A Congestion Control Algorithm for Data Center Area Communications

Hideyuki Shimonishi, Junichi Higuchi, Takashi Yoshikawa, and Atsushi Iwata
System Platforms Research Laboratories, NEC Corporation
1753 Shimonumabe, nakahara-ku, Kawasaki, Kanagawa 211-8666, Japan
Tel +81-44-431-7633, Fax +81-44-431-7644
h-shimonishi@cd.jp.nec.com

***Abstract*—Ethernet is extending its applications to data-center area, i.e. very short distance, communications such as CPU-CPU and CPU-I/O interconnections, where short end-to-end (MAC-to-MAC) data transfer delay is the key performance factor. In this paper, we propose an extended Ethernet MAC mechanism providing end-to-end reliable and congestion controlled packet transfer that minimizes data transfer delay. The proposed MAC mechanism employs a packet retransmission mechanism and a delay-based congestion control algorithm with inline initial bandwidth probing. Since inline bandwidth probing with small number of data packets tends to overestimate the initial bandwidth, we introduced two extended mechanisms; two-stage probing and RTT prediction. The RTT prediction mechanism detects congestion several RTTs earlier than using measured RTT and thus minimize queuing delay. The two-stage probing mechanism adaptively starts up data transfers without overflowing the link. The simulation results show that the proposed mechanism achieves roughly 10 times shorter transfer delay compared to TCP and 2-3 times shorter delay compared to existing delay-based congestion controls.**

I. INTRODUCTION

Ethernet is extending its applications to very short-distance communications such as CPU-CPU and CPU-I/O interconnections in large scale cluster systems. We have proposed a scalable I/O interconnection technology called ExpEther [1]. ExpEther transparently tunnels PCI-Express packets over Ethernet; thus, PCI devices are connected via Ethernet without any modifications to existing hardware and device drivers. Therefore, ExpEther provides a flexible, scalable and cost-effective communication method for computer systems connecting a large number of CPUs and I/Os compared to existing interconnection technologies such as InfiniBand or PCI-Express switches.

One of the typical applications is remote disk mount over Ethernet. In contrast to a NAS (Network Attached Storage) system, ExpEther connects the disk as if they are locally mounted and thus provides fast and efficient connection. Or any remote I/O devices such as graphic cards or accelerators can be connected via Ethernet.

Since PCI-Express is a (packetized) computer-bus technology and provides a channel equivalent to a physical wire, the challenge of ExpEther is to achieve very small data transfer delay, as well as high reliability without any packet losses or congestion. In contrast to FTP or WWW, CPU-CPU and CPU-I/O interconnections have very different traffic characteristics. Rather than generating a large burst of data, relatively small block of data and its acknowledgement is frequently exchanged. Therefore, short end-to-end (MAC-to-MAC) delay is the key performance factor. Large data transfer delay would inefficiently consume link bandwidth just waiting for the acknowledgements from the receiver.

TCP (Transmission Control Protocol) [2] is used to provide reliable communications on top of IP and Ethernet. Since TCP employs loss-based congestion control algorithm, which continues to increase sending rate until it gets packet losses, its queuing delay can grow till its maximum and it takes time to converge to the appropriate sending rate. In very short distance communications, larger data transfer delay adds up to significant performance degradation of CPU-CPU or CPU-I/O effective bandwidth. Recently there are a number of TCP variants having delay-based congestion control algorithm [8][10][11], but they are also not very efficient because of small burst sizes. Another drawback of TCP is its high processing cost that heavily consumes CPU resources. TOE (TCP Offload Engine) is often used to offload the processing to specialized network interface hardware. Nevertheless it would be too costly to have all TCP/IP/Ether protocol stacks on every CPU and I/O devices.

