UNIVERSITY OF CALIFORNIA
SANTA CRUZ

CONGESTION CONTROL PROTOCOL FOR NAMED DATA NETWORKING

A dissertation submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Abdulazaz Albalawi

December 2016

The Dissertation of Abdulazaz Albalawi
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Brad Smith

Professor Chen Qian

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

Congestion Control Protocol for Named Data Networking

by

Abdulazaz Albalawi

The original use of the Internet was to share expensive resources by addressing endpoints using addresses. These days the Internet is mostly used to access distributed content rather than to share resources, but we still end up addressing endpoints in order to access distributed content. Named Data Networking (NDN) is an approach where content names are addressable, rather than endpoints. The main approach in NDN consists of moving the Internet towards a content distribution architecture and providing better support to mobile devices. One of the design concerns of NDN is congestion control, which is the topic of this thesis. Recent studies on congestion control for NDN suggest either using an end-to-end approach or a hop-by-hop approach. Our Congestion Control Protocol (CCP) for NDN is designed to detect and control congestion at the consumer end without the use of RTT measurements or congestion signaling. CCP is a receiver-driven, window-based protocol that can detect and control congestion by making use of measurements of delay along the forwarding path. We implement CCP using the ndnSim simulator and compare it to an end-to-end AIMDD (additive increase/-multiplicative decrease) scheme that behaves the same way as the Transmission Control Protocol (TCP) used in the Internet today.

Chapter 1

Introduction

This thesis focuses on providing a mechanism for congestion control in the Named Data Networking (NDN) architecture. NDN [1] is an alternative approach to the current Internet architecture. The main problem the Internet was designed to solve was to share expensive computing resources. This is why the internet was designed to use end-host addresses to allow clients to communicate with servers. However, the major use of Internet today is not to access remote computing resources but rather to access remote content and services that may be located at multiple sites.

There are two types of packet in NDN, Interest packets and data packets. To request a remote content in NDN the consumer (client) issues an Interest packet for that content. The Interest packet (or simply Interest) states the name of the content, a nonce, and other information. Each Interest retrieves at most one data packet. Data packets must follow the reverse path traversed by the Interests. Because NDN uses names instead of end-host addresses, content can be cached along the way. This allows middle nodes to serve Interests based on the content stored in their caches. Compared to a CDN (Content Delivery Network), NDN provides routing of content at the network layer while CDN provides routing of

content at the application layer.

Because of the design differences between NDN and the current Internet architecture, a new approach is needed to provide reliable transmissions between two remote application processes, i.e., a new transport-layer approach. Depending on the caching policy, many protocols[4][2] assume that data chunks might be served by different sources. Using a single round-trip time (RTT) measure can give the wrong indication of congestion in the network, as content could be served by multiple sources on multiple paths. Furthermore, keeping multiple RTT values for multiple sources could further add complexity on the receive side. Also, in TCP an out-of-order delivery can cause duplicate acknowledgments that can be wrongly interpreted as an indication of congestion in the network. Data packets can be delivered out-of-order based on the caching policy used in the network, which cannot be used as an indication of congestion anymore as in TCP. Protocols that follow a hop-by-hop approach require the network router's assistance to detect and control congestion in the network. Some of these protocols control congestion by dropping Interests using Interest shapers to force a timeout at the consumer end which can be costly.

Most of the end-to-end protocols for congestion control proposed for NDN use the increase and decrease of RTT estimates as the primary indication of congestion in the network. In this thesis we propose a new protocol for congestion control in NDN that is not based on using RTT estimates. We call this protocol Congestion Control Protocol (CCP).

Instead of an RTT estimate, CCP uses the relative delay that data packets experience with respect to one another as the primary indication of congestion in the network. Using the relative delay, CCP is able to detect if congestion is increasing along the Interest or data path, which cannot be done using only RTT

estimates. Also, in CCP we assume a caching policy that always retains complete content objects. To the best of our knowledge, CCP is the first protocol for congestion control in the context of NDN that operates on the basis of measuring relative delays, rather than RTTs.

