TCP Congestion Avoidance using Proportional plus Derivative Control

Dirceu Cavendish, Hikaru Kuwahara, Kazumi Kumazoe, Masato Tsuru, Yuji Oie
Department of Computer Science and Electronics
Kyushu Institute of Technology
Fukuoka, Japan 810-0004
Email: {cavendish,kuma,tsuru,oie}@ndrc.kyutech.ac.jp, kuwahara@infonet.cse.kyutech.ac.jp

***Abstract*—We introduce a transmission control protocol with a delay based congestion avoidance that utilizes a proportional plus derivative controller. The derivative part of the controller is shown to be well suited to effectively control TCP sessions, given the relative shallow buffer space of network elements. We demonstrate the competitive performance of the protocol via open source based network experiments over a research network and the Internet.**

***Keywords*—high speed networks; TCP congestion avoidance; Packet retransmissions; capacity estimation; path bottleneck; Proportional plus derivative controller.**

I. Introduction

Recent advances in TCP protocols have departed from window transmission regulation based on binary information into multi-bit information [2], [9]. Rich feedback information indeed holds the promise of better regulating packet transmission by reducing traffic oscillations typical of binary based control mechanisms. Moreover, recent advances in TCP flow probing have made available path estimators that may be useful for regulating TCP traffic transmission [10].

In our prior work, we have introduced a delay based TCP window flow control mechanism that uses path capacity and storage estimation, called Capacity and Congestion Probing - CCP [5]. The idea is to estimate bottleneck capacity and path storage space, and regulate the congestion window size using a proportional controller. We have shown that CCP has competitive performance as compared with widely known TCP protocols, such as Reno and Cubic, outperforming them in the presence of random packet loss scenarios such as in wireless bottlenecks.

However, quite often path storage space is limited, due to shallowness of network elements' buffers. In such cases, a large proportional gain parameter is needed to increase throughput performance, which increases the protocol aggressiveness. Motivated by the shallowness of buffers in the Internet network elements, in this paper we propose to add a derivative component to the proportional controller used to regulate TCP congestion window. The idea is to react to the derivative of the available path storage space, in addition to the absolute storage space value. When buffers are limited, a derivative component should help regulate better traffic input into the TCP session.

In this work, our contributions are as follows. We show the feasibility of a window flow control mechanism based on a proportional plus derivative controller, based on non intrusive path capacity and buffer storage estimators. As previously, we use a control theoretic framework that ensures stability regardless of session characteristics (path capacity and round trip time) and cross traffic activity; we design a congestion window regulation scheme within the framework of a TCP protocol, called TCP-Capacity and Congestion Proportional plus Derivative - CCPD; we demonstrate TCP-CCPD performance via a comprehensive set of open source based transpacific network experiments. The material is organized as follows. Related work discussion is provided on Section II. Section III introduces the modeling and control theoretic approach of the window regulation scheme, whereas section IV reviews the path estimators used to implement the window regulation. Section V describes the TCP-CCPD protocol, and section VI addresses its performance evaluation. Section VII addresses directions we are pursuing as follow up to this work.

II. Related Work

TCP protocols fall into two categories, delay and loss based. Advanced loss based TCP protocols, such as HS-TCP and Scalable TCP use packet loss as primary congestion indication signal, performing window regulation as $w_k = f(w_{k-1})$, being ack reception paced. Most f functions follow an Additive Increase Multiplicative Decrease strategy, with various increase and decrease parameters. TCP NewReno and Cubic are examples of AIMD strategies. Delay based TCP protocols, on the other hand, use queue delay information as the congestion indication signal, increasing/decreasing the window if the delay is small/large, respectively. Examples of these are TCP-Vegas and FAST TCP. CCP and CCPD fall into the second category, delay based protocols.

Although TCP-Vegas relies on estimates of round trip propagation delay, its window adjustment function is of the form $w_k = f(w_{k-1})$, paced by one rtt per adjustment [2]. TCP-Vegas aims at keeping a small number of packets buffered in the routers along the path. Fast TCP, on the other hand, tracks both minimum and average rtt values of a session, in order to update the window, still via a $w_k = f(w_{k-1})$ function. Although both of these algorithms can be tuned to convergence [6], parameter tuning depends on particular characteristics of each session, such as propagation delays and link speeds [7]. In contrast, CCP and CCPD both rely on a technique for dead-time delay systems to ensure stability regardless the characteristics of a TCP session [12]. This allows us to fine tune the algorithm's parameters to trade throughput for packet loss, without the danger of driving the system to instability. In addition, CCPD derivative component allows a faster reaction to transients when buffer space is limited. Unique to CCP and CCPD window control is that they do not follow a $w_k = f(w_{k-1})$ control law, which, together with built in stability mechanism, allows the protocols to be responsive to cross traffic disturbances at widely different network scenarios and traffic conditions.

