An Adaptive Redundancy Scheme for TCP with Network Coding

Hamlet Jesse Medina-Ruiz, Michel Kieffer, Beatrice Pesquet-Popescu

An Adaptive Redundancy Scheme for TCP with Network Coding

Hamlet Medina Ruiz*†, Michel Kieffer*†, and Béatrice Pesquet-Popescu†

*L2S - CNRS - SUPELEC - Univ Paris-Sud

3 rue Joliot-Curie, 91192 Gif-sur-Yvette, France

†Institut Télécom, Télécom ParisTech and CNRS LTCI

Signal and Image Processing Department,

46 rue Barrault, 75634 Paris Cedex 13, France

‡LRI - CNRS - Univ Paris-Sud

91405 Orsay, France

Abstract

To address the inability of the standard TCP protocol to distinguish between congestive losses and random packet losses produced by the noisy channel, in this paper we propose an adaptive algorithm to dynamically adjust the redundancy factor R of the TPC/NC protocol proposed by Sundararajan et al. by adding some additional functionalities to the original network coding layer. We define a loss differentiation scheme to adjust R, based on the Vegas Loss Predictor and the collective feedback information of ACKs and duplicates ACKs, which are both indicators of the network condition. In that way the source adjusts intelligently R based on the network conditions, avoiding unnecessary TCP rate reduction due to packet losses induced by the noisy channel and preventing the network from entering in a congestion state. Our TPC/NC implementation with our adaptive scheme is full-duplex, and can manage multiple TCP connections simultaneously. Simulation results over realistic network scenarios using

OPNET Modeler show that our adaptive scheme in conjunction with the standard TCP/NC produces better TCP-throughputs than the standard TCP/NC, TCP-Reno, TPC New Reno, and TCP Reno with SACKS.

I. INTRODUCTION

Network Coding (NC) is a technique that offers a lot of potential in today's applications by allowing nodes to send out packets that are linear combinations of previously received information, instead of delivering the information to their destination in the standard store-and-forward-manner [1], [2]. The main benefits of NC are the potential throughput improvements and a high degree of robustness, which is translated into loss resilience [3]. Despite all the benefits of NC , we still lack of many real network coding implementations.

In [4] Sundararajan et al. propose a new TCP-friendly protocol that interfaces network coding with TCP by introducing a new coding layer in the TCP/IP stack between TCP and IP, with the idea of using the benefits of NC and to improve TCP throughput. In this layer TCP segments are encoded at the sender and decoded at the receiver. In [5] the authors also describe several important practical aspects of incorporating NC with TCP-Reno, as the TCP-compatible sliding window code, and new rules for acknowledging bytes to the TCP layer. Redundancy of information added by the sender is used to cover non congestion losses from TCP congestion control, and simulation results in [4], [5] show an improvement in TCP throughput for a constant value for R.

Setting R without having an estimation of the network conditions, as the packet loss rate of the channel and the congestion in the network at any point in time, could lead to TCP times outs when R is too low to match TPC's sending rate to the rate at which data is received at the receiver because of the losses, or, on the other hand, if R is too large, the losses can be

recovered, the congestion in the network could increase and the TCP throughput decrease . As the correct approach is to increase the transmission rate when the channel is dropping packets [6], [7], we would like to increase R in these situations to compensate losses, as for example when TCP-Reno probing for available bandwidth believes that it has reached the network congestion state when it wrongly identifies the reception of duplicates ACKs (DUPACKS) caused by the noisy channel as a sign of congestion, resulting in an unnecessary reduction of the transmission rate, and in a similar manner decrease R when the reception of DUPACKS are really caused by a congested network.

We propose a new adaptive algorithm to adjust R that uses the feedback information available at hand, namely packet drops and packets backlogged based on the Vegas Loss Differentiation algorithm [8] to define a loss differentiation scheme, and based on this information adjust R, helping the TCP congestion control mechanism to properly reduce its congestion windows only when there exists congestion in the network, and reacts to packets drops in a smooth manner.

This paper evaluates the performance of TCP/NC with adaptive R vs the standard TCP/NC protocol, as well as with the standard TCP congestion control flavors, namely TCP-Reno, TCP-NewReno, and TCP Reno with SACKS. In what follows, the TCP/NC protocol is briefly described in Section II, details about the adaptive algorithm are given in Section III, simulation results are described in Section IV, and some conclusions and future research directions are drawn in Section V.

II. Principles of TCP/NC Protocol

TCP/NC protocol introduces a network coding layer between TCP and IP in the protocol stack. The encoder at the sender buffers packets generated by TCP in the coding buffer, and for every arrival from TCP, it transmits R random linear combinations of the most W recently

arrived packets in the coding buffer. To convey information about the linear combination, the NC header is added to the coded packet, which contains information about the coefficients used to mix the packets, the starting and last byte SEQ of each packet involved in the mix, and the first byte that has not been ACKed. The original packets are retained in the coding buffer until an appropriate TCP ACK arrives from the receiver side.

On the receiver side, upon receiving a new linear combination, the decoder places it in a decoding buffer, appends the corresponding coefficients to the decoding matrix (which contains the coefficients used to mix already seen packets received), and performs Gaussian elimination. This process helps identify the "newly seen" packet (if it exists). Then the receiver sends a TCP cumulative ACK to the sender requesting the first unseen packet in order, like in conventional TCP. The Gaussian elimination may result in a new packet being decoded. In this case the decoder delivers this packet to the receiver TCP. Any ACK generated by the receiver TCP is suppressed and not sent to the sender. These ACKs may be used for managing the decoding buffer. An important point is that the introduction of the new NC layer does not require any change in the basic features of TCP. For more detail, the reader is referred to [5].

III. Adaptive TCP/NC Protocol

A. Adaptive redundancy factor

The main idea behind coding is to mask the losses on the channel from TCP. In other words, we wish to correct the losses without relying on the ACK's.