The rest of this thesis is organized as follows: Chapter 2 describes the related work of several protocols for congestion control in NDN. Chapter 3 presents CCP, and Chapter 4 evaluates the performance of CCP through simulations. Chapter 5 provides a summary about our protocol and insight of our future work.

Chapter 2

Related Work

Considerable research has been conducted in different areas of the NDN architecture, including content routing, caching policies and mechanisms, and congestion control. This section reviews different congestion-control schemes proposed for NDN.

Many of the congestion-control schemes for NDN are called receiver-based protocols. They are based on using RTT estimates as the primary indication for congestion in the network, which is the same approach followed in TCP. Other proposals depend on network assistance to detect and control congestion in the network, either by shaping the rate at which Interests are sent or by using congestion signaling.

ConTug[4] and ICP[2] are among the first proposed transport protocols for Information Centric Networking (ICN). They are receiver-driven window-based protocols. Both protocols mimic the behavior of TCP and assume data chunks can be served from multiple sources. In ConTug, the retransmission timeout is set to the maximum RTT that is measured during a session. This approach can lead to long timeout values for the Interest packets in a scenario when one chunk might be served from the original provider or a remote cache, and the rest of the

chunks are being served from a nearby cache. If one of the chunks of the nearby caches is lost then the consumer will have to wait for a long time before RTO is triggered and an Interest for the lost data packet is retransmitted. The same scenario can be applied to the way in which ICP sets the timeout value. In ICP, the timeout value is set to the mean of the maximum and minimum RTT that was measured throughout the session.

Most of the congestion control protocols for NDN follow a hop-by-hop approach taking advantage of NDN's "stateful" forwarding plane. Rozhnova et al. [5] propose a hop-by-hop Interest shaper for congestion control. Each router shapes Interests going upstream in order to control the rate of returning data thus avoiding congestion in downstream links. However, the authors assume that the size of data packets and Interests are fixed and all link capacities are known. This was later improved by Wang [11] by incorporating congestion signaling using NACK packets, which was has also been proposed by Yi et al. [12]. Recently, Schneider et al. [15] proposed PCON, a congestion-control scheme that sends congestion signaling to consumers by marking returning data packets. The protocol does not require routers to have knowledge about the link capacity, and does not assume that the size of Interests and data packets are fixed. The protocol detects congestion in the network by measuring packet queuing time over an interval and compares it to a threshold using CoDel AQM [14]. One of the main differences between PCON and other hop-by-hop protocols such as the one in [5] is that PCON does not reject Interests when it detects congestion. Instead, it signals congestion back to consumers by explicitly marking data packets being sent back.

The major design aspects of our protocol, CCP, is that congestion control is not driven by Interest timeouts, using RTT measurements, or sending congestion signaling from routers. Both ICP and ContTug suggest using RTT estimates

to determine congestion in the network and follow Karn's algorithm [6] which states that an RTT estimate for a retransmitted packet cannot be used in the RTT estimation. The disadvantage of this approach is that during the period of congestion (when a packet is lost) no accurate estimates can be made. This leads to inaccurate estimates of RTT during congestion, which may cause the consumer to retransmit prematurely or after undue delays. Moreover, our protocol does not require congestion signaling from routers or knowledge of the link capacity like some of the hop-by-hop approaches do. However, in CCP we assume a caching policy that always retains complete content objects and both Interest and data packets sizes are fixed.

Chapter 3

Congestion Control Algorithm

A major motivation in the design of CCP is that increases and decreases in RTT estimates should not be used as primary indications of congestion in the network. RTT estimates are not efficient to determine whether congestion is increasing or decreasing along the Interest or data path. Figure 3.1 shows an example of two Interests sent by the same consumer and their corresponding data packets. Using only RTT measurements could lead to an incorrect conclusion of congestion developing in the data path for the second packet. However, the true cause for the increased RTT for the second data packet is congestion along the Interest path, not the data path. To solve this problem, consumers in our protocol determine if congestion is increasing or decreasing in either the reverse or the forward path for data packets using *relative delay*.

