A Consideration on Congestion Control for CCN

Yu Wang, Takeshi Muto, Zhou Su and Jiro Katto
Graduate School of Fundamental Science and Engineering
Waseda University, Tokyo, Japan
E-mail: wang@katto.comm.waseda.ac.jp

Suphakit Awiphan
Department of Computer Science, Faculty of Science,
Chiang Mai University, Chiang Mai, Thailand
E-mail: suphakit.a@cmu.ac.th

***Abstract*—Content centric network (CCN) has been developed as a new and revolutionary approach towards networking. CCN aims at replacing the current forwarding mechanism based on IP addresses with a mechanism based on named contents. In CCN, content is divided into several chunks, and when chunks are transferred, routers on the path can cache these chunks. So content chunks in CCN can be retrieved from several resources (e.g. routers, data base, etc.) [1]. This makes simple implicit-feedback transport protocols (e.g. TCP-Reno) not suitable for CCN, because there may be multiple data sources in a transmission. In this paper, we experiment and discuss the congestion control algorithms of CCNx, and discuss the possibilities to use TCP-Reno like congestion control and adaptive timeout in CCN. More importantly, we propose to use multi-thread congestion control in CCN according to its characteristics.**

Keywords: Content Centric Network, Receiver driven congestion control, Loss-based congestion control, Timeout, Multi-thread congestion control

I. INTRODUCTION

CCN [1] has been proposed by Palo Alto Research Center (PARC) to define the new Internet architecture which focuses on named contents rather than IP addresses. With CCN, several benefits (e.g., caching at each network level which reduces congestion and delay, building security at data level, and the mobility to switch to different nodes during a communication) are provided. However, the change of communication paradigm from IP to CCN poses many problems (e.g., naming, routing, etc.).

II. BACKGROUND

A. Content Centric Networking (CCN)

In CCN, there are two kinds of messages: *Interest and Data*, and both of them are identified by names. User discovers data by sending an *Interest* message. Any node having data, which satisfies the Interest, can respond with Data (Content Object).

Because CCN provides in-network caching mechanism, it helps to reduce the network load and delay significantly. But this also challenges those node-based congestion controls.

III. CONGESTION CONTROL OF CCN

Congestion happens when a link or node is carrying so much data, and it can cause huge delay and packet loss. This affects the bandwidth badly. So congestion control helps to avoid or reduce congestions. In CCN, congestion is realized following some basic rules [1]:

- One interest retrieves at most one data packet
- Interest packets serve the role of window advertisements in TCP. Receivers can dynamically vary the window size by varying the interests that it issues to realize congestion control

Existing CCN prototype implementation CCNx [4] uses timeouts to detect loss, both in the end-system and in routers. And it implements its congestion control as in Table 1. The expired time is default set to 4 seconds.

Figure 1 shows the window size changing under the congestion control in Table 1. The transmission is done between two nodes in LAN. Bandwidth is large enough to satisfy the file transmission even if under its maximum window size. And the average transmission bit rate is very low due to roughly changing window size and also 4s's untimely reaction of timeouts.

It is actually not easy to set timeouts to a fixed value that is small enough for timely reactions while avoiding false alerts for packets that are simply delayed. So we want to explore the possibilities of other congestion control algorithms in CCN.

TABLE I. CCN CONGESTION CONTROL REALIZATION IN CCNCATCHUNKS2

Algorithm: Congestion control in ccncatchunks2

```
At Up-Call arrival from the daemon:
If (it signals an EXPIRED interest)
    {current_window = 1;
     re-sent the expired interest;}
else if (it signals an OUT-OF-ORDER Data)
    current_window --;
else if (it is a regular Data && current_window <
MAXIMUM_WINDOW)
    current_winodw ++;
```

*** MAXIMUM_WINDOW <= 127**
*** hole-filling in ccncatchunks2 doesn't function well now**

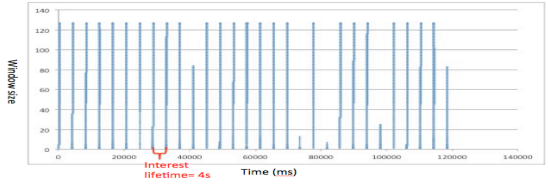

Fig. 1. Window size changing with UDP connection between two nodes in LAN using ccncatchunks2

A. *Receiver-driven TCP-Reno like congestion control*

This algorithm is TCP-Reno like algorithm, which contains slow-start, congestion avoidance, fast retransmit, and fast recovery. Unlike TCP-Reno, it is receiver-driven. Receivers control the window size by controlling the sending of the interests.

Comparing the window size changing between Figure 1 and Figure 2, CCNx congestion control has a higher variability of the window size, while in case of Reno like congestion control window size is comparatively more stable. In general, it is better to have a stable window size rather than a variable one. Because non-stable window size always means non-stable throughput, and for some real time streaming video services, this variability requires large de-jitter buffer so decreasing the quality of experience.

