

Studying Congestion Control with Explicit Router Feedback Using Hardware-based Network Emulator

Kiyohide Nakauchi

Katsushi Kobayashi

National Institute of Information and Communications Technology
4-2-1, Nukui-kitamachi, Koganei, Tokyo 184-8795, Japan
{nakauchi, ikob}@nict.go.jp

ABSTRACT

Congestion control with explicit router feedback, such as TCP Quick-Start and XCP, is a promising way to enhance the performance of data transport especially in high-speed networks. When designing such transport protocols, the trade-off between the granularity of feedback information and additional cost of routers for feedback processing and state keeping should be discussed. However, the feasibility is never convinced in terms of its implementation and deployability because there is no experimental high-speed network environment to evaluate the additional cost. In this paper, we study congestion control with explicit router feedback using hardware-based high-performance network emulator. The network emulator consists of a general-purpose network processor and two Gigabit Ethernet (GbE) ports, where new explicit router feedback functions can be easily implemented and evaluated. The goal of this paper is to gain an insight into feasibility of explicit router feedback and to give a possible direction to design such congestion control mechanisms. To this end, we first discuss the deployability of fine-grained explicit router feedback through the experience of developing the high-performance network emulator and evaluations of the performance. In addition, based on the discussion, we propose a unified explicit router feedback framework. The framework can be applied to a broad class of transport protocols and can provide fine-grained feedback information fundamentally required by end-hosts to design a novel congestion control mechanism or to significantly enhance the performance of traditional congestion control in specific situations. As one possible application of the unified explicit router feedback framework, we show the extension of TCP Limited Slow-Start [2]. Through the implementation and evaluation, we make sure that a sender can shift from slow start phase to congestion avoidance phase without any packet drop, and can achieve more effective bandwidth utilization.

1. INTRODUCTION

As the Internet is evolving, high-bandwidth links ranging from 1 to 10 Gbps are increasingly in fashion around the world. The applications like bulk-data transfer, high-quality video streaming, networked collaboration, and computational grids benefit from this deployment and the performance of such applications over wide-area networks has become a critical issue.

Congestion control positively using explicit router feedback that indicates network conditions, such as TCP Quick-Start [3] and XCP [5], is a promising way to enhance the performance of data transport in high-speed networks. Note that explicit router feedback is also useful to assist recently-proposed TCP variants, such as HighSpeed TCP [1], Scalable TCP (SCTP) [6], FAST TCP [4], and BIC-TCP [12], which do not require any router feedback, to cope with high bandwidth-delay product networks. When designing such transport pro-

ocols with explicit router feedback, the trade-off between the granularity of feedback information and additional cost of routers for feedback processing and state keeping should be discussed. However, the feasibility is never convinced in terms of its implementation and deployability because there is no experimental high-speed network environment to evaluate the additional cost.

In this paper, we study congestion control with explicit router feedback using hardware-based high-performance network emulator. The network emulator is a hardware system and consists of a general-purpose network processor (Intel IXP2400) and two Gigabit Ethernet (GbE) ports. The core emulation software is implemented on network processor. The emulator has enough computation resource to provide additional processing for all forwarding packets, so that we can develop a powerful gigabit PC router, where new explicit router feedback functions can be easily implemented and evaluated.

The goal of this paper is to gain an insight into feasibility of explicit router feedback and to give a possible direction to design such congestion control mechanisms. To this end, we first discuss the deployability of fine-grained explicit router feedback through the experience of developing the high-performance network emulator and evaluations of the performance. As the result of a simple set of performance evaluations, we find the emulator can forward all the received packets with maximum processing delay of 150 ns at the wire rate. The emulator can also achieve wire rate (1 Gbps) forwarding with additional UDP header processing including masking, comparing, and writing. Note that these evaluations are conducted under the emulation mode where target delay is set to 100 ms. These results show explicit router feedback with limited header processing functions at routers would be feasible in terms of implementation limitations. Our main advantage is that we have already developed the powerful network emulator, and have built experimental environments to evaluate fine-grained explicit router feedback.

