Shared Content Addressing Protocol (SCAP)

Optimizing multimedia content distribution at the transport layer

Koen De Schepper, Bart De Vleeschauwer, Chris Hawinkel, Werner Van Leekwijck
Bell Labs
Alcatel-Lucent
Antwerp, Belgium
koen.de_schepper@alcatel-lucent.com

Jeroen Famaey, Wim Van de Meerssche, Filip De Turck
Department of Information Technology
Ghent University-IBBT
Ghent, Belgium

Abstract— **In recent years, the networking community has put a significant research effort in identifying new ways to distribute content to multiple users in a better-than-unicast manner. Scalable delivery is more important now video is the dominant traffic type and further growth is expected. To make content distribution scalable, in-network optimization functions are needed such as caches. The established transport layer protocols are end-to-end, and do not allow optimizing transport below the application layer, hence the popularity of overlay application layer solutions located in the network. In this paper, we introduce a novel transport protocol, the Shared Content Addressing Protocol (SCAP) that allows in-network intermediate elements to participate in optimizing the delivery process, using only the transport layer. SCAP runs on top of standard IP networks, and SCAP optimization functions can be plugged-in the network transparently as needed. As such, only transport protocol based intermediate functions need to be deployed in the network, and the applications can stay at the topological end points. We define and evaluate a prototype version of the SCAP protocol using both simulation and a prototype implementation of a transparent SCAP-only intermediate optimization function.**

Keywords - Buffering, Caching, Content Distribution, Multicast, Networking, Retransmission, SCAP, Scheduling, Streaming, Time-Shift, Transport protocol, Transparent

I. INTRODUCTION

Content delivery over the Internet and IP networks in general is increasing enormously, with a clear trend towards video being the dominant traffic type. Next to existing IPTV and VOD systems, new internet video services with their typical consumption patterns have emerged, such as live streaming, video on reservation and user generated content, all having different timing and latency requirements for delivery. Next to RTP and RTSP-based video delivery, another recent trend is the rise of Internet video using the standard HTTP infrastructure, running on top of TCP as transport layer. A number of mechanisms are used such as HTTP progressive download, HTTP adaptive streaming, and proprietary formats tunneled over HTTP. As TCP is an end-to-end transport protocol, scaling of the delivery infrastructure typically is achieved by distributing HTTP proxy servers nearer to the end-users and redirecting requests towards the most appropriate proxy, as such introducing application-layer functions into the network. The deployment of scalability and optimization functions at the application level requires a lot of preparation, including application design impact and operational management and configuration. Our first objective is to define a transport protocol that allows applications to stay at the topological end points, and needs only transport protocol layer packets to be processed in the network for the optimized content delivery functions. Our second objective is to define the transport protocol such that it enables transparent and gradual deployment of intermediate functions and can operate on the existing IP infrastructure. Additionally, our third objective is to exploit the different timing requirements of the different applications. We believe that using deadlines at the lower layers in combination with announcing content requirements upfront, will result in additional improvements compared to real-time delivery or near real-time (greedy flows with fair-share bandwidth division and limited client buffers as in most current HTTP based video services). In this paper we define a basis for an internet transport protocol called Shared Content Addressing Protocol (SCAP).

This paper is structured as follows: Section 2 describes some work related to scalable content delivery. The SCAP protocol is explained in detail in Section 3, and Section 4 describes a back-to-back transparent proxy as an example for a possible in-network function. Section 5 presents the results of a simulation and prototype evaluation. Finally we indicate items for future work in Section 6, and conclude with Section 7.

II. RELATED WORK

In this section we discuss some work done in the domain of scalable content distribution.

CDN solutions make use of the popularity of HTTP protocol and its capabilities for redirection to proxy servers and application replicas. These services are good for offloading the core of the network, but are not generally economically viable for deeply deployed caches, very close to the clients (such as caches for access and aggregation networks). To achieve significant cache hit rates with the commonly known cache replacement algorithms (e.g. LRU, LFU), cache sizes should be large enough. For small caches, timing information in combination with sufficient upfront content request advertising [5] and recommendation based request overlap [13] can improve the cache performance dramatically. Though these mechanisms can also be applied on HTTP based content distribution, they introduce extra application state and algorithms, and don’t result in a solution for our targets.

This work was performed partially within the PHANTER research project and supported in part by the I.W.T. (Flemish Institute for Scientific and Technological Research in Industry)

With the Subscribe-GET (S-GET) extension to HTTP, the authors of [9] propose a mechanism to setup a standardized HTTP based multicast tree between HTTP proxies to support scalable live traffic. It is used to support scalable real-time multicast of datagrams over HTTP. Per channel, a delivery tree is built between HTTP proxies that multicast any datagram sent by the server to all subscribed clients. As this is a solution for content that is only relevant for real-time consumption by many, it is not supporting content reuse over time (timeshift) and transparent gradual deployment.

For scalable delivery of content also peer-to-peer application overlay multicast trees are well studied [3].The main advantage is server offload in the absence of network support for multicast. Their major drawbacks are a substantial overhead on the local network (if sufficiently localized, otherwise even larger global overhead), and the consumption of extra resources at the peer nodes.

