A Transport Protocol for Content-Centric Networks

Somaya Arianfar and Jörg Ott
Aalto University

Lars Eggert
Nokia Research Center

Pekka Nikander
Ericsson Research NomadicLab

Walter Wong
University of Campinas

I. Introduction

In the current Internet architecture, the hosts and the network have very different roles. Hosts generate and consume packets; the network is in charge of delivering those packets. The Internet architecture has no inherent notion of "content". In the Internet, content resides in applications that themselves reside on specific hosts. In order to access content across the Internet, a hosts first needs to determine a host that holds (a copy of) the content of interest and then it needs to obtain the specific IP address at which that hosts resides at the time.

In content-centric networking, content becomes a first-order element. It is liberated from the shackles of Internet application silos, and the role of the network changes from transporting topologically addressed packets between hosts to delivering uniquely identifiable content to the hosts requesting it. With this approach, hosts no longer need to identify which other host stores a copy of the content of interest; they simply request a named piece of content from the network and let the network to worry about where to retrieve it from.

Several content-centric networking architectures have recently been proposed, including Van Jacobson's CCN [1] and the PSIRP [2]. The focus of these efforts has so far been mostly on the architecture of the inter-networking functions required for content-centric networking, *e.g.*, identification of content, routing, etc.

This work describes ConTug, a receiver-driven transport protocol that can reliably and efficiently retrieve large pieces of content from the network in a way that is congestion-controlled. The work highlights the transport aspects of content-centric networks and their design challenges.

II. Conceptual details

In content-centric networks, to retrieve a piece of content, a host – the *requester* or *receiver* – issues a *request* for it to the network. The network is responsible for forwarding requests towards the original source that announced the content item. A piece of *content* is a sequence of bytes identified by a globally unique and persistent *identifier* (ID). Pieces of content larger than maximum network segment size require segmentation before they can be transmitted; a common design approach – as this work assumes – is that each resulting *segment* of a large content item is a uniquely identifiable piece of content in its own right.

While we do not assume consistent host names in content-centric environment, some forwarding-level identifiers are needed to tell the network where to send segments to. For this purpose, we use the notion of *forwarding channels*, or

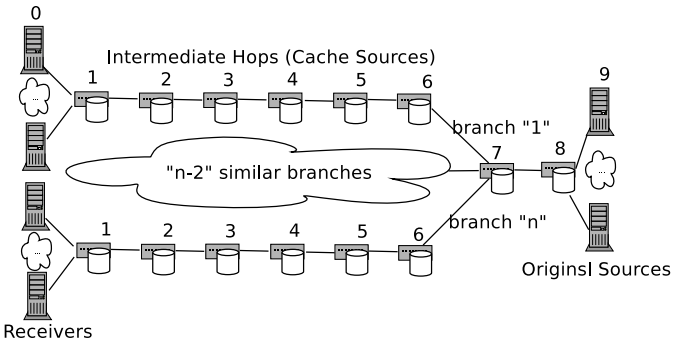

Fig. 1. Example topology with requesters on the left, original sources on the right, and potential caches along the path.

channels for short. Briefly, a channel is an anycast or multicast like network tree, originating at the sender and terminating at one or more locations in the network. A packet traveling along a channel will be forwarded towards one or more of the ends, until someone processes the packet or it hits a dead end.

As discussed in our previous work [3], the network has the freedom of moving and storing the content in different places. Therefore, intermediate nodes on the channel are free to cache some or all content items that are being transmitted across them as they see fit. When such a *real time cache* sees a request for a content item that it has a copy of, it may directly respond with the cached item instead of forwarding the request towards the original source. Caches are consequently also *fractional sources*, trying to reduce the flow completion time (FCT), in case of repeated content requests.

In such an environment, the requested segments will often flow to a requester from several sources at the same time. From the receiver point of view, the different active sources responding to a stream of segment requests in a content network appear to act unpredictably. Because a requester issues requests to the network – instead of sending them to a single, individually known source – it has no direct control over which source will respond. Adding to this apparent unpredictability there are two other factors. First, sources may only cache an arbitrary (non-contiguous) fraction of the segments of a content item, so a source that happened to responded to the some of past segment requests may "disappear" when requests are made for segments it does not have available. Second, whether a given source decides to respond may depend on its load level as well as network load. Fig. 1 illustrates this operation in simple topology with requesters on the left, original sources on the right, and potential caches along the path.

