Self-Adaptive TCP Protocol Combined with Network Coding Scheme

Sicong Song[1], Hui Li[1*], Kai Pan[1], Ji Liu[1], Shuo-Yen Robert Li[2]
1. Shenzhen Key Lab of Cloud Computing Technology & Application, Shenzhen Graduate School, Peking University, Shenzhen, China, 518055
2. Dept. of Information Engineering, The Chinese University of Hong Kong, China
E-mail: songsc0707@yahoo.com.cn, lih64@pkusz.edu.cn, pankai0905@gmail.com, jliu@pkusz.edu.cn bobli@ie.cuhk.edu.hk

Abstract*—A Self-Adaptive Network Coding TCP protocol is proposed for dynamical adaptation of the redundancy factor in the network, including the case of a wireless network. It trims packet loss effectively via redundant packets of network coding. It also adapts certain traffic information, to be stored in the header of TCP or ACK packets, thus enables the sender to dynamically adjust the redundancy factor of the network. Simulation of traffic fluctuation in the real network shows better utilization of communication channels and better throughput by the proposed protocol than TCP-Vegas as well as NC-TCP.

Keywords-network coding; packet loss; TCP

I. INTRODUCTION

Network coding is a technique where, instead of simply forwarding the packets the nodes receive, they will combine several packets together for transmission in order to be used for attain the maximum possible information flow in a network. It has emerged as an important potential approach to the operation of communication network, including the case of a wireless network where network coding can trim losses effectively. The major advantage of network coding is masking packet loss by mixing data across time and across flows [1-3]. In lossy networks, network coding can mask the packet loss via redundant packets, thus decrease the delay caused by the timeout and to raise the utilization of the channels. However, we still seem far from seeing widespread implementation of network coding across network. Since network coding can bring benefits in terms of throughput and robustness [4,5], how to put it into practice in real communication network is the main problem that needs to be solved. To do so, firstly, we need to plant network coding into TCP properly with minor changes to the protocol stack, thereby allowing incremental development. We therefore see a need to find a sliding-window approach as similar as possible to TCP for network coding that makes use of acknowledgments for flow and congestion control [6]. Such an approach would necessarily differ from the generation-based approach more commonly considered for network coding [7, 8]. Secondly, we need to solve the delay of encoding and decoding caused by network coding, which can do harm to the performance of networks.

TCP-NC protocol was presented in 2008 [9] which successfully implemented the network coding into TCP with minor changes to the protocol stack. The key idea was adding a network coding layer between transport layer and IP layer to masks packet losses from congestion algorithm. In fact, masking losses from TCP was considered earlier by using link layer retransmission [10]. Yet it has been noted in [11] and [12] that the interaction between link layer retransmission and TCP retransmission is complicated and the performance may suffer due to independent retransmission protocols at different layers. TCP-NC modifies the ACK echo system, and brings in a new notion "see packets". The biggest difference compared to the original mechanism is that under network coding the receiver does not obtain original packets of the message, but linear combinations that are then decoded to get the original message once enough such combinations have arrived. The "see packets" notion can perfectly adapt to these changes, and before explain the notion, they introduce a definition that will be useful throughout the paper [3]. In NC-TCP, packets are treated as vectors over a finite field F_q of size q. All the discussion here is with respect to a single source that generates a stream of packets. The k^{th} packet that the source generates is said to have an index k and is denoted as $\mathbf{p_k}$. As a result, a node is said to have seen a packet $\mathbf{p_k}$ if it has enough information to compute a linear combination of the form ($\mathbf{p_k} + \mathbf{q}$), where $\mathbf{q} = \sum_{l>k} \alpha_l \mathbf{p}_l$, with $\alpha_l \in F_q$ for all $l > k$. Thus, $\mathbf{q}$ is a linear combination involving packets with indices larger than k. To conclude, there are two main differences in our scheme. First, whenever the source is allowed to transmit, it sends a random linear combination of all packets in the congestion window. Second, the receiver acknowledges every sequence number of seen packet. Additionally it brings in a redundancy factor R, which is used for masking the packet loss. For example, if the loss rate is about 10%, then the optimal R equals to 1/ (1-10%) ≈ 1.11, this means the sender will send one more redundant packet every ten packets NC-TCP achieves a goal,

 ISBN: 978-1-61208-166-3

that is, planting network coding into TCP properly. In some communication networks, where the loss rate is roughly constant, via setting the redundancy factor *R* to an optimal number, a better throughput can got compared to the original TCP.