III. Proportional plus Derivative Window Congestion Control

We make use of the control theoretic approach of [3] to design CCPD protocol. In what follows, we limit ourselves to summarize the results needed for CCPD protocol design, augmenting them with a derivative component.

A. Network and Queue Models

The network consists of $N = \{1, 2, \ldots, n\}$ nodes and $L = \{1, 2, \ldots, l\}$ links. Each link i is characterized by: transmission capacity $c_i = 1/t_i$ (segments/sec); propagation delay td_i. The network traffic is generated by source/destination pairs (S, D), where $S, D \in N$. To each (S, D) session, there is a number of TCP sessions associated, each of which having a fixed path $p(S, D)$ over which all segments of a given session travel. Each source is characterized by its maximum transmission speed, $c_s = 1/t_s$, dictated by its network interface card.

We further assume that each switch maintains a single queue for all sessions exiting a given outgoing link. Let $x_{i,j}(t)$ be the occupancy at time t of the queue associated with link i and $session_j$, and $B_{i,j}$ a corresponding buffer size.

For the model of the dynamic behavior of each queue, we assume a deterministic fluid model approximation of segment flow [11]. Considering the queue associated with the TCP session TCP_j at link i, if the level of queue occupancy at time t is $x_{i,j}(t)$, its input rate $u_{i,j}(t)$, and cross traffic $d_{i,j}(t)$, a fluid model of the queue system is given by:

$$\frac{dx_{i,j}(t)}{dt} = \begin{cases} u_{i,j}(t) + d_{i,j}(t) & \text{if } x_{i,j} > 0 \\ \max(0, u_{i,j}(t) + d_{i,j}(t)) & \text{if } x_{i,j} = 0 \end{cases} \tag{1}$$

B. Window Control Model

In order to control the queue level $x(t)$ for a specific session, we use a proportional plus derivative controller. Letting B be the size of the bottleneck buffer, we compute the difference between the buffer size and the current queue level $x(t)$. This difference, the error $e(t)$, is multiplied by a positive gain K_p, so that $K_p e(t)$ is the regulated input rate of the TCP source. In addition, a difference between the current queue level $x(t)$ and the previously received queue level $x(t-t_p)$ is multiplied by a positive parameter K_d and added to the input rate.

Figure 1 depicts the block diagram of the continuous time model of a TCP session. T_{ff} denotes the propagation delay from the flow controlled source to the bottleneck queue, the most congested intermediate node, whereas T_{fb} is the propagation delay incurred by traffic carrying feedback information from the intermediate node to the destination node, plus the delay incurred from the destination node back to the source node. Therefore, $RTT = T_{ff} + T_{fb}$ is the round trip propagation delay incurred by a segment carrying feedback information. In Figure 1, a generic proportional controller $Kp^*(t)$ is depicted, rather than a simple proportional controller Kp. In addition, a derivative component $Kd^*(t)$ is also depicted, acting on variations of the input signal. The cross traffic rate is $d(t)$, hence variations in $d(t)$ represent cross traffic variations, or disturbances, accounting for cross traffic impact on a TCP session, whereas B is the bottleneck queue size.

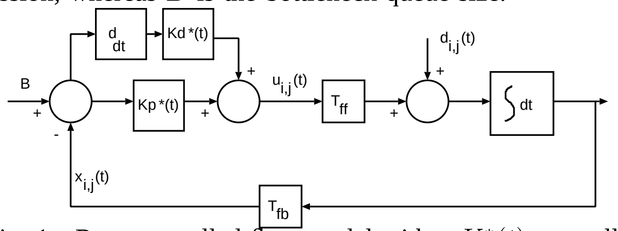


Fig. 1: Rate controlled flow model with a $K^*(t)$ controller

We use a Smith Predictor [12] to ensure stability in large bandwidth delay product paths regardless of the proportional and derivative gain parameters used. Therefore, we substitute the controllers $Kp^*(t)$ and $Kd^*(t)$ in Fig. 1 with controllers such that the resultant system has a delay free feedback loop in cascade with pure delays [3]. The proportional plus derivative controller plus the feedback loop predictor gives rise to the following input rate control equation:

$$\begin{aligned} u(t) = {} & Kd[B - x(t) - in\_flight\_traffic(t)] + \\ & Kd\frac{d}{dt}[B - x(t) - in\_flight\_traffic(t)] \end{aligned} \tag{2}$$

Eq. 2 implements a proportional plus derivative control action with the difference that the actual queue level is increased by the amount of data transmitted during the last round trip delay ($in\_flight\_traffic(t)$). Finally, a discretization of the above continuous system yields the window adjustment equation as:

$$\begin{aligned} w_k = {} & Kp[B - x_k - in\_flight\_segs_k] + \\ & \frac{Kd}{t_k - t_{k-1}}[x_{k-1} + in\_flight\_segs_{k-1} + \\ & -x_k - in\_flight\_segs_k] \end{aligned} \tag{3}$$

where x_k the buffer level at discrete time k, and $in\_flight\_segs_k$ the number of segments transmitted in the last round trip delay.