3.1 Measuring Packet Delay

The delay that packets experience with respect to each other as they are transmitted over the network is referred to as *relative delay*. In CCP, the consumer is able to calculate the *relative delay* for every data packet by a timestamp returned

from the provider or caching site in the data packet header. In CCP we assume a caching policy that retains complete content objects. Thus, no synchronization between the consumer and the provider is required. The consumer keeps a table of two times for every data packet: (a) The arrival time of Interests at the provider, and (b) the arrival time of data packets at the consumer.

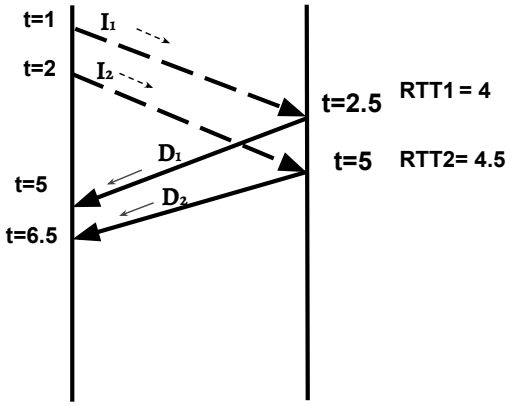

Figure 3.1: Example of RTT ambiguity

The provider timestamps the arrival time of the Interest to the data packet header in order for the consumer to calculate the *relative delay*. From these values, the consumer can determine whether congestion is increasing or decreasing along the Interest path or data path. Using these time values for any two data packets (where j >i) the consumer calculates:

- ΔI: The *relative delay* for Interests; delay along the Interest path
- ΔD: *relative delay* for data packets; delay along the Data path.

To determine whether delay is increasing or decreasing along the data path or Interest path, the following equation is used:

$$RD_{\mathrm{j,i}} = \Delta D_{\mathrm{j,i}} - \Delta I_{\mathrm{j,i}} \tag{3.1}$$

Where RD$j, i$ represents the delay experienced by packet j with respect to packet i. Figure 3.2 shows the arrival of two Interests 1 and 2 at the provider and the arrival of two data packets corresponding to Interest 1 and 2 at the consumer.

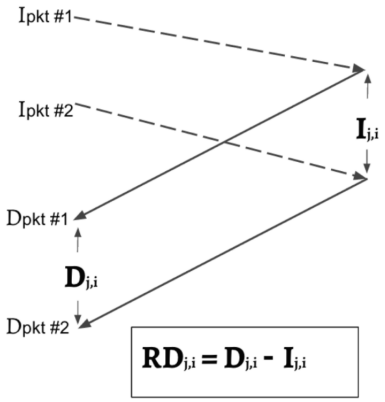

Figure 3.2: Transmission of 2 data packets and corresponding relative delay measurements

Figure 3.3 shows an example on how to use the *relative delay* measurements to determine the level of congestion in the network. In case (a) the *relative delay* for the two data packets is equal to 0. This means both data packets experience the same amount of delay along the data path. In case (b) the second data packet experiences more delay than the first data packet whenever the *relative delay* is larger than 0. Finally, in case (c) the *relative delay* is less than 0, this means the second data packet experienced less delay than the first data packet.

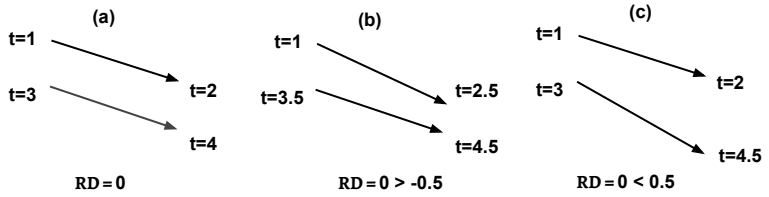

Figure 3.3: Detecting congestion using relative_delay

3.2 Congestion Control Algorithm

The algorithm used in CCP for congestion control follows the same approach used by TCP Santa Cruz [7]. The state machine in Figure 3.4 is the same used by TCP Santa Cruz. It shows the state of the network's bottleneck queue, i.e. whether it is decreasing, increasing or remaining the same using the *relative delay* measurements of the data packets. The positive *relative delay* value represents additional queuing in the network. The negative *relative delay* value represents less queuing in the network. If the *relative delay* is equal to 0, data packets experience no delay with respect to each other. There are two main states of CCP startup and congestion detection.