A more radical clean slate approach is the Content Centric Networking (CCN) paradigm [7]. CCN focuses on a content naming solution replacing the current internet, decoupling document naming from the location where the document resides. CCN gives human readable names to documents and chunks documents in large packets using an application dependent chunk naming scheme. Due to the globally unique content naming per packet and its pull based protocol behavior with the request/response (interest/data) mechanism, it also allows packet level reuse over time and between different users. CCN redefines the complete network stack, envisioning as extra advantages global content mobility and security as part of the underlying mechanism. While CCN can run over IP, using it as a data link layer, the final objective is to replace the existing internet. Questions rise around scalability due to the large amount of names to be known in the network and security handling overhead. Another open question is if all types of communication paradigms can be handled efficiently with the CCN naming mechanism, as all communication needs to use CCN as communication means (both shareable static content as private and live communication).

Protocol independent redundancy elimination (PIRE) is another line of work related to content optimization by reuse in the network [10][2]. It removes duplicate byte series in network packets going over the same link between 2 nodes. Results show that there is a lot of redundant information, and that a lot of identical parts of info is passed between clients and servers. The advantage of PIRE is that redundancy elimination is automatic, without the need for specific application actions. PIRE is protocol independent, and can eliminate redundancy even for SCAP, between different servers and protocols. A disadvantage of PIRE is that it does not offload the server, because the content must be delivered before it can be analyzed and eliminated.

III. SCAP PROTOCOL

SCAP is a transport protocol specialized in content distribution and streaming from a server to multiple clients. We designed the protocol to retrieve static content that can be reused in the network between clients, even over time. SCAP has specific information in its protocol headers that make intelligent in-network functions possible without the help of application layers. First, a simple unique content identification scheme allows identifying each byte uniquely. Secondly, the content range (comprising offset and size) that needs to be transported is provided. The third enabler is the linear stream deadline information. It is a simple expression to determine the deadline for each requested byte in the content range.

The protocol allows retrieving flexibly addressable ranges of content, with accompanied linear mapped deadlines. The client application provides this information to the transport layer, allowing all nodes to combine requests for the same content, to store and reuse previously received content and to schedule the receive, store, delivery and replacement of that content. Intermediate nodes can decide - independently from each other - which content to store and for how long. Clients can request content with a content size from a few bytes to gigabytes and with deadlines from mere milliseconds up to several days. A multimedia presentation that knows upfront the scenario to be played can request content upfront, providing extra room for scheduling, buffering and caching optimizations.

The following subsections explain the key concepts of the SCAP transport protocol. We describe the message structure, request and content identification, and the dynamic behavior between client and server. Client and server can either reside on end-points or on intermediate network elements. An intermediate node should be seen as a server for the original client and a client for the original server, and should behave accordingly. Intermediate functions are using the protocol, but are not part of the protocol. Intermediate functions are described in the next section, and are allowed to process messages in any way, as long as the in- and outgoing flows respect the protocol as described in this section.

A. *Content identification*

In SCAP, content is identified uniquely and efficiently by the combination of IP address (IPv4 or IPv6) and a local 128 bit identifier. Each IP address has its own 128 bit address space. Each origin server is responsible for mapping content correctly in the local address spaces of each IP address it is responsible for. It is the responsibility of the application to transport this low level address provided by the server to the client, using commonly known mechanisms (for instance embedding a URL in an HTML page or specifying the parameters in a video manifest file).

TABLE I. MESSAGE LAYOUT

Message	Layer 3: IP	Layer 4: SCAP
Request	Protocol = SCAP Source address (S) = client Destination address (D) = server	Flags = Rq Request id (Ri) Content address (Ca) Content size (Cs) Deadline begin (Db) Deadline end (De) Receive window (Wr)
Response	Protocol = SCAP Source address (S) = server Destination address (D) = client	Flags = Rp Content address (Ca) Content size (Cs) Deadline packet (Dp) Expiration time (Te) Request id (Ri)

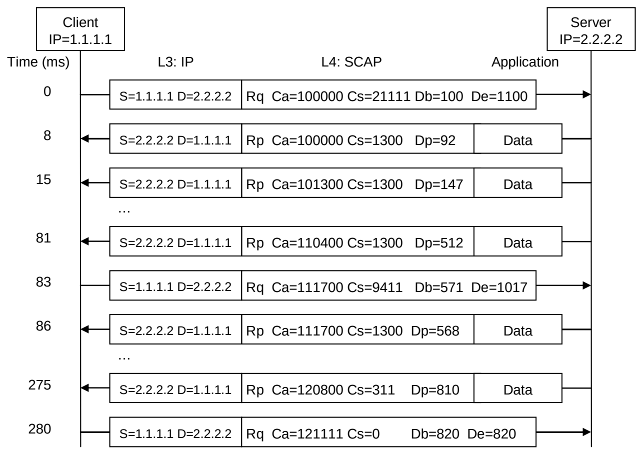

Figure 1. Example of SCAP messages in a request session.

Typically, content is addressed sequentially in ranges. A content range is identified by a start address and a size. In the example in Fig. 1 the client requests a range of content identified as 2.2.2.2:100000~21111, which means that it needs content from the server with IP address 2.2.2.2, starting at local content address 100000 and with a content size of 21111 bytes.

