MIT Libraries | DSpace@MIT

MIT Open Access Articles

An algorithm for improving Sliding window Network Coding in TCP

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation: Karafillis, P., K. Fouli, A. ParandehGheibi, and M. Medard. "An Algorithm for Improving Sliding Window Network Coding in TCP." 2013 47th Annual Conference on Information Sciences and Systems (CISS) (March 2013).

As Published: http://dx.doi.org/10.1109/CISS.2013.6552263

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: http://hdl.handle.net/1721.1/90429

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike

Massachusetts Institute of Technology

An Algorithm for Improving Sliding Window Network Coding in TCP

P. Karafillis, K. Fouli, A. ParandehGheibi, and M. Médard[*]

***Abstract*— Sliding-window Network Coding (NC) is a variation of Network Coding that is an addition to TCP/IP and improves the throughput of TCP on wireless networks. In this article, two implementations of a new algorithm are proposed in order to decrease the total transmission time, and to increase the decoding throughput throughout the transmission. The algorithm's main process identifies then retransmits the number of outstanding lost packets and is implemented in two different ways. The End of Transmission (EOT) implementation applies the process only once at the end of the transmission, whereas the "Pseudo-block" (PB) implementation applies the process at regular intervals throughout file transmission. The discrete event simulator *ns*-2 is used to implement and test the benefits of the proposed algorithm. Our extensive simulation results show that both implementations provide a sizeable decrease in average transmission time. For the first implementation (EOT), the average time to receive data decreased by 8.04% for small files (under 1 MB) compared to TCP/NC. The second implementation, PB, reduces file transmission times by up to 70% for larger files (GB range). Furthermore, PB creates a more even decoding throughput and allows for a smoother transmission. In this work, PB is shown to decrease the average standard deviation of the decoding throughput by over 60%. This decrease in decoding delay demonstrates the potential of sliding window NC in future streaming applications.**

***Keywords*— Network Coding, TCP, Sliding Window, Packet Erasure Channel**

I. Introduction

THE transmission of data in computer networks requires methods to ensure that the transmitted information is free of errors and losses. One of the most widely used protocols for data transmission is the Transfer Control Protocol (TCP) [1]. Originally developed in 1974, TCP and its various adaptations such as TCP Reno, New Reno and TCP Vegas use the same basic structure, the Sliding Window Acknowledgment System. This system was developed to primarily ensure that information packets were transmitted correctly and without exceeding the transmission rate of the wired networks that were prevalent at the time (as opposed to the mix of wired and wireless networks of today). In wired networks, the main concern during data transmission was congestion, whereas in wireless networks there is the additional concern of data loss due to wireless connection deficiencies [2].

[*] P. Karafillis is with The Winsor School, Pilgrim Road, Boston MA, 02215 (e-mail: pkarafillis@winsor.edu).

K. Fouli, A. ParandehGheibi, and M. Médard are with the Massachusetts Institute of Technology, 77 Mass. Av., Cambridge MA, 02139 (e-mail: {fouli,parandeh,medard}@mit.edu).

Using messages commonly referred to as Acknowledgments (ACKs), TCP monitors if all the packets sent within a packet window have arrived safely at their destination. It also monitors how quickly the recipient can receive data, in order to prevent congestion. For each packet received, the recipient sends an ACK to the sender carrying a sequence number greater by 1 than the last byte it received. If a packet is dropped or lost, the recipient will resend the ACK for the last packet it received, resulting in a duplicate ACK. Duplicate ACKs sent from the recipient indicate that not all of the data sent within a window of packets have arrived. After receiving three duplicate ACKs, the size of the sliding window is halved. TCP interprets all transmission errors as congestion, and the applied correction is the reduction of the transmission rate. Although this method ensures that data have been transferred correctly, it also significantly reduces throughput, even when there is a small amount of losses. It has been demonstrated that a packet loss rate of 2% causes an over 60% decrease in throughput [3].