Linear combinations generated using NC should be able to correct any loss of the W packets involved in the linear combinations that have not yet been ACKed. R linear combinations on average are sent to the IP layer to compensate for the loss rate of the channel, and to match TCP's sending rate to the rate at which data are actually sent to the receiver. If there is too little

redundancy, we will not receive enough linear combinations to decode (losses are not masked from TCP), leading to timeouts and consequently low throughput.

On the other hand, if the redundancy is too large, losses are recovered, but TCP congestion window advances smoothly, the throughput is reduced due to low code rate and the congestion increases.

Assuming a constant redundancy factor equal to the reciprocal of the probability of successful reception is impractical, as this probability changes over time due to changes on bit error probability of the channel and the congestion in the network.

The main idea of this paper is to adjust dynamically R to an optimal rate based on the actual network conditions, which can be detected by collecting the feedback information available. Our objective is to adjust R in such a way that TCP congestion window slows down its sending rate vigorously to alleviate congestion only when we have an indication of a really congested network, and in the worst case to maintain its rate when loss packets are due to the noisy channel . When the channel is dropping packets we try harder by increasing the redundancy to mask the channel losses from TCP, and we decrease the redundancy when congestion is present, allowing TCP to sense the congestion and reduce its rate to alleviate it. To accomplish this objective, we make use of the TCP Vegas loss predictor to detect when congestion is present in the network, in conjunction with the number of duplicate ACK's generated by the receiver.

1) Vegas loss predictor: TCP Vegas uses a loss predictor to decide whether the network is congested or not based on rate estimators [8], [9], [10] , estimating the backlogged packets in the buffer of the bottleneck link. We implement the Vegas Loss Predictor at the NC layer to know when the network experiences congestion and to adjust R in a more intelligent way. As in [8], we set $BaseRTT$ to be the RTT of a segment when the connection is not congested (in

practice the minimum of all measured round trip times, which is reset after a time out) as a reference to derive the expected throughput the network can accommodate.

The expected throughput is given by $window/BaseRTT$, where $window$ is the difference between the SEQs of the newest and oldest byte involved in the linear combinations currently in transit.

The measured throughput is given by $windowAcked/RTT$, where $windowAcked$ is the number of bytes ACKed (seen by the receiver NC layer) during the last RTT, and RTT is the sample RTT of the received segment.

To estimate the cause of the packet losses, the parameter qv is calculated as

$$qv = \left(\frac{window}{BaseRTT} - \frac{windowAcked}{RTT} \right) BaseRTT \tag{1}$$

and compared with two thresholds α and β to know the network state. If $qv \geq \beta$, the network is considered congested (too much backlogged), if $qv \leq \alpha$, possible losses due to random errors (little backlogged), and finally if $\alpha < qv < \beta$, the predictor assumes that the network state is the same as in the previous estimation. In our test the best performance was achieved with the common values used in Unix OS flavor implementations, $\alpha = 1$ and $\beta = 3$.

A detailed accuracy analysis of the Vegas predictor and other loss differentiation algorithms can be found in [9].

2) New Adaptive algorithm for R: In this algorithm we define a loss differentiation scheme with the help of the Vegas Loss Predictor [8], and the number of DUPACKS generated by the receiver to detect if a packet loss is due to the noisy channel or to a congested network by comparing these two measures.

We propose an implicit approach to adjust R based on this differentiation scheme, allowing the sender to react intelligently to the cause of the packet loss, due to either congestion or random

errors, and therefore prevents the network from entering in congestion or an unnecessary decrease of the congestion window.

Basically, each time an ACK is received, the quantity in (1) is evaluated. If the value of (1) is below the threshold that indicates a congested network and some DUPACKS have been received, it is because packets are being dropped by the noisy channel and not by buffer overflows in the router queues'. In these situations the correct approach is to increase R to mask the losses. On the other hand, if a network congestion is detected using (1), we decrease R, allowing TCP to also reduce its rate and alleviate the congestion. Initially we set R to a value that takes into account the loss in throughput due to the finiteness of the field to reduce the number of DUPACKS due to dependent linear combinations received at the receiver, and when congestion is detected R is set again to this value. The algorithm is specified below using pseudo-code.

In order to successfully implement Algorithm 1 in the NC layer, some minor improvements are added to the standard TCP/NC protocol. Timer functions regarding the RTT calculation, and variables to keep track of the number of loss events based on the reception of DUPACKS, the value of $window$ and $windowAcked$ in (1) have been implemented.

Our implementation of the TCP/NC protocol with the proposed algorithm is full-duplex, and also supports multiple instances, one for each TCP connection present [11], which is identified by the TCP socket. To put it clear, the algorithms for the source and receiver side are shown is pseudo code below. This specification assumes only one instance and one-way TCP flow. The source side algorithm has essentially to respond to two types of events, the arrival of a packet from the source TCP, and the arrival of an ACK from the receiver via IP. On the receiver side, the algorithm has to respond to the arrival of a packet from the source, and the arrival of ACKs from the TCP sink.

Algorithm 1 Calculate qv

$\alpha = 1$, $\beta = 3$, $R = 1.1$ {Initialization}

Each time an ACK is received: Call timer routines and compute qv (1)

if $qv \geq \beta$ **then**

$R \leftarrow 1.1$

else if $qv \geq \alpha$ **then**

if $cntDupAck \geq 2$ **then**

$R \leftarrow R + 2/10$;

else if $cntDupAck == 1$ **then**

$R \leftarrow R + 1/10$

else

$R \leftarrow R - 1/10$

end if

else

$R \leftarrow 1.1$

end if

IV. Simulation Scenario And Results

A. Simulated Topology

We use a client/server OPNET model to evaluate and compare the performance of TCP/NC with adaptive redundancy vs the standard TCP/NC protocol, TCP-Reno, TCP-NewReno and TCP-Reno with SACK congestion control mechanisms [12], [13], [14], in a scenario when channel and congestion losses are simultaneously present.