B. *Request and response messages*

The protocol is built around two basic message types: a request message and a response message. Fig. 1 shows an example flow for one request session from a client towards a server and in Table 1 a high level representation of the message format is shown. A client can request content by sending request messages. Request messages contain the start address and size of a consecutive range of bytes a client is interested in, together with timing requirements for the delivery. In Fig. 1, a request is sent for 21111 bytes of content (= Cs), where the first byte is expected within a time span of 100 ms (= Db), and the last byte 1 second later at 1100 ms (= De). Request messages are updated and sent repeatedly (in this example 83 and 280 ms after the first request) to refresh the request, until all content has been received. Refresh messages can be updated, due to either changed timing requirements, or partial completion of it. In the example, the message at 83 ms requests only the part of the content that is not yet received and the last message at 280 ms closes the request session by sending a request with content size set to 0.

The server sends response messages, each carrying a part of the requested content. The response message header includes the content identification of the content range it carries.

Both request and response messages contain a unique request identification that is the concatenation of the L3 network address of the client and a L4 request id value. Since request messages are sent multiple times for the same request session, and the requested content range changes over time (typically, shrinks or slides as the request is gradually completed), a request identification is needed for fast and unambiguous lookup of the request states on the server, and for fast mapping of the response messages to the request states on the client. In the example in Fig. 1, the request identification field is left out for brevity, but is the same for all messages that are shown, since they handle the same client request session.

A client can change its mind about the timing requirements, or can even give up its request, for instance when pausing playback of a video stream. To take this into account, the request should be rescheduled in time by updating the deadlines in a new request message with the same request id.

Content is acknowledged by means of a new request message with the same request id, but with a higher start address. As such, the server is informed that the given client is no longer interested in the part before the new start address. Similarly, an application can skip content, even when the data was not received at all.

A client expects to receive content for a request in an earliest-deadline-first order. In case a gap is detected while content is being received (i.e. not the next expected address), the client must send a new request with a new request id for the missing gap that was detected. The original request will acknowledge both the missing gap and the received content. This mechanism results in a selective NACK mechanism, similar to the TCP Selective Acknowledgment mechanism [8], but reuses the normal request message instead of extra options which keeps the protocol handling simple.

For the case that content is lost, and there is no later content received to create a detectable gap, a second mechanism is defined. It is similar to the TCP fast retransmit mechanism **Error! Reference source not found.**, and is based on receiving duplicate acknowledgments. If the server receives a specified number of duplicate acknowledging request messages with an identical start address smaller than the last send address by the server, the server should retransmit messages from this start address. As such, this mechanism allows dealing with packet loss in a way that complements the first mechanism. As long as there are newer packets, the first mechanism will trigger a new request for the missing gap. If this gap is at the end of a range, the client will not request the missing gap based on the first mechanism, as it is not detected, and the server will wait for the final acknowledgment. Only the second mechanism based on the duplicate acknowledgements will trigger the server to resend the lost content. Note that duplicate acknowledgements for the last packet(s) are triggered by timeouts, and can take a long time to arrive.

When a request is fulfilled, a final request message with content size 0 must be sent, allowing clean up of network and server state for this request. In Fig. 1 the message at time 280 is an example of such a final acknowledgment message. To avoid unnecessary delivery of content that is no longer asked for, requests can time out. If a request received by a server is not refreshed within a request timeout window, the server can forget this request. A client can refresh the request by sending a request message with the same request id at a certain interval, sufficiently below the request timeout threshold.

Some extra fields that are shown are outside the scope of this paper, but are necessary for receive window management, congestion control, content expiration, etc …

C. *Linear stream deadlines*

As shown in the example in Fig. 1, SCAP allows requesting content with timing requirements. In general, client applications are aware of the deadlines of content they need. If the network is aware of these deadlines, the delivery of content over multiple clients and servers can be optimized by evaluating the scheduling priority of several competing requests.

For continuous media delivery we typically have a specific arrival curve of the content stream. Assuming a constant bit rate, each byte has a deadline that is c seconds later than the previous byte. If a stream has a constant bit rate r in bytes per second, then $c = 1/r$. In SCAP the deadlines of the first and last bytes are specified in the request. Every intermediate byte x has a deadline d_x that is obtained through linear interpolation, as:

$$d_x = d_b + x \, . \, c = d_b + x \, . \, (d_e - d_b) / s \qquad (1)$$

with d_b and d_e the deadline of the beginning and the end of the range, s the content size, and d_x the deadline of byte x. This combination of a start and end deadline, with its linear interpolation, mapped to a range, is what we call a linear stream deadline. In the remainder of the document this is also simply called the deadline if applicable to a range. Deadlines are expressed in milliseconds with a value that indicates the time span between now and the latest expected delivery time to the client. A deadline is positive if it is in the future, 0 if it is now, and negative if it is expired. We used a signed 32 bit fixed-point data representation for simplicity, since 32 bit allow to cover a range of [- 24 days, + 24 days] with a resolution of milliseconds.

When a node keeps state of a request, it decrements the deadlines according to progression of time. The time that a message is "on the wire" should be compensated by the request sender. This is the round trip time between the 2 SCAP aware nodes and can easily be calculated by subtracting the local deadline by the received deadline in responses.