In this paper, we propose an extended Ethernet MAC mechanism, which we call EFL (Ethernet with Flow Label). To provide an end-to-end (MAC-to-MAC) reliable path over Ethernet, EFL extends MAC with packet retransmission mechanism and congestion control mechanism. The retransmission mechanism is a simple one consists of block data transfer, rather than byte data transfer of TCP, and Go-back-N ARQ mechanism. The congestion control algorithm employs a delay-based congestion control because it has an advantage of maintaining small queuing delay. To improve transfer delay of short data transfer, we introduced an initial bandwidth probing mechanism. The mechanism is a quick inline bandwidth measurement introduced for agile rate adaptation at the beginning of a data transfer. The mechanism measures an appropriate sending rate as soon as a new data transfer starts and quickly adjusts the sending rate. Since we assume applications that generate a series of relatively short data transfers, agile rate adaptation is very important to improve communication efficiency.

Inline bandwidth measurement, which uses data packets as a probe, has been proposed in some literatures [6][9]. However, the measurement requires large number of packets and long time interval, typically several RTTs, to get accurate bandwidth information. In fact, it is shown that the measurement with a small number of probe packets tends to overestimate the bandwidth [6]. To get bandwidth information quickly with a small number of probing packets and still avoid overloading the path with inaccurate bandwidth estimation, we propose the following two mechanisms. The first one is two-stage probing mechanism, which sends a small number of packets, e.g. 10 or 20 packets typically, at the beginning and continues to send packets at a very small rate until first ACK packet arrives. The first group of packets is used to measure the bandwidth roughly and quickly, and the subsequent packets are used to verify and adjust the measured bandwidth. After the initial bandwidth probing, the second mechanism is used with delay-based congestion control. Since delay-based congestion control relies on RTT measurements and it takes time to observe increased RTT after a congestion period actually starts, reaction to the congestion is delayed. Thus, to minimize queuing delay, the second mechanism utilizes the observation that the change of receiving rate precedes the change of measured RTT, and predicts RTT changes.

Explicit congestion feedback mechanisms like XCP [3], IEEE 802 Congestion Management [4], or Quick-Start [5] would be useful to quickly obtain accurate congestion information or appropriate initial sending rate. However, the need for design new Ethernet switches having such mechanism would spoil the advantage of using standard Ethernet for CPU-CPU and CPU-I/O interconnection.

II. REALABLE ETHERNET TRANSPROT

EFL extends MAC with packet retransmission mechanism and congestion control mechanism. As shown in Fig. 1, end points having EFL MAC, either at server (CPU) or I/O device side, are connected via Ethernet. When there are any packet losses in the network, sender side EFL MAC recognizes the losses and retransmits the lost packets. Also, when there is any congestion in the network, the sender side EFL MAC regulates its sending rate to mitigate such congestion.

As shown in Fig. 2, EFL defines an extended MAC header having sequence number and time stamp. It also defines a new MAC packet for acknowledgement having acknowledged sequence number. The sequence numbers are used for packet loss detection and retransmission. They are defined as packet count, rather than byte count, because EFL encapsulate PCI-Express packet with MAC headers and do not re-segment byte data stream. Time stamps, which records the time the packet is sent, in both forward data packet and ACK packet are used to measure RTT. In our implementation, the size of the extension header is 9 bytes, whereas TCP/IP header is 40 bytes. This small header overhead is significant especially for PCI-Express tunneling because typical length of PCI-Express packets is as small as 128 or 256 bytes.

In our implementation, we use Go-back-N ARQ mechanism for packet retransmission because it does not