In addition to storing and forwarding, Network Coding (NC) enables nodes to combine or separate transient bits, packets, or flows through coding and decoding operations [4,5]. NC can be used as a backward-compatible enhancement to TCP in order to improve efficiency in lossy networks such as wireless networks [3,6]. This work builds upon the seminal TCP/NC proposal by Sundararajan et al. [6]. TCP/NC applies random network coding [7] for individual flows. The NC layer generates coded packets by linearly combining [5,7,8] all packets within a certain coding window or buffer. Any packets thus generated from t uncoded packets may be seen to represent an equation in t variables. Once t such coded packets have arrived at the destination, the t original packets can be decoded as if they had each arrived separately.

In TCP/NC, redundancy is used preemptively against packet losses: instead of resending packets after losses have been detected, the number of transmitted packets t is multiplied by a redundancy factor R, ensuring that enough packets are transmitted so that t packets arrive at their destination. The redundancy factor is theoretically the reciprocal of the fraction of packets that successfully traverse the network. For instance, if the loss rate p were 20%, then theoretically $R = \frac{1}{(1-p)} = 1/.8 = 1.25$. It has been shown through simulation and analysis that TCP/NC achieves significantly higher throughput than traditional versions of TCP in lossy networks [3,4,9].

TCP/NC uses a sliding coding window [6]. Hence, the

packets in the coding window can be decoded only after enough packets have been received to enable decoding; i.e., a count of *t* equations has been received for the transmission of *t* packets. This means that there are periods of time when the recipient has received packets but cannot decode them until additional redundant packets are received. If the redundant packets do not cover the losses, a TCP-imposed time-out may occur, and the recipient has to wait for the missing packets to be retransmitted in order to decode and use the data.

This article demonstrates that these two issues (i) having to wait for the successful transmission of an adequate count of packets prior to decoding and (ii) end of connection timeouts and retransmissions, cause time delays and large variations in the data *decoding rate* for a simple source and sink network. These issues can cause severe fluctuations in the throughput of decoded data, thereby presenting a problem for the use of NC in applications where a streaming rate has to be maintained, such as multimedia streaming applications [10]. In order to improve the performance of network coding with respect to issues (i) and (ii) identified above, an algorithm that can reduce or eliminate the end-of-connection retransmissions and decrease the decoding delay was developed. The algorithm's main process computes then transmits a number of coded packets based on the number of outstanding lost packets. It is implemented in two different ways. The End of Transmission (EOT) implementation triggers the process only once at the end of the transmission whereas the "Pseudo-block" (PB) implementation applies the process at regular intervals throughout transmission. The number of outstanding lost packets is carried by a new ACK header field.

The discrete event simulator *ns*-2 [11] is used to implement and test the benefits of the proposed algorithm. Simulation results show that both implementations provide a sizeable decrease in average transmission time. Under a 5 Mbps connection, EOT decreases the average time to receive data by up to 8.04% for small file sizes (under 1 MB) compared to TCP/NC. PB reduces the average file transmission times by more than 29% for medium-sized files (1MB-10MB) and more than 70% for large files (1GB-10GB) compared to TCP/NC. PB further creates a more even decoding throughput and allows for a smoother transmission: PB is shown to decrease the average standard deviation of the decoding throughput by over 60%.

Our article is organized as follows. Section II describes our proposed algorithm as well as the modeling involved in our simulations. Section III illustrates the performance gains of EOT- and PB-enhanced TCP/NC over conventional TCP/NC [6]. In Section IV, we focus on PB by conducting extensive simulations to demonstrate the benefits of PB-enhanced TCP/NC for a wider array of file sizes and packet loss rates. A discussion and conclusion are finally provided in Sections V and VI, respectively.

II. Modeling and Simulation

Ns-2 Simulator and Network Coding

This work builds upon the seminal TCP/NC proposal by Sundararajan et al. [6]. Using the *ns-2* discrete-event simulator [11], a simple network was modeled with one source, one sink, and no intermediate nodes. A number of TCP variations are provided in *ns-2*, including TCP Reno, New Reno, and Vegas [12]. TCP/NC is implemented through modifications to TCP Vegas. In our work, decoding is assumed to be instantaneous.

