the router based calculations represent smaller file downloads better. Which type of calculation is finally taken depends on the knowledge of the architecture and on the purpose of the analysis.

The efficiency of CDNs and caches is usually compared by measuring the hit rate. ViNo can also be used to analyze the impact of the hit rate to the delay.

In our experiments the objects' sizes are Zipf distributed with an alpha value of 1.0 (strongly skewed). This means that

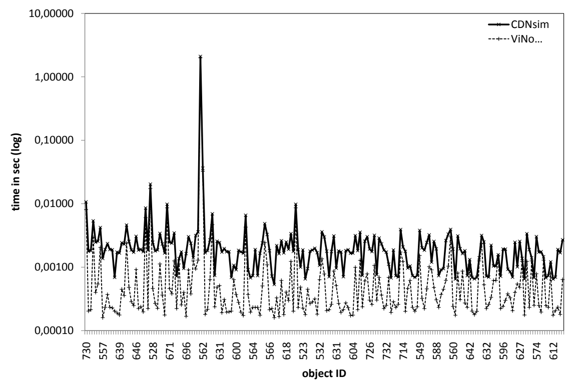

Fig. 6. Comparison of download time ViNo generic vs. CDNSim

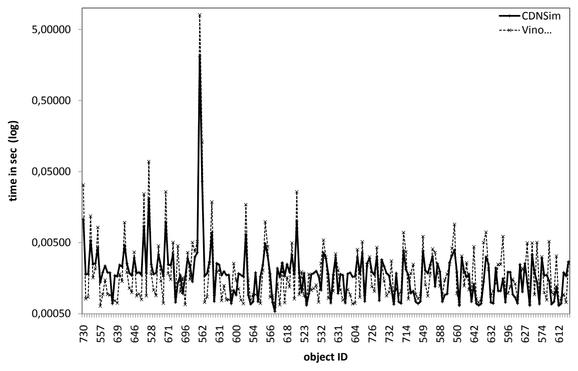

Fig. 7. Comparison of download time ViNo routers vs. CDNSim

80 % of the overall objects' size are represented by 20 % of the objects. This fact has a huge impact on the surrogates' cache size. In CDNSim one cache was able to store 109 MBs, which lead to hit rates of around 80 %. The reason is that most surrogates handle small files and the cache misses only occured in the beginning of the simulation until all objects were loaded from the origin server (i.e., the actual cache size was around 100 %).

If the surrogate servers had used popularity based prefetching the hit rate would have reached 100 %. This leads to the question on how much delay improvement prefetching would make. For this investigation we took ten random surrogate servers out of the simulation and calculated the delay reduction as shown in TABLE III. It can be seen that the number of units (i.e., the size of the original files) to serve vary a lot. E.g., surrogate s1191 only serves 19 files, whereas surrogates s1158 and s1164 serve almost the same amount of units, but the number of objects differ by a factor of 10. In general, those surrogate servers serving large files have advantages if prefetching is used. A CDN provider for

SID	no. clients	no. objects	no. units	prefetch	pipeline
s1110	2	2158	26290	1.57733	1.57727
s1132	1	475	4033	0.24198	0.24190
s1140	3	46	210	0.01260	0.01255
s1158	1	214	18352	1.10110	1.10109
s1164	3	2632	18597	1.11571	1.11571
s1188	4	3007	49100	2.94594	2.94591
s1191	2	19	69	0.00414	0.00408
s1194	2	6658	77183	4.63089	4.63077
s1196	4	3355	56281	3.37679	3.37672
s1198	6	3291	39795	2.38764	2.38758

TABLE III
SURROGATE DELAY REDUCTION IN SECONDS ON PREFETCH AND ON PIPELINING

videos can reach better surrogate efficiency and therefore startup delay minimization if popularity based prefetching per surrogate is applied. However, the popularity measures must include different factors, e.g, region, as we might assume that clients in Europe have different interests than in America, aso.