Although our main interest is in TCP sender window regulation, Eq. 3 can also be used to account for retransmissions of lost segments. This is because the window regulation scheme itself does not distinguish between fresh and retransmitted segments, as long as they are both accounted for by in_flight_segs. Notice that Eq. 3 is not a pure recurrent equation, of the form $w_k = f(w_{k-1})$. Therefore, a current window size is not solely dependent on the value of the previous window size, as in most TCP protocols. [1] In addition, theoretically Eq. 3 would need to be recomputed at a given minimum frequency,

[1]To be precise, w_k depends on the values of all w_i within a full round trip time, due to in_flight_segs term.

which would require timers associated with transport sockets that require locks for safe access. Instead, a TCP congestion avoidance implementation of Eq. 3 does not require timers because, as soon as a control window worth of packets is transmitted, no new packets are allowed into the network until an acknowledgement is received. At that time, we can recompute Eq. 3 appropriately.

We use Kp parameter for throughput regulation, whereas Kd is used to quickly respond to variations in path buffer space. That is, although large values of Kp increase session throughput, segment loss may be experienced, depending on cross traffic behavior. Segment losses can be mitigated by responding quickly to queue build ups via Kd parameter of the derivative component of the controller.

Summarizing, window regulation through Eq. 3 allows for a trade off between segment loss and throughput via parameter Kp. Hence, tuning of Kp can be exercised for traffic engineering purposes. For paths through network elements with shallow buffers, Kd is tuned so as to provide quick response to variations in available space. To end this section, we mention that the control window prescribed by Eq. 3 allows the TCP sender to send a certain number of segments at line speed, if wished, without impairing controllability or stability of the session. This is indeed the typical behavior of a TCP session. Moreover, each TCP session sees its own buffer size B and a buffer level caused by its own traffic plus cross traffic on its path, caused by other TCP sessions and UDP traffic crossing its path.

IV. Path estimators

Eq. 3 requires estimators for bottleneck buffer size $\hat{B}$ and buffer level $\hat{x}$. As the estimators used in this paper are a more accurate version of the estimators used in [5], in this section we simply summarize the description of the estimators, for completeness.

A. Capacity estimation

The capacity estimation method of our choice is based on packet pair dispersion [10] techniques. The idea is to measure dispersion of the delay of packet pairs sent back to back . If both probing packets of size MSS of a packet pair sample do not suffer any queueing delay, and the dispersion between them is d, the slowest link capacity can be estimated as:

$$\hat{C} = \frac{MSS}{d} \tag{4}$$

The capacity estimation method is described in detail in [4]. In this paper, we implemented a version of this method with high resolution clocks, which allows us more precision, as well as the bottleneck capacity estimation of a wider range of speeds. Fig. 2 reports capacity estimation results for short and long rtt path scenarios, using two K_p parameter values. We can see that capacity estimation accuracy does not depend on the rtt nor the set of parameters used.

B. Buffer size estimation

Let rtt_{max} and rtt_{min} be the maximum and minimum rtts experienced by segments of a given session. A reasonable estimator for the bottleneck buffer size would then be:

$$\hat{B} = \hat{C} * (rtt_{max} - rtt_{min}) \tag{5}$$

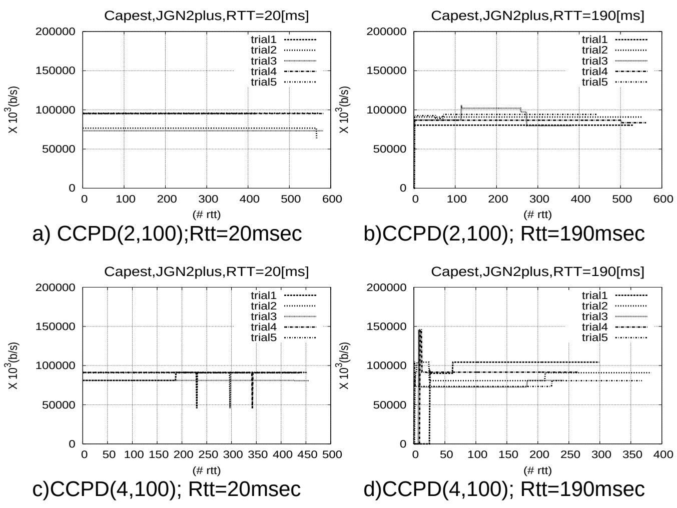


Fig. 2: Capacity estimation

However, a precise estimation would be achieved only when the bottleneck buffer is full, so that rtt_{max} is the rtt of the segment once stored at the last buffer slot of the buffer, otherwise the estimator will underestimate the buffer size. On the other hand, rtt_{min} may represent more than pure propagation delays, if during the estimation period the bottleneck buffer never empties. In this case, however, one may argue that the extra buffer space, taken by a persistent traffic, is never available anyways, so this extra space is perceived by a TCP session as an additional propagation delay. Fig. 3 report buffer size estimation results for short and long rtt path scenarios, using two K_p parameter values. Buffer size estimation accuracy does not depend on the parameter values used. However, long rtt sessions result in larger buffer size estimation, as expected.

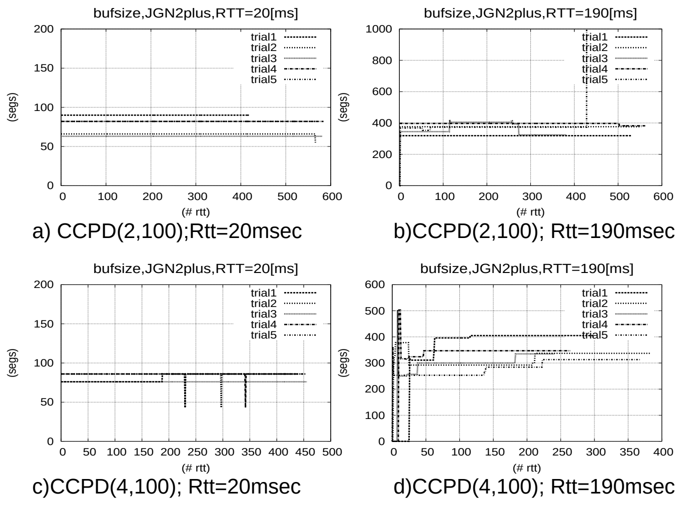


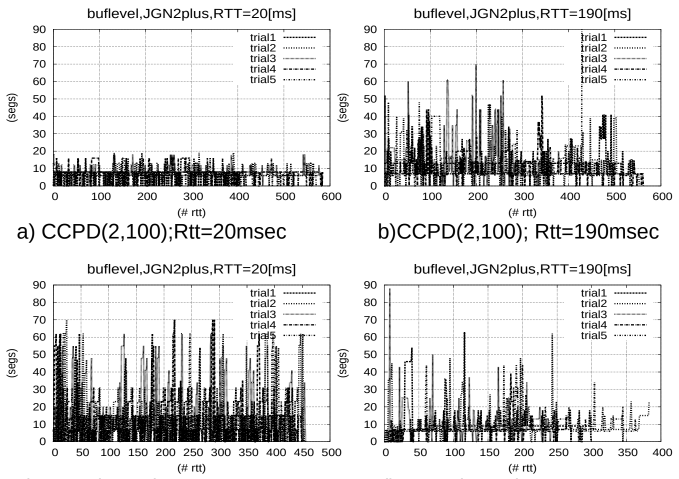
Fig. 3: Buffer Size Estimation

C. Buffer level estimation

If one tracks each segment rtt, the current buffer level $x(t)$ can be estimated by $\hat{x}(t) = (rtt(t) - rtt_{min}) \times \hat{C}$. Since sample rtt values typically include high frequency variations, a smoothed average rtt value $rtt_s(t)$ is used instead, so:

$$\hat{x}(t) = \hat{C} * (rtt_s(t) - rtt_{min}) \quad (6)$$

Fig. 4 report buffer level estimation results for short and long rtt path scenarios, using two K_p parameter values. Buffer level estimation does depend on both the parameter values used, as well as the session rtt.



a) CCPD(2,100);Rtt=20msec b)CCPD(2,100); Rtt=190msec

c)CCPD(4,100); Rtt=20msec d)CCPD(4,100); Rtt=190msec

Fig. 4: Buffer Level Estimation

V. TCP-CCPD PROTOCOL

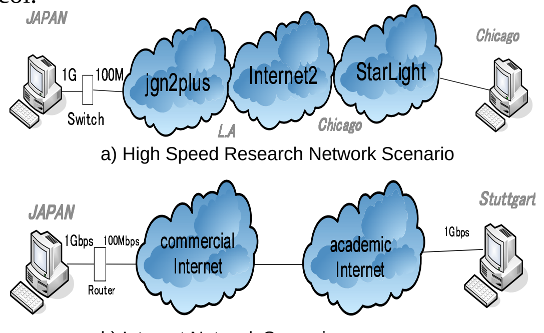
Our design follows the TCP framework: slow start, congestion avoidance, fast retransmit, and fast recovery phases, with adaptations to capacity and congestion probing as follows:

- **Slow Start :** We use a plain TCP slow start mechanism so as to focus on characterizing the performance of the congestion avoidance mechanism proposed in this paper. The only difference is that the bottleneck capacity and the buffer size estimation are passively performed during slow start as well as during congestion avoidance.
- **Congestion Avoidance :** In congestion avoidance, capacity estimators are updated continuously, so that the CCPD session can track changes in the path characteristics due to cross traffic dynamics. In particular, a capacity segment sample is set at every rtt interval to avoid interference between samples, provided that the control window is increased by at least two segments via Eq. 3. High accuracy clock helps accurate computation of the derivative component of the controller.
- **Fast Retransmit and fast recovery :** Duplicate acks cause segments to be retransmitted. During retransmission, the congestion window is maintained at the same size until all segments transmitted during that window are acknowledged. Moreover, for each duplicate ack received, the congestion window is increased to allow the retransmission of the missing segment. During recovery, rtt measurements become problematic, since segments may have to be retransmitted several times, artificially increasing their rtt. Since CCPD relies on rtt measurements, the protocol does not react to dupacks, avoiding estimators' contamination with inflated rtt values.

Regarding implementation cost, since no additional packets are used, there is no bandwidth overhead incurred by CCPD. Regarding scalability, the protocol requires OS kernel timers of small enough granularity to detect time differences that scale with bottleneck capacity speed. We have upgraded our previous estimators' implementation with high accurate clocks, where nanoseconds accuracy allows us to probe path bottleneck capacity in excess of 100 Gbps.

VI. PERFORMANCE EVALUATION

We now report on a series of open source based experiments on a high speed research network as well as the Internet (5. The research network is used to analyze our protocol properties and performance in detail, as we are able to control cross traffic and path routes. The Internet scenario is used to investigate protocol feasibility on paths with realistic cross traffic. We contrast the CCPD performance with two well known TCP protocols: NewReno, with a loss based congestion avoidance; Cubic, the Linux TCP algorithm of choice; and CCP [5], our previous delay based congestion avoidance protocol.



a) High Speed Research Network Scenario

b) Internet Network Scenario

Fig. 5: TCP Protocol Evaluation: Network Scenarios

A. PD controller parameters tuning

We use the research network scenario to tune CCPD parameters. We have selected a path with medium rtt, 40msecs, between two machines in Japan, as a commonplace path. However, the control theory behind equation 3 ensures system stability regardless the path rtts and bottleneck capacity speeds. Table I reports on 100MByte file delivery completion time for various Kp and Kd parameters. We have included CCP results, for comparison. From these results, we have selected $Kp = 2$ and $Kd = 100$ as default CCPD parameters, as a tradeoff between throughput performance and variance.

	CCPD(2,100)	CCPD(2,1000)	CCPD(4,100)	CCPD(4,1000)	CCP(2)	CCP(4)
trial 1	20791.4	17872.1	19364.5	20945.8	56636.3	25303.4
trial 2	19615.9	24074.5	17046.9	16810.4	38083.9	23647.3
trial 3	19162.0	15854.5	14419.3	20013.4	59176.1	22777.5
trial 4	20054.5	16103.8	15288.4	18280.9	30264.5	35769.7
trial 5	20818.6	18916.6	26469.8	20625.3	16608.3	33574.1
avg	20088.5	18564.3	18517.8	19335.2	40153.8	28214.4

TABLE I: 100MB delivery time(msec) : rtt=40msec

B. Transport protocols' performance with no packet loss

In this experiment set, we characterize the performance of the TCP protocols when session path is clear of congestion and packet losses. Tables II and III show the completion time of a file of 100MBytes delivered over the research network for short (20msecs) and long (180msecs) path scenarios. CCP is characterized for K_p parameter, whereas CCPD is characterized for K_p and K_d parameters. In terms of transfer speed performance, Table II shows that all protocols perform similarly for the short rtt and no packet loss scenario. For long rtts (Table III), we see that Cubic, CCP(4) and CCPD(4,100) are the fastest protocols, being the most aggressive TCP congestion avoidance schemes. Hence, the least aggressive protocols (Reno and CCPD(2,100)) perform poorly in long rtt scenarios with no cross traffic. For better understanding of the dynamics of the congestion avoidance, we include a characterization of the cwnd control window for a single long rtt trial in Fig. 6. CCP(4) and CCPD(4,100) have similar cwnd dynamics over large rtts.

