Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 6, 2012

P. Sarolahti
J. Ott
Aalto University
C. Perkins
University of Glasgow
March 5, 2012

# TCP Segment Caching
draft-sarolahti-irtf-catcp-00.txt

## Abstract

This document describes Content- and Cache-Aware TCP (CATCP) that allows caching of TCP segments to be re-used between different connections transmitting same data. When there is redundant data to multiple receivers, this can lead to significant load reductions and performance improvements.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2012.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction . . . . . 3
2. Terminology . . . . . 3
3. Protocol Overview . . . . . 4
4. New TCP Options . . . . . 5
4.1. CA-TCP Enabled Option . . . . . 5
4.2. Content Label Option . . . . . 6
4.3. Content Request Option . . . . . 7
5. Operation . . . . . 8
5.1. Sender Behavior . . . . . 8
5.2. Receiver Behavior . . . . . 11
5.3. Controller Behavior . . . . . 11
5.4. Segment Cache Behavior . . . . . 13
6. Open Issues . . . . . 14
6.1. Correctness of Data . . . . . 14
6.2. Consistent Segmentation . . . . . 14
6.3. Interaction with middleboxes . . . . . 14
6.4. Other Issues . . . . . 15
7. Security Considerations . . . . . 15
8. Acknowledgments . . . . . 15
9. References . . . . . 15
9.1. Normative References . . . . . 15
9.2. Informative References . . . . . 16
Authors' Addresses . . . . . 16

## 1. Introduction

Many current Internet applications are content-oriented, where the main application primitive is to locate and fetch a named content resource. The most common example is the world-wide web, that uses URLs to identify a particular content resource. Other content-oriented applications are the various peer-to-peer file sharing systems, or the traditional FTP. To enhance the efficiency of content delivery, popular content is replicated to multiple servers, and cached by intermediate on-path caches. Usually these caches are application-specific, commonly focused on the web traffic.

This document describes a TCP extension called Content- and Cache-Aware TCP (CA-TCP), that identifies the content in TCP payload in a new TCP option [RFC0793]. This information can be used my new types of intermediate network caches, to enable generic TCP segment caching independent of the upper layer application protocol. These network caches can transmit the cached TCP segments on behalf of the original TCP sender to any number of receivers. All acknowledgments flow to the original sender, and it can keep track of the progress of transmission. The extension requires modifications only at the TCP sender, and works with any normal TCP receiver. If the network path contains intermediate network caches that support CA-TCP, the communication performance of transmitting same data to multiple receivers can be significantly improved. If there are no such caches on the path, the communication behavior is identical to normal TCP.

Interoperating with the network caches, CA-TCP can reduce the load at the TCP senders, and reduce the overall congestion in the network. Through short-term caching between multiple simultaneous receivers of the same data, CA-TCP can also be used to enable a form of "pseudo-multicast": a cache can replicate the data sent by the TCP sender to multiple receivers. This can be useful with, for example, with TCP-based live video streaming.

There are also open issues to be solved with CA-TCP. A verification mechanism is needed for the content that can be cached, to protect against false injected TCP segments at the caches. The cached content needs to be consistently segmented among different receivers to make use of the cached data. These and some other issues are discussed in more detail in Section 6.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document is an Experimental specification, but uses the normative language as described above. In other words, implementation of this document is optional, but if a host implements CA-TCP, the normative instructions of this document MUST be followed.

In addition we use the following terms:

- Segment cache: A CATCP-aware cache that can store TCP segments based on the TCP options in the packets.

## 3. Protocol Overview

With CA-TCP the applications can identify data they are transmitting using a content label that is indicated to the TCP implementation by an API extension. This content label is transmitted with TCP segments using a new Content Label Option, that enables TCP segment-level caching at segment caches that support CA-TCP.