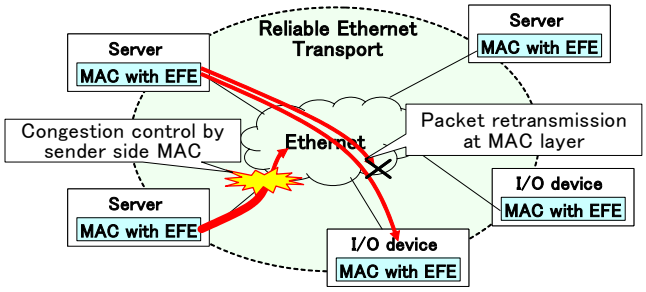

Fig.1: Reliable Ethernet Transport with EFL

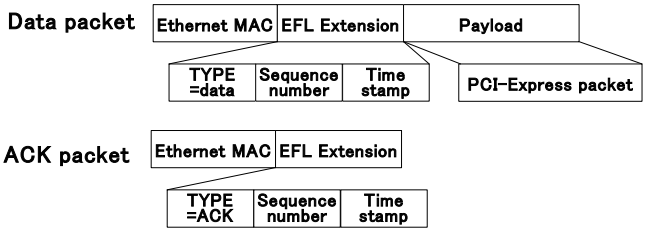

Fig.2: EFL frame format

requires reorder buffers at a receiver and thus the receiver hardware is greatly simplified. Since we assume that packet loss events are exceptional very rate through delay-based congestion control, the simple Go-back-N mechanism would not significantly deteriorate the efficiency. Selective repeat may further improve the efficiency, but it would be minor with the assumption. For detecting packet loss, we employed both fast retransmission mechanism triggered by duplicate ACKs and retransmission timeout. Since we assume local Ethernet where packet reordering rarely happens, we employ double duplicate ACKs, rather than triple duplicate ACKs. For detecting retransmission timeout, as well as for RTT measurement, we have very fine grain timers, an order of 100nsec.We also employed delayed ACK mechanism to reduce the number of ACK packets. Since PCI-Express packet is very small, we configured that an ACK packet is sent for every ten data packets.

III. CONGESTION CONTROL ALGORITHM FOR VERY SHORT DISTANCE COMMUNICATIONS

As shown in Fig. 3 and 4, the congestion control algorithm has four states for sending rate control: 1) idle state, 2) bandwidth probing state, 3) delay probing state, and 4) congestion avoidance state. In the following, sending rate control in each state is explained.

1) Idle state

If there are no data to send, a sender is in this state. When the sender finishes sending all data packets and receives all acknowledgements for the data packets, the sender switches to this state.

2) Bandwidth probing state

When a new group of packets arrives at the sender, the sender switches to *Bandwidth probing state* and sends first N data packets at a maximum bandwidth. These data packets are used as a probe to measure appropriate initial sending rate. N is the number of packets a sender can transmit without hearing from the network. It should be small enough to avoid the risk

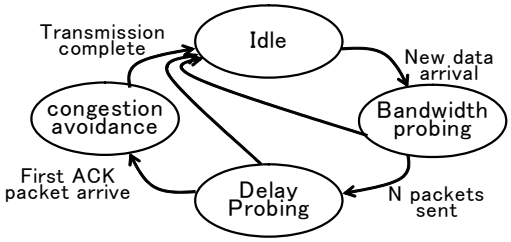

Fig. 3: State transition diagram (1)

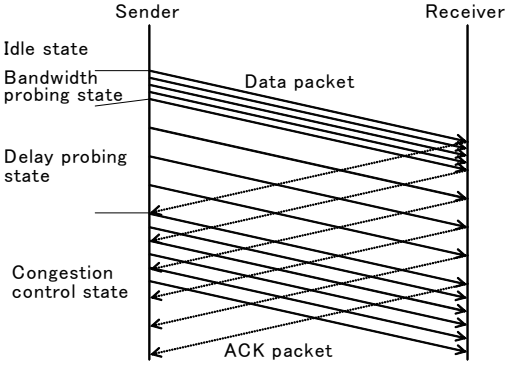

Fig. 4: State transition diagram (2)

of instantaneous congestion. In our implementation, we set *N=20*, i.e. 5KB of data with 256 byte encapsulated PCI-Express packet, considering that TCP initial window size is 4 segments, i.e. 6KB of data with 1500byte MTU. These initial packets are also designed to send short messages quickly without waiting for acknowledgements.

After receiving all ACK packets acknowledging the reception of all the packets sent in this state, the sender calculates the sending rate R as follows:

$$R = \frac{D}{T_2 - T_1} \tag{1}$$

where T_1 and T_2 are the reception time of the first and last ACK packets, respectively, and D is the amount of data sent by these data packets. Since T_1-T_2 is regarded as the time for the data packets to traverse through a bottleneck queue, the above equation estimates the bandwidth that the data packets went through the bottleneck link.