TCP-DNC protocol is presented in 2009 [13], which focuses on reducing the decoding delay and redundancy by adding some information in packet's header. It inherits the coding approach and "see packets" notion presented by the NC-TCP scheme [9]. In the receiver, the TCP-DNC brings in a new factor "loss", which indicates how many combinations the sender needs to retransmit enable the receiver decode all the combinations it has received. The "loss" factor will be sent back to the sender, and the sender uses this factor to decide how many redundant packets should be sent and how many original packets should be coded. By doing this, this new scheme can avoid the retransmission of the useless redundant packets, and due to sending redundancy packets coded by the appropriate number of original packets, it significantly reduces the decoding delay and improves the performance of the networks.

We propose a new scheme named SANC-TCP protocol, which mainly optimizes the scheme based on NC-TCP. To be concrete, in NC-TCP, the redundancy factor *R* is constant, we need to know the loss rate of the network circumstance, and set *R* to the optimal number. However, when the system is under lossy networks, especially wireless network where the loss rate is not constant, the constant redundancy factor *R* may cause problems, either sending bunches of useless redundancy packets or being not able to mask the packets loss. Both will impair the performance of the network. As a result, we need to find a scheme to adjust *R* adaptively to the real system, aiming to better the utility of the networks and decrease the retransmission of the useless redundant packets. Our new scheme, SANC-TCP, adds some feedback information in the ACK header, to indicate the current network state, thus enable the sender to dynamically change the *R* according to the real system.

In Section I, we get an overview of the NC-TCP scheme, and describe the basic theory for background; In Section Ⅱ, we introduce the arithmetic of the Active-R NC-TCP Protocol; In Section Ⅲ, we prove the fairness of our new scheme compared to the old one; In Section Ⅳ, we demonstrate the effectiveness of the new protocol, and show its advantage over the old others. Finally, in Section Ⅴ, we make a succinct conclusion of the whole article.

II. SELF-ADAPTIVE NC-TCP PROTOCOL

In this section, we will describe the basic ideas of the SANC-TCP protocol and the arithmetic for dynamically adjusting the redundancy factor *R*.

The SANC-TCP aims to better the utilization of channels by dynamically adjusting the redundancy factor *R* in unknown lossy networks. To fulfill this target, we make some minor changes to the original protocol stack via adding two variables to the ACK header, i.e., *loss* and *echo_pktID*. At the receiver, the difference which is indicated by *loss* between the largest packet index in the coefficient vector and the number of seen packets implies the number of packets the sender needs to retransmit. Another variable *echo_pktID* indicates the packet ID of which packet generates this ACK. At the sender, once it receives a new ACK, it checks the *echo_pktID*. When *echo_pktID* = 10 or *echo_pktID* > 10 for the first time, it starts to adjust the *R*. First, the sender picks up the variable *loss* from the header of ACK, then figures out the value of *diff_loss_new*, that is, *diff_loss_new* = *loss* – *loss_old*, where diff_loss_new indicates the effect of the redundant packets that sended in the latest turn. The new *R* = 1 + (*diff_loss_new*/10)*2 + *diff_loss_old*/10 , and the original *diff_loss_old* = 0. The current variables *echo_pktID*, *diff_loss_new*, *loss* and *R*, that is *W* = *echo_pktID*, *diff_loss_old* = *diff_loss_new*, *loss_old* = *loss*, *R_old* = *R* is also recorded; For example, if the sender receives a new ACK, and the *echo_pktID* in the ACK equals to 10, then the sender decides to adjust the *R*. Suppose one packet lose among the first ten packets, then the *loss_new* = 1. Meanwhile, the *loss_old* = 0 originally. So, the new redundancy factor *R* = 1 + (1/10)*2 + 0 = 1 + 0.1*2 + 0 = 1.2. After this, the sender keeps checking *echo_pktID* from every new ACK. When *echo_pktID* = *W* + 10**R*, or *echo_pktID* > *W* + 10**R* for the first time, adjust the *R*. At this time, *R* = *R_old* + (*diff_loss_new*/10)*2 + *diff_loss_old*/10. Record the current variables *echo_pktID*, *diff_loss_new*, *loss* and *R*, that is *W* = e *echo_pktID*, *diff_loss_old* = *diff_loss_new*, *loss_old* = *loss*, *R_old* = *R*. If the result of *R* is smaller than 1, set the *R* to 1. For example, if the previous *echo_pktID* = 200, *R* = 1.1, and the sender did not receive ACK which contain *echo_pktID* = 211 or *echo_pktID* = 212. Then, when it receives the ACK whose *echo_pktID* = 213, the sender starts to adjust the *R*. At the receiver when this ACK is generated, if l*oss_new* = 20, *loss_old* = 19, then *diff_loss* = 20 – 19 = 1; This time, at the sender, if *diff_loss_old* = 1, then *R* = 1.1 + (1/10)*2 + 1/10 = 1.4.