B. *Adaptive timeout*

Another way to detect the packet loss and do retransmission is to use proper timeouts of interest. In figure 1, we find it very difficult to set the timeouts to a fixed value. So eRTT (estimated Round Trip Time) can be used as a dynamic parameter to decide timeouts of each interest and has a fast retransmission. Now we are still implementing this.

C. *Multi-thread congestion protocol:*

TCP-Reno like protocol is based on point-to-point communication, so timeouts based on eRTT and fast retransmission in TCP-Reno may not be appropriate for CCN because data source change frequently, and CCN users may retrieve Data chunks from a number of different nodes/cache as shown in Figure 3.

To solve this problem, and make the Reno and adaptive timeout more applicable to CCN, we propose multi-thread congestion control in CCN.

User realizes congestion controls separately and simultaneously for different data sources that contain different chunks of a requested content. And in each received chunk, information about remaining chunks in that source is written to inform the user the timeouts for those remaining chunks. Therefore, each data source will have its own independent eRTT, and this is used for setting timeouts of next requested chunk in that Data Source.

In Figure 4, we use an example to illustrate how multi-thread congestion control works. Here user wants to retrieve content with different chunks stored in two separate Data Sources. At first, user sends interest for chunk 1, and the window size is initially 1 and timeouts is large. Data source A send back content 1 and telling user that I have remaining chunks 3, 5, 8, 10…Then user sets a window size W_A =2 for data source A and get the $eRTT_A$ between data source and user. After that, user sends interest 2 with large timeouts and interests 3, 5 with timeouts based on eRTT. When chunk 2 (including information about source B's remaining chunks: 2, 4, 6, 7, 9…) is received from data source B, user sets W_B=2 for Data Sources, and calculate $eRTT_B$ between user and source B. When interest 6 is expired, W_B is adjusted and interest 6 is retransmitted, but W_A will not be influenced.

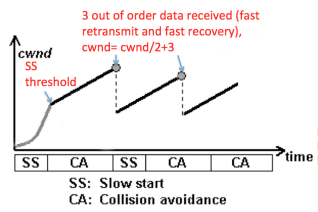

Fig. 2. Window size changing for TCP-Reno like congestion control

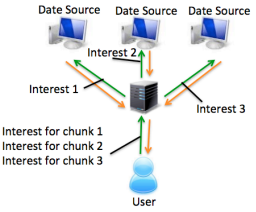

Fig. 3.Users may retrieve chunks of a data from several sources

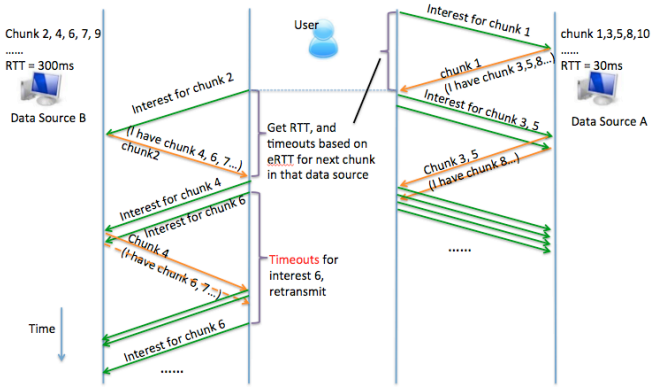

Fig. 4. Example for multi-thread congestion control

By doing it this way, we can realize multi-thread congestion control. And both Receiver Driven TCP-Reno and adaptive timeout can be applied to CCN.

CONCLUSION

This paper experiments and discusses the congestion control in CCNx, and proposes the possibilities to use TCP-Reno like congestion control and adaptive timeouts in CCN. According to the characteristics of CCN that it may retrieve data from multiple sources, we propose multi-thread congestion control in CCN. And later, we will try to carry out our experiments.

ACKNOWLEDGMENT

This paper has been carried out in the framework of the FP7/NICT EU-JAPAN GreenICN project.

REFERENCES

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

[2] Suphakit Awiphan, Takeshi Muto, Yu Wang, Zhou Su and Jiro Katto " Video Streaming over Content Centric Networking, Experimental Studies on PlanetLab" ,ComComAP, Hong Kong, China, April. 2013, pp.19-24

[3] Paolo VIOTTI, "Caching and Transport Protocols Performance in Content-Centric Networks", Master Thesis, September 2010

[4] CCNx Project, [Online], http://www.ccnx.org

[5] Stefano Salsano, Andrea Detti, Matteo Cancellieri, Matteo Pomposini, Nicola Blefari-Melazzi, "Transport-Layer Issues in Information Centric Networks", ACM SIGCOMM Workshop on Information-Centric Networking (ICN-2012), Helsinki, Finland , August 2012.

[6] L. Saino, C. Cocora, G. Pavlou, "CCTCP: A Scalable Receiver-driven Congestion Control Protocol for Content Centric Networking", In IEEE ICC 2013 - Next-Generation Networking Symposium (ICC'13 NGN), Budapest, Hungary, June 2013