In addition, based on the discussion above, we propose a unified explicit router feedback framework. In traditional explicit router feedback mechanisms using in TCP Quick Start and XCP, desired information that could be obtained from router feedback depend on the kind of transport protocols, and individual router feedback mechanisms are proposed for specific purposes. We believe a unified fine-grained explicit router feedback framework that can be easily applied to a broad class of transport protocols would be useful. This is for the same reason that recently proposed Datagram Congestion Control Protocol (DCCP) [7] prepares several congestion control profiles and allows applications to choose between the profiles. The framework can provide fine-grained feedback information fundamentally required by end-hosts to design a novel congestion control mechanism or to significantly enhance the performance of traditional one in specific situations. Note

that the framework includes traditional explicit router feedback only with header processing of masking, comparing, and writing, such as TCP Quick Start, and can cope with such mechanisms.

As one possible application of the unified explicit router feedback framework, we show the extension of TCP Limited Slow-Start [2]. In the extended Limited Slow-Start, a sender is informed of minimum buffer size and maximum available bandwidth along the path from the sender to a receiver using the unified explicit router feedback framework, and the sender set Limited Slow-Start parameters `max_ssthresh` and `ssthresh` to appropriate values, respectively. Through the implementation and evaluation of the extended Limited Slow-Start, we make sure that a sender can shift from slow start phase to congestion avoidance phase without any packet drop, and can achieve more effective bandwidth utilization.

This paper is organized as follows. In section 2, we describe the feedback granularity for congestion control and limitation of existing congestion control with explicit router feedback. In Section 3, we show the specification of the hardware-based network emulator. We show an overview and implementation of a unified explicit router feedback framework and its application to TCP Limited Slow-Start in Section 4 and in Section 5, respectively. Finally, we conclude in Section 6.

2. FINE-GRAINED EXPLICIT ROUTER FEEDBACK

In this section, we picture the granularity of recent explicit router feedback mechanisms and discuss its feasibility.

As the congestion control mechanisms become sophisticated, the diversity of information in the network desired by end-host is increasing. For example, the following explicit information is noticed by routers in recently proposed congestion control mechanisms.

- **bottleneck bandwidth** : This information is generally useful to calculate the upper limit of sending rate. For example, in PTP [11], a single PTP packet collects information from routers and determines the bottleneck bandwidth along the path from the sender to the receiver.
- **available bandwidth** : This information is generally useful for rate control. For example, TCP Quick-Start provides the mechanism that each router along the path reduces the requested rate indicated in the request packet to its available bandwidth if the requested rate is larger than available bandwidth.
- **queue size** : This information is generally useful to control a large congestion window or to calculate the upper limit of congestion window. TCP Limited Slow-Start [2] would benefit much if this information is available at end-hosts.
- **queue length** : This information is generally useful for congestion window control, especially for TCP modifications for High-bandwidth product networks. We include congestion indication, such as Explicit Congestion Control (ECN) [10] and AntiECN [9], in this category in a broad sense.
- **loss rate** : This information is generally useful for congestion window control, and would be also used to estimate the cause of packet losses in wireless environments. Explicit Transport Error Notification (ETEN) [8] includes a cumulative mechanism to notify end-hosts of aggregate congestion statistics along the path.

Fine-grained explicit router feedback could explicitly provide this information that routers actually measure. Though explicit information provided by routers is currently used for specific transport protocols, it could enhance the performance

Table 1: Parameters of Network Emulator System

function	parameters
delay and jitter	probability distribution, delay over 500 ms
packet loss	probability, burst loss
packet duplication	probability
packet reordering	probability
bandwidth control	bandwidth, algorithm
packet rewriting	comparison, mask bit, overwriting

of traditional congestion control without any router feedback, too. However, desired information that could be obtained from router feedback depend on the kind of transport protocols, and individual router feedback mechanisms are proposed independently. A unified fine-grained explicit router feedback framework that can be easily applied to any transport protocols would be useful, if the performance benefit exceeds the additional cost.

We believe the capability of developing such powerful routers using high-performance network emulator described in Section 3 make fine-grained explicit router feedback feasible to an agreeable degree if the additional cost is feasible.

3. NETWORK EMULATOR

To evaluate the performance of transport protocols, we implemented a high-performance network emulator. The network emulator is a hardware and software system and consists of a general-purpose network processor and two Gigabit Ethernet (GbE) ports. The core emulation software is implemented on a network processor, and we can flexibly adopt desired network models. The emulator has enough computation resource to provide additional pre-defined processing for all traversing packets at wire rate as a specification, so that we can develop a powerful gigabit PC router, where new explicit router feedback functions can be easily implemented and evaluated.