The linear interpolation could be seen as restrictive, but note that a content request can be split in multiple ranges with linear deadlines to approximate more complex deadline functions. The granularity of the approximation depends on the client buffer size and response time constraints of the receiver.

IV. SCAP Functions

Intermediate SCAP functions are transparent to clients, servers and other intermediates, which facilitates their incremental introduction. Non-SCAP elements use standard IP mechanisms to route the SCAP messages to the next node. This can be seen as the basic non-optimizing SCAP function, and any extra functionality is possible as long as it is desired or improves the overall efficiency of content delivery. SCAP functions can be implemented on the packet level for maximum throughput, with minimal soft-state. The amount of required soft state depends on the capability and complexity of the deployed function. The SCAP-aware functions can be part of a switching/routing device, or can be attached to a non-SCAP-aware node, which is configured to redirect all or a subset of the SCAP traffic to this SCAP-aware function. Since SCAP uses a dedicated IP protocol id, its redirection is supported by most of the existing network devices.

Different functions can be envisioned, ranging from measurements, shaping, policing/admission, billing, buffering, caching, scheduling and scaling functions. The advantage of understanding the requests from the clients is that most functions can perform control functionality on requests before the server is informed and responses with data arrive. For instance policing can be performed by inspecting and manipulating the requests coming from the clients, before they are sent to the server.

Depending on the needs of the function different system architectures can be used to implement such a function. A SCAP-aware node can be implemented as a back-to-back client-server or as a packet-processor/forwarder. A back-to-back implementation is consuming all incoming packets, storing state in internal structures, and generating packets independently from the incoming packets. Alternatively, a packet processor/forwarder implementation is steered by update and forward rules, generated by a control plane. The implementation choice depends on the constraints and limitations of the function. A measurement function is the simplest function, and can just inspect the request messages, without the need to filter or manipulate them.

Below a few relevant scenarios and aspects are detailed.

A. *Back-to-back message flow*

Fig. 2 shows the message flow for a back-to-back SCAP proxy. It is demonstrated for the first messages of a simple scenario. A client sends a message (1) towards the origin server to request a range of content. A SCAP router is allowed to intercept the message, and finds out that a part of the requested content (2) is already in the local buffer or cache. The router sends this content as a response message (3) towards the client. The source IP address of a response message must be kept as the address of the origin server, since that IP@ is part of the content identification. The response carries the deadline of the most urgent byte in that packet (typically the first byte), aged by the processing time (here as an example 10 ms). The client can use it to calculate the roundtrip time if he compares this value with his own expected deadline value. The router also sends a request message (4) towards the server for the missing part of the content. The source address of the request message is changed to the source address of the router, to assure that the response arrives to the SCAP function, instead of following potentially an alternative path directly to the client. This action is only necessary if the intermediate function generates the request (and request id) and needs to process the responses for this request. The deadlines are recalculated for the partial range, and aged by the time necessary for internal router processing (here as an example: 20 ms). The request arrives at the origin server, where the content is originally stored. The server sends response messages (5) to the router. The router can then send this content further to clients that have asked for that content. The request-ids are not shown, but should be unique per client IP address. Since the SCAP router behaves as a single client towards all other servers (or other intermediate

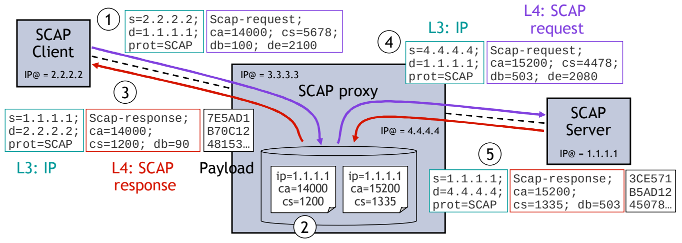
SCAP Client
IP@ = 2.2.2.2
1
s=2.2.2.2; d=1.1.1.1; prot=SCAP
Scap-request; ca=14000; cs=5678; db=100; de=2100
L3: IP
L4: SCAP request
4
s=4.4.4.4; d=1.1.1.1; prot=SCAP
Scap-request; ca=15200; cs=4478; db=503; de=2080
IP@ = 3.3.3.3
SCAP proxy
3
s=1.1.1.1; d=2.2.2.2; prot=SCAP
Scap-response; ca=14000; cs=1200; db=90
7E5AD1 B70C12 48153…
L3: IP
L4: SCAP response
Payload
IP@ = 4.4.4.4
ip=1.1.1.1 ca=14000 cs=1200
ip=1.1.1.1 ca=15200 cs=1335
2
SCAP Server
IP@ = 1.1.1.1
5
s=1.1.1.1; d=4.4.4.4; prot=SCAP
Scap-response; ca=15200; cs=1335; db=503
3CE571 B5AD12 45078…

Figure 2. SCAP back-to-back message flow.

proxies), it must also allocate unique request-ids for outgoing requests that it issues. The SCAP router behaves as multiple servers towards its clients, proxying the real servers.