Another solution for the CDN provider could be to apply pipelined transport on a miss, i.e., a surrogate forwards a unit immediately after download from the origin. For a web site that consists of three units this is described as:

$$\begin{aligned} c = (u_0 \leftarrow_{BW} u_1) &\leftarrow [(u_0 \leftarrow_{BW} u_2)||(u_0 \leftarrow_{BW} u_1)] \\ &\leftarrow [(u_0 \leftarrow_{BW} u_3)||(u_0 \leftarrow_{BW} u_2)] \leftarrow (u_0 \leftarrow_{BW} u_3) \end{aligned}$$

In comparison to $c_1 \leftarrow c_2 = (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3) \leftarrow (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3)$ for the pure sequential transport. The delay for the pipelined composition is calculated as the sum of all sub-compositions (i.e., $c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow c_4$).

$$\begin{aligned} &delay(c) \\ &= \sum_{i=1}^{4} delay(c_i) \\ &= delay(u_1, BW) \\ &+ \max(delay(u_2, BW), delay(u_1, BW)) \\ &+ \max(delay(u_3, BW), delay(u_2, BW)) \\ &+ delay(u_3, BW) \\ &= \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} = 0.24ms \end{aligned}$$

If the units would be transported sequentially as in $c_1 \leftarrow c_2$ the delay would be calculated as the sum of the sub delays, i.e:

$$\begin{aligned} &delay(c) \\ &= delay(u_1, u_2, u_3, BW, ..., BW, seq) \\ &+ delay(u_1, u_2, u_3, BW, ..., BW, seq) \\ &= \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} \\ &= 0.36ms \end{aligned}$$

The transport would need 3+3=6 time slots. The pipelined transport reduces the number of time slots to 4, in more general (for a 2-stage pipeline):

$$delay_{pipelined} = \frac{delay_{sequential}}{2} + 1 \ time\ slots$$

If the surrogates analyzed before used pipelined transport on a miss the delay would reduce in comparison to a sequential transport as shown in TABLE III. Which of the both techniques a CDN provider chooses is a matter of implementation.

B. Non-sequential Multimedia Caching

If resources at a surrogate server are more limited and the access patterns more flexible than in the CDNSim case before, a CDN provider might be interested in a more efficient replacement and prefetching policy. This analysis was done in [1] and extended results are shown in the following.

We assume that the units are self-contained and equipped with metadata comprising further information about the content. Furthermore, we assume that a smart user application exists that provides information about user intentions (see [24]). User intentions are metadata about semantic roles a user can be categorized to. Such a role could be, e.g., ”informational” denoting users looking for many but unspecific data and ”transactional” denoting users wishing to buy a specific content. A semantic group of units is therefore a number of units that is of interest for a category of users. Note that the unit groups are not disjoint, but a user mapped to a given role is expected to request those units that are mapped to that role. This favors units that are more popular than others. Units in each group are ordered according to their popularity. This knowledge is exploited in the proposed cache admission policy.

The initial content of the cache is prefetched at system start and regards the most popular units of all groups, depending on the cache's size. Subsequently, the next fitting unit from a user group will be prefetched. The next fitting unit is the unit following the currently requested unit regarding popularity within the current group. This policy is called ”simple cache admission policy” and formally defined as follows:

$$\text{prefetch=} \begin{cases} u_{next} & \text{if hit } u_{current} \\ 0 & \text{if hit } u_{current} \text{ \&\& hit } u_{next} \\ u_{current} \leftarrow u_{next} & \text{else} \end{cases}$$

However, this policy is inefficient, since unpopular units are prefetched as well. Therefore, the second admission policy is based on a rank calculation (r_{all}) over all groups for each unit. If the calculated global rank is below a predefined rank (rank 0 is the highest rank level), the unit is considered for prefetching. The impact of low popularity is minimized using the logarithm of the group rank.

$$r_{all} = \frac{1}{n}\sum_{i=1}^{n} \ln r_i$$

If a unit is in the top 5 of one group and less popular in another group, it is more likely that this unit is cached than a unit that has an average popularity within several groups.

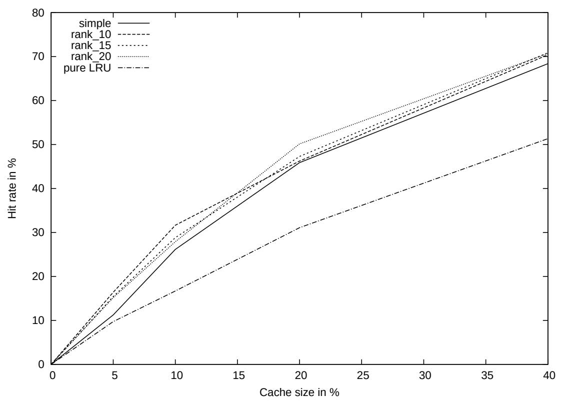

Fig. 8. Hit rate comparison of pure, simple and rank-based admission using LRU.

