PURSUIT
Publish Subscribe Internet Technology
FP7-INFSO-ICT-257217

TR 12-0002

ConTug: A Receiver-Driven Transport Protocol for Content-Centric Networks

Document Properties:

Title of Contract	Publish Subscribe Internet Technology
Acronym	PURSUIT
Contract Number	FP7-INFSO-ICT 257217
Start date of the project	1.9.2010
Duration	30 months, until 28.2.2013
Document Title:	ConTug: A Receiver-Driven Transport Protocol for Content-Centric Networks
Date of preparation	2011-04-06
Author(s)	Somaya Arianfar (AALTO-HIIT), Pekka Nikander, Lars Eggert, Jörg Ott, and Walter Wong
Responsible of the deliverable	Somaya Arianfar Email: somaya.arianfar@aalto.fi
Target Dissemination Level:	PU
Status of the Document:	Completed
Version	1.0
Document location	http://www.fp7-pursuit.eu/
Project web site	http://www.fp7-pursuit.eu/

ConTug: A Receiver-Driven Transport Protocol for Content-Centric Networks

Somaya Arianfar, Pekka Nikander, Lars Eggert, , Jörg Ott, and Walter Wong

April 2011

Contents

Abstract

This report presents ConTug, a new, receiver-driven transport protocol for content-centric networks. It discusses the specific constraints of content-centric networks and describes the design of ConTug, highlighting a number of general challenges encountered while designing a transport protocol for such networks. It also presents ns-3 simulations that study how ConTug behaves in a number of simple content-centric networking scenarios, focusing on comparing the performance of ConTug with that of standard TCP and theoretical limits in similar scenarios. These early results show that even a simple design like ConTug is able to achieve an acceptable level of efficiency and provide sufficient congestion control.

Chapter 1

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. Under the original "smart edges, dumb core" paradigm of the Internet, hosts perform all higher-layer functions, such as congestion control, guaranteeing reliable delivery, establishing some degree of fairness between concurrent transmissions, and so on. The network is limited to performing those functions that facilitate the end-to-end delivery of host-generated packets, such as discovering topology and computing routes.

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. It also needs to determine which transport and application layer protocols to use in order to access the content. "Accessing content" typically means to retrieve a copy of the content for local use or storage.

One consequence of this architecture is that IP packets are only labeled with the information that is required for their end-to-end delivery by the network—namely, source and especially destination IP addresses. The actual content that is transmitted in a packet or a sequence of packets, *i.e.*, a flow, is supposed to be opaque to the network.

Although this separation of roles is appealing in principle, it has *de facto* ceased to be applied. Devices in the network routinely look past the IP header in order to determine which type of content is being transmitted in a flow, *e.g.*, by inspecting the transport protocol header. In other cases, devices in the network attempt to identify specific content items by interpreting the payload of a flow and, in some cases, even modifying payloads on the fly. Although these practices appear to violate the original architecture of the Internet, valid operational, security, and other concerns have caused them to become ubiquitous.

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 piece of content from the network and let the network to worry about where to retrieve it from. Therefore, the network is more independent and, at the same time, has more control over traffic management, as content can now be cached and replicated as needed. Additionally, hosts experience a faster and more resilient service, as the availability of each piece of content is decoupled from the reachability of specific host and as the network can typically make better decisions of where to transfer content from than most hosts could.

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

This paper begins exploring the transport aspects of such an architecture, *i.e.*, considering the case of a network where hosts request chunks of content from the network, and the network delivers those chunks from anywhere they happen to be stored, *e.g.*, from distributed, cached replicas.

In particular, this paper describes the design of ConTug, a receiver-driven transport protocol that can reliably and efficiently retrieve large pieces of content from the network. ConTug benefits from opportunistic caching in the network, pulling data chunks from any number of cached copies in a way that is congestion-controlled and to some degree fair.

Chapter 2

Background and Related Work

As mentioned before, in the Internet, hosts generate and consume packets, whereas the network itself is in charge of delivering those packets. Hosts are responsible for implementing end-to-end transport functions, such as for congestion control, reliable delivery, or maintaining fairness. The vast majority of the bytes on the Internet today are transmitted using TCP [3].

Because content is not a first-order element of the Internet architecture, there is no uniform way to determine which hosts hold copies of items of interest. Consequently, each application needs to implement its own scheme. A uniform resource identifier (URI) [4], for example, requires a lookup – a level of indirection that can be used to influence to which peer hosts a TCP connection will be opened.

Content-centric networking is fundamentally different from Internet-style networking; a fact that substantially affects the design of content-centric transport protocols. Because pieces of content are uniquely identifiable, hosts no longer need to identify the topological location(s) where a piece of content is stored. Instead, a much simpler style of request-response networking becomes possible. Hosts issue requests for data without needing to specify the intended source, and the network routes those requests to potential sources of the content, any number of which may then respond. Content networking architectures that aim to be deployed on current hardware need to be designed with packet-based networking in mind. This means, for example, that pieces of content larger than the maximum supported network packet size need to be segmented. Once that has happened, it is the job of the transport protocol to request each of the segments from the network, and reassemble the segmented responses into the original piece of content.

Such transport protocols tend to become receiver-driven, because the network forwards the segment requests to potentially multiple data sources, and only the receiver knows when the entire transfer has completed. This approach promises substantial performance improvements, because requested segments can flow towards the receiver from well-connected nearby sources whenever possible. On the other hand, new mechanisms are needed to provide congestion control and establish fairness and reliability, because the traditional Internet approach of adapting the transmission rate to the probed end-to-end path characteristics is no longer feasible—there no longer is a single well-defined path to a single well-defined source.