B. *Buffering and Caching*

Since SCAP publishes the required content, size and deadlines upfront, intermediate nodes can detect overlapping requests. They are allowed to combine content overlaps by requesting only the most urgent ranges, and later when receiving content sending it to multiple interested clients, possibly with additional delay by buffering it. At a second stage, content that is already buffered for another node can be reused to send to a new interested client. If the new request had a more urgent deadline, it can be sent to the second client before it is sent to the first client. If the deadline is later, then the content can be buffered longer than originally planned. As long as there are clients known to wait for stored content, we define the storing action as buffering. If content is kept inside a node without any known client expressing a need for it, we define this action as caching. Caching is useful if memory is available after buffering content for all known requests. Also if content is estimated to be more popular (statistically needed sooner) than buffered content with the longest deadline, it can be useful to reduce buffer capacity to keep content cached.

The scheduling timing for receiving (open the receive window), storing (buffering and caching) and sending (assigning priority and performing congestion control) is not defined as part of the transport protocol. It is up to the clients, servers and intermediate function to decide what actions are possible and beneficial to improve the overall content delivery efficiency. A topic for further research is to define the most optimal delivery strategy.

C. *Intermediate soft-state management*

In Fig. 3, a detail of the intermediate soft-state is shown for a back-to-back implementation. A Response List is keeping a list of all incoming requests that need responses to be sent to. The entries are indexed via the client IP address and the unique request id given by that client. An entry contains further the state that is needed to handle the incoming request dynamics (request, send and acknowledgement addresses and timeout), and has a link in the content index to ranges that cover the consecutive content range that is requested by the client. This Response List is used by the response scheduler, which is responsible for the scheduling of response packets on the links of the node.

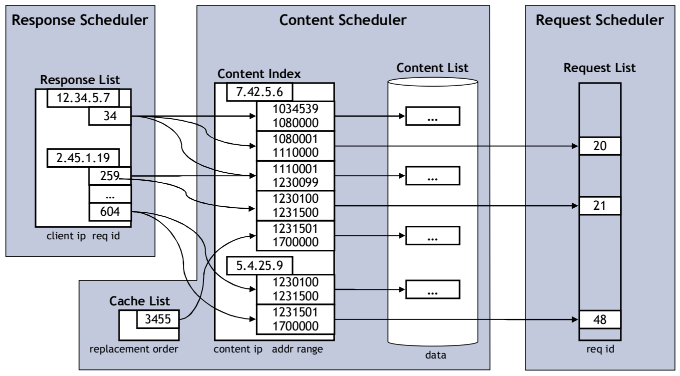

Figure 3. SCAP back-to-back internal state.

The responses determine the entries in the Content Index. This index is a unique overview of the all requested content and on its turn determines the entries in the Content List and Request List. If the content in a range is available in the node, then the Content Index points to a Content List entry, if not, it points to a Request List entry. Content that is not referenced by any incoming request can be referenced by the Cache List or else is freed again. The Content Scheduler is responsible for managing the content stored in the node. It determines the amount of content that is requested, received, buffered and cached. The entries in the Request List are used by the Request Scheduler, which is responsible for sending request packets (for new requests, or to refresh and acknowledge existing requests). Any new entry in the Request List gets a unique request id assigned by this node because requests are sent with the IP address(es) of this node. When response packets are received, the range in the Request List entry shrinks and grows in the preceding Content List entry. Entries in the Content List are shrunk when acknowledgements are received from clients and make preceding entries referenced by the Response or Cache List grow.

V. EVALUATION

We evaluate the protocol in the context of optimization functions in access networks. These nodes are highly distributed (a large number of devices are deployed), serve a relatively low number of users per node (hundreds to thousands), and have limited storage capabilities (cost reduction is an important objective). We evaluate the benefit of SCAP in an access network for making multimedia delivery more scalable. We compare native SCAP transport using an intermediate SCAP aware access node, with HTTP based delivery using an intermediate HTTP proxy. Typically an access network has many access nodes, each serving a relatively small amount of clients with limited available resources. In this context, we focus on the memory and bandwidth usage efficiency.

A. *Scenario and Topology*

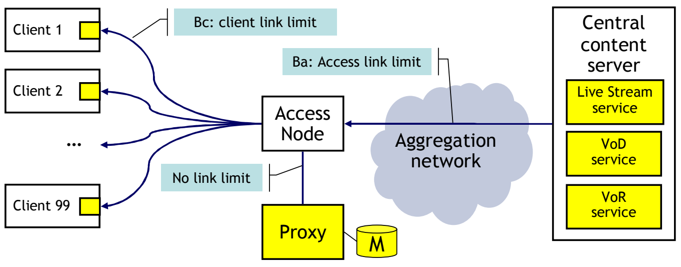

Figure 4. Evaluated topology.

In this evaluation, both simulations and prototype runs were performed. For both, the same topology and input data were used.

The topology in Fig. 4 represents 3 different applications on a central content server, one access node extended with an intermediate proxy and 99 video clients, each requesting one video session for 1.5 hours viewing time. The available buffer memory on the clients is not limited to give the intermediate function a high level of scheduling freedom. The available bandwidth on the link between the content server and access node (Ba) was varied in order to study the effect of bandwidth bottlenecks on performance. The link between the access node and every client (Bc) was given a fixed 20 Mbps bandwidth limit. This bandwidth is reserved for the generated SCAP or HTTP traffic and resembles actual bandwidths assigned to managed multimedia applications.