	Reno	Cubic	CCP(2)	CCP(4)	CCPD(2,100)	CCPD(4,100)
trial 1	1229.7	1227.3	1410.7	1226.2	1459.6	1230.8
trial 2	1221.2	1226.9	1219.3	1222.6	1225.9	1224.1
trial 3	1223.7	1455.8	1220.3	1219.7	1220.6	1224.4
trial 4	1220.0	1453.6	1458.0	1447.2	1222.6	1231.8
trial 5	1218.6	1223.2	1200.3	1222.9	1220.9	1228.3
avg	1222.64	1317.36	1301.72	1267.72	1269.92	1227.8

TABLE II: 100MB delivery time(msec): 0 PER ; rtt=20msec

	Reno	Cubic	CCP(4)	CCPD(2,100)	CCPD(4,100)
trial 1	44383.5	23114.0	21548.3	34490.6	19117.5
trial 2	44449.1	23197.8	21868.4	30846.7	22878.3
trial 3	44385.2	23134.4	19116.3	29957.6	21982.2
trial 4	44382.6	23117.9	21788.6	28430.8	22641.4
trial 5	44385.4	25043.9	21485.8	37542.3	23516.9
avg	44397.2	23521.6	21161.5	32253.6	22039.3

TABLE III: 100MB delivery time(msec): 0 PER ; rtt=190msec

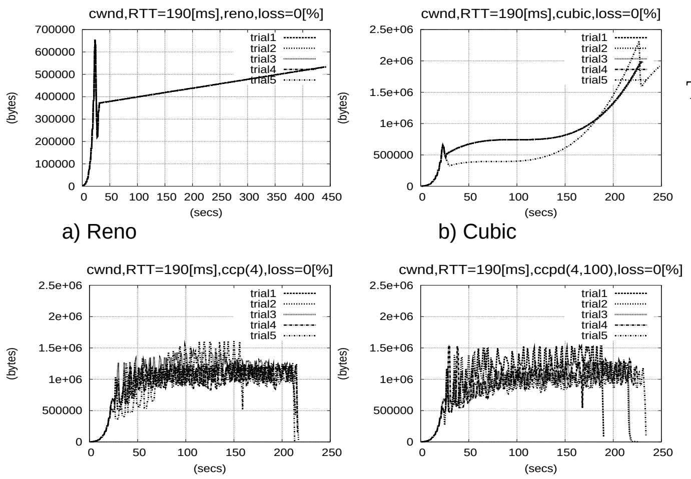


Fig. 6: Cwnd(t) without random packet loss : rtt=190msec

C. Transport protocols' performance with random packet loss

In this experiment set, we characterize the performance of the TCP protocols when the session experiences random packet losses. Tables IV and V show the completion time of a file of 100MBytes delivered over the research network, when a 10^{-4} packet drop (PER) is exercised by a link emulator placed at the bottleneck link of the session, for 20msec and 180msec rtt scenarios, respectively. CCP is characterized for $K_p = 4$, whereas CCPD is characterized for $K_p = 2, 4$, and $K_d = 100$. In terms of transfer speeds, we see that all protocols completion time get severely affected by the packet loss, if compared with no loss results, except CCP and CCPD for long rtt scenario. For short rtt scenario, the protocols with most impacted file completion times are the most aggressive protocols, namely Cubic and CCP. The least affected protocol is CCPD(2,100), arguably the least aggressive protocol. Figure 7 depicts the cwnd dynamic behavior of one of the long rtt trial for all protocols. We see that for all protocols except CCP and CCPD, there is a large drop in cwnd size on every packet loss experienced. Because CCP and CCPD rely on estimators that are not related with packet loss, but rather packet delays, not affected by random losses, their performance do not get affected by random losses significantly.

In summary, two factors significantly affect the performance of the protocols: packet loss level, and rtt size. For short rtt scenarios and no packet loss, all protocols deliver similar completion time performance. For high packet loss, long rtt scenarios require aggressive protocols for superior performance, while short rtt scenarios favor less aggressive protocols in delivering faster completion time.

	Reno	Cubic	CCP(2)	CCP(4)	CCPD(2,100)	CCPD(4,100)
trial 1	19781.9	1502.4	1849.3	2364.3	1295.4	1274.0
trial 2	17402.6	2082.8	1981.0	1728.1	1988.2	1711.2
trial 3	20899.0	3634.5	1596.4	1421.4	1623.7	1229.3
trial 4	7699.5	1726.7	2463.0	3603.7	1144.3	1519.9
trial 5	7047.1.2	1304.6	1198.5	1604.0	1508.7	1953.5
avg	14566.0	2050.2	1817.6	2144.3	1512.0	1537.6