To make it clear, we independently describe the actions which are taken on the sender and receiver side. Provided we have introduced a network coding layer between the transport layer and the IP layer.

(1) Receiver side: The receiver side algorithm has to respond to two types of events – the arrival of a packet from the sender, and the arrival of ACKs from the TCP sink.

1. Wait state: If any of the following events occurs, respond as follows; else wait.
2. ACK arrives from TCP sink: If the ACK is a control packet for connection management, deliver it to the IP layer and return to the wait state; else, ignore the ACK.
3. Packet arrives from the sender side:
   a) Remove the network coding header and retrieve the coding vector.
   b) Add the coding vector as a new row to the existing coding coefficient matrix, and perform Gaussian elimination to update the set of seen packets.

 ISBN: 978-1-61208-166-3

c) Add the payload to the decoding buffer. Perform the operations corresponding to the Gaussian elimination, on the buffer contents. If any packet gets decoded in the process, deliver it to the TCP sink and remove it from the buffer.

d) Count the variable *loss* which equals to the difference between the largest packet index in the coefficient vector and the number of seen packets; Pick up the value of *pktID* from the received packet's header, record it to *echo_pktID*.

e) Generate a new ACK with sequence number equals to that of the oldest unseen packets and add two variables loss and echo_pktID to the ACK header.

(2) Sender side: On the sender side, the algorithm again has to respond to two types of events – the arrival of a packet from the sender TCP, and the arrival of an ACK from the receiver via IP.

1. Set NUM to 0;

2. Wait state: If any of the following events occurs, respond as follows; else wait.

3. Packet arrives from TCP sender:

a) If the packet is a control packet used for connection management, deliver it to the IP layer and return to wait state.

b) If packet is not already in the coding window, add it to the coding window.

c) Set NUM = NUM + *R*. (*R* = redundancy factor)

d) Repeat the following ⌊NUM⌋NUM] times:

i) Generate a random linear combination of the packets in the coding window.

ii) Add the network coding header specifying the set of packets in the coding window and the coefficients used for the random linear combination. Add the variable *pktID* to the network coding header.

iii) Deliver the packet to the IP layer.

e) Set NUM:= fraction part of NUM.

f) Return to the wait state.

4. ACK arrives from receiver:

a) Pick up the variable *echo_pktID*, to judge if it is time to adjust the value of *R*.

i) If echo_pktID = W + 10**R*_old or echo_pktID > W + 10**R*_old for the first time, start to reset the value of *R*.

Ⅰ) Extravagate the value of *loss* from the ACK header, *diff_loss_new* = *loss* – *loss_old*;

Ⅱ) Then *R_new* = *R_old* + 2*(*diff_loss_new*/10) + *diff_loss_old*/10.

Ⅲ) Record variables, such as, *R_old* = *R_new*; *diff_loss_old* = *diff_loss_new*; *loss_old* = *loss*; *W* = *W* + 10**R_new*.

ii) else doing nothing and move to state b).

b) Remove the ACKed packet from the coding buffer and hand over the ACK to the TCP sender.