Each application on the content server offers another type of content. First, the live stream content type (LS) represents an infinite video stream where content is made available progressively in time. Second, the video on demand content (VoD) represents pre-recorded streams of a finite length that are requested just before the user wants to start viewing them. Finally, video on reservation (VoR) is an alternative to video on demand, where content is requested up to 1.5 hours before usage. In the presented evaluation scenario, live streams and video on demand content have a bit rate selected at random from the set (3, 8) Mbps, while video on reservation has a bit rate randomly selected from the set (4, 15) Mbps. A higher quality (and consequential bandwidth) is assumed for VoR as an incentive for the user to reserve its content upfront.

A constant bit rate is assumed for all types of video. Each content server contains 6 unique content items of its type. Each of the three services is requested by 33 clients with a content item that is uniform randomly selected from the available 6. Live stream content is requested with a deadline of 1 second, which means that the user expects the live stream to start playing one second after selecting the requested content. The live stream content selected by the client has a delay of 10 seconds compared to the live feed. This makes it possible to download the content up to 10 seconds upfront, but the start of the playout begins already after 1 second. For video on demand the deadline is 10 seconds, while for video on reservation it is chosen at random from the interval [30, 180] minutes. Video on demand and video on reservation are fully available on the server, and download is only limited by link capacities. The scenario runs over a 6 hour period. Requests arrive at uniform random moments within this timeframe.

In both the simulation and the prototype we measure the minimum deadline value that the intermediate is scheduling. If deadlines are not met (go below zero), both the prototype and the simulation continue scheduling the content, with negative deadline values as a result. The minimum deadline (in seconds) over the complete experiment is plotted for different access link capacities to compare the different scheduling methods. If this time is at least zero, all deadlines are met, and a perfect delivery is achieved. The minimum link bandwidth for which all deadlines are met (B_M) for all clients is a measure for the performance of the scheduling, buffering and/or caching algorithm. To evaluate the content reuse performance, also the hit ratio is calculated as one minus the ratio of the total content size transported over the Ba link and the total content size transported over the Bc links. The utility measure gives an indication of how well peak bandwidth requirements can be smoothened out over time. It is the ratio between the link capacity (B_M) and the average bandwidth over the access link.

For the SCAP prototype and the simulator we have not used buffer control, buffer replacement and congestion control mechanisms. Enough memory is available to buffer any excessive incoming rate which is not consumed by a compensating outgoing rate. Link congestion is avoided by limiting the rate at the source to the bandwidths defined for the links. Simulations are done using fair-share scheduling resembling TCP, but buffer and request reuse as in SCAP were used in stead of LRU caching. The HTTP proxy prototype on the other hand uses the LRU replacement algorithm with a fixed cache size that is at least the peak buffer size used by the SCAP proxy, and link capacity is limited using the Traffic Control settings in the Linux kernel (tc command).

B. *Simulation*

A simulation implementation for scheduling and buffering was built in order to compare earliest deadline first (EDF) with fair share (FS) and real-time (RT) scheduling. The simulator abstracts the underlying network layers and packet-based transport mechanisms. Instead, it uses a stream-based event-driven approach. These streams are modeled as request flows between adjacent network nodes. Every flow is associated with a single-hop outstanding request and keeps a state in the form of the address of the next byte that needs to be sent towards the receiver, its deadlines and a current bit rate. Before and after every event, the simulator recalculates the state of all flows and node buffers for the new time, and searches for the next soonest event. These events can be for instance the start and end of a flow or deadlines of flows that meet each other. These event actions are performed in a loop for every event until all requests are handled. Further, the network latency, node processing delay and packet loss ratio on the network links are assumed to be neglectable and set to zero. The simulator makes abstraction of the actual transport protocol, and just applies the correct scheduling algorithm. For all scheduling algorithms the memory is used only for pure buffering and buffer reuse. Once content is not requested anymore, it is discarded and its memory is freed again.

In Fig. 5 the minimum deadline per access link bandwidth (Ba) limitation is plotted for the different scheduling policies and topologies.

The values for minimum bandwidth (B_M), link utility and hit ratio are shown in Table 2.

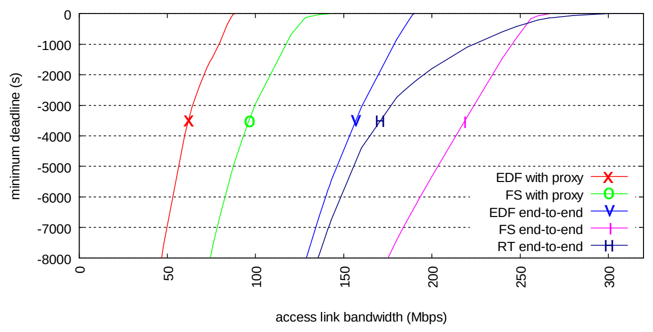

Figure 5. Simulation minimum deadlines per access link bandwidth.