TABLE IV: 100MB delivery t(msec): 10^{-4} PER; rtt=20msec

	Reno	Cubic	CCP(4)	CCPD(2,100)	CCPD(4,100)
trial 1	133872.7	38054.4	24262.5	33819.9	23247.3
trial 2	107217.8	39091.7	21772.8	32684.9	22374.0
trial 3	117845.4	65617.6	24245.2	30335.9	21342.4
trial 4	126682.7	45681.4	22435.5	29964.2	21837.4
trial 5	118829.2	38538.0	22616.5	31382.9	23557.9
avg	120889.6	45396.6	23066.5	31637.6	22471.8

TABLE V: 100MB delivery t(msec): 10^{-4} PER ; rtt=190msec

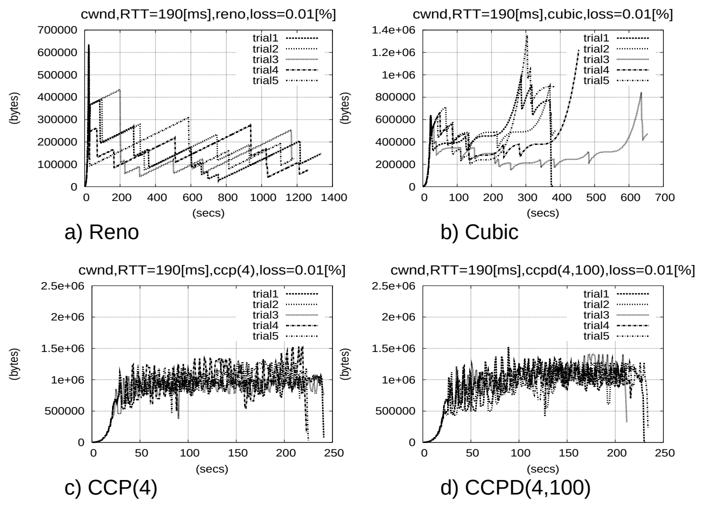
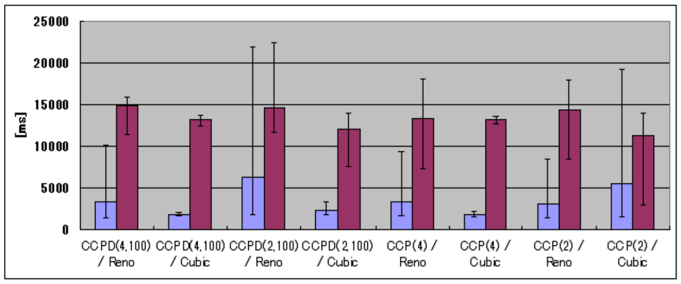


Fig. 7: Cwnd(t) with random packet loss

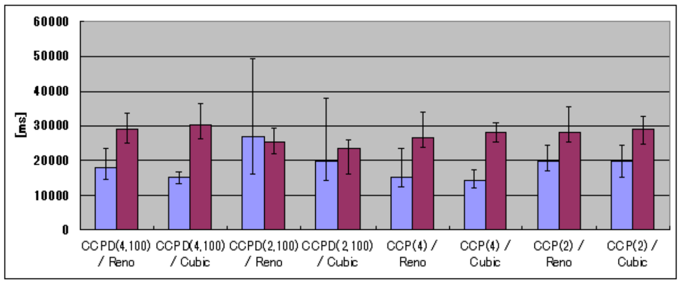
D. Benchmarking CCPD against other TCP protocols

In this subsection, we benchmark CCPD against Reno, Cubic, and CCP TCP protocols. Two parallel TCP sessions are initiated for the same file of 100MByte size, over the research network and Internet scenarios. We recall that the Research Network has very little cross traffic. We collect completion time performance for short and long rtt types of session. Results are shown in Figs. 8 and 9. Each pair of bars indicate average completion time over five trials for two competing TCP protocols. A range bar on top of the histogram bar indicates minimum and maximum values across the trials.

As the two sessions come simultaneously into the network, their completion time performance are comparable.