Following the approach above, the sender adjusts the redundancy factor *R* from time to time, thus to dynamically change the R according to the real system. The algorithm to adjust the redundancy factor *R* in the sender is showed in Figure 1.

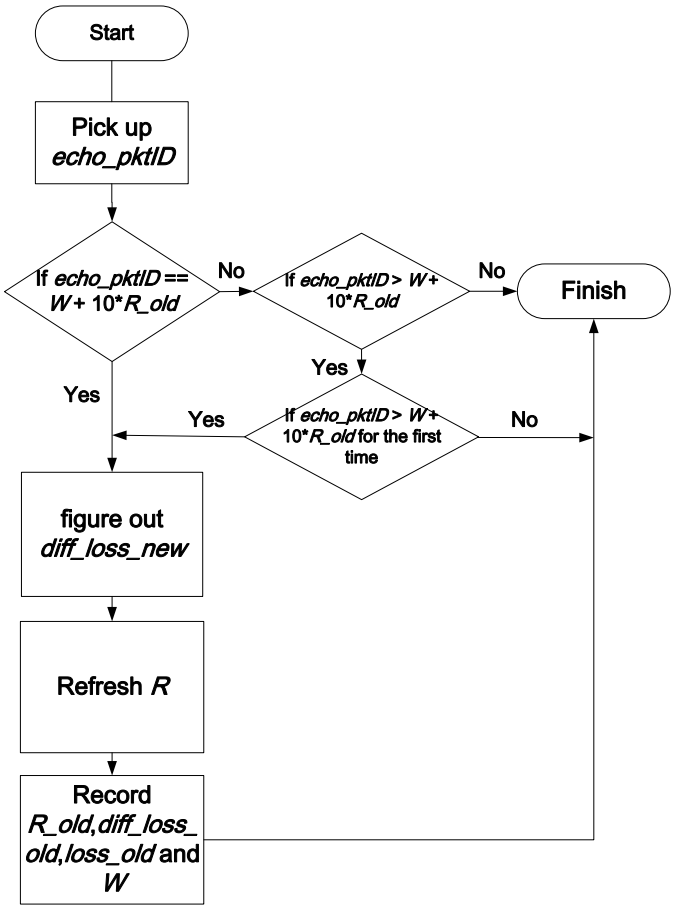

Figure 1. The algorithm to adjust the redundancy factor *R* in the sender

III. FAIRNESS OF THE NEW PROTOCOL

We use the network simulator-2 [14] to access the performance of different protocols in network. The topology for all the simulations is a tandem network consisting of 8 hops (hence 9 nodes), shown in Figure 2.

In this system, there are two flows generated by two FTP applications. One is from node 0 to node 7, and the other is from node 1 to node 8. They will compete for the intermediate channels and nodes. All the channels have a bandwidth of 1 Mbps, and a propagation delay of 10ms. The buffer size on the channel is set to 200. The TCP receive window size is set to 40

 ISBN: 978-1-61208-166-3

packets, and the packet size is 1000 bytes. The Vegas parameters are chosen to be α =28, β = 30, γ =2.

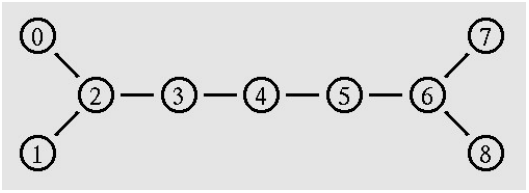

Figure 2. A tandem network consisting of 8 hops

By fairness, we mean that if two or more similar flows compete for the same channel, they must receive an approximately equal share of the channel bandwidth. In addition, this must not depend on the order in which the flows join in the network. It is well known that depending on the value chosen for α and β, TCP-Vegas could be unfair to an existing connection when a new connection enters the bottleneck link. In our simulation, we first choose a certain value of α and β (in this case, α =28, β =30) that allows fair sharing of bandwidth when two TCP-Vegas flows without our modification. Then, we choose the same value of α and β, and figure out the fairness characteristic under three different situations:

Situation 1: a TCP-Vegas flow competes with an SANC-TCP flow.