3) Delay probing state

After sending initial N data packets, the sender switches to *Delay probing state*. In this state, the sender has not heard from network yet and thus continues to send data packets at a minimum sending rate not to stress the network. As described above, inline measurement with a small number of probe packets tends to overestimate the bandwidth. Therefore, although the calculated initial sending rate would be useful as a prompt measurement; it should be beneficial to continue to monitor the network to verify and adjust the initial sending rate. The packets sent in this state are used for this purpose.

When the sender receives an ACK packet responding to the data packet sent in this state, the sender measures RTT and updates the initial sending rate. If the ACK packet has RTT larger than the minimum RTT value, i.e. round trip propagation delay between the sender and receiver, the sender recognizes the existence of cross traffic and reduces the initial sending rate. The rate update uses the same equation used in the Congestion avoidance state, e.g. Eq. (2), but the parameters would be reconfigured, i.e. smaller alpha and larger beta, to be more conservative.

4) Congestion avoidance state

When the sender receives first ACK packet, it switches to *Congestion avoidance state*. In the beginning of this state, the sender starts packet transmission using the calculated initial sending rate, and then the rate is kept updated using RTT information. Like other delay-based congestion controls, such as TCP-Vegas [7] and FAST TCP [8], the sending rate is calculated based on the difference of the amount of backlogged data in a bottleneck queue and its target value. The sending rate R is updated upon each reception of each ACK packet as follows;

$$R = R + \left(\alpha - (RTT - RTT_{MIN})R\right)\frac{\beta}{RTT} \tag{2}$$

where RTT_{min} is the minimum RTT value. Parameter α is the target value for backlogged data, and equivalently, determines the speed of rate increase. Parameter β is a control gain.

5) Packet loss

When a packet loss is detected through duplicate ACK or retransmission timeout, sending rate is halved, like most TCP algorithms do. In the bandwidth probing state, sending rate is reduced to 0 because the rate is not determined yet in this state.

6) RTT prediction mechanism

To minimize transfer delay of relatively short data transfers, agile rate adaptation is required and thus larger α value is preferred. However, we have observed that larger α value can result in larger or oscillated queuing delays, and thus non-negligible packet losses. To minimize and stabilize the queuing delay, it should be essential to quickly detect and react to congestion as soon as the congestion begins.

Therefore, we propose a RTT prediction mechanism to detect an incipient congestion before measured RTT grows. To detect the incipient congestion, we utilize the observation that the receiving rate changes faster than measured RTT. The receiving rate can be estimated at the sender using ACK packets. As the traffic volume at a bottleneck queue increases and congestion begins, it takes time the queue length grows enough. The sender recognizes the increased queuing delay after a round-trip propagation delay plus queuing delay. Thus, as the queuing delay increases, it takes more time for the sender to recognize the increased RTT. On the other hand, the output rate of a flow from the queue changes immediately after the congestion begins, and thus, the change can be recognized by the sender within one RTT.

In a steady state, we have the following relationship.

$$R_{RECV} = D_{FLY} / RTT \tag{3}$$

where R_{RECV} is the receiving rate and D_{FLY} is the amount of data on the fly (which corresponds to the window size of TCP). In a transient state, R_{RECV} changes faster than measured RTT changes, thus, D_{FLY} / R_{RECV} can be regarded as a foregoing index of measured RTT. Since D_{FLY} can be approximated by $R \cdot RTT$, we calculate the predicted RTT, $RTTp$, as following

$$RTTp = RTT \frac{R}{R_{RECV}} \quad (4)$$

This equation indicates that:

- In a steady state when queue length is constant, the receiving rate is equal to the sending rate and thus $RTTp = RTT$.
- When congestion begins and the queue length starts growing, the receiving rate becomes less than the sending rate and thus $RTTp > RTT$.
- When congestion is ceasing and the queue length begins shrinking, the receiving rate becomes larger than the sending rate and thus $RTTp < RTT$.

IV. Performance Evaluations

We have evaluated the proposed mechanisms using ns2 simulator [12]. Now, we are developing prototype MAC hardware and will report experimental evaluation results in the near future. We have confirmed that the packet retransmission mechanisms work properly and we will show the results for the congestion control algorithm in the following evaluations.