In addition to the TCP sender and receiver, the CA-TCP framework consists of a Controller, which is a middlebox that intercepts the data segments and acknowledgments and determines the next data expected to be transmitted, and segment caches that maintain packet level caches of labeled content. These caches are then used to share the cached content between TCP connections, with the segment caches intercepting CA-TCP acknowledgments, labeled using a new content request TCP option, and transmitting data from their cache in response. This helps reduce the load at the TCP sender, and in the upstream path. The receiver may be modified to add content request options to acknowledgments, or they may be added by a middlebox near the edge of the network known as the controlling node.

TCP acknowledgments are delivered to the original sender as normal, allowing it to keep track of the transmission progress and update its transmission window accordingly. When transmitting a segment on behalf of the sender, the segment cache updates the content request TCP option to inform the sender and upstream segment caches that it has sent data, preventing them from injecting the same data again.

A CA-TCP connection starts in the usual way, with an end-to-end three-way handshake. Once the connection is established, data transfer can proceed as normal, with unlabeled content, or the sender can add a content label to identify the payload as being a particular content item.

For example, when fetching an HTTP resource, the client would initiate the connection and send the HTTP GET request as unlabeled content in the usual way. The server would accept the connection and

send the headers of the response as unlabeled content, then it may assign a content label for the response data, differentiating between variants (e.g., encodings) of the content as negotiated with the client. The content label can be changed in the middle of a connection, if the object under transmission changes. Data that should not be cached is not given a content label.

## 4. New TCP Options

Because CA-TCP is an experimental mechanism, the new options use the experimental TCP options according to the guidelines given in [I-D.ietf-tcpm-experimental-options]. Therefore no IANA allocation for new TCP option types is needed at this time.

### 4.1. CA-TCP Enabled Option

The CA-TCP Enabled TCP Option is shown in Figure 1. The option is sent in the beginning of connection, together with the SYN and SYN-ACK segments, to indicate that CA-TCP is supported.

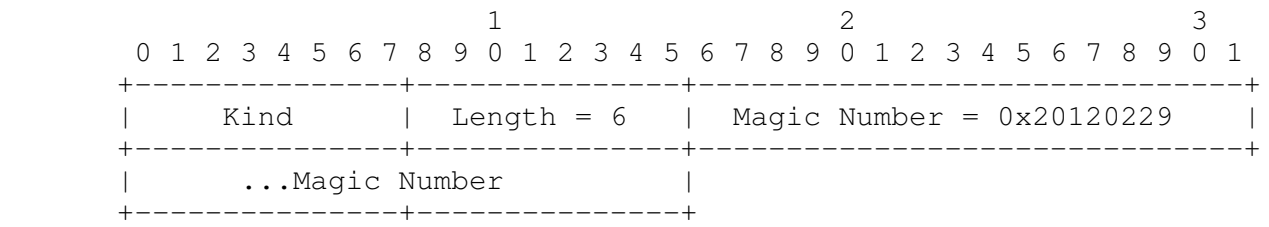


Figure 1: CA-TCP Enabled Option

The option fields are as follows:

- Kind: set to 253 in TCP SYN segment, and 254 in TCP SYN-ACK segment. These are the experimental TCP option codes, available without IANA allocation. Note that TCP receivers are not required to be aware of CA-TCP, but a CA-TCP segment cache SHOULD add this option to TCP SYN-ACK segment, if it was not there already.

- Length and Magic Number are set as indicated above. The use of Magic Number is described in [I-D.ietf-tcpm-experimental-options], and we use hexadecimal value 0x20120229 for CA-TCP.

## 4.2. Content Label Option

The Content Label Option (Figure 2) is attached to TCP segments containing data that can be cached by the segment caches. This option SHOULD NOT be used, if a TCP sender has not received a valid CA-TCP Enabled Option in a SYN-ACK, as this likely indicates that there are no segment caches on the path, and the Content Label Option would be useless.