Situation 2: an SANC-TCP flow competes with another SANC-TCP flow.

Situation 3: five SANC-TCP flows compete with each other.

In Situation 1, the loss rate is set to 0%, and the SANC-TCP flow starts at 0.5s while TCP-Vegas flow is 200s later. The SANC-TCP flow ends at 800.5s, while TCP-Vegas flow ends at 1000.5s. The system is simulated for 1100s. The current throughput is calculated at intervals of 2.5s. The evolution of the two flows' throughput over time is shown in Figure 3 which indicates, when TCP-Vegas flow joins in the channel, it quickly shares an equal amount of bandwidth of the channel with the previous SANC-TCP flows, thus proving the fairness of new SANC-TCP.

In Situation 2, the loss rate is set to 0%, and one of the SANC-TCP flows start at 0.5s while the other one is 300s later, and they both end at 1000.5s. The system is simulated for 1100s. The current throughput is calculated at intervals of 2.5s. The evolution of the two flows' throughput over time is shown in Figure 4 which is similar to Figure 3. The latter flow quickly shares an equal amount of bandwidth of the channel with the former one after it joins in the system. This also demonstrates that the fairness of SANC-TCP.

In Situation 3, five different SANC-TCP flows start independently at 0.5s, 100.5s, 200.5s, 300.5s, 400.5s. According to the result showed in Figure 5, when each flow comes into the channel, they quickly share equal amount of the channel's bandwidth compared to others, and thus, it proves that the SANC-TCP is strictly fair.

IV. EFFECTIVENESS OF THE NEW PROTOCOL

Backed-up by the simulation, we now try to prove that our new protocol SANC-TCP has a better throughput rate and utilization of the channels under unknown lossy channels, compared to NC-TCP. In part A, we compare the throughput rate and the utility of three different protocols TCP-Vegas, NC-TCP, SANC-TCP under the same lossy channels, with different loss rate every measured time. For the NC-TCP, we set the redundancy factor at the optimum value corresponding to each loss rate. In part B, we set the redundancy factor to a constant number 1.11. The loss rate of the channels is varied from 10% to 45%. We will compare the throughput rate and the utilization of the channels between NC-TCP flow and SANC-TCP flow. Finally, in part C, we consider a situation called bursty loss situation, where there will be a sudden large loss rate for a short time in the system. We compare the performance of three different protocol flows under bursty loss situation.

Fairness

The topology setup is identical to that used in the fairness simulation, except that now we only use one FTP flow, which is from node 0 to node 7. We set the same loss rate on the channels between node 2 and node 6. For example, if we set loss rate to 0.1 on every channels between node 2 and node 6, we get the total loss rate $1-(1-0.1)^4 = 0.3439$. When simulation starts, the FTP0 flow starts at 0.5s, and the intermediate channels start to lose packet in a certain rate at 0.6s. The simulation time is set to 1000s.