1) *Evaluation model*

Figure 5 shows the network model. In this evaluation, we used dumb-bell topology to investigate basic behaviors of the proposed algorithms. Evaluations with more complex network topology using simulator and/or prototype hardware system would be one of the future works. Evaluations using real CPU-CPU and CPU-I/O traffic in the testbed system, and discussions on parameter setting using the real traffic would also be our future works.

All link bandwidth is 10 Gbps and round trip propagation delay between senders and receivers is 100us. Switches employ tail-drop buffers with 1MB capacity. Packet size is 192 byte including extended MAC header and PCI-Express packet as a payload.

We have compared the following four congestion control algorithms; 1) *Plain*, plain delay-based algorithm using Eq. (2), 2) *BW probe*, delay-based algorithm with bandwidth probing using Eq. (1) and (2), 3) *Proposed*, delay-based algorithm with two-stage probing and RTT prediction, and 4) *TCP*, TCP-NewReno. Since there are a number of delay-based protocols, we modeled them into 1) and 2) and tested the models to make the results more understandable and comparable, rather than testing each one of them.

2) *Evaluation of RTT prediction*

We first show an example of predicted RTT. Traces of actual RTT, measured RTT, and predicted RTT are shown in Fig. 6. The actual RTT is a theoretical RTT value calculated from instantaneous queue length, link bandwidth and propagation delay, namely *actual RTT = queue length / link bandwidth + propagation delay*. In this particular example, two flows are sending from the beginning of the simulation and extra two flows start at 10msec.

When congestion occurs at 10msec, queuing delay increases first and the measured RTT increases subsequently. Although the raise of measured RTT and predicted RTT starts

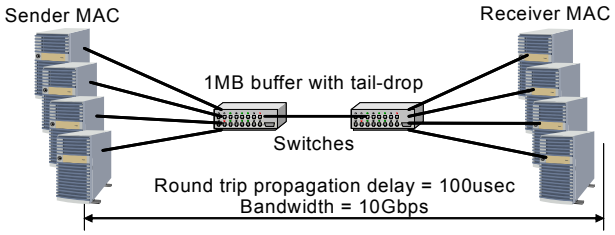

Fig. 5: Network model

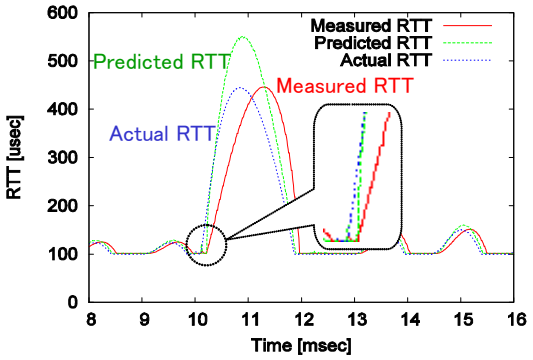

Fig. 6: Predicted / measured / actual RTTs (2+2 long-lived flows)

slightly after the raise of actual RTT by 1RTT=100usec (dotted circle in the figure), the increase of predicted RTT catches up very soon and then follows the increase of actual RTT. While predicted RTT and actual RTT record their peak value almost at the same time, measured RTT records its peak value 400usec afterwards. Although predicted RTT is not a real prediction of RTT and can not go faster than propagation delay, it sensitively captures the trends of the queue behavior by measuring the rate changes and uses them to

3) *Evaluation with long-lived flows*

To discuss transient behavior of the proposed algorithm, we used several long-lived flows; first flow (flow 1) starts at 0msec, and then, two flows (flow 2-3) start at 10msec, 4 flows (flow 4-7) start at 20msec, 8 flows (flow8-15) start at 30msec, 16 flows (flow 16-31) start at 40msec. At 50msec, all flows except for the first one terminate.