Algorithm 2 Source Side Algorithm

1 $NUM \leftarrow 0$.

2 Initialize Algorithm 1 parameters

3 Wait state: If any of the events occurs at points 4 and 5, respond as follows; else, wait

4 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 the packet is not already in the coding window, add it to the coding window.

c) $NUM \leftarrow NUM + R$

d) Repeat the following $\lfloor NUM \rfloor$ 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

iii) Deliver the packet to the IP layer.

iv) Update the oldest and newest byte SEQs of the linear combinations in flight.

v) $NUM \leftarrow NUM - \lfloor NUM \rfloor$

vi) Return to the wait state.

5 ACK arrives from receiver:

a) Call Algorithm 1

b) Remove the ACKed packet from the coding buffer

c) Hand over the ACK to the TCP sender.

Algorithm 3 Receiver Side Algorithm

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 source side.

a) Remove the NC header and retrieve the coding vector.

b) Call Algorithm 1

c) Add the coding vector to the coding coefficient matrix, and perform Gauss Jordan elimination to update the set of seen packets.

d) Add the payload to the decoding buffer. Perform the operations corresponding to the Gauss Jordan 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.

e) Generate a new TCP ACK with sequence number equal to that of the oldest unseen packet.

The simulated network topology is presented in Figure 1. It consists of two 100BaseT subnet LANs, subnet A and B, respectively connected via gateway routers through a WAN Network represented by the CoreNetwork.

The two gateway routers in subnet A and B are connected using the *Point To Point Protocol* (PPP) and *Digital Signal 1* (DS1, 1.544 Mbps) links, see Figures 2 and 3, respectively.

Subnet A consists of one FTP server running TCP/NC, one Video Streaming Server, and one VoIP workstation that are connected to the gateway router by 100 Mbps links (see Figure 2). In Subnet B there are the FTP client running TCP/NC, the video client, and one VoIP workstation

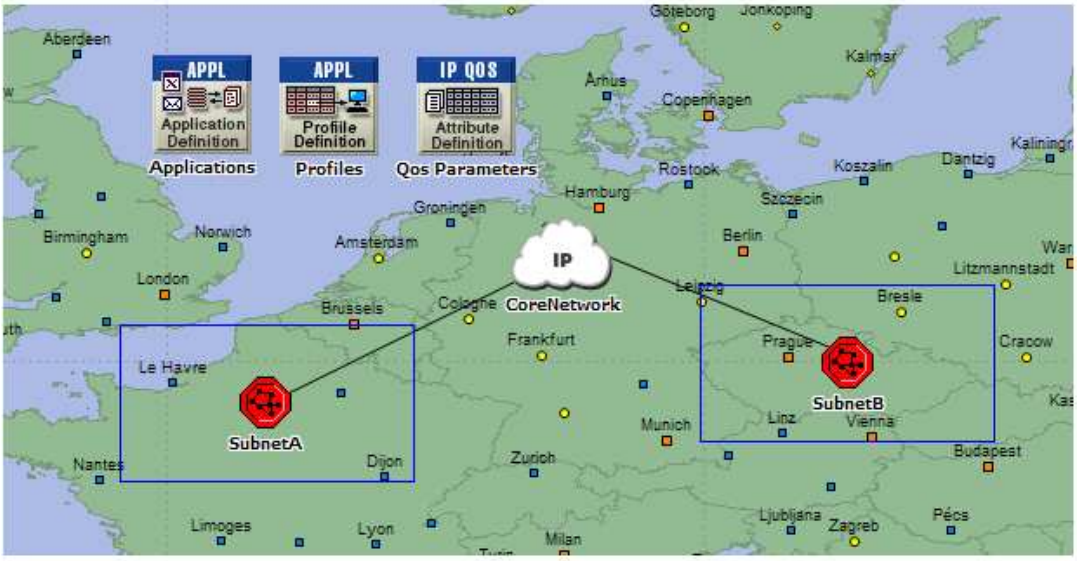

Fig. 1. Topology of the wide Area Network

also connected to the gateway router by 100 Mbps links (see Figure 3).

B. Simulation Setup

In this scenario the network carries traffic from three different applications, FTP, Video, and VoIP. VoIP traffic has PCM quality speech, and Video traffic is a low resolution video streaming, with a frame inter-arrival time of 10 fps and frame size of 128×120 pixels.

Each type of traffic is marked with a priority using the IP Type of Service (ToS) field. FTP traffic is configured to have the best effort service, VoIP and Video streaming traffic with Interactive Voice service and streaming multimedia service, respectively.

Video Streaming and VoIP use RTP protocols with normal TCP Reno for the control channel.