In our simulations, the available bandwidth is 5Mb/s, the queue size is 100 packets, and the receive window is 100 packets. Each packet is 1000 B (i.e., 1 KB) and each ACK is 40 B. The Vegas parameters are $\alpha = 28$, $\beta = 30$,$\gamma = 2$ [13].

Acknowledgements (ACKs)

TCP ACKs are also adjusted as follows. An array containing the identities of packets involved in the coding (i.e, coding window) is maintained and transported in the header of each coded packet. When a coded packet is received, the decoding module loops through the coding window array of the received packets until it finds a packet within this array that it has not yet acknowledged. The sink then acknowledges this packet. This model of NC assumes that all packets are linearly independent and therefore that each packet provides the mathematical equivalent of a degree of freedom. Furthermore, acknowledgements are assumed to be carried by a lossless channel [6].

Redundancy

A fixed redundancy rate R is implemented. During the algorithm execution, a variable N (referred to as redundancy accumulation index) is initialized to zero. For each packet to be sent, N is incremented to $N+R$ and the integer part of N is subtracted. The subtracted integer indicates the number of packets that are sent in place of a single packet [6]. Such a transmission algorithm applies the redundancy rate dynamically on a packet-per-packet basis.

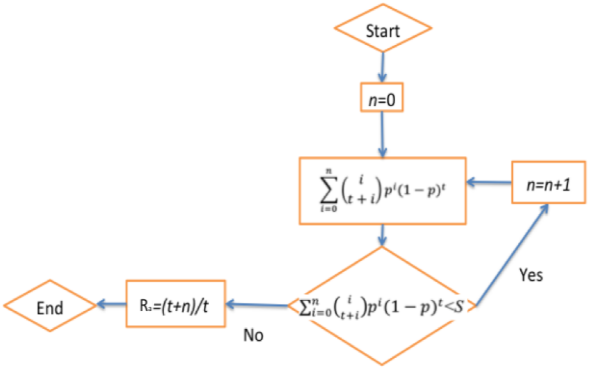

Figure 1: *Algorithm for the calculation of the adjusted redundancy R_a*

For a given packet loss rate p, and the theoretical redundancy value is given by $R = \frac{1}{(1-p)}$. In this work, we use an adjusted redundancy factor R_a that is calculated to achieve

a transmission success rate *S*. Unless stated otherwise, this threshold success rate is set to be equal to 90%. The adjusted redundancy value R_a is calculated using the **Redundancy Algorithm** of **Fig. 1**.

The Proposed NC Improvement Algorithm

In order to address the issues of NC described in the introduction of this article (decoding delay and end of connection retransmissions), we propose that the following algorithm be run in conjunction with TCP/NC:

Step 1: At the sink, the total count of lost packets is determined by subtracting the largest packet sequence number in the coding window from the sequence number that the NC-enabled sink is currently acknowledging. This number is then inserted into the ACK header and transmitted to the source.
Step 2: At the source, the total count of lost packets is updated each time an ACK is received.
Step 3: There are two different implementations of the algorithm in this step:
End of Transmission Implementation (EOT): In TCP, the application layer sends a FIN (finish) flag to the Transport Layer indicating that it no longer needs the connection to send data [1]. In the EOT implementation proposed here, upon detection of FIN and in addition to the final packet, the source sends a number of coded packets equal to the total number of lost packets in order to cover any outstanding losses.
Pseudo-Block Implementation (PB): At regular intervals in the count of transmitted (non-redundant) packets, a number of coded packets equal to that of the lost packets is sent. In this work, an interval (i.e., block) of 100 packets is used. EOT is additionally implemented by the source.

Collecting Data

In our work, the time of arrival and the sequence number of all received packets are recorded. The instantaneous decoding throughput of the connection can thus be calculated by finding the number of packets decoded per time interval. The interval used here is .1 seconds.