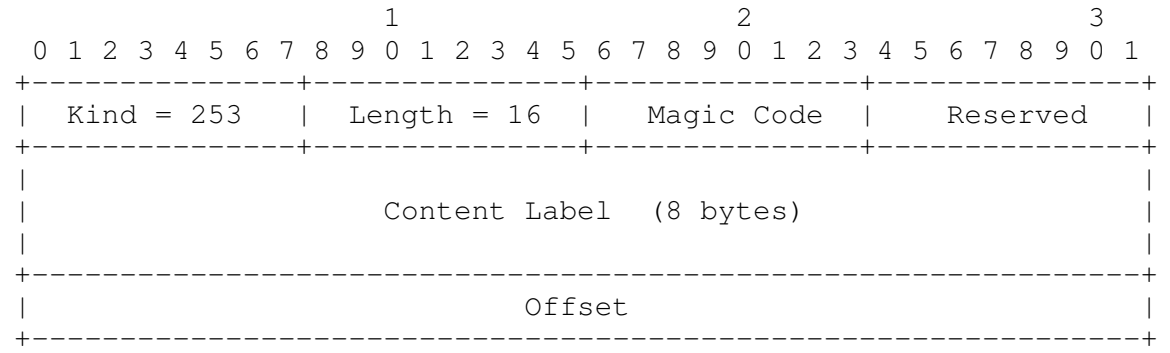


Figure 2: Content Label Option

The option fields are as follows:

- Kind and Length: as indicated in the picture.
- Magic Code: This is the last 8 bits of the Magic Number in the CA-TCP Enabled Option, i.e., 0x29. Using a full 32-bit Magic Number is not desirable for Content Label Option because of the size constraints in TCP Options. An 8-bit "magic code" allows simultaneous operation of multiple experimental options sharing the same option kind numbers. If the length and the magic code are not correct, the Content Label Option MUST NOT be processed by the CA-TCP segment caches.
- Reserved: SHOULD be set to 0 for now, reserved for future use.
- Content Label: Identifies the application layer content. Application chooses a content label that is unique by high probability, and communicates that to the TCP implementation through the API. This label is only used as a caching identifier (together with the content offset), and the TCP implementation is indifferent about the contents of this field. The application MUST NOT use the same content label if the application payload has changed from the earlier use of the content label.

- Offset: Indicates the relative distance of the TCP payload to the start of the content object in bytes. When a TCP sender starts to send cachable data under a new content label, the offset is initialized to 0 (unless the sender starts transmission from the middle of content). For subsequent segments the offset value is increased by the same amount as the TCP sequence number is increased. This information is used to identify the cached segments, in addition to the content label. Relative offsets are needed, because TCP connections start at random sequence numbers that cannot be used for caching.

## 4.3. Content Request Option

The Content Request Option (Figure 3) is carried together with TCP acknowledgments. A CA-TCP segment cache uses it to indicate the highest sequence number sent, and to indicate whether it has sent data in response to the acknowledgment, to prevent upstream nodes sending excess data in response to single acknowledgments.

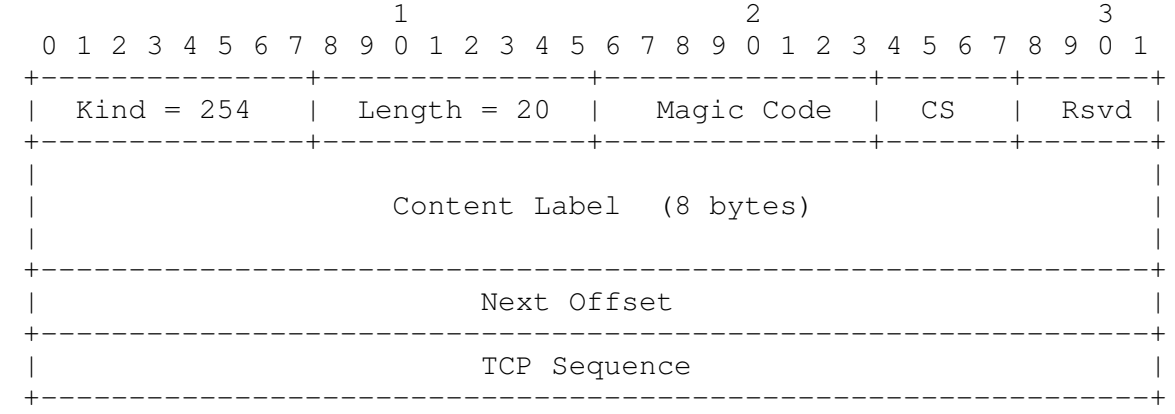


Figure 3: Content Request Option

The option fields are as follows:

- Kind and Length: as indicated in the picture.

- Magic Code: This is the last 8 bits of the Magic Number in the CA-TCP Enabled Option, i.e., 0x29. If the length and the magic code are not correct, a segment cache MUST NOT process this option.

- CS (CanSend): Number of TCP segments that can be sent in response to this TCP acknowledgment. With this field a CA-TCP segment cache can control the number of data segments transmitted in response to the acknowledgment. Even though the field length

allows larger values, the value of CS SHOULD NOT be larger than 2, i.e., at most two outgoing segments are allowed for an incoming acknowledgment. The management of CS field is described in more detail in Section Section 5.3.1.

- Rsvd (Reserved): SHOULD be set to 0 for now, reserved for future use.

- Content Label: Identifies the content expected to be transit for this connection. When receiving this acknowledgment, a CA-TCP segment cache uses this field, together with the 'Next Offset' field to check if it has data in its cache it can transmit in response to the acknowledgment.

- Next Offset: Indicates, relative to the beginning of the content, what data is expected to be transmitted next. A segment cache or TCP sender uses this field, together with the 'Content Label' field to determine what data is to be transmitted next. When a segment cache transmits data, it increases the Next Offset field before forwarding the TCP acknowledgment, to prevent multiple transmissions of the same data.

- TCP Sequence: Tells the next TCP sequence number that should be used by a segment cache for sending a data segment in response to this acknowledgment. This information is needed by a segment cache to build a valid TCP header for outgoing data segment. Segment caches cannot be assumed to maintain a full TCP flow state of ongoing connections. The other header fields can be constructed based on the acknowledgment, as outlined in Section Section 5.4.

# 5. Operation

In the following we specify the operation of CA-TCP Sender, CA-TCP Controller that intercepts the normal TCP acknowledgments, and Segment Cache, that stores the TCP segments that arrive with a Content Label option. The TCP receiver follows normal TCP operation.

## 5.1. Sender Behavior

A TCP sender starts connection with normal TCP SYN handshake. If the sender is going to use CA-TCP during the connection, it MUST add CA-TCP Enabled Option (with Kind number 253) to the SYN segment. If a TCP implementation with CA-TCP support does not know at connection establishment time whether it is going to use CA-TCP, a safe action is to add the CA-TCP Enabled option to all new connections. However, a TCP sender MAY leave the option out, in which case it MUST NOT use

CA-TCP during the remainder of the connection.

The TCP sender checks if incoming SYN-ACK segment carries a CA-TCP Enabled option (with Kind number 254). If this option is not included, the TCP sender MUST NOT use CA-TCP during the remainder of the connection.

#### 5.1.1. Outgoing Data

Any time during the connection a TCP sender may or may not attach the content label to the outgoing data segments. A TCP sender can also change the content label during the connection, if the content item under transmission changes. For example, in a persistent HTTP connection, the HTTP headers would be sent without a content label, and a static, cachable HTTP body would be transmitted with the content label option., and for multiple HTTP responses in the same connection, the different body elements would use different content label.

The content label is chosen by the sending application, and the TCP implementation is indifferent about its content. The receiving TCP implementation will ignore the Content Label option, and receiving application will not therefore know about its existence. The content label should be chosen such that the likelihood of a label collision at segment caches is as small as possible, for example using a digest composed from the content itself.