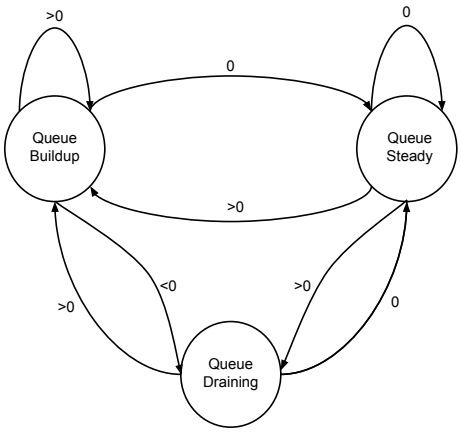

Figure 3.4: TCP Santa Cruz [7] bottleneck queue different status "filling, draining or maintaining"

Congestion Detection

During the congestion detection state, CCP calculates the *relative delay* for every data packet then compares it to an average of the *relative delay* over a time period. This can provide an immediate and better indication of congestion in the network every time a data packet is received by the consumer. As mentioned

before, depending on the caching policy, if we assume chunks could be served from multiple sources, an immediate calculation of the *relative delay* can determine whether contents are being served by a close cache or by a distant provider. As close caches should indicate more throughput, the consumer can increase its Interest window size as the content now is being served from a closer cache. The *relative delay* value for every two data packets is referred to as *delay sample*. The first value of the *delay sample* is only calculated after the arrival of the second data packet. Thus, the CWNDI (Congestion Window of Interests) is set to two Interests at the beginning of the session.

The algorithm used for congestion detection by CCP is shown in Algorithm 1. The algorithm works by comparing every new value of the *delay sample* to a threshold, called *delay threshold*. The *delay threshold* determines how much queuing delay is tolerated in the network. For example, if the value of the *delay threshold* is set to 0, this means no delay is tolerated in the network. This can lead to underutilization of the network's available capacity, as the bottleneck queue will always be empty.

It is clear that the *delay sample* value will fluctuate from one data packet to another due to congestion in the network. Because of this fluctuation, any given *delay sample* value could be atypical. An average of the *delay sample* over a time period will give a better estimate of the delay in the network. The *average delay* is an Exponentially Weighted Moving Average (EWMA) of the combination of old values of *average delay* and the difference between the new value of *average delay* and the *delay sample*. Upon obtaining a new value of the *delay sample*, the protocol updates the *average delay* according to the following equation:

$$average\_delay = \alpha \times average\_delay + (1 - \alpha) \times delay\_sample \tag{3.2}$$

In CCP, during congestion detection the *delay threshold* is set to be equal to the *average delay* over a time period referred to as *time interval*. The *time interval* is set to 100 ms. Consumers measure the *delay sample* of each data packet using the *relative delay* function. If the *delay sample* of each packet exceeds the *delay threshold*, the consumer will consider the network is congested and reduce the Interest window size by one according to Algorithm 2. However, if the *delay sample* is less than or equal to the *delay threshold*, the consumer will increase the size of the sending window by one.

Algorithm 1 CCP Congestion Detection

```
1: function ONDATA(dataPkt)
2:     delay_sample ← dataPkt
3:     UPDATE_AVERAGE(delay_sample)
4:
5:     If startup = true then
6:         delay_threshold = max_delay
7:         time_interval = max_interval
8:         startup = false
9:     end If
10:
11:    If CurrendTime ⩾ time_interval then
12:        time_interval += 100 ms
13:        delay_threshold = average_delay
14:    end If
15:
16:    UPDATE_WINDOW(delay_sample, delay_threshold)
17: end function
```

Startup

Startup is part of our congestion control strategy that ensures new consumers joining the network are able to utilize the link's available capacity (see Algorithm 1). Consumers will begin startup at the beginning of the connection. The *time interval* for startup is set to be equal to *max interval* which in CCP is set to be