Some proposed transport protocol designs for such networks exist. One such proposal is CCN [1]; a content-oriented communication model driven by data consumers' interests. In order to retrieve items of interest, consumers broadcast their interests into the network and any intermediate node owning any piece of the requested content replies. The CCN architecture does not support a transport layer in its traditional sense. Reliability is achieved as combinational work of application and forwarding modules. Application ultimately makes sure it has every piece of the content that it is interested in. The congestion control operation is done hop-by-hop, by controlling the number of interestests sent upward on each link/interface; when a router is overloaded by incoming data traffic from any specific neighbor, it slows down or stop sending Interest packets to that neighbor. Once congestion occurs, the application times out and retransmits the Interest.

While we do not argue against the removal of a seprate layer as transport in this architecture, the coice of hop-by-hop congestion control does not seem to achive a significant performance gain in different scenarios. It may not even be possible to use this kind of hop-by-hop control in cases where the data-forwarding is done without keeping any interest table in the routers [**?**]. Finally in general, since the ultimate interest rate is adjusted by the final consumer, the optimality of timeout calculations and bandwidth utilization remains as a big question in CCN [1].

In more traditinal content routing proposals such as TRIAD [5], the normal TCP connection is opened between the source and the receiver(or downstream caching point) after the high-level object source is identified by the infrastructure. While it is a viable approach we are looking at a more optimized way, where we can make a network's resources including links, caches and servers behave like a single pooled resource. In same line as other recent proposals like Multipath TCP [6], our aim is to build mechanisms for shifting load between the various paths and sources of the network in short periods of time.

Content orientation is also found in message-based networks such as delay-tolerant networks (DTNs), whose support for arbitrary message sizes allows a piece of content to be carried in a single message. This blends a part of the transport functionality with the network layer. For DTNs, there are descriptions for content-centric retrieval paradigms [7] and opportunistic caching schemes [8].

A few Internet transport protocols and mechanisms also have similarities to transports aimed at content-centric networks. WebTP [9] and PROMPT [10] propose receiver-driven transport protocols. More recent proposals aimed at request/response communication include RCP [11].

BitTorrent [12] is a file-sharing application that uses multiple TCP connections to transmit chunks from different peers at the same time. BitTorrent can be seen as an application-layer content-oriented network where users "request" content by joining a swarm, which lets them obtain metadata about content parameters, such as piece length and piece identifier list, via a tracker or peer-to-peer. BitTorrent uses simultaneous TCP connections as well as a "background" UDP-based transport protocol similar to LEDBAT [13]. All these Internet based proposals count on deterministic sources, which are not reliable components in the content-centric environment.

Chapter 3

Conceptual Model

As mentioned above, content-centric networks operate in a very different fashion than host-oriented networks such as the Internet. The consequence is that none of the standard Internet transport protocol designs match the requirements of content-centric networks. This section defines some terminology and discusses the basic operation of a transport protocol for content-centric networks, before discussing the emerging properties and constraints of such a transport design.

3.1 Basic Operation

In content-centric networks, a piece of *content*, also referred to as a piece of *data* or a *file*, is a sequence of bytes identified by a globally unique and persistent *identifier* (ID), *e.g.*, an application document stored on disk, a live video transmission, etc. Each content item has an *original source* that announces the availability of the content item to the network, and that is the authority that has change control over the item and its *meta-data*.

Packet networks can only forward packets up to a maximum packet size. Pieces of content larger than this limit require segmentation before they can be transmitted. A common design approach – as this paper assumes – is that each resulting *segment* of a large content item is a uniquely identifiable piece of content in its own right. E.g. anything larger than 1280 bytes (defult minimum MTU size of IPV6), needs to be identified.

To retrieve a piece of content, a host – the *requester* or *receiver* – issues a *request* for it to the network. The request contains a *requester identifier* – a capability-like opaque tag that may be used to send replies back to the requester. The network is responsible for forwarding requests towards the original source that announced the content item.

Intermediate nodes in the network between requesters and sources are free to cache content items that are being transmitted across them as they see fit. They may cache all segments belonging to a content item, or only a fraction of them. When such a *cache* sees a request for a content item that is 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 sources, but not the original sources.

When a host requests a content item that is shorter than the maximum packet size of the network, the response will contain the requested data. When it is longer, the response will contain meta-data about the requested item, including the segment identifiers that make up the requested content, content length, and security and integrity information.

To retrieve such larger content items that consist of multiple (potentially many) segments, a receiver-side mechanism is needed to issue requests for each of the segments to the network. This mechanism needs to avoid overloading the network, it needs to avoid overloading the requester itself, and it needs to verify that the entire content item has been correctly received before passing it on to the application. In other words, it needs to implement congestion control, flow control, reliability and integrity protection.

3.2 Emerging Properties

From the description above, it becomes clear that requested segments will often flow to a receiver from several sources at the same time. The receiver usually does not know any of the sources *a priori*; a receiver will usually learn about a new source when it receives the first response from it. This is very different from TCP or other Internet transport protocols, where a receiver retrieves the content item from pre-known sources.

Here the argument can go in different directions.One way of controlling the congestion in hop-by-hop basis. the functionality of each hop though may differ. Some proposals like CCN suggest that the functionality of each hop can be limited to slow-down forwarding the interests towards the node that is overloading it. Some other proposals like the ones assumed in DONA, assume similar to web-proxies the full functionality of Internt stack is available in each caching point, therefor caching points can form end-points of TCP connection between each other to transfer parts of the data which are avilable to them. In this scenraio one can consider network caching points as bit-torrent clients which are orgenized hierarchically in the best possible way at the edges of each ISP network. It will mean th eoperation can be done again by opening different connections between receiver and each cache. because if it happens between the caches it is expected to add to the latency by too many send and and receives in the middle points in the network.But this will not help to shift the load during a connection. In addition it also creates privacy concerns about tracking pieces of content that each customer has been interested in. If it happens between the caches, each cache should synchronize between the arived requests and received data. So each data should go to the application level once and then comes back. Opening direct flows somehow geoperdizes the whole point of having pub/sub based content centric networks. Though it claims that the role of source location and IP address has reduced so it is ok with content-centric logic.