3.1 Specification

The network emulator supports a multi-network-clouds model. Table 1 shows the parameters we can set for each network-cloud. The emulator is controlled as per-packet basis.

- **delay and jitter** : We can set delay for each packet by microsecond order. Delay distribution is based on desired probability distribution. We can also set delay over 500 ms.
- **packet loss** : The emulator can drop packet at the desired rate $1/N$, where $1 \leq N \leq 2^{30}$. We can also set the number of burst packet losses (less than $2^{31} - 1$) in one congestion event.
- **packet duplication** : The emulator can duplicate a packet once every desired number of packets (less than $2^{31} - 1$).
- **packet reordering** : The emulator can replace a packet with the following packet once every desired number of packets (less than $2^{31} - 1$).
- **bandwidth control** : We can decide the bandwidth of each output link. We can also select bandwidth control algorithms such as Leaky bucket or Token Bucket.
- **packet rewriting** : The emulator masks the header of Ethernet, IPv4, IPv6, TCP, and UDP, and compares the header with data (pattern matching). Then the emulator overwrites control field for emulator (4 Byte) and Data field (8 Byte) in a header. After that, the emulator recalculates IP Header Checksum / TCP Checksum / UDP Checksum, if necessary.

The emulator assures wire rate throughput for a frame with a maximum size of 8KB.

The hardware price of evaluation board is \$5,000, but the

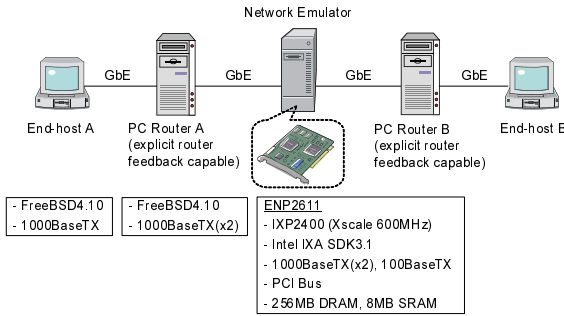


Figure 1: System Architecture of the Overall Network Emulation System

price of the product would be lower.

3.2 Architecture

We adopted Intel IXP2400 as a general-purpose network processor, and ENP2611 released by Radisys Inc. as IXP 2400’s development platform, respectively. ENP2611 has three 1000 BaseSX ports (only two of them are available for 1 Gbps) for packet processing, one 100BaseTX / 10BaseT port as a controller’s interface, PCI bus, and a serial interface. ENP2611 has 256MB DRAM (maximum: 1GB) mainly used for packet buffer and 8MB SRAM (maximum: 16MB) mainly used for control information buffer.

IXP2400 consists of two blocks; controller CPU and Micro Engine (ME) for packet processing. Xscale (600MHz) is adopted as controller CPU, and Linux and VXworks are adopted as OS. All the devices except for packet processing interfaces can be accessed from Xscale, and existing device drivers can be used without any modification. IXP2400 has eight 600 MHz MEs, and the MEs are given from Xscale instruction that defines procedures in MEs. In each ME, ME can cope with eight threads at a time. Packets processed by MEs are sent / received through a SPI-3 interface with 2.4 Gbps bandwidth.

Note that IXP2800 with a SPI-4 interface that can cope with 10 Gbps link bandwidth has been already announced. As IXP2800 belongs to the same product family and has similar architecture as IXP2400, transplantation of systems developed on IXP2400 to IXP2800 would be relatively easy.

Finally, we describe the system architecture of the overall network emulator system in Fig. 1. In the current state, the network emulator does not have any router functions. However, we vision the network emulator implemented with full router functions. We are working on this implementation and we could present the full implementation in the final paper.

3.3 Performance Measurement

We measured the performance of the network emulator. First, we measured jitter of delay in comparison with dummynet, which was well-known software-based network emulator. Then, we measured packet forwarding performance with additional UDP header processing. In our experiments, we set target delay to 100 ms. In the case of dummynet, we use a PC with 3.4 GHz Pentium 4 Xeon Processor. The tester we use can provide the measurement granularity of 10 ns.