500 ms. Also, the consumer sets the *delay threshold* to be equal to *max delay* which is set to 50 ms. The startup strategy can also help achieve fairness when new consumers join the network. Existing consumers will detect an increase in delay when a new consumer joins the network, causing them to reduce their Interest sending rate and allowing the new consumer to increase its sending rate. Once startup exceeds the *time interval*, the consumer enters the congestion detection state. An ideal value of *max delay* and the *max interval* would be a value that allows new consumers to be able to achieve fairness and be able to utilize the network's resources. We leave choosing the right value of *max delay* and *max interval* as part of our future work.

Algorithm 2 CCP Window Update

```
1: function UPDATEWINDOW(delay_sample,
2:                       delay_threshold)
3:     If delay_sample ⩽ delay_threshold then
4:         CWNDI += 1
5:     else If delay_sample > delay_threshold then
6:         CWNDI -= 1
7:     end If
8: end function
```

Timeout

The timeout interval in the protocol should be greater than or at most equal to the *delay threshold*, or unnecessary retransmission of Interests would be sent. However, the timeout interval should not be too large, to avoid the protocol retransmitting the Interest too late when a data packet or an Interests is dropped. Some papers suggest using the maximum RTT with some constants[4], other paper suggest using an average of the maximum and minimum RTT as the retransmission timeout. In this protocol we follow the TCP approach of setting the retransmission timeout to be equal to the average delay of the network plus some

margin.

In CCP, we set the margin to be equal to the variation of *delay sample* and the *average delay*. The next function shows our variance function for the retransmission timeout:

$$delay\_variation = \beta \cdot delay\_variation + (1-\beta)|-delay\_sample - |average\_delay|| \tag{3.3}$$

The timeout interval in the protocol is set to be equal to:

$$TimeoutInterval = 4 \cdot delay\_variation + average\_delay \tag{3.4}$$

Based on this information we compute the retransmission timeout RTO as follows:

$$RTO = min[max\_delay, max[min\_delay, TimeoutInterval]] \tag{3.5}$$

where *min delay* is the minimum delay bound on timeout. In case of retransmitted Interests the consumer will still be able to calculate the *relative delay* for data packets. This is the case for both the arrival of the original data packet and the retransmitted one, as data packets in our protocol will carry a timestamp for their retransmission time either from the provider or from a nearby cache. This actually provides a better indication of the network status than some protocols that follow Karn's algorithm [6] in measuring RTT. We leave the evaluation of the Timeout strategy as part of our future work.

Chapter 4

Performance

This chapter evaluates the performance of CCP under different scenarios. First the performance results of the protocol are shown for a basic configuration with a single consumer and provider with a bottleneck link between them. Moreover, the chapter includes an evaluation of the capability of the protocol in ensuring flow fairness between multiple consumers sharing the same link path. The performance of the protocol was measured through simulation using the ndnSIM[8] network simulator. The ndnSIM is a NS-3[9] module that implements the Named Data Networking (NDN) communication model to support protocol evaluation in both wired and wireless settings.

4.1 Basic Bottleneck Configuration

The first experiment shows the protocol performance over a simple network consisting of a single consumer and a single provider scenario. We compare our protocol with a pure end-to-end AIMD scheme that behaves the same way as TCP, where consumers can only infer congestion via a retransmission timeout and use AIMD window control to avoid congestion, which is used by most end-to-end

protocols in NDN.

The topology of the network depicted in Figure 4.1 consists of a single consumer sending interests for data packets of 32 Kbytes via one intermediate router connected by a bottleneck link of a bandwidth of 40 Mbps and a latency of 5ms. The provider is connected to the middle router via a 200Mbps bandwidth and a 50ms latency link. The router's queue is set to be equal to the BWDP of this configuration which is equal to 1500KB. In addition, all packets have the same size. Also, there is no limit on caching at the router. The imaginary 320MB content file consists of 10,000 chunks in total. This topology is similar to a topology that had already been used in previous paper by Zhang et al. [13].