Content-centric networking comes with potential for a wide range of benefits such as content caching to reduce congestion and improve delivery speed, simpler configuration of network devices, and building security into the network at the data level. Reduces congestion and latency - doesn't send irrelevant or redundant information through network pipelines. CCN transport is for dynamic links, ours is for dynamic storage and servers loads. Our method is as good as the case when one opens differnet connections, but it acts even better when the load over the connections needs to be shifted.

From the requester 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.

Figure 3.1 illustrates this operation in simple topology with requesters on the left, original sources on the right, and potential caches along the path. For simplicity, the requests from a requester towards the original source will always follow a (single, linear) path through the network, because intermediate nodes will never forward requests away from the source. In other words, all potential responding sources for a content item are on the request path for a given segment.

This poses interesting design challenges if a receiver-driven transport protocol attempts to perform some transmission control per known source, because 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. TCP has a much easier problem to solve, because it only deals with a single control loop to a single source, instead of needing to drive multiple control loops without being able to influence which source will get to respond to a newly issued segment request. CCN [1] addresses this problem by restricting all transmissions to only occur between adjacent nodes and introducing state in the intermediate nodes. This paper attempts to avoid the need for keeping such states in the forwarding nodes, but consequently needs to deal with multiple interacting control loops.

The fact that a receiver pulls content from the network as a whole (*i.e.*, from all potential sources), rather than instructing a single individual source

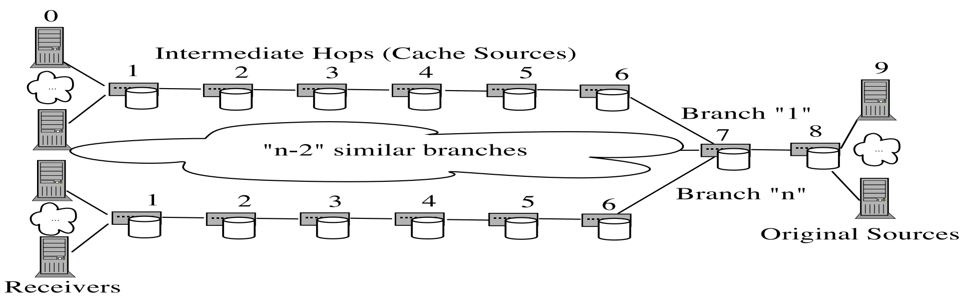

Figure 3.1: Example topology with requesters on the left, original sources on the right, and potential caches along the path.

to "send this data to me" as in the Internet, causes the resulting transport protocol to be necessarily receiver-driven. Sources only see requests for individual segments, and they never know if they will see another request from the same requester again, making sender-side control impossible. Consequently, transmission state and control need to reside at the requester. Other nodes in the network, including the original source, need not keep state related to specific transmissions, although they may, *e.g.* for optimization purposes.

Content networks usually attempt to minimize redundant transmissions of the same content over a network link by maintaining downstream cached replicas, in order to serve repeated requests (including retransmissions [14, 15]). This is one of the reasons why content networks can offer improved performance compared to Internet-style communications for popular content items.

Chapter 4

ConTug Transport Protocol Design

This section describes the details of the ConTug protocol design. Although ConTug targets operation over the PSIRP architecture [2], we believe that the overall design is suitable for adoption to other content networks.

4.1 Environmental Details

4.1.1 Forwarding Channels

While our design does not assume host names, some forwarding-level identifiers are needed to tell the network where to send packets to. For this purpose, we introduce the notion of *forwarding channels*, or *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. A *request channel* is used to send requests towards the original source (and/or possibly towards other "good" sources). A *reply channel* is used to send segments (or other replies) back to a requester.

To identify *reply channels*, a request may collect a reverse forwarding identifier as it passes from node to node, similar to reactive routing protocols or routing capabilities [16, 17].

The problem of identifying a specific *request channels, i.e.*, as a forwarding and routing problem, falls mostly beyond the scope of our present work. Potential solutions can borrow from a number of approaches, including [18]. For the purpose of this paper, we assume that the receiver gets the needed request channel identifier along with the meta-data; *e.g.*, the identifier may be collected into the meta-data reply.

In the simplest case (as shown in figure 3.1), all request channels for a receiver and the reply channels for the segments of a given content item follow a linear path between the receiver and the item's original source. Channels for different (topologically close) receivers will overlap. Note that channels may be time-varying even for the same receiver as sources appear and disappear due to caching or content relocation operations of the network.

Mapping the channel concept to an IP network would lead to using *anycast* addresses to specify channels for requests, and unicast addresses for replies carrying data. If needed, hop-by-hop options could be used to facilitate processing at the routers. Alternatively, in-packet Bloom Filters, as used in e.g. [19] or [20], may be used as forwarding identifiers.

4.1.2 Segment Format and Identifiers

We use a segment identifier (*segued*) to identify each segment, based on the contents or, more specifically, the data the segment carries. Segment identifiers allow us to address a given piece of data regardless of its location or the protocol used to retrieve it. Based on the type of the file, segIds can be generated differently. For application-level kind of documents, a hash over the segment content can be used as the segId. Alternatively, an algorithm can be used to generate segment Ids on-the-fly for dynamic content, such as real-time applications [21].

Fig. 4.1 shows the conceptual format of ConTug segments. In addition

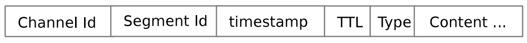

Figure 4.1: The conceptual segment format

to the channel and segment ids, the basic ConTug operation requires a timestamp, a hop-count (TTL), (both explained below), and a type field, to distinguish between requests and responses.