This approach was evaluated with a discrete-event based simulation using Omnet++. The user requests were generated with Medisyn [25], in a similar way as done in CDNSim.

For 100 units with different popularity, 10000 requests per user group were generated. Initially, LRU was implemented as the replacement strategy. The threshold was mapped to ranks 10, 15 and 20. Furthermore, we used CDNSim with the same parameters to compare pure LRU without prefetching to the non-sequential cache.

First, the hit rate comparison is done for the LRU-based admission policies, the results are depicted in Fig. 8. The rank-based admission policy shows an improvement of 5-10 % according to the simple admission and up to 20 % of improvement in comparison to the non-prefetching policy. It can also be seen that the thresholds of the rank-based policy show small differences in hit rate, but the number of prefetches increases the higher the threshold is specified. For further experiments the threshold of rank 10 is seen to be sufficient.

Although LRU supports popular units to remain longer in the cache, for small cache sizes even popular units are often replaced in the case of prefetching. Fig. 9 shows the factor of requests sent to the server in comparison to the client requests. This shows the maybe surprising result for small cache sizes it would be more efficient to send the units directly from the server, because the number of units sent from the server exceeds the number of requests. For the simple admission policy the server ought to send units in vain until 30 % of cache size. Whereas the rank-based policy decreases the load to an efficient level already at a cache size of 10-15 %. For less unnecessary replacements LRU has to be substituted by a replacement policy that considers the unit popularity.

A unit has to be prefetched and cached if it is popular enough. The rank calculation of the admission policy can also be applied to the replacement policy.

The effect on the load is shown in Fig.11. The simple admission policy starts to be efficient from a cache size of 10 % in comparison to LRU replacement. Also the rank-based

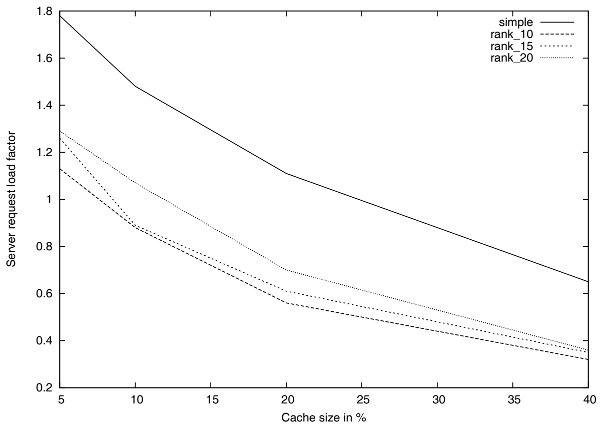

Fig. 9. Factor of server requests compared to user requests (LRU)

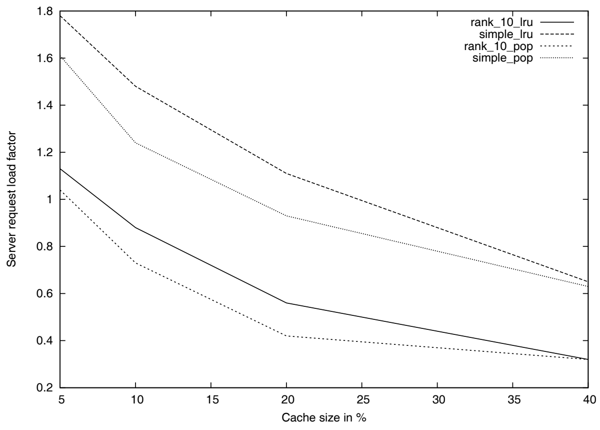

Fig. 11. Factor of server requests compared to user requests (LRU, popularity Relacement)