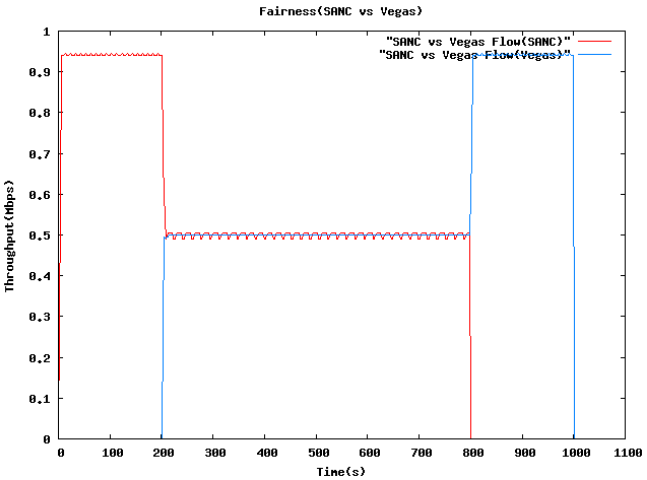

Figure 3. A TCP-Vegas flow compete with an SANC-TCP flow

 ISBN: 978-1-61208-166-3

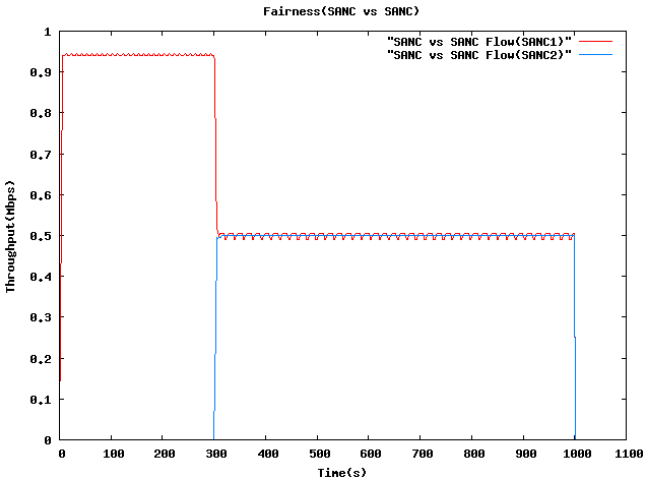

Figure 4. an SANC-TCP flow compete with another SANC-TCP flow

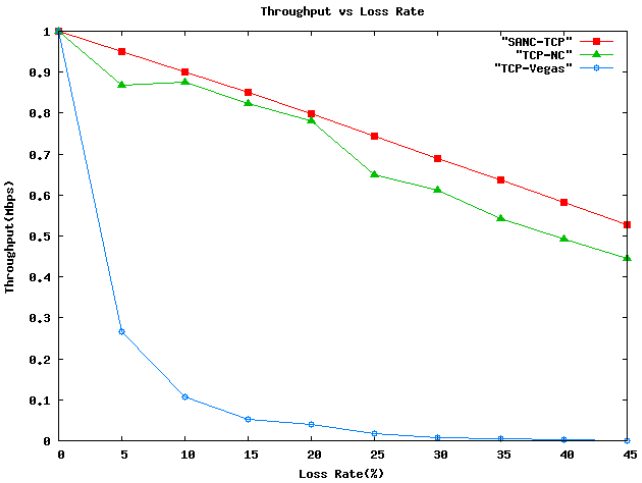

Figure 6. The throughputs of three different flows

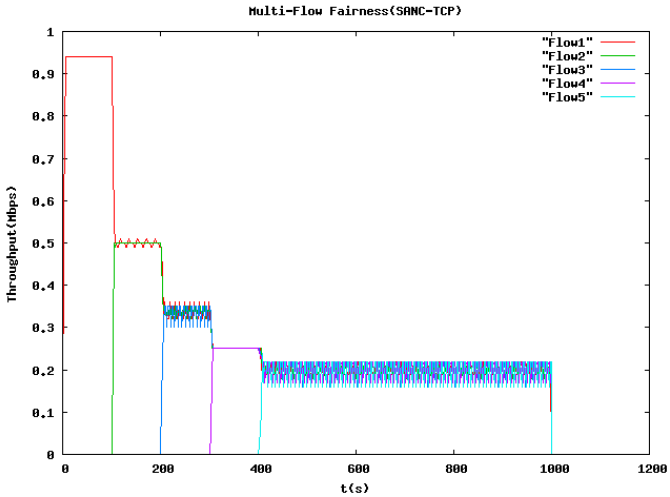

Figure 5. Five SANC-TCP flows compete with each other

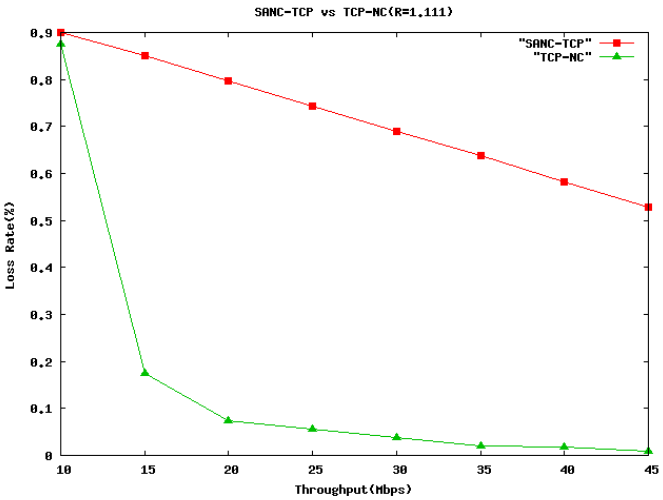

Figure 7. The throughputs of TCP-NC flow and SANC-TCP flow

Simulation results are shown in Figure 6. The X-axis represents the various loss rate, and the Y-axis represents the throughput rate corresponding to different loss rate. As we set the link capacity to 1 Mbps, the Y-axis can also represents the utilization of the channels. The blue line is referred to TCP-Vegas, the green line is referred to NC-TCP and the red is to SANC-TCP. To emphasize, under every different loss rate, the redundancy factor *R* is set to the optimal value. For example, if the loss rate is 20%, then the *R* is set to be 1 / (1-0.2) = 1.25. Figure 5 shows that, when the loss rate is 0%, the throughput of all three protocols almost reaches the optimal value 1Mbps. However, as the loss rate becomes larger, the throughput of TCP-Vegas descends drastically, while both NC-TCP and SANC-TCP are close to the theoretical value of maximum utilization of channels. For example, theoretical value of maximum utility of channels is 1Mbps * (1 – 20%) = 0.8Mbps when loss rate is set to 20%, as we can see NC-TCP and SANC-TCP are both close to it from Figure 6.