III. Comparing NC, EOT and PB

In this section, we compare the End of Transmission (EOT) and the Pseudo-Block (PB) implementations of the proposed algorithm with the NC implementation described in [6]. For the sake of comparison, we compare the behavior of the three algorithms under identical conditions (i.e., identical simulation seed). We define the total transmission time as the length of time from connection request to receipt of all data in the file.

End of Transmission (EOT) vs. NC

In this section, we establish a single TCP connection from source to sink and compute the file completion time while varying four parameters: The delay between the source and the sink is either 50ms or 150ms, the file size is 250, 500, or 1000 packets, the loss rate is either 5% or 15% and the redundancy rate is either the theoretical redundancy rate ($R=\frac{1}{(1-p)}$), or adjusted redundancy R_a. **Table I** shows the average decrease in transmission time for each file size (over all other simulated parameters).

Table I: File Size and Decrease in Transmission Time of NC+EOT vs. NC.

File Size (KB)	Avg. Decrease in Transmission Time [%]	Avg. Decrease in Transmission time (s)
250	6.09	0.249
500	8.00	0.452
1000	10.04	0.756

The average decrease in total transmission time for all simulations with respect to NC was 0.485 seconds, or 8.04%. EOT never had a negative effect on the total transmission time. The effects of EOT on transmission time improved as the file size increased, reaching 10% for 1 MB files. Substantial delay improvements are expected for larger files.

We next plot the instantaneous decoding throughput (i.e., post-decoding throughput seen at the TCP layer) for a single file transmission. **Figs. 2** and **3** show average instantaneous decoding throughput of NC alone, then NC with EOT and PB. This simulation is run for a file size of 3 MB, an RTT of 50 ms, and under packet losses of 10% (**Fig. 2**) then 5% (**Fig. 3**). Since the EOT algorithm is only implemented upon receipt of the FIN flag, EOT and NC results differ only at the end of the transmission. In both **Figs. 2** and **3**, the transmission of packets using EOT terminates consistently earlier than NC.

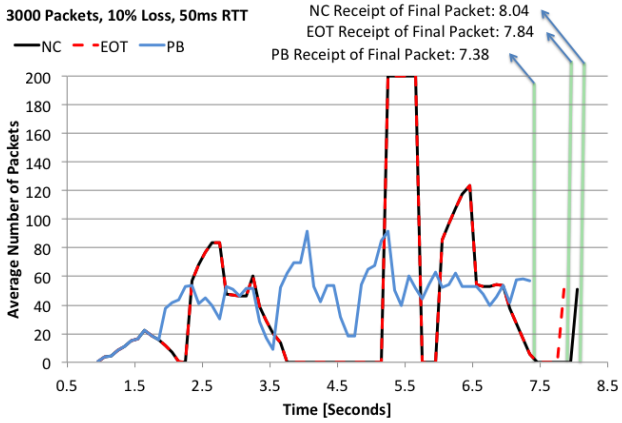

Figure 2: *Comparison of Instantaneous Decoding Throughput between NC, PB, and EOT under 10% Loss Rate*

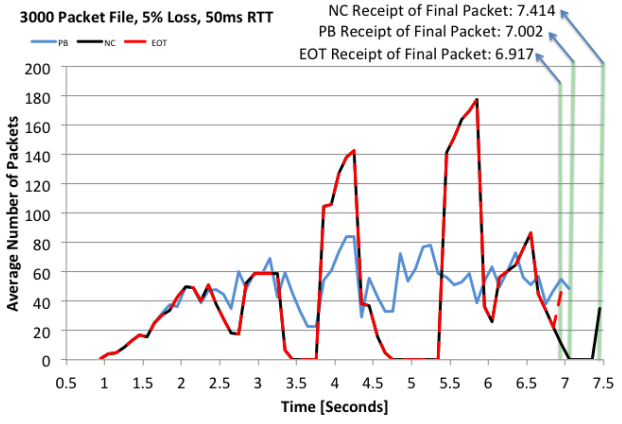

Figure 3:*Comparison of Instantaneous Decoding Throughput between NC, PB, and EOT under 5% Loss Rate*

Pseudo-Block (PB) vs.NC

In **Figs. 2** and **3**, the PB interval is set to 100 packets, as mentioned in the algorithm. **Figs. 2** and **3** show that PB has an identical decoding throughput as NC early in the transmission, before the PB algorithm is first executed. However, soon after the first PB execution, the output becomes distinct from NC.