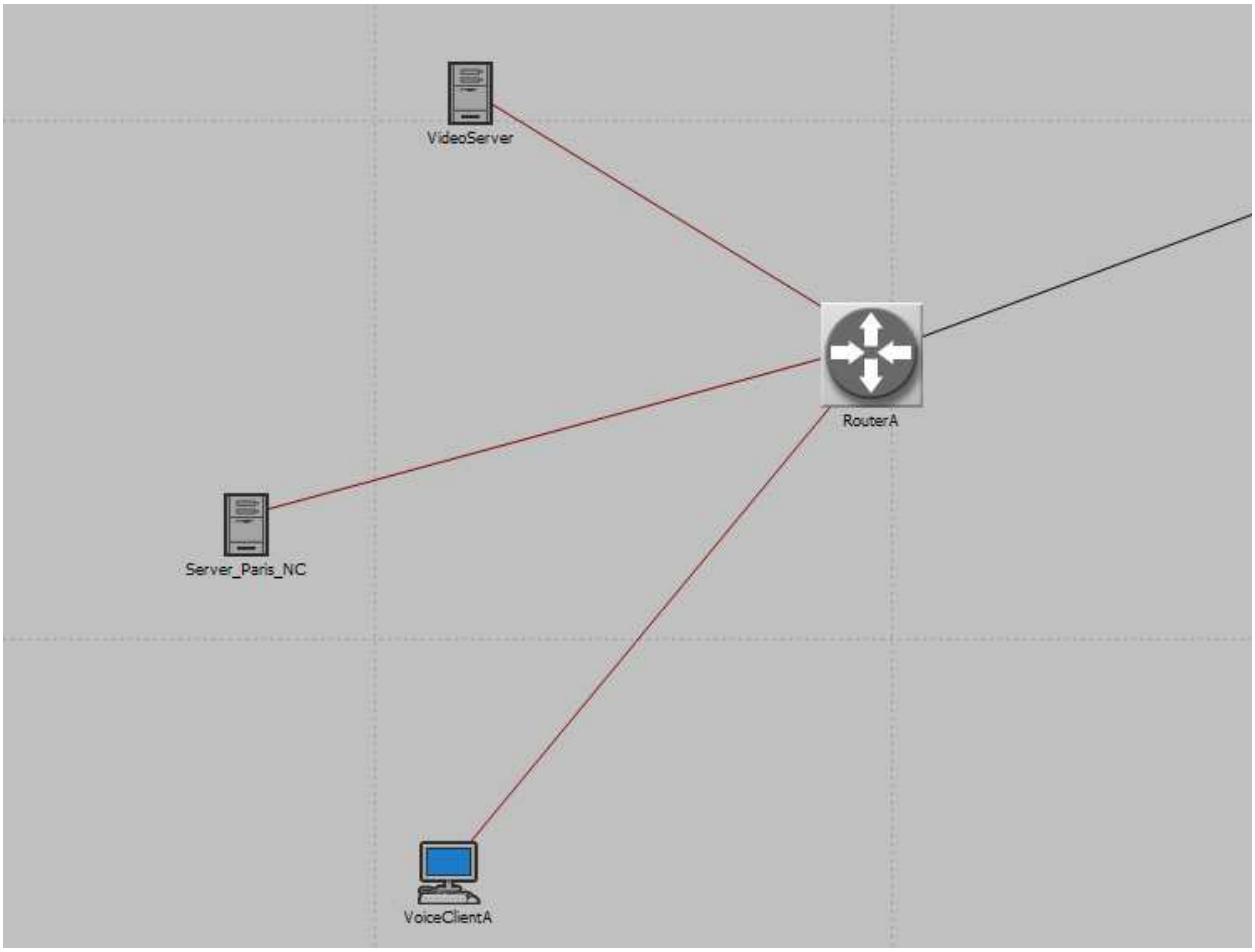

Fig. 2. Servers and VoIP client located in subnetA

A 10 MB file is transferred using the FTP application from the NC Server located in subnet A to the NC Client located in subnet B.

In this scenario the FTP application begins the connection at time $t = 10\,s$ and finishes when the file is totally transferred or by the end of the simulation time which is of 15 minutes. The VoIP and Video Streaming traffic are active between $60\,s < t < 110\,s$ and between $80\,s < t < 110\,s$, respectively, generating congestion in the network during that period of time. In the simulation we have also modeled the loss in throughput due to the finiteness of the field (in our case, its size is 256).