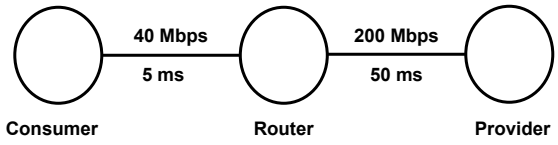

Figure 4.1: Single Flow Topology

Figure 4.2 shows the growth of the consumer's sending window in CCP for the first 3 seconds. Initially the consumer begin with the startup state for the first half second to ensure utilization of the link's capacity. Once the startup phase ends, we see that the consumer start reducing the size of the interest window once packet's delay exceed the average delay (*delay threshold*). Also, we notice that the congestion window reaches a steady state value between 22 and 23 interests. The algorithm maintains this steady state value for the duration of the connection with no occurrence of timeout. This demonstrates the main strength of CCP: transmitting interests at the bandwidth of the connection, without congesting the network and without overflowing the bottleneck queues. Figure 4.3 shows the throughput of our protocol. Notice that CCP was able to fully utilize the link's capacity with an average throughput around 39 Mbps.

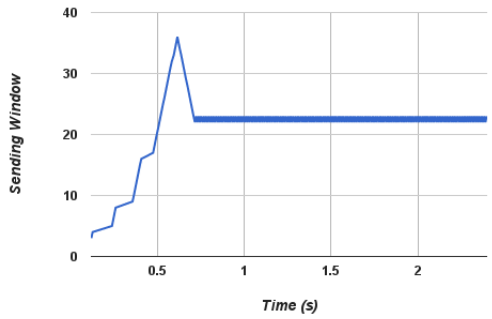

Figure 4.2: CCP Congestion Window

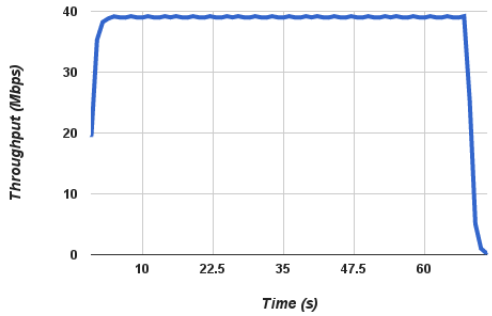

Figure 4.3: CCP Throughput

Figures 4.4 and 4.5 shows the consumer's sending window in the end-to-end AIMD protocol and the average throughput respectively. As can be seen from the figure, the consumer will keep increasing the size of the sending window until a timeout triggered. Once the bottleneck's queue begin to fill up, the router will start dropping incoming packets. Eventually dropped packets will trigger a timeout at the consumer end resulting of a retransmit of the interest and reducing the size of the sending window by half. This is the only way the consumer is able to detect and control congestion in the network. This will result an increase in delay as well as a reduction of throughput.

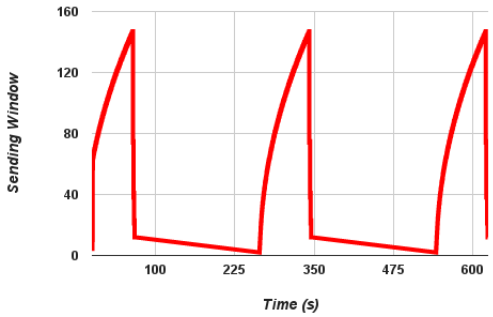

Figure 4.4: end-to-end AIMD Protocol Sending Window

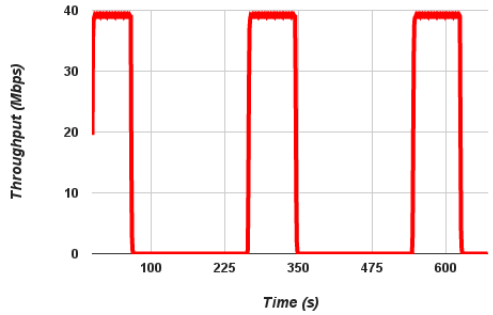

Figure 4.5: end-to-end AIMD Protocol Throughput

4.2 Multi-Flow Scenario

This part of the chapter reviews the behavior of our protocol when multiple flows occur. The topology used in the scenario is depicted in Figure 4.6, which consists of two consumers (Consumer 1 and Consumer 2) and two providers (Provider 1 and Provider 2). The two consumers request different content files of size 1024 Bytes, each one is hosted by one of the providers. In this scenario Consumer 1 starts requesting at time 0 and Consumer 2 starts at time 20s.