Effectiveness

In order to compare the throughput and utilization of the channel between NC-TCP flow and SANC-TCP flow under various loss rate, we set the R to 1.11 in NC-TCP flow case, while the other parameters of simulation environment are totally the same.

As is shown in Figure 7, The X-axis represents the different loss rate which is varied from 10% to 45%, and the Y-axis represents the throughput rate corresponding to different loss rate which can also be understood as utilization of the channel. The green line is referred to NC-TCP flow and the red one is to SANC-TCP flow. When the loss rate is 10%, NC-TCP flow requires high throughput with R equals to 1.11 as the optimal value and approximates SANC-TCP flow. However, as the loss rate becomes larger, the throughput of NC-TCP case descends drastically because it cannot mask the packet loss with the *R* value sticking to 1.11. Adversely, the throughput of SANC-TCP flow is close to theoretical value under every loss rate. For example, the theoretical value of maximum utility of the channel is 1Mbps * (1 – 30%) =

 ISBN: 978-1-61208-166-3

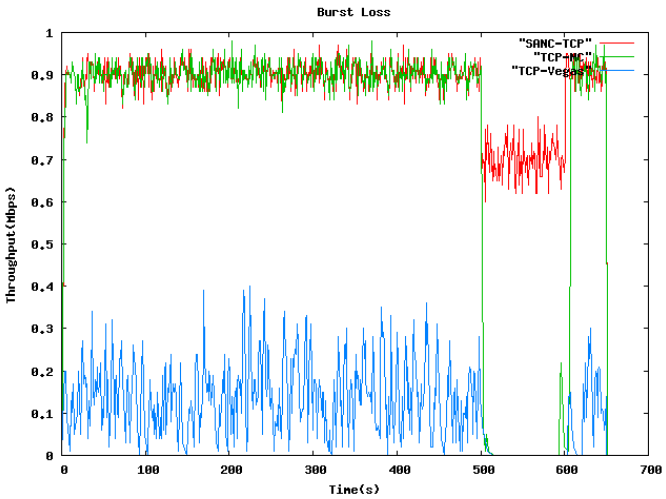

Figure 8. Bursty loss situations

0.7Mbps when loss rate is set to 30%, and the SANC-TCP flow is close to it. In addition, given *R* equals to 1.11, lots of packets will be sent unnecessarily which leads to low performance if there are more than one flow in the network when the loss rate is smaller than 10%. SANC-TCP adjusts *R* to the practical condition and maintains it at the optimal state which avoids wasting bandwidth.