4.1.3 Partial Segment Sources: Short-Term Caching

As described above, ConTug is designed to retrieve content from any number of sources inside the network and thus to take advantage of content cached on routers. While this also helps efficiently repairing packet losses, caching is especially beneficial when serving the same content to many receivers (within a short period of time). According to [22], a few seconds of caching can lead to major savings in network capacity usage (up to some 50%). Therefore, in addition to forwarding segments, forwarding nodes may also cache them: segments are received and enqueued for forwarding as usual, but after forwarding, they are kept in memory with a certain probability (*e.g.*, 30%). If marked for caching, a segment is not dropped until replaced by another one to be cached ,following some cache replacement algorithm.

As a result, any segment passing along a path has some probability of getting cached at one or more nodes, at least for a few seconds. A network node may use such cached copies to satisfy requests, thereby becoming an unpredictable partial source for the requester. [23]

Note that caching segments does not lead to the versioning problems that affect, e.g., web caches. Modifying the content of a segment will, by construction, result in a different segment Id, since the identifier is generated from the very contents. The new version of a segment would thus be accessed via a different segment Id, the new Id being available in the new version of the meta-data file. In the rest of the paper a *source* can mean both a *partial segment sources* formed randomly in the forwarding nodes during the previous transfers or an *original source.*

4.1.4 Pull Based Transport

The ConTug transport is a completely data-oriented pull-based mechanism, where the receiver controls the segment reception based on its local parameters. In our model, the receiver is able to request each of the segments separately. Also, the receiver is able to send multiple concurrent requests into the network before receiving any replies from the network, in order to enhance its download rate.

4.1.5 Initialization

The content retrieval procedure starts with an initialization phase. In this phase, the content meta-data is retrieved from a known location in the network; the source of meta-data may or may not be trusted. In the case that piece of meta-data itself is so large that it does not fit into a single segment, the top-most meta-data segment indicates so, and a recursive procedure is used to first retrieve the rest of the meta-data using ConTug. The meta-data is used to prime the segment Ids for the ConTug transport, without assigning them to a location.

4.1.6 Request/Response Mode

In ConTug's model, receivers retrieve segments through a request/response style of interaction with the network. The requests go through a request channel, which we assume having been provided to the receiver along with the meta-data. The receiver does not need to know the end-point(s) of the channel. This in an important differentiation point from other transport protocols, such as MPTCP [24] or LEDBAT [13].

4.2 Congestion Control and Reliability

Resource unavailability in a ConTug's environment can be caused by different reasons. Any indicator of the resource unavailability can be considered as a sign of congestion, e.g. bandwidth shortage or unavailability of certain range of the segments on that source. All of these causes, are treated the same in ConTug and addressed as *congestion*.

Even though ConTug is flexible enough to support different algorithms for congestion control, we opted to implemented the initial algorithm as similar to TCP as possible. This design choice was made to provide a baseline implementation, allowing a fair comparison between ConTug and TCP, if required. Nevertheless, the protocol is not limited to TCP-like rate control algorithms and we are planning to study alternative approaches.

We now describe the details of ConTug's congestion control and reliability algorithms. The design choices were made based on our implementation and evaluation work, described below.

4.2.1 The Conceptual CWND (CCWND)

In order to control the number of outstanding requests and responses, ConTug uses the *Conceptual* congestion control window (CCWND) 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 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 creates a distinct $CCWND_i$ for each identified segment source i. Their sum yields the overall CCWND, see Eq. 4.1.

$$CCWND = \sum_i CCWND_i \tag{4.1}$$

4.2.2 CCWND Increase

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 $CCWND_i$, with the initial value 2. 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 to be sent to the channel.

4.2.3 RTT Measurement

The RTT estimation uses the timestamps included in the request and response headers. The receiver stamps every segment request and the sources copy the same timestamp value from the request to the response header. In this way no synchronization between the receiver and the network nodes are required. Upon receiving a response segment, the receiver determines the $CCWND_i$ the segment belongs to and updates the corresponding RTT_i estimate using the familiar smooth RTT formula, used in TCP.

$$RTT_i = \alpha * RTT_i + (1 - \alpha) \times Sample_i \tag{4.2}$$

4.2.4 Weak Congestion Indication

Similar to TCP Vegas [25], ConTug uses the increased RTT estimation as a weak sign of congestion. The difference between the expected rate and actual rate is counted as an indication for the amount of resource unavailability in the network.

The expected rate at each time is calculated based on Eq. 4.3.

$$Expected\ Rate_i = \frac{CCWND_i \times Segment\ Size}{BASERTT_i} \tag{4.3}$$

The $BASERTT_i$ is the minimum RTT that receiver sees in the batch of samples coming from source i. The actual rate is then calculated based on Eq. 4.4.

$$Actual\ Rate_i = \frac{CCWND_i \times Segment\ Size}{RTT_i} \tag{4.4}$$

At the beginning if the difference between the expected and actual rate is more than one, the algorithm enters the congestion avoidance phase. In the congestion avoidance phase, if the difference is less than an α value the $CCWND_i$ will increase and if it is bigger than a β value then $CCWND_i$ will decrease, linearly.

4.2.5 Strong Congestion Indication

Strong signs of congestion for ConTug are timeouts. Timeouts in receiving segments from a previously identified source i in the channel Eq. 4.5, shows signs of the less availability toward that source. This occurs when no packet from the source i arrives during the timeout period. In this case the $CCWND_i$ reduces to half. The $CCWND_i$ size can go to less than 1, which means no segment will be requested as part of that $CCWND_i$ anymore.

$$Source\ Timeout = C \times RTT_i \tag{4.5}$$

However, channel timeouts Eq. 4.6, triggered by actual packet losses in the network is a stronger sign of congestion. It happens when a requested segment is not delivered during the timeout. It causes the overall CCWND to reduce to half.

$$Channel\ Timeout = C \times MAX_{i \in \{0,n\}} RTT_i \tag{4.6}$$

C in both formulas is a constant value, currently set to 2 in our experiments.