PB presents major advantages with respect to NC and EOT.

First, it smoothes the profile of the instantaneous decoding throughput, as is clear from **Figs. 2** and **3**. The standard deviation of the average throughput for NC alone is 121.53 packets per second, while the standard deviation of PB is 48.08 packets per second. Therefore, PB causes over 60% decrease in the standard deviation. This decrease clearly indicates that PB decodes packets at a steadier rate when compared with NC and EOT, both the latter incurring periods of no decoding activity followed by decoding peaks involving large numbers of packets. This effect can be seen on **Figs. 2** and **3**, by observing the distribution of decoding events (i.e., the throughput peaks due to the arrival of a sufficient number of coded packets for decoding to occur) for all three examined methods. Decoding events are significantly more frequent using the PB algorithm vs. the prolonged lapses of decoding for NC and for EOT. The higher frequency of decoding events in PB leads to a higher decoder throughput and faster transmission times than NC. In addition, the PB algorithm leads to lower decoding complexity, since the size of the decoding operation (i.e., the number of packets to decode) is controlled through the "interval" parameter, in this case 100 packets. As a result, the PB algorithm is superior to NC and to EOT under stringent streaming requirements.

A marked reduction of the total transmission time in PB vs. NC can also be observed on **Figs. 2** and **3**. PB is further investigated in the next section.

IV. Investigating PB

In this section, we run extensive simulations showing the benefits of PB-enhanced TCP/NC for a wider array of file sizes and packet loss rates. For all simulated scenarios, the source-sink delay is 50ms, the redundancy ratio is derived using the algorithm of **Figure 1**, and the PB interval is set to 100 packets. In the rest of this section, we use the terms "PB" and "NC" to denote PB-enhanced TCP/NC and conventional TCP/NC [6], respectively.

Our first set of simulations compares the total file transmission delay of NC and PB at three packet loss rates: 1%, 10%, and 30%. For each of the loss rates, **Figure 4** plots the transmission delays of both NC and PB for file sizes ranging from 100KB to 10GB. Each histogram data value is the average of 100 simulation runs with different seeds.

We observe that the total transmission time increases as a function of the file size and packet loss rate. More importantly, we note tremendous reductions for PB compared to NC at the three packet loss rates.