TABLE II. SIMULATION RESULTS

	Without proxy			With proxy	
	EDF	**FS**	**RT**	**EDF**	**FS**
B_M (Mbps)	190	267	310	88	143
Utility (%)	88	62	54	85	64
Hit ratio (%)	0	0	0	55	45

Without a proxy (end-to-end), RT scheduling requires a peak bandwidth of 310 Mbps. RT scheduling is not using the scheduling freedom, and content is delivered at the deadline. FS and EDF scheduling start using the bandwidth from the moment a request is known. Therefore they can serve requests upfront, and require less peak bandwidth with higher utility. Without a proxy, the end-to-end flows are limited to the 20 Mbps link capacity of the client links. Only when the flows start competing for bandwidth on the access link, the scheduler at the server will determine the bandwidth share. The FS scheduler is just dividing the bandwidth evenly over the running flows, and the flows with highest bandwidth and shortest deadlines will miss their deadline first. As EDF is giving priority to the flows with the shortest deadline, it requires the smallest link capacity.

With a proxy, FS and EDF scheduling use the full link capacity if at least one request is known. If the bandwidth is higher than the 20 Mbps client link limitation, the content is buffered in the proxy. This not only allows content to be reused over the different requests, but also to fully use the access link capacity whenever requests are known.

The results for our scenario show that EDF without a proxy requires 39% less bandwidth than RT, and with a proxy even 72% less. Compared to FS, EDF requires 29% less bandwidth without a proxy, and 39% less with the proxy. These results are promising, because they show that using SCAP end-to-end can already give a substantial advantage compared to FS and RT based protocols, and that adding intermediate functions is further increasing substantially the performance of the delivery network.

C. Prototype evaluation

In addition to the simulation results, the prototype implementation of the SCAP protocol was evaluated in a physical setup. In the prototype setup, the SCAP protocol with an earliest deadline first scheduling strategy is compared with the EDF simulation results and an HTTP setup containing an HTTP proxy. The same client scenario is used for the prototypes as was used for the simulations. Due to the memory limitations of the in-memory buffer of the SCAP router and the in-memory LRU-cache of the HTTP proxy, time was down-scaled with a factor of 10, and bandwidths with a factor of 4. The results in the graphs are again scaled up. This setup was created on a Emulab setup [12]. The nodes in the Emulab network are equipped with two dual core opteron 2212 processors and 4GB RAM.

The SCAP prototype is a simplified implementation of the back-to-back proxy described above, focusing on a first set of functionalities. Workarounds are defined for functionalities that are out of the scope of this paper. All limitations and restrictions described here should not be applicable to the final protocol or final intermediate function behavior, and are only provided for a detailed understanding of the evaluation results.

The prototype does not implement the Cache List described in Fig. 3. As in the simulator, content that is not referenced by any incoming request is immediately freed again, and thus memory is used only for buffering and buffer reuse. This means that the amount of reuse is only determined by time-shifts due to overlapping requests pending at the same time. This is actually reducing the potential for content reuse, and is a disadvantage for the SCAP runs, and an advantage for the HTTP-LRU runs, which can use all available memory.

The prototype implements neither a congestion control nor receive window strategy. The receive window is always kept completely open, so there is no way to limit incoming flow and limit the buffering, and hence memory usage. The buffering in the proxy is limited only by the incoming link capacity. Therefore the response scheduler uses a configured scheduling rate limitation per request IP address. This rate is used to limit the bandwidths for the SCAP runs.

The response scheduler uses strict earliest deadline first scheduling. If content is available in the Content List for an incoming request, then the request is inserted in a deadline ordered list. The next packet is sent from the first entry in this list, and the request is reinserted in the list for the next not yet sent byte. As a result, requests are preempted by more urgent requests. As long as a request is the most urgent, it will be served with priority. The requester will receive responses only for this request with the full link capacity. If the deadline of a first request reaches the same value as for another request, the two requests will share the link capacity, based on the deadline rate ratio. This resembles exactly with the EDF implementation in our simulator. The response scheduler retransmits after 2 duplicate acknowledgments. Refresh and timeout times are fixed. The request scheduler used 100 ms for the refresh interval, and the response scheduler will use a timeout window of 250 ms before removing an incoming request and all its state, which allows for one consecutive lost request message.

The prototype has additionally a local socket interface that applications can use to request ranges with deadlines, and read from the content stream. Additionally, an API is present to insert content directly in the Content Index and Content List. As such the same prototype process is started in all SCAP-aware nodes, and having the same functionality for the clients, server and intermediate function.

In addition to the SCAP setup, which matched the topology and setup that was also used in the simulation, an HTTP setup was also created with the Squid caching proxy [11]. This setup has the same topology as the SCAP setup, but with HTTP over TCP as the protocol and an HTTP proxy with the least recently used (LRU) cache replacement strategy. The peak buffer memory usage of the SCAP scenario was 20GB, so we configured the LRU proxy cache with the same memory limit. We applied the same scaling factors as for the SCAP prototype, resulting in the same down-scaled cache limit of 500MB. The same client requests with the same timings were applied. The video files were stored as 2 second fragments to facilitate caching in limited memory. The clients and server are C++ developed processes to control the correct request and release of the video content. The clients will download the requests in sequence. Only if the download of the previous fragment is completed, the download of the next fragment is started. The server will release the content controlled for the live streams and immediately for VoD and VoR. The capacity of the links is limited using the Traffic Control settings in the Linux kernel (tc command).