When application sets new content label, the first byte of the indicated content MUST start a new TCP segment. The TCP sender adds a Content Label Option to this segment and subsequent segments until the transmission of the content is complete. The value of Offset field is set to 0 in the first segment, and the subsequent segments set this value to the distance of the first byte of payload to the start of data. The sender also stores the the TCP sequence number of the first byte of content for further processing in an internal variable (SND.Content-Start). This field can be seen as a content-internal sequence numbering that is independent of the randomly initialized value of the TCP sequence number.

If the labeled content changes in any way, for example, if any single bit changes, or the length of the content changes, the application MUST use a different content label than earlier.

TCP sender SHOULD try to maintain consistent segmentation for payload with content label, to improve the efficiency of the caching of the content. This is may not always be possible, but failing to do so is not fatal. Inconsistent segment boundaries just result in not being able to use the possibly earlier cached data.

### 5.1.2. Processing Acknowledgments

If an incoming acknowledgment arrives without the Content Request Option, it is processed normally.

If an acknowledgment contains the Content Request Option, the sender updates its local state in the following way: if the sum of 'Next Offset' field and SND.Content-Start points to sequence that is later than the value of SND.NXT in TCP sender's local state, the value of SND.NXT is set to the sum of 'Next Offset' and SND.Content-Start. This avoids resending data that has already been sent by a segment cache in the network.

It is possible that with a fast segment cache SND.NXT grows faster than the sending TCP application can write to the socket send buffer. This is considered a feature of CA-TCP. The applications are expected to explicitly enable CA-TCP, and to use content labels consistently for the exactly same data. If this principle is followed, this behavior does not cause problems.

SND.UNA in TCP's local state is managed normally, based on the incoming TCP acknowledgments. TCP's retransmission algorithms operate normally, based on incoming duplicate acknowledgments and retransmission timer. Retransmitted segments MAY contain the Content Label Option, if the content in question was assigned with a label.

TCP congestion control operates according to the normal rules [RFC5681]. However, when an incoming acknowledgment arrives, the TCP sender consults the 'CanSend (CS)' field, before increasing the congestion window. The sender's congestion window MUST NOT be increased more than the value of CS, multiplied by the SND.MSS. When data is delivered from segment cache, it is common that the CS value in incoming acknowledgment is 0. In this case the TCP sender does not increase the congestion window at all (even by a fraction, if in congestion avoidance). Note that this makes the TCP sender behavior more conservative than in the normal case.

When data is delivered from the segment caches, it is possible that TCP sender receives acknowledgments for data it has not yet sent. Such data SHOULD be discarded from the socket send buffer without transmitting it. Sampling the round-trip time and updating the RTO estimate is impossible in such cases. Otherwise the RTO estimate is maintained in a normal way [RFC6298]. The RTO timer SHOULD be reset based on the currently estimated value on each new acknowledgment that advances the window, to avoid spurious retransmission timeouts on long periods of cached data.

### 5.2. Receiver Behavior

TCP Receiver operates in normal way, and does not need to be aware of CA-TCP, or be able to parse the options. The receiver ignores the options with their content, and delivers the data to the application normally. A receiver can operate as a CA-TCP Controller, adding the Content Request options to outgoing acknowledgments.

### 5.3. Controller Behavior

CA-TCP Controller is a node that processes normal TCP acknowledgments and adds Content Request option to them, if needed. The Controller can be co-located at a segment cache, or a TCP receiver can act as a controller. Because segment caches operate based on the Content Request options, an ideal location for the controller is close to the receiver, because the segment caches downstream from controller cannot be used for caching. In order to process the acknowledgments appropriately, the controller needs to maintain some per-flow state, as described below.

Controller maintains some flow-specific data for each flow with potentially cachable data, in a structure we call "flow table". The flow table contains the following data for each active flow.