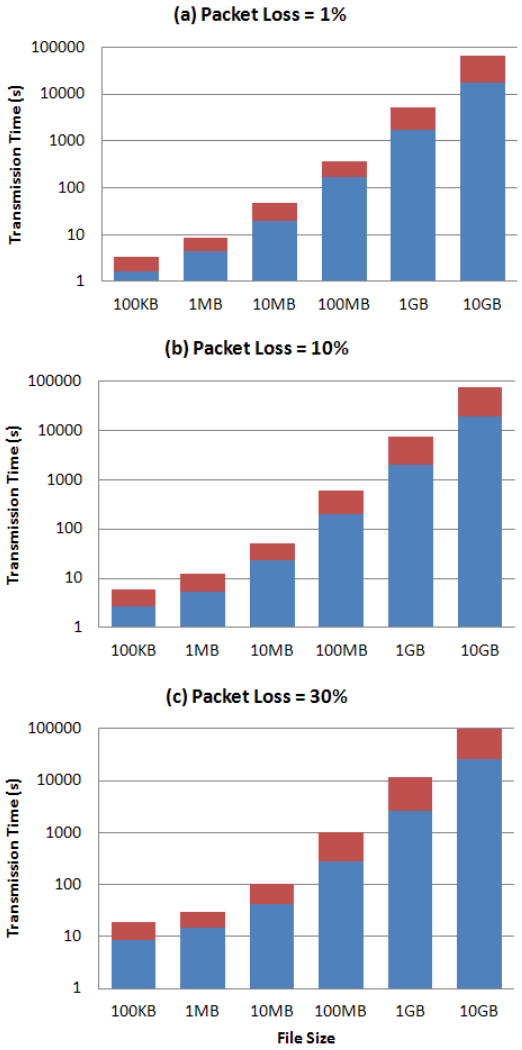

Figure 5: *Total Transmission Time(s) vs. File Size for NC (red) and PB (blue) with a packet loss rate of (a) 1%, (b) 10%, and (c) 30%.*

PB greatly decreases the total transmission time compared to NC. **Table II** lists the average percentage reduction in transmission time of PB compared to NC for all simulated scenarios. We note that while medium-sized files (1MB-10MB) see over 29% reductions in transmission times from NC to PB, these improvements reach 70% for large files (1GB-10GB).

Table II: Average Percentage Gain in Transmission Time of PB vs. NC.

Loss	100KB	1MB	10MB	100MB	1GB	10GB
1%	7.97%	0.53%	25.47%	5.15%	51.34%	64.54%
10%	15.66%	20.49%	11.25%	53.24%	65.23%	65.19%
30%	17.33%	2.20%	29.77%	61.21%	70.10%	72.12%

Our second set of simulations illustrates the instantaneous decoding throughput advantages of PB compared to NC. **Figure 5** is a trace of the instantaneous decoding throughputs of PB and NC in two transmission instances of a 10MB file. The two loss rates shown in **Figure 5** are 1% and 10%. Contrary to **Figs. 2 and 3**, the simulations of **Figure 5** are not

run under identical loss conditions (i.e., identical seed), hence the distinct traces during the early stages of the simulation.

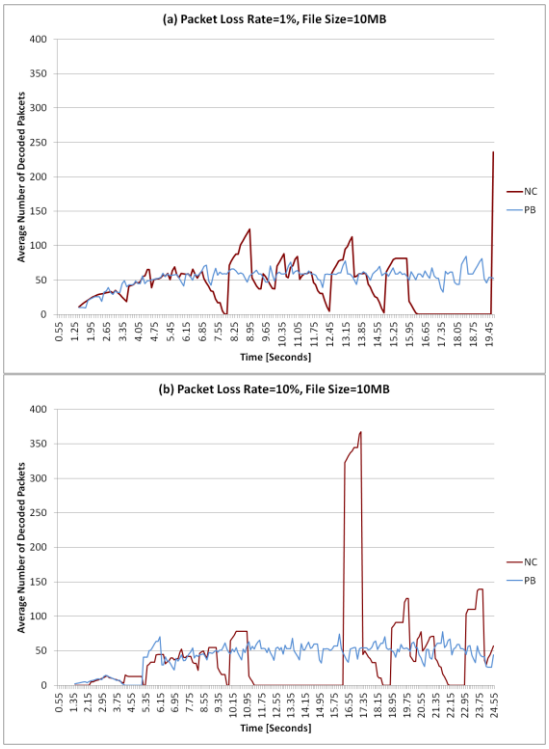

Figure 5: *Instantaneous Decoding Throughput for NC (red) and PB (blue) with a packet loss rate of (a) 1%, (b) 10%, and (c) 30%.*

Figure 5 confirms that the throughput-smoothing effects of PB are carried out through larger file sizes. During large time periods where the decoding throughput of NC drops to zero, the instantaneous decoding throughput of PB remains steady (around 50 packets per unit time). While increasing the packet loss rate from 1% to 10% has little effect on the decoding throughput of PB, the NC trace shows more extreme variations and longer decoding lulls, hence increasing the likelihood of TCP time-outs.