4.2.6 Loss recovery

Contrary to TCP, out-of-order delivery is part of the regular ConTug operation: as different segments may be retrieved from different sources along the path with different RTTs, they may naturally arrive out of the sequence in which they were requested. Hence, segment loss is only triggered by a timeout based on the RTT, as expressed in Eq. 4.6. The timeout mechanism uses timers for each segment request. Whenever a timer expires prior to the segment arrival, the receiver interprets the event as a segment loss. It then sends another request for the lost segment.

4.3 Security

ConTug's security model also follows the content-centric logic, placing the security mechanisms in the segments instead of the channel. In ConTug, segment integrity can be easily achieved by using content-dependent identifiers. When cryptographically generated hashes are used as part of the segIds. The receivers (and even caches) can check segment integrity by taking the cryptographic hash function over the segment and comparing the resulting value with the segId.

ConTug also provides confidentiality through segment encryption, e.g. by using the cryptographic hash over the segment as the encryption key. In this case, segIds are based on the encrypted content, i.e. once the segment is encrypted, a second hash is taken and used as the part of the segId. This mechanism aims to overcome the limitations of the usual key establishment procedures, where a common key must be established among a group of users. By using a key that is based on the content, we decouple content encryption from any other context besides the content itself. Later, the decryption keys may grouped in a separate segment that can be encrypted, e.g. once again with the same procedure, resulting in a master key. The master key may then be distributed using usual key distribution mechanisms.

ConTug relies on network capabilities [16] to protect against fake requests destined to a victim, thereby preventing denial-of-service attacks. With network capabilities the receivers issue capability tokens to the sources, allowing data to be send back through the network. In ConTug, as the requests flow through a channel, the protocol collects a capability that allows sources to send the data back through the reverse channel. One way to prevent reuse of valid channels identifiers by attackers is to limit the capability validity [17], requiring the new capabilities after some time interval.

Chapter 5

Evaluation Setup

For the following scenarios, we restrict ourselves to a simple channel between a receiver and a primarily-unknown original source in the network. Speaking in terms of IP, such a restricted channel represents a unicast path between two points in the network. It is important to note that in ConTug there can potentially be multiple time and space transient sources located on this unicast path.

5.1 Network setup

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 [20], as it inherently supports ConTug's channel concept slightly better than IP does.

Our evaluation topology is quite simple following an expanded dumbbell model with 8 routers on the channel between each pair of requester and original source (Fig. 3.1). Different possible segment sources are identified with id's of #1 to #9, #9 being the original source. The receivers are located on the left side (only 2 or 4 are chosen from each branch for the tests) while the original sources are located at the right side of the figure. Table 5.1 summarizes the basic setup parameters for the experiments.

The basic queuing discipline in the experiments is FIFO. We fixed the TCP payload size to 1100 bytes. The ConTug segment size was set to 1000 bytes, to compensate for ConTug's larger headers, including a 64-byte zFilter, segIds and etc. In the current implementation, the overall header size of ConTug segments is 140 bytes.For now, we do not consider the effects of this extra header overhead in our evaluations.

The routers along the channel have internal caches, which may be enabled or disabled depending on the evaluation scenario. At the beginning of the experiments there are no cache entries at the routers. The caches only

Toplogy	Extended dumbbell
Link Delay	20ms
Bandwidth	100Kb
Queue size	20KB
File size	$\sim$ 2MB
Segment size	$\sim$ 1KB

Table 5.1: Network Simulation Parameters

get filled as segments pass a node during a simulation.

5.2 Theoretical limits

Theoretical throughput in the network normally equals the capacity of the bottleneck link. For instance, if n concurrent flow share a bottleneck link with capacity C, in the fair scenario the throughput of each flow is expected to be,$\frac{C}{n}$. As a result the minimum flow completion time (FCT) can be as short as $n \times \frac{L}{C}$, when L is the flow length.

In reality, though, the FCT for flows with TCP like protocols normally takes longer because the main focus of these protocols is controlling the congestion and keeping the bottleneck link busy. The FCT at the minimum in these protocols equals $[\log_2(L+1) + \frac{1}{2}] \times RTT + \frac{L}{C}$. In our setting the min FCT for single TCP or ConTug flow (with no caching) is expected to be around 189 sec. This value may increase based on the congestion control behaviour. Increased number of similar concurrent flows in a fair situation will also lead to an increase in the FCT (at least to $n \times 189$ sec). ConTug flows in the caching-enabled environments though are expected to improve the throughput and decrease the FCT. ConTug should be able to achieve this by retrieving the data from different nodes in the channel, without going through the same bottleneck link for all the content items. In the next section, we first show a simple comparison between ConTug and TCP then compare the experimental ConTug results with these theoretical values. In the rest of the section we use terms *rate* and *throughput* interchangeably. The measured throughput in the experiments is calculated based on the Eq. 5.1.

$$Throughput = \frac{\#received\ bytes}{measurement\ period} \tag{5.1}$$

Chapter 6

Simulation results

In the following, we call each hop or node on the channel a potential source $\#i$. In our plots, the window and RTT variations are shown for the whole transfer duration. Therefore, the flow completion time can be read from the figures.

6.1 ConTug Basic Behaviour

In order to have a baseline comparison, we first illustrate the ns-3 TCP implementation's flow and queuing behavior in a simple scenario. For the case of having only a single flow, Figs. 6.1a, 6.1b, and 6.1c, respectively, illustrate the ns-3 TCP a) window and RTT changes, b) the rate changes, and c) the queue size changes at different network nodes.