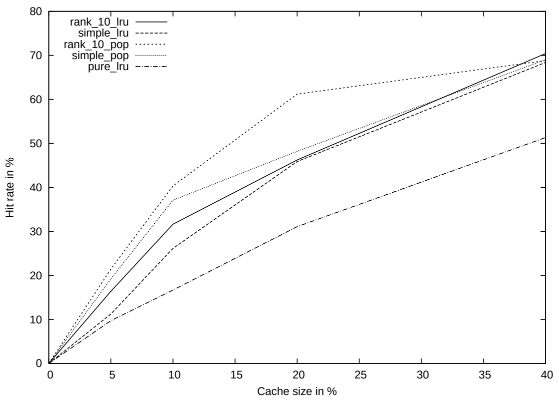

Fig. 10. Hit rate comparison of the admission policies using LRU and popularity replacement

admission policy starts to be efficient from very small cache sizes. However, this policy is complex and might not be used for caches with limited computing resources. For this cases the simple admission policy in combination with rank-based replacement is preferable.

Fig. 10 compares the hit rate evolution for both replacement policies. The comparison to pure LRU shows that the hit rate increases remarkably if applying a combination of rank-based admission and rank-based replacement. For a cache size of 20 % the hit rate differs by 40 %. Video CDN providers would gain remarkable storage cost reductions if applying rank-based caching.

V. Conclusion and Future Work

In this paper we extended the definition of the syntax and semantics of the Video Notation (ViNo) published in [26] and in [1]. Its applicability for describing and analyzing video transport is shown by simple examples and by the evaluation of an existing transport technique (CDN). For this reason we investigated two calculation types: (1) a simple, generic calculation that is always valid for the estimation of the best case delay and (2) a router based calculation that is more precise if the architecture is known in advance.

We showed in a caching scenario by taking advantage of ViNo how the use of prefetching and pipelining reduces startup delay. It is shown that ViNo could support content providers' system design decisions without extensive simulations.

The analysis of the CDN simulation was further used to investigate flexible access patterns in a non-sequential caching technique. The next unit to prefetch is depending on the popularity of this unit in all defined user groups. By comparing these prefetching techniques LRU replacement was found to replace units too frequent. Thus, LRU should be substituted by a popularity based replacement. The popularity based replacement technology improves hit rate and reduces the load of the server remarkably. This proactive caching and prefetching policy can be an efficient technique for multimedia CDNs, because storage and costs would be reduced and quality be increased (smaller start-up delays). The calculation effort of rank-based prefetching combined with rank-based replacement might have huge impact on the performance of a system. Thus, the decision of which cache admission and replacement policy to use depends on the resources available in the system to analyze.

However, ViNo cannot fully substitute a simulation, since the prediction of specific steps in a system (e.g., dynamic routing paths) cannot be made with reasonable effort. ViNo can be used as a tool for approximating general behavior of multimedia transport, e.g., to compute the best case delay on a miss or on a hit. One of ViNo's strength is the expression of different transport techniques, which allows a simple comparison on the first sight.

By using ViNo in research articles authors can explain new

transport techniques. For example, a new flexible approach can be formally described in a few lines. This would banish a lot of ambiguity from the scientific discussion.
Future work will regard further QoS calculations beyond delay, and the definition of unit loss. Another issue is dynamic unit size, which will be needed for semantically meaningful units. In this context ViNo will be applied to compare self-organizing multimedia transport to existing techniques. It is expected that in non-sequential cases the flexible system will outperform the traditional systems regarding startup delay and user experience, even though the proposed caching technique will also have its costs.

VI. Acknowledgements

The results presented in this paper are part of the research efforts for the SOMA (Self-organizing Multimedia Architecture) project, a Klagenfurt University and Lakeside Labs GmbH cooperation (URL: http://soma.lakeside-labs.com/)

Appendix

The following ViNo specification was created by using ANTLR a LL(*) parser generator [27]. Since special signs are not allowed in ANTLR, we used <_Q for $\leftarrow_Q$.

```
SEQ : '<' Q? ;
PAR : '||';
NUMBER : ('0'..'9');
VALUE : NUMBER+;
LETTER : ('a'..'z'|'A'..'Z');
NAME : LETTER (NUMBER | LETTER)*;
Q : ('_' NAME ('=' | '>=' | '<=') VALUE)+;

unit : NAME;

primitive: unit | group;

par: (PAR primitive)+;

seq: (SEQ primitive)+;

pargroup: '[' primitive par ']';

seqgroup: '(' primitive seq ')';

group: pargroup | seqgroup;

comp: primitive ( par | seq )?;
```