In the first experiments, we generate IP traffic at the rate of 100 Mbps, and set packet size to 1518 byte. As a result, the emulator can forward all received packet with delay jitter from -60 ns to +150 ns. In the case of dummynet, we adjust a system clock to 100 μ s (we set FreeBSD kernel option HZ

to 10000). Dummynet can forward all the received packets with delay jitter from -101 μ s to +202 μ s. The average delay is 99.757 ms. These results show our network emulator has enough computation resource and can make more ideal emulation environments. Note that in dummynet, we can configure target delay only by millisecond order and can never achieve such precise jitter as the hardware-based network emulator.

In the second experiments, we generate UDP traffic at the wire rate of 1 Gbps, and set packet size to 1518 byte. The emulator can achieve wire rate (1 Gbps) forwarding with additional UDP header processing including masking, comparing, and writing.

These results show explicit router feedback with limited header processing functions at routers would be feasible in terms of implementation limitations. Note that we do not discuss in this paper whether congestion control with more sophisticated explicit router feedback involving control packet generation or complicated computation such as RTT calculation at routers such as that in XCP is feasible or not.

4. UNIFIED EXPLICIT ROUTER FEEDBACK FRAMEWORK

In this section, we show design rationale and then propose unified fine-grained explicit router feedback framework.

4.1 Design Rationale

The unified fine-grained explicit router feedback framework is required to be applicable to broader classes of transport protocols and to be able to provide fine-grained feedback information fundamentally required by end-hosts to design a novel congestion control mechanism or to significantly enhance the performance of traditional one in specific situations.

Considering these requirements and the granularity of required information in the network described in Section 2, we design a unified fine-grained explicit router feedback framework. The noticeable features of the framework are as follows.

(1) Interpretation of QoS semantics at end-hosts

To achieve flexibility to be capable to a broad class of transport protocols, routers are required to just provide measured or pre-configured information (QoS semantics) as explicit feedback, and interpretation of QoS semantics is required to be imposed on end-hosts. For example, when routers can provide QoS semantics on available bandwidth at the router, whether the value is smallest or not along the path is required to be judged by end-hosts, if possible. This design policy has another advantage to alleviate the load on routers.

(2) Per-packet processing

The framework like TCP Quick-Start that a packet requiring additional processing at routers (request packet) is generated only once in one RTT is vulnerable to DoS attacks such that the flood of request packets is generated by malicious users. To cope with such DoS attacks, routers are required to additionally process every packet without performance degradation.

(3) Maintenance of no per-flow state

In general, keeping per-flow state in routers causes a scalability problem (heavy load on routers) and synchronization difficulty.

(4) Applicability to multicast

If the unified feedback framework can support multicast congestion control, the framework would be more widely deployed. Fine-grained explicit router feedback is also useful for multicast congestion control, especially for receiver-driven multi-rate multicast congestion control. In this case, each end-host can decide its subscription level independently with-

```

struct oqhdr {
    u_char req_mode; // currently only "OQ_TTL" is defined
    u_char req_probe; // "OQ_LINK", "OQ_LOSS", "OQ_QUEUE" are defined
    u_char req_flag; // reserved
    u_char req_ttl; // specify the router to respond

    u_char res_probe; // same as req_probe
    u_char res_len; // length of response data
    u_char res_ttl; // TTL in the first response of the successive ones
    u_char oq_p; // protocol next to this header (TCP,UDP,...)

    struct {
        union u_oq_data data; // data specified by req_probe
    } req, res[OQRESLEN];
};

```

Figure 2: Header Format of the Unified Fine-grained Explicit Router Feedback Mechanism

out causing feedback implosion. TCP Quick-Start and XCP do not support multicast congestion control.

4.2 Overview

Based on the requirements described in Section 4.1, we design the unified fine-grained explicit router feedback framework. The key concept of the framework is that information at each hop (router) is successively but independently collected by end-hosts and the end-hosts grasp the network condition by interpreting the collected information. In this framework, all that routers have to do is only comparing TTL values and writing specified information into the header.

To realize the key concept, the original header is inserted next to IP header. Figure 2 shows the format of the original header format of the proposed mechanism. In the current implementation, we can collect three kinds of information; "OQ_LINK" (link bandwidth and available bandwidth), "OQ_LOSS" (packet loss rate and link error rate), and "OQ_QUEUE" (queue size and queuing delay).