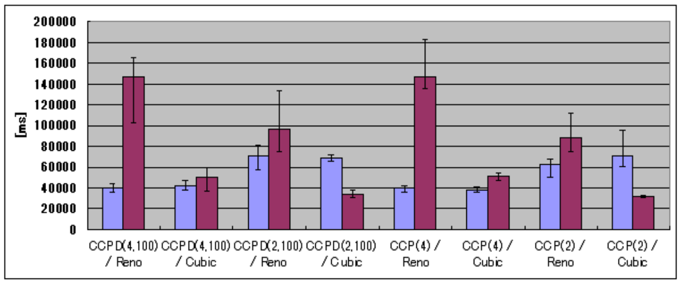


a) Short rtt scenario: rtt=20 msecs

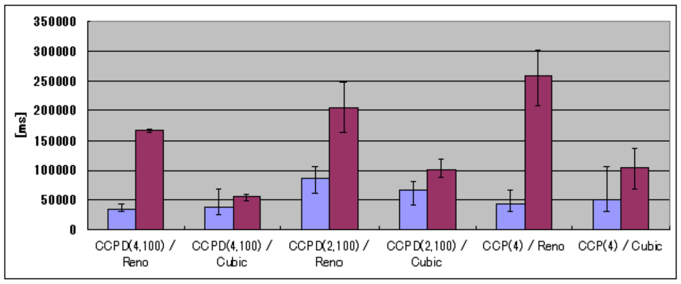


b) Long rtt scenario: rtt=190 msecs

Fig. 8: Research Network: Completion time performance



a) Short rtt scenario: rtt=20 msecs



b) Long rtt scenario: rtt=300msecs

Fig. 9: Internet: Completion time performance

In the research network scenario, where we have practically no cross traffic, CCP and CCPD perform better than all other protocols except CCP(2) and CCPD(2,100), outperformed by Cubic on long rtt scenario. For short rtt sessions, CCPD outperforms CCP and other protocols (e.g. CCPD(2,100)/Cubic vs CCP(2)/Cubic). In the Internet scenario, CCP and CCPD outperform the other protocols on average completion time. When comparing CCPD(x,100)/otherTCP with CCP(x)/otherTCP, we see that CCPD outperforms CCP for long rtt scenario, whereas CCP outperforms CCPD for short rtt scenario. CCPD performance over the Internet needs further investigation.

VII. Future Work

In this paper, we have introduced TCP-CCPD, a transmission control protocol based on control theoretical concepts, and window regulation based on TCP session estimators. The protocol regulates traffic injection by tracking capacity and congestion along the session path during the lifetime of the session, an implementing a proportional plus derivative controller. The derivative component of the controller allows quick reaction to queue build ups. Preliminary experimental results have demonstrated CCPD competitiveness as compared to widely used TCPs. We are in the process of generating more extensive experimental results. In addition, we are currently studying a hybrid CCP/CCPD congestion avoidance mechanism, which activates the derivative part of the controller only in appropriate path scenarios. The goal is to guarantee best performance regardless of the network path characteristics.

Acknowledgment

Work supported in part by JSPS Grant-in-Aid for Scientific Research KAKENHI B (21300024).

References

[1] K.J.Astrom and B.Wittenmark, “Computer Controlled Systems.” *Englewood Cliffs, NJ*: Prentice Hall, 1990.

[2] L. S. Brakmo and L. L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE J. Select. Areas Commun.*, Vol. 13, No. 8, pp. 1465-1480, 1995.

[3] D. Cavendish, M. Gerla, and S. Mascolo, “A Control Theoretical Approach to Congestion Control in Packet Networks,” *IEEE Transactions on Networking*, Vol. 12, No. 5, pp. 893-906, October 2004.

[4] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie, and M. Gerla, “Capstart: An Adaptive TCP Slow Start for High Speed Networks,” *IEEE First International Conference on Evolving Internet*, best paper award, pp. 15-20, August 2009.

[5] D. Cavendish, K. Kumazoe, M. Tsuru, Y. Oie, and M. Gerla, “Capacity and Congestion Probing: TCP Congestion Avoidance via Path Capacity and Storage Estimation,” *IEEE Second International Conference on Evolving Internet*, best paper award, September 2010.

[6] M. Chen, J. Zhang, M. N. Murthy, and K. Premaratne, “TCP Congestion Avoidance: A Network Calculus Interpretation and Performance Improvements,” *Proceedings of INFOCOM05*, Vol. 2, pp. 914-925, March 2005.

[7] J-Y. Choi, K. Koo, J. S. Lee, and S. H. Low, “Global Stability of FAST TCP in Single-Link Single-Source Network,” *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 1837-1841, December 2005.

[8] S. Floyd and K. Fall, “Promoting the Use of End-to-End Congestion Control in the Internet,” *IEEE/ACM Transactions on Networking*, Vol. 7, No. 4, August 1999.

[9] J. Martin, A. Nilsson, and I. Rhee, “Delay-Based Congestion Avoidance for TCP,” *IEEE/ACM Transactions on Networking*, Vol. 11, No. 3, pp. 356-369, 2003.

[10] R. Kapoor, L-J Chen, L. Lao, M. Gerla, and M. Y. Sanadidi, “CapProbe: A Simple and Accurate Capacity Estimation Technique,” *Proceedings of SIGCOMM 04*, Portland, Oregon, pp. 67-78, Sept. 2004.

[11] L. Kleinrock, “Queueing Systems. Volume II: Computer Applications,” *Wiley*, 1976.

[12] O.J.Smith, “A controller to overcome dead time,” *ISA J.*, Vol.6, No.2, pp. 28-33, Feb. 1959.