V. Discussion

PB and EOT can provide benefits such as decrease in total transmission time and increase in decoding events. However, they can only do so with the knowledge of the current number of lost packets, N_{lp}. In the simulations, this data is transmitted in a new field of the acknowledgement header. In a true implementation, this field would be a one byte addition to the TCP acknowledgement header.

While in the simulation the number of lost packets was transmitted between source and sink through acknowledgement headers, it is also possible to calculate the value of lost packets without adding a new field in the acknowledgement header. If, instead, the source were to increment the value of N_{lp} one RTT after a non-redundant packet is sent, and decrement N_{lp} each time an ACK is received, the source will be able to keep an accurate count of N_{lp}. However, this method relies heavily on an accurate and stable RTT.

PB creates the benefit of a smoother throughput due to the increase of the number of decoding events throughout the transmission. PB also decreases the size of the decoding buffer. In contrast, for NC, where a decoding event does not occur for long periods of time, the decoding buffer grows, hence increasing the likelihood of congestion.

VI. Conclusions and Future Work

In this article, two implementations of a new algorithm are proposed in order to decrease the total transmission time of sliding-window NC, and to increase the decoding throughput throughout the transmission. This is achieved by determining the number of outstanding lost packets and transmitting an equivalent number of coded packets at the End Of Transmission (EOT) and additionally in regular intervals during transmission (Pseudo-Block, PB).

Under a 5 Mbps connection, EOT is shown to decrease the average file transmission time by up to 8.04% for small file sizes (under 1 MB) compared to NC. Our extensive discrete-event simulations show that, compared to NC, PB decreases the file transmission time by more than 29% for medium-sized files (1MB-10MB) and more than 70% for large files (1GB-10GB). In addition, PB decreases the burstiness of decoding and allows for a smoother transmission, making sliding-window NC a viable and competitive option for streaming applications. In our simulations, PB decreases the average standard deviation of the decoding throughput by over 60%.

There are several promising possibilities for future testing of this algorithm. Implementing it in a real network will further demonstrate its effectiveness. Besides, testing sliding-window NC in more complex topologies will determine the performance of the proposed algorithms more accurately.

References

[1] "The TCP/IP Guide" in *http://www.tcpipguide.com/*.
[2] B. Sardar and D. Saha, "A survey of TCP enhancements for last-hop wireless networks," IEEE Comm. Surveys and Tutorials, Vol. 8, pp. 20–34, 2006
[3] M. Kim, "Network Coding for Robust Wireless Networks", p. 106, Ph.D. Thesis, MIT, Feb. 2012, http://web.mit.edu/minjikim/www/papers/2012-phdthesis.pdf
[4] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," IEEE Trans. Inf. Theory, vol. 46, pp. 1204–1216, 2000
[5] R. Koetter and M. Médard, "An algebraic approach to network coding," IEEE/ACM Trans. Netw., vol. 11, pp. 782–795, 2003
[6] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in Proc. of IEEE INFOCOM, Apr. 2009, pp. 280–288
[7] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," IEEE Trans. Inf. Theory, vol. 52, pp. 4413–4430, October 2006
[8] S.-Y. R. Li, R.W. Yeung, and N. Cai, "Linear Network Coding", IEEE Transactions on Information Theory, Vol. 49, No. 2, Feb. 2003
[9] M. Kim, M. Médard, and J. Barros, "Modeling network coded TCP throughput: A simple model and its validation," in Proceedings of ICST/ACM Value tools, May 2011
[10] M. Wang and B. Li., "R^2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming", IEEE Journal on Selected Areas In Communications, Vol. 25, No. 9, Dec. 2007
[11] "Network Simulator (*ns-2*)," in *http://www.isi.edu/nsnam/ns/*
[12] J. Mo, R. J. La, V. Anantharam, and J Walrand. "Analysis and Comparison of TCP Reno and Vegas". INFOCOM '99. Proc. IEEE, Vol. 3, pp. 1556-1563, March 1999
[13] L. S. Bramko, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in Proceedings of the SIGCOMM '94 Symposium, August 1994