The sender adds the original header to each request packet with setting `req_ttl` from the same value as TTL in IP header to 1 cyclically in the decreasing order. At the routers supporting the framework, `req_ttl` in the original header is compared with TTL in IP header, and if these values are same, the router writes information specified by `req_probe` into `data` field. Note that this request packet is only processed at this router along the path. The receiver writes the collected information together into `data` field like a stack in the original header in the response packet, and returns the packet to the sender. As a result, per-hop information can be eventually collected by end-hosts.

Note that the proposed framework can provide the same router functionalities as TCP Quick-Start and XCP (except for RTT calculation) with little performance degradation. There are two reasons of performance degradation. The first is that our framework requires N packets to collect router information along the path, where N is hop count from the source to the destination, while TCP Quick-Start and XCP can collect those information using only one packet. However, in high-speed networks with large flows, we could ignore the performance degradation because N packet are sent in succession at the rate-paced intervals. The second is that both TCP Quick-Start and XCP interpret QoS semantics at routers immediately when a packet is received, while our framework require at most one RTT to interpret QoS semantics until the packet arrives end-hosts.

5. EXTENSION OF TCP LIMITED SLOW-START

In this section, we show experiments to evaluate the efficiency of the framework. As one possible application of the unified explicit router feedback framework, we show the extension of TCP Limited Slow-Start [2]. In the extended Limited Slow-Start, a sender is informed of minimum queue size and available bandwidth along the path from the sender to a receiver using the unified explicit router feedback framework, and the sender determines optimal Limited Slow-Start parameters `max_ssthresh` and `ssthresh`. We implement and evaluate the extended Limited Slow-Start.

5.1 Limited Slow-Start

The current TCP Slow-Start and Quick-Start procedures result in increasing the congestion window (`ccwnd`) by thousands of segments in a single round-trip time. Such an increase can easily result in thousands of packets being dropped in one round-trip time, because the burst (which size equals `ccwnd`) causes queue overflow on the path and consequent retransmissions. It is difficult to implement precise rate-based pacing function on the current Network Interface Card (NIC) with software. Limited Slow-Start prevents this burst packet drops without rate-based pacing function.

In Limited Slow-Start, `ccwnd` is updated as follows:

```

For each arriving ACK in Slow-Start phase:
  If (ccwnd ≤ max_ssthresh)
    ccwnd += MSS;
  else
     $K = \text{int}(\text{ccwnd} / (1/2 \times \text{max\_ssthresh}))$ ;
    ccwnd +=  $\text{int}(\text{MSS}/K)$ ;

```

Thus, during Limited Slow-Start, `ccwnd` is increased by $1/K$ MSS for each arriving ACK, instead of by 1 MSS as in standard Slow-Start.

When `ssthresh` \geq `ccwnd`, Slow-Start phase is exited, and the sender is in the Congestion Avoidance phase. `max_ssthresh` is recommended to be set to 100 MSS [2], but there is no recommended value of `ssthresh`.

5.2 Experimental Setup

Experimental environments for evaluation in this section are as follows: Limited Slow-Start is implemented on NetBSD 2.0 PC with a 2.8GHz Pentium 4 Xeon processor and 1 GB RAM. The network emulator is configured to create a bottleneck link. In the network emulator, bandwidth limit and delay is set to 800 Mbps (leaky bucket) and 100 ms ($\text{RTT} = 200$ ms), respectively. In the experiments in this section, we preliminarily implement additional router functions of writing available bandwidth into the packet header on FreeBSD 5.1 PC with 2.8GHz Pentium 4 Xeon processor and 1 GB RAM. Value of available bandwidth is manually configured to 800 Mbps in advance in the router in Section 5.5. Note that the goal of the experiments in this section is not to evaluate the load on the router, so that these implementations have no effect on the performance of TCP NewReno. Throughput is measured on a receiver using iperf 1.7.0. MSS is 1460 Byte. Buffer management scheme is droptail.

5.3 Evaluations of Limited Slow-Start

In order to make clear the effect of the unified explicit router feedback framework, we first show the basic performance of original TCP NewReno with Limited Slow-Start in our assumed environments.