In Fig. 7 and Fig. 8 sending rate of individual flows in each group is plotted for plain and proposed algorithms, respectively. With the probing mechanism, sending rate of flow 1 quickly reaches to the link capacity as soon as it starts, whereas it takes 13msec for the flow to reach the link capacity without the probing mechanism. At 10msec, flows 2 and 3 start with the estimated initial rate of 4Gbps and they quickly reduce the rate in response to the increased queuing delay. Then these three flows converge to the same sending rate. As other flows come in, they also quickly converges to the fair share rate. We configured that all flows measures their minimum RTT values before the congestion occurs and they actually measure the same minimum RTT value, thus, as a nature of delay-based protocols, sending rate of all active flows converges to the fair share value sometime after new flows are added. In a steady state, all three delay-based protocols, including plain, BW probe, and proposed algorithms, converge to the fair share rate but the Plain and

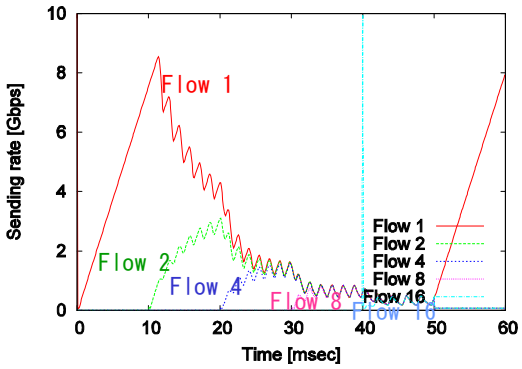

Fig. 7: Throughput of individual flows (*Plain*)

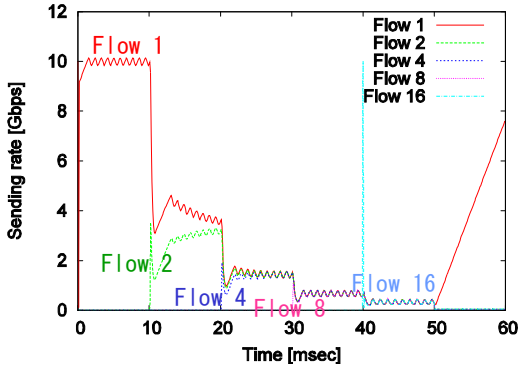

Fig. 8: Throughput of individual flows (*Proposed*)

Probing algorithm, both without RTT prediction, oscillates the sending rates more than the proposed algorithm.

Traces of their RTTs are shown in Fig. 9. When new flows come in, both BW probe and proposed algorithms experience larger queuing delay due to larger initial sending rate, but the proposed algorithm roughly halves its maximum queuing delay. For example, when flows 2 and 3 start, the bandwidth probing estimates initial sending rate to be 4Gbps, thus, the sum of the sending rate of flow 1-3 becomes 18Gbps and the queue at the bottleneck link starts to built up. As a reaction to this sudden congestion, the proposed algorithm quickly reduces the sending rate than BW probe algorithm does, i.e. proposed algorithm takes less than 100usec to reduce the sending rate whereas BW probe algorithm takes 200usec. Their peak queuing delays at this moment are 120usec and 250usec, respectively. This figure also shows larger oscillation of RTT regarding plain and BW probe algorithms. Without the RTT prediction mechanism, reactions to congestions are delayed and their maximum queuing delay in a steady state reaches 250usec, whereas the proposed algorithm has the maximum queuing delay of 100usec during steady state.

4) Evaluation with short data transfers

To discuss the behaviors of the proposed algorithm in more realistic situation, we tested the algorithms in an environment where a large number of on-off flows compete at the bottleneck link. Each flow generates a series of short data, whose average size is 100KB and its distribution is Pareto (Shape=1.2). Inter-arrival time of the data is exponentially

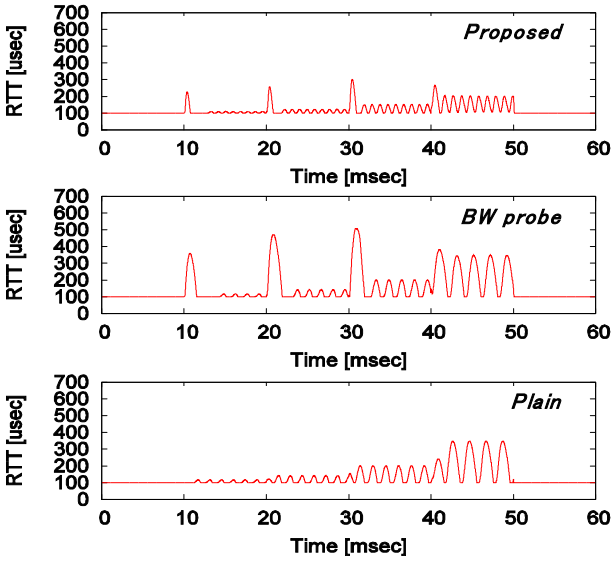

Fig. 9: Traces of RTTs (long-lived flows)