Figure 4.7 shows the change in the size of the sending window for both consumers at the time Consumer 2 joined the network. The figure highlights the effect of CCP startup state on existing flows. At time 20s Consumer 2 joined

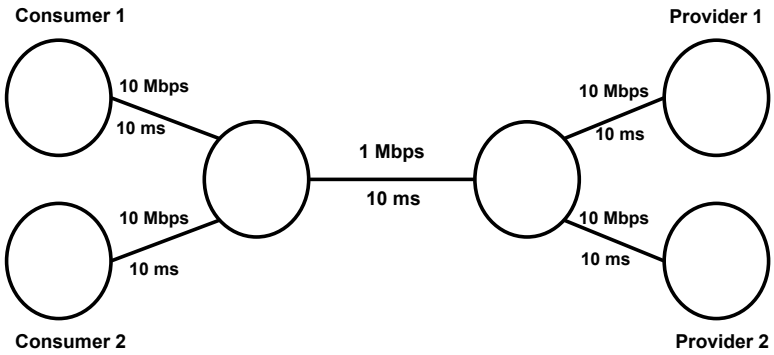

Figure 4.6: Multi-Flow Topology

the network, it started sending interests with a high *delay threshold* for the first half second. This caused an increase of delay in the network which resulted in a reduction of the window size for Consumer 1. Figure 4.8 shows the average throughput for both consumers for the whole duration of the connection. When Consumer 1 started sending interests at time 0, CCP allows the consumer to utilize the network resources efficiently with an average throughput of 953.4 Kbps for the first 20s. Once Consumer 2 joined the network, it caused a reduction in the sending rate for consumer 1. Even though, CCP didn't provide full fairness to Consumer 2 it was able to cause Consumer 1 to reduce its interest sending rate. However, once Consumer 1 finished sending interests, Consumer 2 immediately starts utilizing the network available capacity. We leave the topic of improving fairness in CCP to future work.

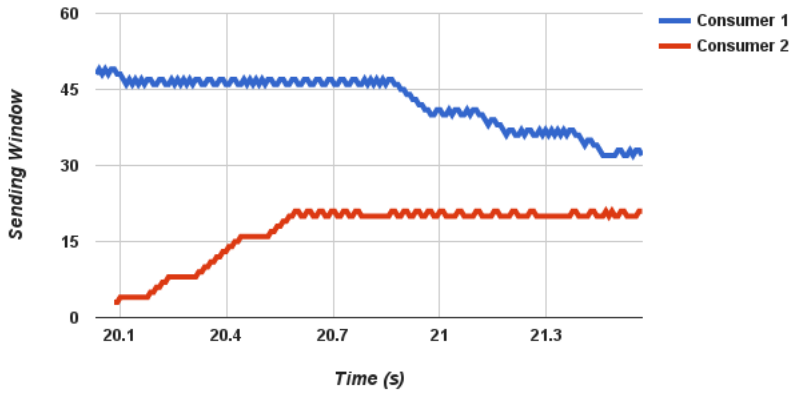

Figure 4.7: CCP Window: Affect of startup state on exciting flows when Consumer 2 joined the network