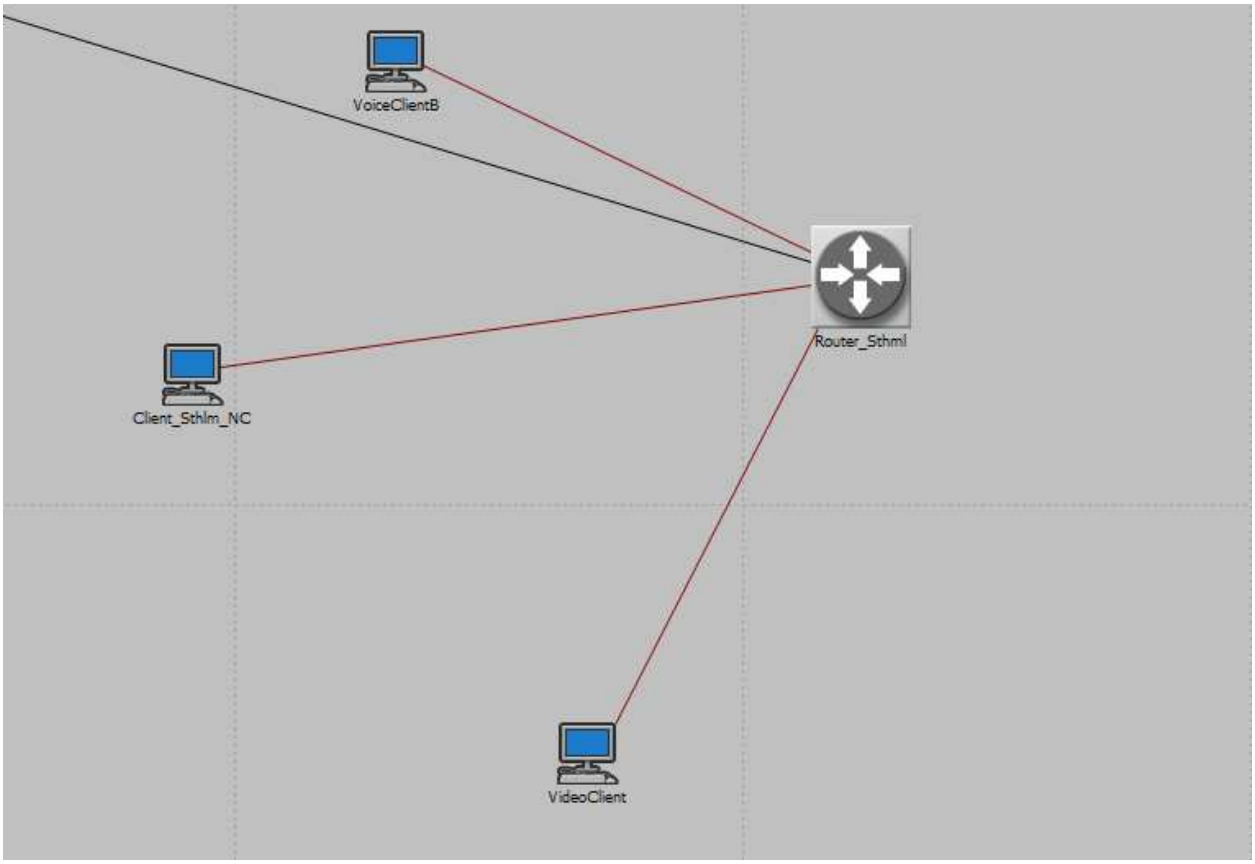

Fig. 3. Clients Located in Subnet B

We assume for the rest of the paper a simulation time of 15 minutes, a latency equal to 100ms on the DS1 link, a FIFO queueing discipline with a buffer space equal to 500 in each router interface queue (value that is much larger than the bandwidth-delay product) , and the TCP simulation parameters shown in Table I. The parameters in Table I are used for all the TCP flavors.

Performance parameters as the congestion window size, file download response time and goodput are studied through the simulations.

TABLE I

OPNET TCP SIMULATION PARAMETERS

TCP Parameters	Value
MSS (bytes)	1500
Slow start initial count (MSS)	2
Receive buffer size (bytes)	65535
Maximum ACK segment	2
Initial RTO (sec)	3.0
Minimum RTO (sec)	1.0
Maximum RTO (sec)	64
RTT gain	0.125
Deviation gain	0.25
RTT deviation coefficient	4.0
Duplicate ACK threshold	3

C. Simulation Results

Using the simulation setup explained in section IV-B, we first study the effect of the parameter R on the goodput of the standard TCP/NC protocol and our TCP/NC approach as a function of the loss rate of links (the loss rate being the probability of a packet getting lost on the link). Packets in the network are subject to these losses in the forward and the reverse direction. Figure 4 shows the plot of the TCP/NC goodput as a function of the loss rate when the parameter R is varied between 1.0 and 1.5, as well as when R is dynamic. From this plot, it is clear the important role played by parameter R. We can see that for the range of R and loss rates tested, when R is varied over time according to Algorithm 1, we outperform the standard TCP/NC .

In order to compare the performance obtained with our approach with the standard TCP congestion mechanisms, in Figure 5 is plotted the TCP goodput for TCP-Reno, TCP-NewReno,