distributed whose average is 10msec. The number of flows is 150, which means the link is fairly loaded. In this evaluation, data transfer time is defined as a time interval from the time first packet is sent by a sender to the time last packet is received by a receiver.

Figure 10 shows the distribution of data transfer time sorted by their data size. The bandwidth probing mechanism is effective for relatively small data transfers whose data size is less than 10MB, and both BW probe and proposed algorithms achieves 3 or 4 times smaller transfer time than the Plain algorithm. And the proposed algorithm achieves further smaller transfer time; it almost halves transfer time for very short transfers compared with the BW probe algorithm, because of the smaller queuing delays. The plain algorithm with its slow startup has the smallest queuing delay but, on the other hand, it exhibits large transfer time except for very short data transfers whose transmission would be finished in a few round trips. In this evaluation, we also tested TCP-NewReno with and without slow start mechanism. Since TCP with slow start caused frequent retransmission timeouts and resulted in serious performance degradation, we only show the results without slow start in the figure. Although TCP achieves comparable transfer times with other algorithms for long data transfers, transfer time for short data is very poor due it its slow convergence to the appropriate sending rate, as well as large queuing delay and frequent packet losses due to its loss-based behavior.

We also compared the distribution of data transfer time of those algorithms. In Fig. 11, cumulative distribution of transfer time is plotted for different algorithms. This figure shows that, with the proposed algorithm, 95% of data transfers are done within 0.85msec, whereas with BW probe, plain, and TCP, they are done within 2.3msec, 3.5msec and more than 10msec, respectively. Also the portions of data transfers that are done within 1msec are 96%, 89%, 55% and 14% using the proposed, BW probe, plain, and TCP, respectively.

RTT traces are shown in Fig. 12. Average queuing delay of the proposed, BW probe and plain algorithms is 68us,

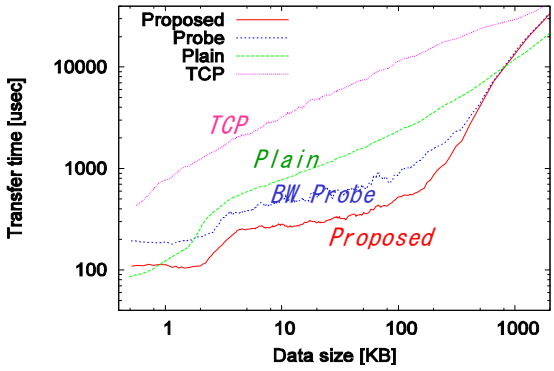

Fig. 10: Transfer time of individual data (on-off flows)

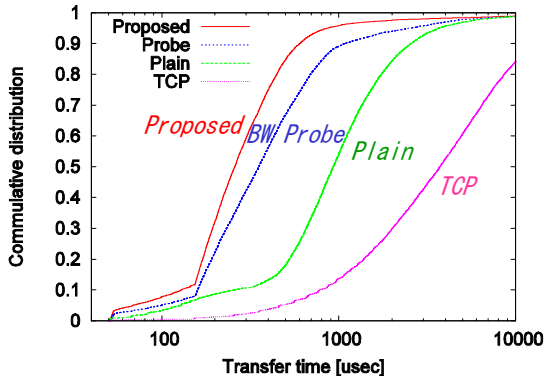

Fig. 11: Cumulative distribution of transfer time (on-off flows)

140us, and 34usec, respectively. The plain algorithm has the smallest queuing delay. This is because, with the plain algorithm, flows start with a small initial sending rate and most of the flows finish its transmission before reaching maximum sending rate. The proposed algorithm almost halves its queuing delay compared to the BW probe algorithm. In this evaluation, short data transfers frequently occur and these cross traffic may result in inaccurate RTT prediction. Our investigation actually shows that the predicted RTT is not very accurate in tracing small oscillation of RTT, however, it successfully captures the major trends of queue length changes faster than the changes of RTT. Consequently, the proposed algorithm can detect major congestion quicker, which results in smaller RTTs.