We now compare the behavior between basic, non-caching ConTug and improved version of TCP Tahoe (the only available version on ns-3). The congestion indicators in Contug is closer to TCP Vegas than Tahoe. But considering general similarities of Tahoe and Vegas [25], ConTug's comparison with Tahoe is sufficient to show the basic similar behaviour of ConTug

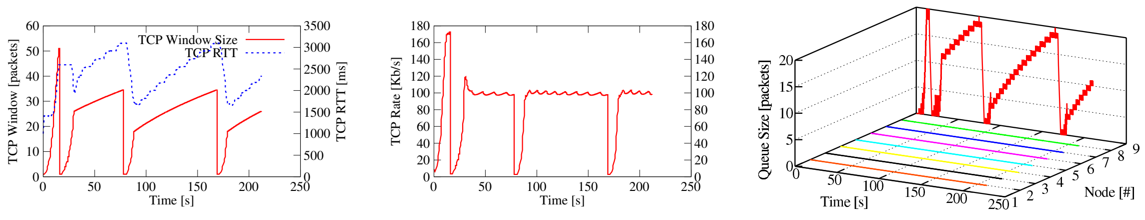

Figure 6.1: Single TCP transfer between requester and original source; showing (a) TCP window and RTT on the left, (b) instantaneous rates in the middle and (c) queue occupancies at the different intermediary nodes on the right.

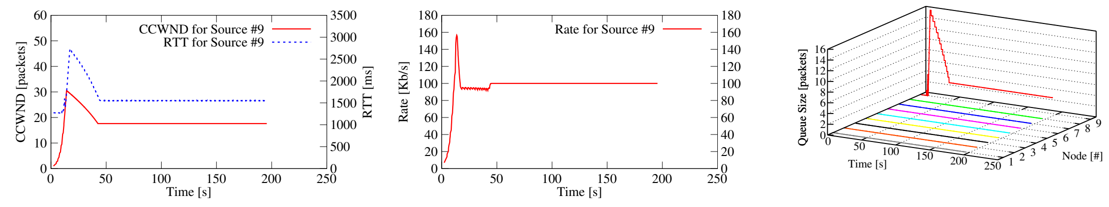

Figure 6.2: Single ConTug transfer in a scenario with empty caches, *i.e.*, all responses come from the original source (node #9); showing (a) CCWND and RTT on the left, (b) instantaneous rates in the middle and (c) queue occupancies at the different intermediary nodes on the right.

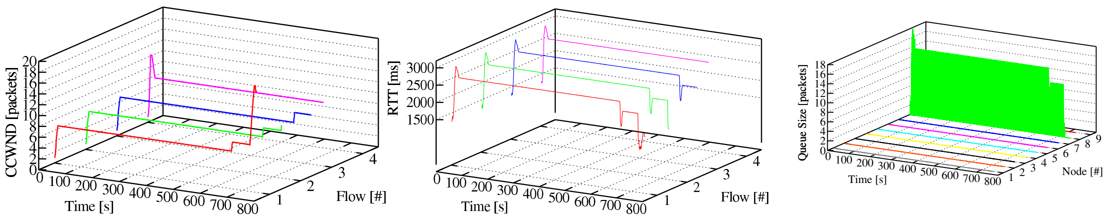

Figure 6.3: Four concurrent ConTug transfers in a scenario without caching; showing (a) CCWNDs on the left, (b) RTTs in the middle and (c) queue occupancies at the different intermediary nodes on the right.

vs. TCP. Our first simulation investigate ConTug's behavior in a scenario with only one flow on a channel with no caching. That is, no transient segment sources are created during the transfer. Fig. 6.2a illustrates ConTug's window and RTT changes, comparable to those of TCP in Fig. 6.1a. As ConTug uses the RTT estimation as sign of congestion, it shows more stable behaviour compared to the version of TCP in Fig. 6.1a. Fig. 6.2b shows ConTug's stream utilization rate (cf. Fig. 6.1b). Since there is no caching on the channel, both TCP and ConTug use the capacity of all the links on the channel almost in the same way. The ConTug receiver though, reacts to the congestion faster than ns-3 TCP; managing to get a bit higher throughput and lower FCT.

Fig. 6.2c illustrates the queue size variations for the ConTug sources in the same scenario. As for TCP, the queue fills up an the farthest point in the channel, close to the only available source (node #9). The difference though is in the amount of queueing in the network. In TCP, the queue varies steadily between the minimum and maximum sizes, Fig. 6.1c. For ConTug, the queue length remains close to the expected minimum, with the RTT estimation approach.

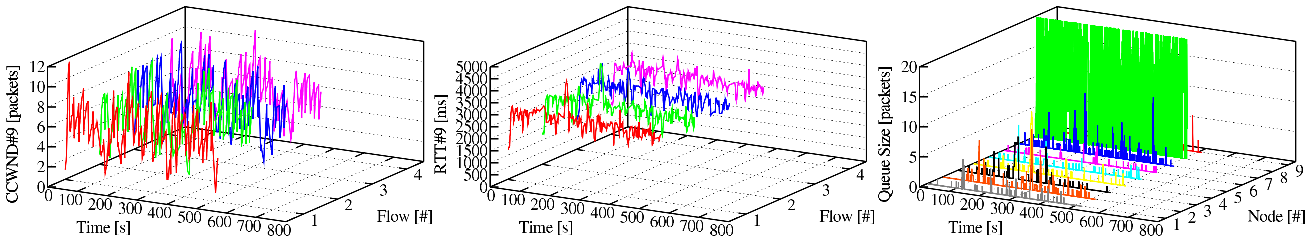

Figure 6.4: Four concurrent ConTug transfers in a scenario *with* caches. For each transfer, plots show only measures taken on responses arriving from the *original source* (node #9); showing (a) CCWNDs on the left, (b) RTTs in the middle and (c) queue occupancies at the different intermediary nodes on the right.