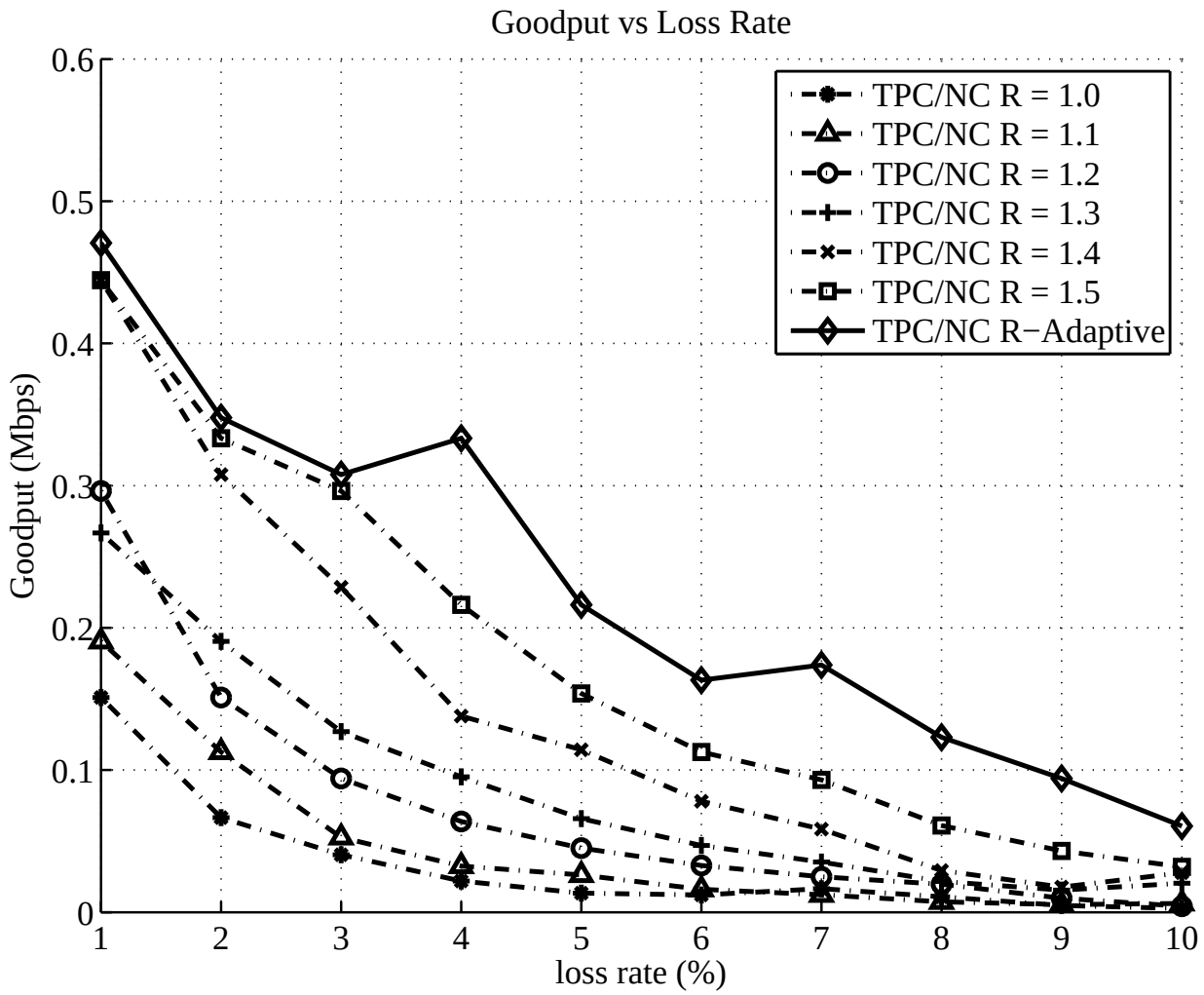

Fig. 4. Goodput vs Loss Rate for TCP/NC and TCP/NC with adaptive R

and TCP-SACKS congestion control for the same range values of the loss rate. We clearly appreciate the improvement obtained in goodput with our approach.

In Table II we also compare the download time response experienced by the FTP client since the FTP connection is established, in the case when the loss rate is equal to 3%.

In Figure 6 is shown the evolution of the SEQs sent by the FTP/NC Server as a function of time for a loss rate equal to 5% for different values for the parameter R. We can observe

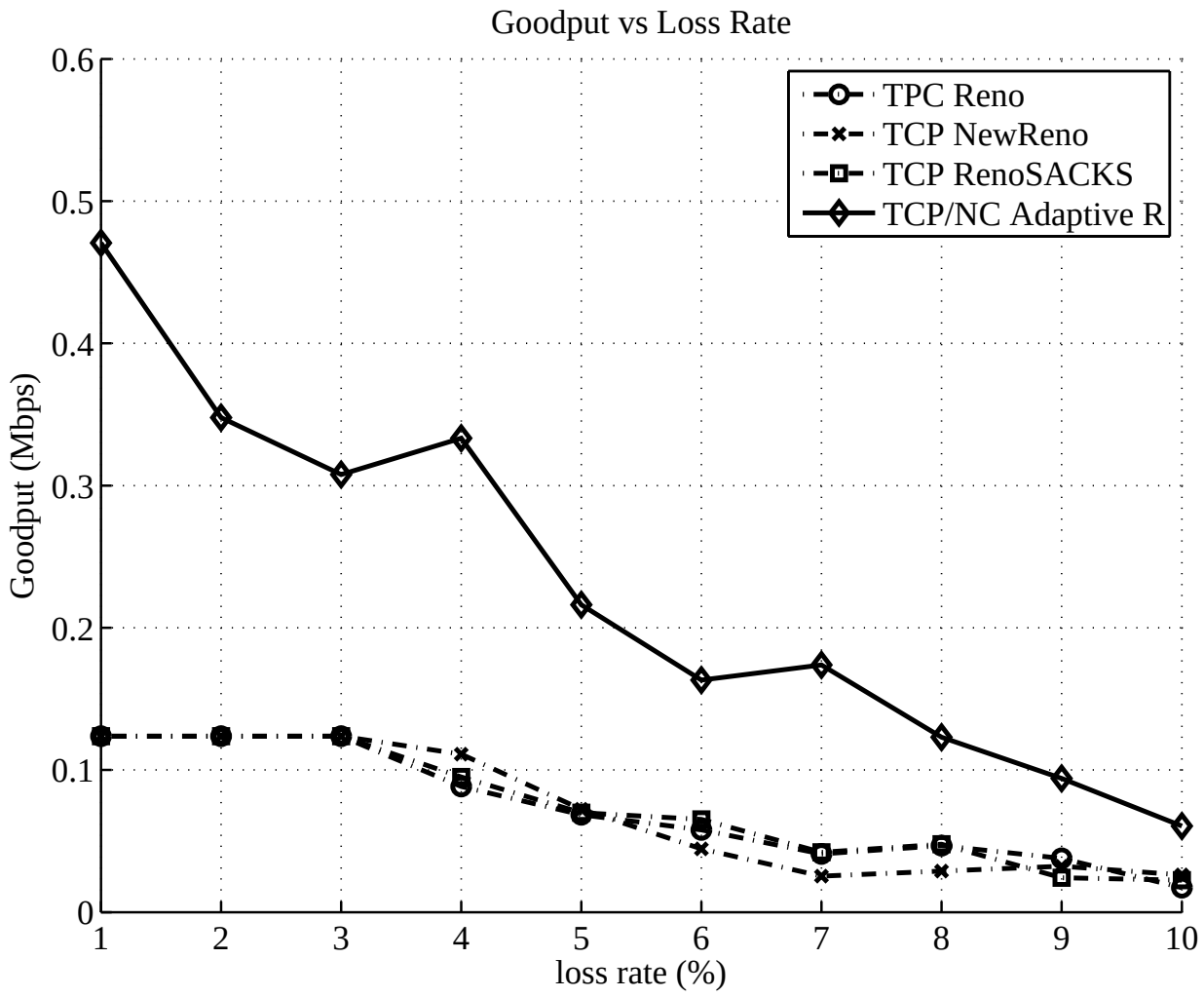

Fig. 5. Goodput vs Loss Rate for TCP/NC with adaptive R and different TCP Flavors

TABLE II

DOWNLOAD FTP RESPONSE TIME FOR A 10MB FILE AND LOSS PROBABILITY EQUAL TO 3%

TCP Variants	Download Response Time (s)
Reno	511.70
NewReno	523.85
RenoSACKs	447.30
TCP/NC Adaptive R	232.02

that for R values between 1.0 and 1.5 we have more almost flat segments in the curve, which represent drops seen by the TCP congestion window. Their cause is either errors on the noisy channel or network congestion. With our approach, we identify the underline cause of the loss, which allows us to take the correct action over R.