V. CONCLUSION

In this paper, we proposed the extended Ethernet MAC mechanisms providing end-to-end (MAC-to-MAC) reliable transport for very short-distance communications such as CPU-CPU and CPU-I/O interconnections. To minimize queuing delay and thus data transfer time, we have proposed a delay-based congestion control algorithm with two-stage probing and RTT prediction mechanisms. The two-stage probing mechanism mitigates the congestion caused by overestimated initial bandwidth probing and the RTT prediction mechanism detects congestion earlier than the measured RTT indicates congestion. The simulation results have shown that the RTT prediction mechanism detects

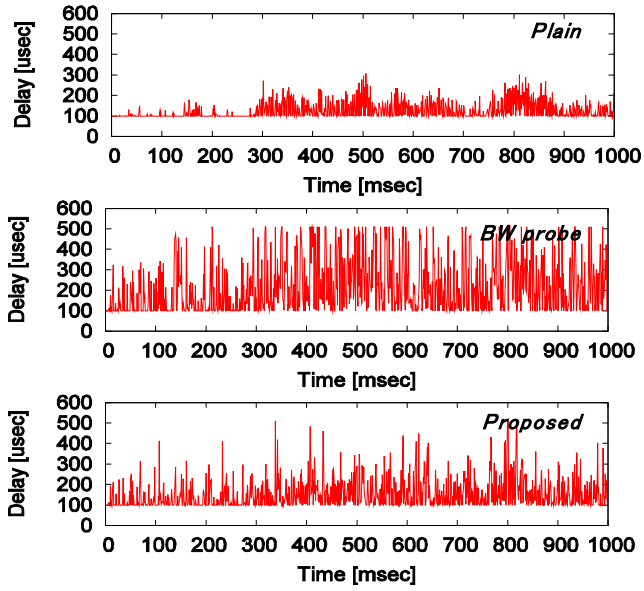

Fig. 12: Traces of RTTs (on-ff flows)

congestion several RTTs earlier than measured RTT increases. We have also shown that the proposed algorithm achieves roughly 10 times shorter data transfer delay compared to TCP and 2-3 times shorter data transfer delay compared to the plain delay-based congestion control. We are now developing prototype hardware system of the proposed mechanism, and will report further investigation results in the near future.

REFERENCES

[1] J. Suzuki, et. al., “ExpEther – Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform”, to be presented at HOT Interconnects 14, 2006.

[2] S. Floyd, et. al., “The NewReno Modification to TCP's Fast Recovery Algorithm”, RFC 2582, IETF, 1999.

[3] D. Katabi, et. al., “Congestion Control for High Bandwidth-Delay Product Networks.”, SIGCOMM02

[4] IEEE 802.1Qau - Congestion Notification

[5] S. Floyd, et. al., “Quick-Start for TCP and IP”, RFC 4782, IETF, 2007.

[6] Ren Wang, et. al., "Adaptive Bandwidth Share Estimation in TCP Westwood", In Proc. of Globecom 2002,

[7] L. Brakmo, et. al., “TCP Vegas: New techniques for congestion detection and avoidance.” SIGCOMM94

[8] David Wei, et. al., “FAST TCP: motivation, architecture, algorithms, performance”, IEEE/ACM Trans. on Networking, 14(6), 2006.

[9] Cao Le Thanh Man, Go Hasegawa and Masayuki Murata, "ImTCP: TCP with an inline measurement mechanism for available band-width," Computer Communications Special Issue: Monitoring and Measurements of IP Networks, September 2004.

[10] D. Leith, et. al., "Delay-based AIMD congestion control", In PFLDnet 2007, 2007.

[11] H. Shimonishi, T. Hama, and T. Murase, “TCP-AdaptiveReno: Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm”, In Proc. of PDLFnet, 2005

[12] The network simulator– ns-2, http://www.isi.edu/nsnam/ns