- Source and Destination IP address and port -- for identifying a flow.
- Content.Label -- Content label currently in transit in a flow. This can be 0, if no labeled content is currently in transit.
- Content.Offset -- The TCP sequence number at the start of the currently transmitted content object.
- Content.Next -- The next untransmitted content byte in the flow, relative to the beginning of the content.
- Congestion control state -- see Section 5.3.1 for discussion on congestion control at the Controller.

When a TCP SYN segment with CA-TCP Enabled option arrives at the controller, it creates a new entry in the flow table using the source and destination IP address and TCP port, and initializes the other parameters for the flow. Storing the flow table entry is optional: controller may choose to ignore the option, for example because of resource constraints or a policy. In this case the TCP connection continues to operate normally, without segment caching.

When a TCP SYN-ACK segment arrives at the controller (or is sent by a

receiver that also acts as a controller), and the controller has the corresponding flow state for the flow, it SHOULD add CA-TCP Enabled option to the segment before passing it forward, with option kind set to 254, as described in Section 4.1.

When a TCP data segment with Content Label option arrives at the controller, and it has the corresponding flow state initialized, the controller stores the currently transmitted content label to the appropriate flow table entry if it is different from earlier stored content label. The controller also stores (or verifies) the TCP sequence number at the start of the content to the flow table ('Content.Offset'). 'Content.Offset' is calculated by subtracting the Offset field at the Content Label option ('Option.Offset') from the TCP sequence number in the current segment. This information is needed to build correct Content Request options for TCP acknowledgments for the same TCP flow. If Option.Offset + TCP.Length (the length of TCP segment payload) is higher than 'Content.Next' in the local flow table, the controller sets Content.Next to Option.Offset + TCP.Length. Note that for a correctly working connection, the difference between offset and TCP sequence number should not change during the transmission of the content. However, it is possible that TCP sender starts transmission of content from a different offset than 0.

When a TCP data segment without Content Label arrives, and the flow is found in the flow table, the controller erases the current label from the table, if one was stored. If labeled content was previously in transit, but a segment without content label option is received, that tells the transmission of labeled content is finished (for example, transmission of HTTP body is over, and data for new HTTP request arrives).

When a TCP acknowledgment arrives that does NOT yet have a Content Request option, but corresponds to a flow that is stored to flow table, the controlling node reviews from the flow table if there currently is labeled content in transmission. In positive case, it adds a Content Request Option to the segment containing the current content label. The controller also copies the 'Content.Next' value from the local flow state to the 'Next Offset' field of the TCP Option, and sets the 'TCP Sequence' field in the option as Content.Offset + Content.Next. Now the contents of the option refer to the next content offset and TCP sequence that can be transmitted either by the sender caches, or by the TCP sender. The Controller also sets the CS (CanSend) field using the congestion control algorithm described below. After this the Controller forwards the acknowledgment.

When a TCP acknowledgment arrives with Content Request option, the

Controller just ignores the segment and forwards the acknowledgment. In this case there is another controller on the downstream path that already manages the TCP flow.

#### 5.3.1. Congestion Control

The Controller MUST take care of congestion control for the flows that are in maintained at the flow table. A simplified congestion control algorithm is considered sufficient, because the original TCP sender is responsible of retransmitting data and ultimately maintains the normal sender-side congestion control algorithm.

For example, the following algorithm is sufficient.

- The controller maintains a congestion window for each flow that indicates how many segments are allowed to be in transit. The congestion window is initialized to 3.

- When an incoming acknowledgment advances the window, the congestion window is increased according to congestion avoidance algorithm.

- When three consecutive duplicate acknowledgments arrive at the controller, the congestion window is halved.

The controller then compares the current congestion window (cwnd) to the number of outstanding unacknowledged TCP segments for the flow (FlightSize), and sets the CanSend field in outgoing Content Request option to cwnd - FlightSize. If the difference is more than 2, the CanSend field SHOULD NOT have larger value than 2, to limit bursts.

The controller does not manage retransmission timer or take RTT samples. If a retransmission timeout is required, the TCP sender takes care of the needed retransmissions.