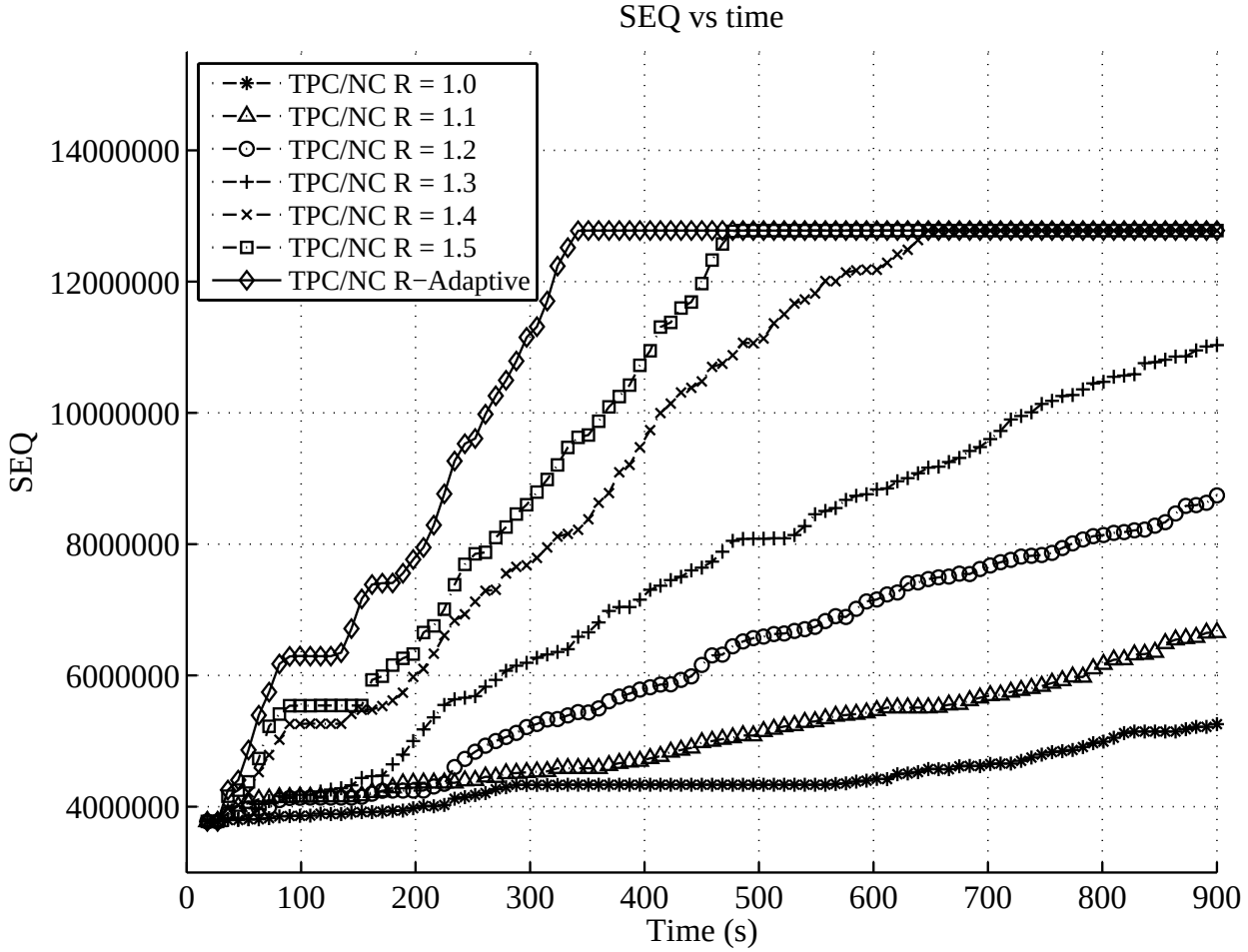

Fig. 6. Sent SEQ vs Time for TCP/NC and TCP/NC with adaptive R

Our algorithm also adapts to the reduction of the throughput due to the finiteness of the size of the field. Even without errors in the channel and no congestion, there is a non null probability that the TCP sender rate does not match the TCP receiver rate and we can not

decode, leading to time outs. If the receiver receives dependent linear combinations, it generates DUPACKS for the last unseen packet to the source. When the source receives the DUPACKS, it runs Algorithm 1 and verifies that the received DUPACKS are caused by the noisy channel (dependent linear combinations received with no congestion) and will increase R to mask losses and avoid unnecessary reduction in the TCP congestion window.

Finally, in Figure 7 we show the evolution of the TCP congestion window ($cwnd$ in bytes) as a function of the redundancy factor R during first 450s of simulations for a loss rate of 3%, using adaptive R and contant R equal to 1.3. We observe that at the moments when $cwnd$ curve presents almost flat segments there are peaks in redundancy curve. This represents events when DUPACKS are received, caused either by the noisy channel or by the finiteness of the field. It is also important to notice, that our scheme adapts well to a burst of errors, which can be viewed by comparing the oscillating behavior of the $cwnd$ for $R = 1.3$ and the smoothness of the $cwnd$ using our scheme in some time intervals, without having a too large value for R. Increasing R when DUPACKS are received by non congestion causes prevents the unnecessary reduction of TCP sending rate.

V. Conclusions and perspectives

This paper considers the issue of adapting over time the redundancy factor R in the original TCP/NC protocol.

We proposed a new adaptive algorithm to dynamically adjust R using packet drops and packets backlogged information based on received DUPACKS and the Vegas Loss Differentiation algorithm, respectively, to discover the underline cause of a packet loss in the network and take the correct action over R.