The next set of the figures illustrate ConTug behavior when a few flows start at the same time. Fig. 6.3a illustrates the overall window changes for 4 simultaneous ConTug flows. In this case, the overall CCWND is equal $CCWND_9$, as there is only one source, #9. Fig. 6.3a speaks for an acceptable level of fairness in the network; there are no signs of starvation.As can be seen the average FCT in this case is around 750 sec, which suggest each flow has been able to achieve its expected throughput (refer to sec. 5.2).

Fig. 6.3b displays RTT estimate changes for each flow. We can see that RTT estimates remain at the same level for all flows. Finally, Fig. 6.3c illustrates the queuing behavior on the channel. A queueing bottleneck is formed at the merging point of the flows, which in our case is at node 8. The bottleneck is kept busy during the whole transmission time.

6.2 ConTug With Caching

Our next ConTug simulation examines the general behavior of ConTug in caching-enabled channel. All caches are initially empty, and they get filled in with segments during the transfers. In our simulations, each segment has a 30% chance of getting cached along any node it passes. The goal is to be as close as possible to real scenarios, where the caches are not pre-planned but just happen to be created on a channel.

Fig. 6.4a illustrates the window changes of 4 simultaneous receivers; we use $CCWND_9$ as an indicative variable. The plots show that the estimated $CCWND_9$ changes are qualitatively different from those of the non-caching case, illustrated in Fig. 6.3a. On the caching-enabled channel, some of the segments are found in sources closer to the receiver than #9, resulting in changing CCWND for the source #9. Since there are only a few flows and there is no pre-caching on the channel, the FCT drops only slightly, but still

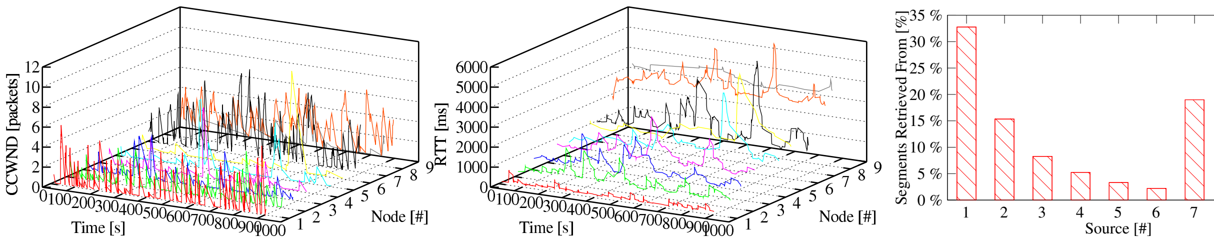

Figure 6.5: 32 concurrent ConTug transfers in a scenario *with* caches. Plots show measures taken for a *single* receiver(transfer); showing (a) CCWNDs used with different sources on the left, (b) RTTs to different sources in the middle and (c) which fraction of segments was retrieved from which source.

it is less than theoretical minimum of 750 sec in a single source channel. In this scenario, caches mainly become useful to serve timeout-triggered retransmission requests. Fig. 6.4b also shows that the RTT estimates for source #9 are higher on the caching-enabled channel compared to a non-caching channel. That is, retransmissions create some additional queueing at the routers, resulting in higher RTT values. But the upper limit for the RTT is expected to be controlled as shown later. From the link utilization point of view, the expected bottleneck link do not become fully empty, as shown in Fig. 6.4c. Thus, if there is capacity and interest, segments may get sent all the time.

6.3 ConTug's Efficiency With Caching

After proving the correct behavior of ConTug, we evaluated its basic efficiency parameters. In this experiments there are multiple simultaneous flows in the network retrieving the *same* file, all starting to request the initial phase at same time. It means routers in the channel can cache random number of passing segments and form transient sources to reply to some of the later arriving requests for same segments. Fig. 6.5a illustrates the variations of different $CCWND_i$s at one of the random receivers. Comparing with Fig. 6.5c, the plot easily mirrors the dependency of $CCWND_i$s to the amount of data served from each sources.

Fig. 6.5b 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 with its RTT based congestion indication is successful in keeping the overall RTT and congestion low enough.

In order to show the efficiency gains that ConTug achieves with this behavior, Fig. 6.5c illustrates the proportion of the segments that the receiver

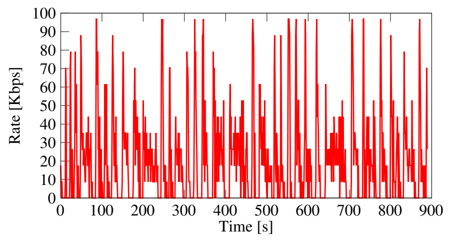

Figure 6.6: Throughtput of a *single* ConTug transfer, with 32 concurrent flow and caching

Flows(#)	Mean FCT	Standard Deviation
4	511.34	0.03
8	613.42	3.4
16	782.81	4.22
32	883.34	18.83

Table 6.1: Mean FCT of ConTug flows requesting same content in caching enabled network

retrieves from different sources. The results show that a large proportion of the segments were retrieved from the closest sources. More than 30% of the segments where retrieved form the first and ∼15% from the second. The number of retrieved segments from the other sources are lower but still non-negligible, due to our random caching approach. In the extended scenarios these savings can play a significant role in the overall bandwidth savings, but we leave more detailed studies on this topic for the future work.

Fig. 6.6 shows the instantaneous throughput that the sample flow gets from the channel. It can be easily seen that the overall average throughput is higher than the expected maximum throughput from a single source. It obviously shows ConTug is able to use the available unknown caching resources in the channel to increase its throughput, without any time or space binding to a specific source. The average FCT shown in Table 6.1 is another sign that the mean throughput of each ConTug flow is higher than the theoretical maximum throughput that TCP can achieve.