### 5.4. Segment Cache Behavior

The segment cache intercepts arriving TCP segments with Content Request option, and transmits segment(s) from cache, if the requested segments can be found, based on content label and Next Offset fields in the option.

If a segment cache has a segment in storage that starts from the sequence number indicated by the Next Offset field AND that has the same content label, AND if the CS field is larger than 0, the segment cache can transmit the cached segment towards the receiver that sent the acknowledgment. The TCP header is built based on the incoming acknowledgment, and the TCP sequence number is copied from the

Content Request option. The ACK flag is not set in the segment, and the advertised window is set to a small value (exact value TBD), because the segment cache does not know the state of the buffer at the TCP end host.

After this the CS value in Content Request option is decremented by one, and the Next Offset and the TCP sequence are incremented by the length of the cached data segment payload. If the CS value is still larger than 0, the segment cache can send another segment based on the same procedure, if the segment is in cache, again decrementing the option fields appropriately.

After the segment cache is done with processing the Content Request option, it forwards the segment towards TCP sender.

## 6. Open Issues

### 6.1. Correctness of Data

Using content labels will give a malicious data source a tool to inject data under a false content label unless some measures are taken to verify the validity of the content at the receiver. Many applications have such verification mechanisms built in, but at the moment there are no valid common transport layer mechanism to check the correctness of labeled content. A sending application must also use the content labels consistently: if any part of the content is changed, the label must be changed.

### 6.2. Consistent Segmentation

The potential benefit of caching depends on having consistent segmentation of data. If the segment boundaries vary between connections, the efficiency and cache hit rate suffers.

### 6.3. Interaction with middleboxes

Network address translation is not expected to affect the behavior of CA-TCP, and its behavior is tested with some NAT implementations. However, some middleboxes may perform in-window checks that filter out acknowledgments that appear to arrive out of window, as may easily happen with CA-TCP, as acknowledgments may arrive for data that was not sent by the original sender. Some middleboxes may alter segment boundaries, leading to similar problems as discussed above for consistent segmentation.

### 6.4. Other Issues

The control loop between a cache and receiver may be significantly faster than the loop between the TCP end hosts. Sender can interrupt the transmission of data at any time by sending a segment without a content label. However, during the time the segment is in transit, a cache may have transmitted new data if there are cache hits. Currently we believe this is not a critical problem for labeled content.

CA-TCP requires a good portion of TCP option space for labeled data. There are also other enhancements that are hungry for TCP option space, such as Multipath TCP [I-D.ietf-mptcp-multiaddressed], and the option space limitation of 40 bytes prevents the different enhancements to be used together. One solution to this problem would be to come up with enhancements to extend the available option space, such as discussed in [I-D.eddy-tcp-loo].

If the communication path changes during a TCP connection, the traffic may bypass the original controller. This is not a fatal problem, but just causes the rest of the TCP connection be transmitted between the sender and the receiver.

## 7. Security Considerations

The main security issue is related to the correctness of data, as discussed in Section 6.1. Good solutions for this problem are currently being investigated.

## 8. Acknowledgments

This work has been partially funded by the PURSUIT EU project.

## 9. References

### 9.1. Normative References

[I-D.ietf-tcpm-experimental-options]
Touch, J., "Shared Use of Experimental TCP Options", draft-ietf-tcpm-experimental-options-00 (work in progress), January 2012.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

## 9.2. Informative References

[I-D.eddy-tcp-loo] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[I-D.ietf-mptcp-multiaddressed] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-multiaddressed-06 (work in progress), January 2012.

## Authors' Addresses

Pasi Sarolahti
Aalto University
Department of Communications and Networking
P.O. Box 13000
FI-00076 Aalto
Finland

Email: pasi.sarolahti@iki.fi

Joerg Ott
Aalto University
Department of Communications and Networking
P.O. Box 13000
FI-00076 Aalto
Finland

Email: jo@netlab.hut.fi

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
United Kingdom

Email: csp@csperkins.org