References

[1] A. Sobe and L. Böszörmenyi, "Non-sequential multimedia caching," in *International Conference on Advances in Multimedia MMedia2009*. IEEE Computer Society, 2009, pp. 158–161.
[2] M. Lux, O. Marques, K. Schöffmann, L. Böszörmenyi, and G. Lajtai, "A novel tool for summarization of arthroscopic videos," *Multimedia Tools and Applications*, vol. 46, no. 2, pp. 521–544, January 2010.
[3] G. S. Blair and J.-B. Stefani, *Open Distributed Processing and Multimedia*. Addison-Wesley Longman Publishing Co., Inc., 1998.
[4] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, and Y. Manolopoulos, "Cdnsim: A simulation tool for content distribution networks," *ACM Transactions on Modeling and Computer Simulation*, 2009.
[5] J. Jin and K. Nahrstedt, "Qos specification languages for distributed multimedia applications: a survey and taxonomy," *Multimedia, IEEE*, vol. 11, no. 3, pp. 74–87, July-Sept. 2004.
[6] J. Altmann and P. Varaiya, "Index project: user support for buying qos with regard to user's preferences," in *Quality of Service, 1998. (IWQoS 98) 1998 Sixth International Workshop on*, May 1998, pp. 101–104.
[7] X. Gu, K. Nahrstedt, W. YUAN, D. Wichadakul, and D. Xu, ""an xml-based quality of service enabling language for the web"," *Journal of Visual Language and Computing, special issue on multimedia languages for the Web*, vol. 3, pp. 61–95, 2002.
[8] S. Frolund and J. Koistinen, "Qml: A language for quality of service specification, hpl-98-10," HP Laboratories, Tech. Rep., 1998.
[9] I. Foster and C. Kesselman, "The globus project: a status report," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, Mar 1998, pp. 4–18.
[10] O. Lampl, E. Stellnberger, and L. Boeszoermenyi, "Programming language concepts for multimedia application development," in *Modular Programming Languages*. Springer, September 2006, pp. 23–36.
[11] O. Lampl and L. Böszörmenyi, "Adaptive quality-aware programming with declarative qos constraints," in *Internet and Multimedia Systems and Applications, EuroIMSA 2008*, M. Roccetti, Ed., 2008.
[12] D. C. Bulterman and L. W. Rutledge, *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer Publishing Company, Incorporated, 2008.
[13] Y. Zhao, D. L. Eager, and M. K. Vernon, "Scalable on-demand streaming of nonlinear media," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1149–1162, 2007.
[14] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, pp. 331–373, 2003.
[15] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1999, pp. 1310–1319 vol.3.
[16] K. L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, 2001, pp. 36–44.
[17] J. Yu, C. Chou, Z. Yang, X. Du, and T. Wang, "A dynamic caching algorithm based on internal popularity distribution of streaming media," *Multimedia Systems*, pp. 135–149, October 2006.
[18] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Disc: Dynamic interleaved segment caching for interactive streaming," *Distributed Computing Systems, International Conference on*, vol. 0, pp. 763–772, 2005.
[19] G. Pallis and A. Vakali, "Insight and perspectives for Content Delivery Networks," *Commununications of the ACM*, vol. vol. 49, no. 1, 2006.
[20] A. Vakali and G. Pallis, "Content Delivery Networks: Status and Trends," *IEEE Internet Computing*, vol. 7, no. 6, 2003.
[21] "Omnet++ discrete event simulator." [Online]. Available: http://www.omnetpp.org
[22] "Inet framework." [Online]. Available: http://inet.omnetpp.org
[23] "Cdnsim." [Online]. Available: http://oswinds.csd.auth.gr/˜cdnsim/; accessed 12/09
[24] C. Kofler and M. Lux, "Dynamic presentation adaptation based on user intent classification," in *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*. New York, NY, USA: ACM, 2009, pp. 1117–1118.
[25] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, "Medisyn: a synthetic streaming media service workload generator," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2003, pp. 12–21.
[26] A. Sobe and L. Böszörmenyi, "Towards self-organizing multimedia delivery," Reports of the Institute of Information Technology, Klagenfurt University, TR/ITEC/12/2.08, Tech. Rep., 2008.
[27] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.