Chapter 7

Discussion and Future work

ConTug's throughput, when used without caching, is very similar to TCP and the theoretical limit, while the protocol is completely receiver driven and supports any number of stateless partial senders. Furthermore, the protocol easily adopts to situations where the sources come and go during the transfer. With caching, we gain some savings; *e.g.*, reduced FCT in the case of multiple receivers asking the same file simultaneously.

Compared to CCN [1], we expect ConTug to scale better in terms of memory consumption and flow completion time, mostly due to ConTug not needing to keep any per-request state at the routers.

One specific area of improvement is to combine multiple segment requests into a single packet. As we use cryptographic content hashes as our segIds, a typical 1KB packet can carry only some 50–100 segIds. Even a typical small request, of 100–200 bytes, can carry a few segIds. As an alternative, we are considering using Bloom filters to encode a set of segIds into a request.

A potential problem with combined requests is the reduced ability to quickly reacting to changing conditions. When sending segment requests one at time, the receiver can moderate its sending rate, *e.g.*, based on changes in the RTT estimates, while when sending large packets, each encoding a number of segment request, leaves no space for such moderation. This is likely to require more sophisticated rate control at the receiver, and perhaps completely new concepts for congestion and queuing at the intermediate nodes.

Another important piece of our planned future work is to study alternative rate control and fairness algorithms and analyze them in more complex situations, such as ones with different topologies and interactive protocol runs.

Chapter 8

Conclusion

In this report, we have initiated the discussion on the challenges faced when designing transport protocols for content-centric networks. In particular, we presented ConTug, a new, simple, receiver-driven transport protocol, designed specifically for such networks. Our early evaluation results, obtained via ns-3 simulations, indicate that ConTug works at least as efficiently as TCP over similar network topologies.

When employed with in-network fractional random caching, our initial results indicate more efficient bandwidth usage and shorter flow completion times than with TCP or with ConTug without caching. However, in order to validate the results for more complex scenarios, further studies should be carried out.

Bibliography

[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] C. Labovitz, D. McPherson, S. Iekel-Johnson, J. Oberheide, F. Jahanian, and M. Karir, "Internet Observatory Report," *Proc. NANOG-47*, 2009.

[4] M. Mealling and R. Denenberg, "Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations," RFC 3305, 2002.

[5] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the internet," in *Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems - Volume 3*, ser. USITS'01. Berkeley, CA, USA: USENIX Association, 2001, pp. 4–4. [Online]. Available: http://portal.acm.org/citation.cfm?id=1251440.1251444

[6] D. Wischik, C. Raiciu, and M. Handley, "Balancing Resource Pooling and Equipoise in Multipath Transport," *(under submission)*, 2010.

[7] M. Demmer, K. Fall, T. Koponen, and S. Shenker, "Towards a Modern Communications API," in *Proc. ACM HotNets-VI*, November 2007.

[8] J. Ott and M. Pitkänen, "DTN-Based Content Storage and Retrieval," in *Proc. IEEE AOC*, June 2007.

[9] R. Gupta, M. Chen, S. Mccanne, and J. Walrand, "WebTP: A Receiver-Driven Web Transport Protocol," in *Proc. IEEE INFOCOM*, 1999.

[10] T. Balraj and Y. Yemini, "PROMPT - Destination Oriented Protocol for High Speed Networks," in *Proc. IFIP WG6.1/WG6.4 Workshop on Protocols for High-Speed Networks*, 1990.

[11] H.-Y. Hsieh, K. han Kim, Y. Zhu, and R. Sivakumar, “A Receiver-Centric Transport Protocol for Mobile Hosts With Heterogeneous Wireless Interfaces,” in *Proc. ACM MobiCom*, 2003, pp. 1–15.

[12] B. Cohen, “The BitTorrent Protocol Specification,” http://www.bittorrent.org/beps/bep_0003.html, 2008.

[13] S. Shalunov, “ Low Extra Delay Background Transport (LEDBAT),” Internet Draft draft-ietf-ledbat-congestion (Work in Progress).

[14] U. Legedza, D. Wetherall, and J. V. Guttag, “Improving the Performance of Distributed Applications Using Active Networks,” in *Proc. IEEE INFOCOM*, 1998, pp. 590–599.

[15] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch, “Ricochet: Lateral Error Correction for Time-Critical Multicast,” in *Proc. ACM/USENIX NSDI*, 2007.

[16] T. Anderson, T. Roscoe, and D. Wetherall, “Preventing Internet Denial-of-Service with Capabilities,” *ACM SIGCOMM CCR*, vol. 34, 2004.

[17] C. Rothenberg, P. Jokela, P. Nikander, M. Sarela, and J. Ylitalo, “Self-Routing Denial-of-Service Resistant Capabilities using In-Packet Bloom Filters,” in *Proc. EC2ND*, 2009.

[18] A. Carzaniga, M. Rutherford, and A. Wolf, “A Routing Scheme for Content-Based Networking,” Technical Report CU-CS-953-03, Department of Computer Science, University of Colorado, 2003.

[19] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, “Revisiting IP Multicast,” *ACM SIGCOMM CCR*, vol. 36, no. 4, p. 26, 2006.

[20] 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.

[21] D. Trossen (ed.), “Update on the Architecture and Report on Security Analysis,” PSIRP Project, Deliverable D2.4, 2009.

[22] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, “Redundancy in network traffic: findings and implications,” in *SIGMETRICS*, 2009.

[23] 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

[24] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-Path TCP: a Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1260–1271, 2006.

[25] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "Tcp vegas: new techniques for congestion detection and avoidance," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 24–35, 1994.