Simulation results have shown an improvement in the goodput and download FTP time

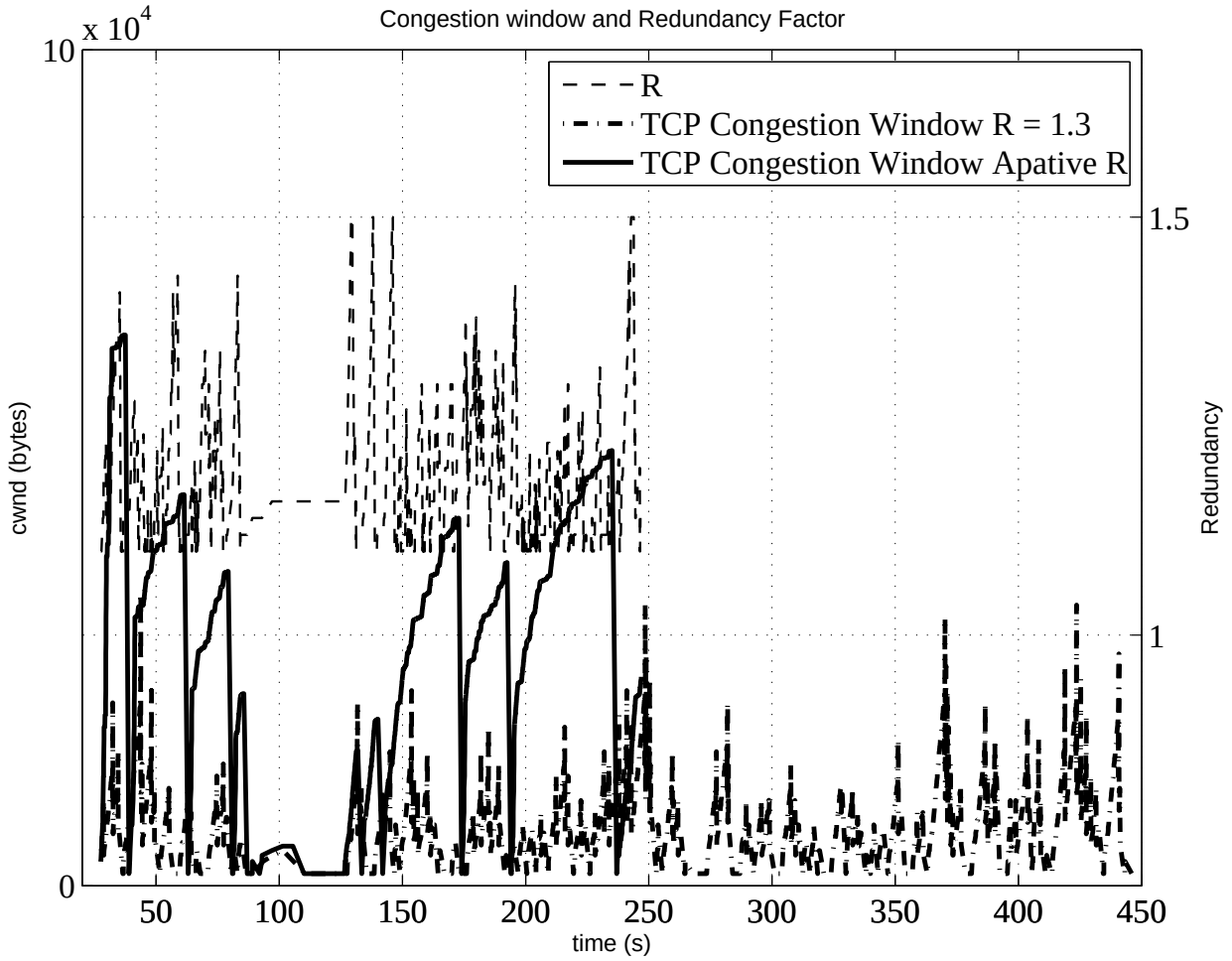

Fig. 7. Evolution of the redundancy and the congestion window for 1 % loss probability

response of TCP/NC with adaptive R over the standard TPC/NC protocol and different standard flavors of TCP congestion control mechanism.

In future work, simulations over heterogenous networks, a mathematical proof of the stability of the TCP/NC protocol with our adaptive redundancy scheme, as well as a characterization of the throughput behavior as a function of the loss rate will be considered.

ACKNOWLEDGMENTS

This work has been partly supported by DIM-LSC NC2, DIM-LSC SWAN. Michel Kieffer was partly supported by the Institut Universitaire de France.

REFERENCES

[1] R. Ahlswede, N. Cai, S. yen Robert Li, and R. W. Yeung, "Network information flow," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] S. yen Robert Li, S. Member, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, 2003.

[3] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 63–68, Jan. 2006. [Online]. Available: http://doi.acm.org/10.1145/1111322.1111337

[4] J. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *INFOCOM 2009, IEEE*, april 2009, pp. 280–288.

[5] J. K. Sundararajan, S. Jakubczak, M. Médard, M. Mitzenmacher, and J. Barros, "Interfacing network coding with TCP: an implementation," 08 2009. [Online]. Available: http://arxiv.org/abs/0908.1564v1

[6] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[7] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach, 3rd edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[8] L. Brakmo and L. Peterson, "TCP vegas: end to end congestion avoidance on a global Internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, oct 1995.

[9] F. Martignon and L. Fratta, "Loss differentiation schemes for TCP over wireless networks," pp. 586–599, 2005.

[10] Y. Tian, K. Xu, and N. Ansari, "TCP in wireless environments: problems and solutions," *Communications Magazine, IEEE*, vol. 43, no. 3, pp. S27–S32, march 2005.

[11] W. R. Stevens and G. R. Wright, *TCP/IP illustrated (vol. 2): the implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[12] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[13] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno and sack TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, Jul. 1996. [Online]. Available: http://doi.acm.org/10.1145/235160.235162

[14] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," United States, 1999.