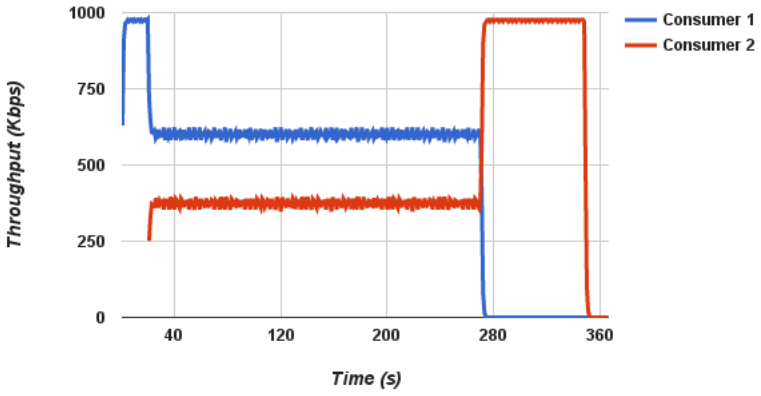

Figure 4.8: CCP Throughput: Consumer 1 starts 20s before Consumer 2

Chapter 5

Conclusion

Named Data Networking (NDN) is an alternative approach to internet architecture, and addresses contents' names rather than endpoints. NDN's main approach moves the internet toward a content distribution architecture and provides better support to mobile devices compared to TCP/IP architecture. Moreover, it also secures the data instead of the communication. One of the design issues of NDN is congestion control. This paper focuses on this issue by presenting the design and evaluation of CCP, a Congestion Control Protocol for Named Data Networking.

Because of the design nature of NDN compared to the TCP/IP model, congestion control can be different. Many recent proposed solutions for congestion control in NDN suggest using RTT measurements as the primary indication of congestion in the network. Other solutions suggest a hop-by-hop approach either by shaping interests or congestion signaling. The Congestion Control Protocol (CCP) is designed to detect and control congestion by making use of measurements of delay along the forward path to indicate on which path congestion is developing. CCP doesn't infer congestion by timeouts, RTT measurements, or congestion signaling . Through the use of timestamps returned by the provider

with every data packet, the consumer is able to estimate the level of congestion at the bottleneck of the connection including retransmitted packets as will.

For our future work we points out a few possible solutions that could further improve the performance of CCP such as:

- Defining synchronization in case of a caching policy that allowed content to be served from multiple sources.
- An evaluation of the effectiveness of the timeout strategy in CCP.
- Determining the right value of CCP variables such as *max delay* and *max interval* that can help achieve fairness and utilization of the network resources during startup.

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.

[2] G. Carofiglio, M. Gallo, and L. Muscariello. ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking. In IEEE NOMEN 12, March 2012.

[3] G. Carofiglio, M. Gallo, L. Muscariello, and M. Papalini. Multipath Congestion Control in Content-Centric Networks. In IEEE NOMEN'13, April 2013.

[4] S. Arianfar, P. Nikander, L. Eggert, and J. Ott, âĂIJContug: A receiver-driven transport protocol for content-centric networks,âĂİ in IEEE ICNP 2010 (Poster session), 2010.

[5] N. Rozhnova, Y. Wang, A. Narayanan, D. Oran, and I. Rhee. An Improved Hop-by-hop Interest Shaper for Congestion Control in Named Data Networking. ICN 2013

[6] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. In Computer Communication Review, volume 17 No. 5, pages 2 âĂŞ 7, August 1987.

[7] Christina Parsa and J.J. Garcia-Luna-Aceves. Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media.

[8] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005. (NDNsim)

[9] F. Urbani, W. Dabbous, and A. Legout. (2011, November) NS3 DCE CCNx quick start. INRIA.

[10] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notifi- cation (ECN) to IP. IETF RFC 3168, September 2001.

[11] Y. Wang Caching, Routing and Congestion Control in a Future Information-Centric Internet

[12] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang. A Case for Stateful Forwarding Plane. Technical Report NDN-0002, July 2012.

[13] F. Zhang, Y. Zhang, A. Reznik, H. Liu, C. Qian, C. Xu. A Transport Protocol for Content-Centric Networking with Explicit Congestion Control

[14] S. Braun, M. Monti, M. Sifalakis, and C. Tschudin. An empirical study of receiver-based aimd flow-control for ccn. In IEEE ICCCN, 2013

[15] K. Schneider, C. Yi, B. Zhang, L. Zhang A Practical Congestion Control Scheme for Named Data Networking In ICN , 2016