Bursty

In real wireless networks, the loss rate is affected by various reasons. Sudden large loss, we call it bursty loss, is one of the phenomena that occur in the system. To evaluate the performance of the three different protocol flows under bursty loss situation, we set a circumstance where the loss rate of the system is kept as 10%, except for the time from 500s to 600s, the loss rate is changed to 30%. We use the same topology as Part A and Part B.

As is shown in Figure 8, the X-axis represents the simulation time, and the Y-axis represents the throughput or the utilization of the channel. The blue line is referred to TCP-Vegas flow, the green line is referred to NC-TCP flow whose redundant factor R is set to the optimal value of 1.11 and the red line is referred to SANC-TCP flow. During the time when the loss rate is kept in 10%, the NC-TCP flow and SANC-TCP flow can both nearly reach the theoretical value of the throughput. However, when the time comes to 500s, the loss rate is suddenly changed to 30% until 600s. According to Figure 8, NC-TCP flow suffers a lot during the time from 500s to 600s, the throughput is almost drop to 0. Comparably, the SANC-TCP shows its robustness to the bursty loss, and maintains the theoretical value of throughput during 500s to 600s.

V. CONCLUSION AND FUTURE WORKS

Network coding is an effective tool to fight against non-congestion losses. However, due to the different loss rate in different period of time in wireless networks, the NC-TCP with constant redundancy factor *R* cannot effectively solve the non-congestion losses problem by retransmitting redundant packets. In this work, we propose a new approach to dynamically adjust *R* to the real networks. As the redundancy factor *R* is no longer constant, we can change it according to the real current circumstance, thus better the performance under lossy networks where the loss rate is not constant.

For future work, we plan to focus on the encoding and decoding delay problem which stands in the way for the network coding technology to implement in the real system.

VI. ACKNOWLEDGEMENT

This work has been supported by 973 Program 2012CB315904; NSFC 60872010; SZJC201005260234A; SZZD201006110044A; GDNSF No.9150 6420 1000 031.

REFERENCES

[15]

[1] R. Ahlswede, N. Cai, S. Y. Li, and R. W. Yeung, "Network Information Flow," IEEE Trans. On Information Theory, vol. 46, pp. 1204-1216, 2000.

[2] T. Ho, "Networking from a network coding perspective," phD Thesis, Massachusetts Institute of Technology, Dept. of EECS, May 2004.

[3] J. K. Sundararajan, D. Shah, and M. Medard, "ARQ for network coding," in IEEE ISIT 2008, Toronto, Canada, Jul, 2008.

[4] S. Katti, H. Rahul, W. Hu, Databi, M. Mcdard, and J. Crowcrofg, "XORs in the Air: Practical Wireless Network Coding," in IEEE/ACM Transactions on Networking, 16(3): 497-510, June 2008.

[5] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding Aninstant primer," ACM Computer Communication Review, Jan. 2006.

[6] C. Fragouli, D. S. Lun, M. Medard, and P. Pakzad, "On feedback for network coding," in Proc. of 2007 Conference on Information Sciences and Systems (CISS 2007).

[7] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in Proc. of Allerton Conference on Communication, Control, and Computing, 2003.

[8] S. Chachulski, M. ennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in Proc. of ACM SIGCOMM 2007, August 2007.

[9] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP," in IEEE INFOCOM, Apr 2009.

[10] S. Paul, E. Ayanoglu, T. F. L. Porta, K.-W. H. Chen, K. E. Sabnani, and R. D. Gitlin, "An asymmetric protocol for digital cellular communications," in Proceedings of INFOCOM, 1995.

[11] A. DeSimone, M. C. Chuah, and O.-C. Yue, "Throughput performance of transport-layer protocols over wireless LANs," IEEE Global Telecommunications Conference (GLOBECOM '93), pp. 542-549 Vol.1, 1993.

[12] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," ACM Wireless Networks, vol. 1, no. 4, pp. 469-481, December 1995.

[13] J. Chen, W. Tan, and L. X. Liu, "Towards zero loss for TCP in wireless networks," in Performance Computing and Communications Conference(IPCCC), 2009 IEEE 28th international.

[14] "ns-2 Network Simulator," http://www.isi.edu/nsnam/ (Sep 20th, 2011)

 ISBN: 978-1-61208-166-3