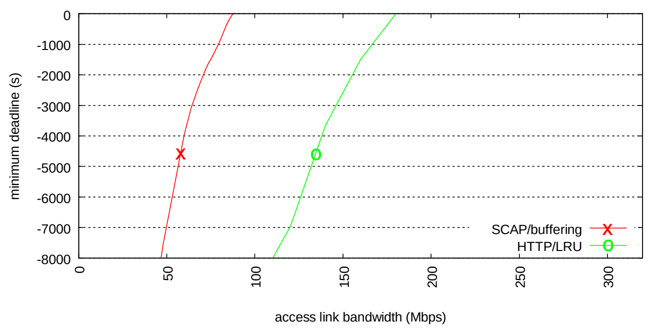

Figure 6. Prototype minimum deadlines per access link bandwidth.

TABLE III. PROTOTYPE RESULTS

	SCAP / buffering	**HTTP / LRU**
B_M (Mbps)	88	180
Utility (%)	85	78
Hit ratio (%)	55	16

In table 3 the results are shown for the prototype runs. The SCAP results are, as expected, equal to the simulation results. Due to the small cache size and the high interval caching [4] opportunities, the SCAP prototype has a 3.45 times higher hit ratio compared to the LRU cache.

Fig. 6 plots the minimum deadline per access link bandwidth (Ba) limitation for the two prototype runs.

The results show that using end-to-end SCAP (as simulated) almost performs as good as a proxied HTTP/LRU setup with a small cache size. Adding the SCAP proxy is further halving the required bandwidth.

VI. FUTURE WORK

In this paper, the focus was on the transparent intermediate optimization concept. We shortly discuss what we think is needed to further evolve the protocol. A receive window strategy with buffer and caching management must be defined. A (near) optimal and simple heuristic is needed that balances the amount of memory usage between 4 types of storage: actual receive window storage, buffer storage for a single flow, buffer reuse between flows and cache storage for statistically popular content. For an internet wide deployment, the security aspects must be taken into account. Also TCP friendliness with slow start and congestion control are important aspects to look into. Further, investigation of the impact of different scheduling strategies and the interworking with congestion control on larger networks is needed. The impact on scalability of using ranges and different strategies for handling them should be further investigated. Finally, exploration of possible functions and a study on their interworking on a global scope should be performed.

VII. CONCLUSION

In this paper we introduced a novel transport protocol that can be used for transparent network optimization. This protocol supports the scalable delivery of content from one server to multiple clients. Content is uniquely identified and deadline information that expresses "when the content is required" is also provided. The SCAP protocol allows intermediate network devices to take this deadline information into account when scheduling the transmission of data packets. By uniquely identifying content it enables application agnostic caching and buffering. We have evaluated the SCAP protocol both in a simulation environment and in a physical setup using a prototype implementation. The prototype behavior matches the simulation results. In the simulation environment, using earliest deadline first scheduling allowed a significant bandwidth reduction when compared to traditional real-time and fair share scheduling techniques. In the prototype test setup, a transparent SCAP function was compared with an HTTP/LRU proxy cache. The tests showed an extra reduction on bandwidth requirements by exploiting the timing information when buffering and reusing content. The presented results show on the one hand that providing and using deadline information improves the efficiency of network optimization, and show also that intermediate nodes can be deployed transparently in the network without impact on clients and servers.

REFERENCES

[1] M. Allman, V. Paxson, W. Stevens. "TCP Congestion Control", RFC 2581 (Proposed Standard), April 1999.

[2] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications", in Proceedings of the 11th international joint conference on Measurement and modeling of computer systems 2009, Seattle, WA, USA

[3] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," available at http://www.cs.umd.edu/projects /nice/papers/compare.ps.gz.

[4] A. Dan and D. Sitaram, "A Generalized Interval Caching Policy for Mixed Interactive and Long Video Environments", Proceedings of SPIE Multimedia Computing and Networking Conference, San Jose, CA, 1996.

[5] J. Famaey, W. Van de Meerssche, S. Latre, S. Melis, T. Wauters, F. De Turck, K. De Schepper, B. De Vleeschauwer, and R. Huysegems, "Towards intelligent scheduling of multimedia content in future access networks", in proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2010.

[6] R. Huysegems, B. De Vleeschauwer, and K. De Schepper, "Enablers for non-linear video distribution", Bell Labs Technical Journal, June 2011, pp. 77-90.

[7] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content.", in CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies, pp. 1-12, New York, NY, USA, 2009. ACM.

[8] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018 (Proposed Standard), Oct. 1996.

[9] L. Popa, A. Ghodsi, and I. Stoica. "HTTP as the Narrow Waist of the Future Internet". In ACM SIGCOMM HotNets, 2010

[10] N. T. Spring, and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic", in SIGCOMM 2000, pp. 87–95.

[11] Squid Web Proxy Cache Home Page, "http://www.squid-cache.org/", 2011.

[12] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks", SIGOPS Oper. Syst. Rev., 36(SI):255-270, 2002.

[13] T. Wu, K. De Schepper, W. Van Leekwijck, and D. De Vleeschauwer, "Reuse time based caching policy for video streaming", submitted to CCNC 2012 - 9th Annual IEEE Consumer Communications and Networking Conference.