Our subgoal on keeping no control states in the network while having the freedom of moving the content, poses an

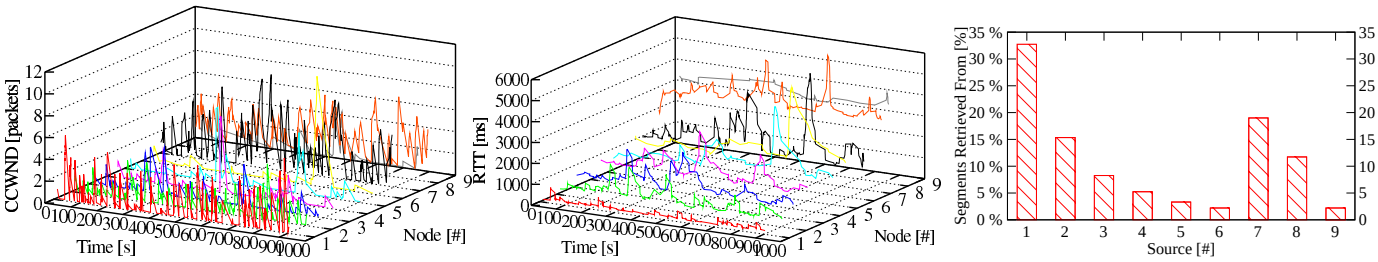

Fig. 2. 32 concurrent ConTug transfers in a scenario *with* caches. Plots show measures taken for a *single* receiver(transfer); showing (a) CCWNDs used with different sources on the left, (b) RTTs to different sources in the middle and (c) which fraction of segments was retrieved from which source.

interesting challenge in our design. The challenge is how to perform some transmission control per known source, because as requester asks for the content the paths to the sources – and hence the control loops used by the protocol – will interact and also *there is no predictability on which source can serve which range of the requests.*

III. Design Overview

To address the conceptual challenges, the ConTug transport is designed as a completely data-oriented pull-based mechanism, where the receiver controls the segment reception based on its local parameters. The receiver is able to either request each of the segments separately or send multiple concurrent requests into the network to enhance its download rate.

Starting with an initialization phase, the receiver gets the meta-information required to retrieve a piece of content. This meta information at minimum contains the channel identifier and the list of segment ids to be requested. In case of real time streaming, instead of the simple list of segment ids, the meta information may contain a seed identifier and a corresponding algorithm to generate the identity of the rest of the segments.

After the initialization phase, the actual content retrieval operation starts. The requester tries to approximate how much resources are available in the network, to serve its requests. The approximation is done over a channel with unpredictable sources without binding to any specific one of them. In order to control the number of outstanding requests and responses, ConTug uses the *Conceptual* congestion control window (CCWND) per channel that operates similar to the TCP congestion window (CWND). Their main difference is that CCWND is kept at the receiver and all estimates are performed by the receiver. Another difference is that ConTug may have to use multiple stochastic conceptual windows for the multiple transient sources on a single channel. To compute the windows, the receiver needs to be able to differentiate between different sources. For example, the sources may be identified by the reply channel identifiers included in the segments, or based on TTL and RTT sample clustering.

The receiver starts requesting segments with one conceptual window, $CCWND_1$. Then, the receiver enters the slow start (later congestion avoidance) ramp-up phase for that window, increasing its size on successful responses. Whenever there is a response from a new source, ConTug assumes more resource availability and creates a new $CCWND_i$. Each response from the source i triggers an increase on the correspondent $CCWND_i$, thereby increasing the overall conceptual window size, allowing more outstanding requests on the channel.

In the design, any indicator of the resource unavailability at the channel , e.g. temporary bandwidth shortage or unavailability of certain range of the segments on a source, is addressed as *congestion*. The requester is able to adapt its request rate to the congestion situation and prevent congestion collapse. To adapt to congestion and source unpredictabilities, ConTug uses the timeouts and increased RTT estimation as signs of congestion and react to them in different ways.

IV. Implementation and Evaluation

In order to evaluate the different features provided by ConTug, we have implemented a new native protocol stack in ns-3. Instead of using IP as the forwarding fabric, we employ an implementation of the so-called zFilter-based host-identity independent forwarding layer [4].

Our primary evaluation topology follows Fig. 1. Different possible segment sources are identified with id's of #1 to #9, #9 being the original source, each able to cache and re-play different proportion of the passing data. Channel has the base RTT of 1500 ms with no queuing. FCT is ~ 5200 s when only one source is available . As an example, Fig. 2a illustrates the variations of different $CCWND_i$s at one of the random receivers. The FCT of the flow is shown on the *X* axis of this plot. Comparing with Fig. 2c, the plot easily mirrors the dependency of $CCWND_i$s to the amount of data served from each sources. Fig. 2b displays the variations of different RTT_i estimates at one receiver. It can be seen that although different segments are retrieved quite randomly from different parts of the channel, ConTug is successful in keeping the overall RTT, FCT, and congestion sufficiently low. ConTug manages to receive the data in a more efficient manner compared to TCP, by retrieving some parts of the content from closer nodes, this can be seen in Fig. 2c .

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. ACM CoNEXT*, 2009, pp. 1–12.

[2] D. Trossen (ed.), "Architecture Definition, Component Descriptions, and Requirements," PSIRP Project, Deliverable D2.3, 2009.

[3] S. Arianfar, P. Nikander, and J. Ott, "Packet-level caching for information-centric networking," Finnish ICT-SHOK Future Internet Project, Tech. Rep., 2010. [Online]. Available: http://users.piuha.net/blackhawk/contug/cache.pdf

[4] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking," in *Proc. ACM SIGCOMM*, 2009, pp. 195–206.