Figure 3 shows throughput (bps) of TCP NewReno with and

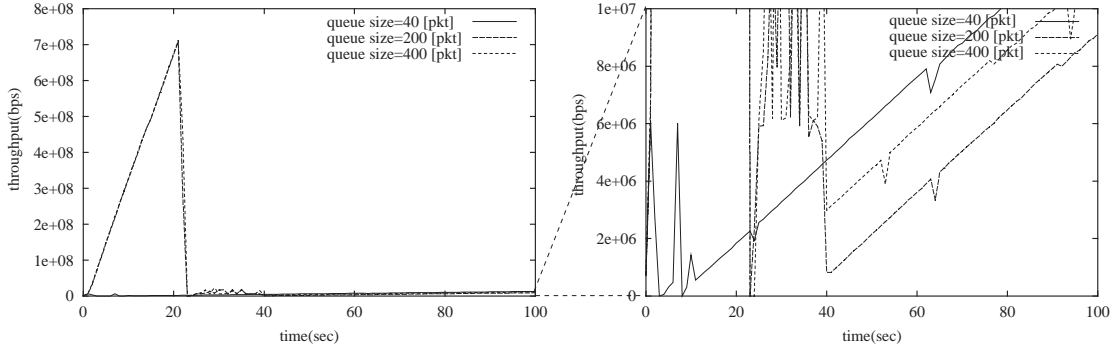


Figure 4: TCP NewReno with Limited Slow-Start. RTT=200ms, BW=800Mbps, max_ssthresh=500pkt.

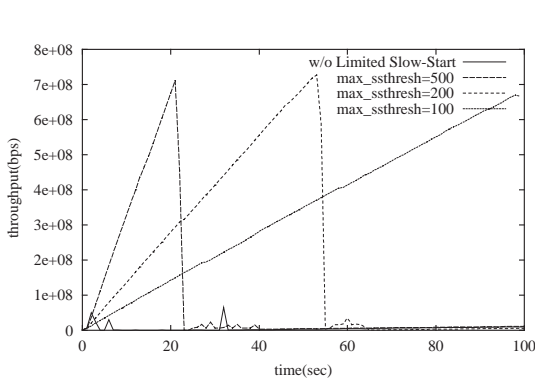


Figure 3: TCP NewReno with Limited Slow-Start. RTT=200ms, BW=800Mbps, Queue Size=200pkt.

without Limited Slow-Start when queue size is 200 packets. Clearly Limited Slow-Start can achieve much higher throughput than TCP NewReno without Limited Slow-Start, the value of `max_ssthresh` significantly affects the performance of TCP NewReno as shown in Section 5.1. So, ideally we should optimally determine `max_ssthresh` suitable for the network environment along the path.

Figure 4 shows throughput (bps) of TCP NewReno with Limited Slow-Start when `max_ssthresh` is 500 packets. When queue size is 40 packets, TCP NewReno with limited Slow-Start cannot achieve high throughput in Slow-Start phase because of queue overflow (burst packet drops) and consequent . On the other hand, when queue size is over 200 packets, TCP NewReno with limited Slow-Start can achieve about 700 Mbps. Thus queue size in the routers also significantly affects the performance of TCP NewReno.

5.4 Extended Algorithms

We apply the unified explicit router feedback framework to Limited Slow-Start to determine optimal `max_ssthresh` based on the informed value of minimum queue size (pkt) along the path.

We also apply the unified explicit router feedback framework to Limited Slow-Start to determine optimal `ssthresh` based on the informed value of available bandwidth (bps) along the path. The expected effect of this setting is that a sender can exit (Limited) Slow-Start phase and enter Congestion Avoidance phase without any packet drop, and can avoid unnecessary retransmit timeout after large number of packet drops.

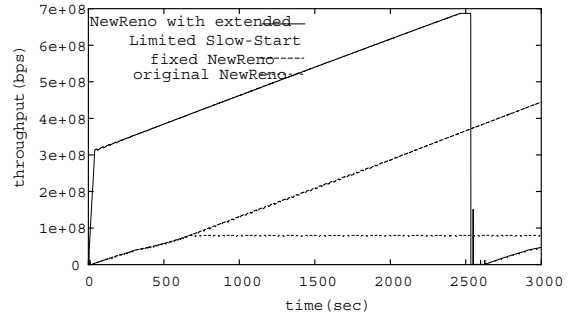


Figure 5: TCP NewReno with Extended Limited Slow-Start. RTT=200ms, BW=800Mbps, Queue Size=200pkt.

In the extended Limited Slow-Start, a sender is informed of minimum queue size and maximum available bandwidth along the path using the unified explicit router feedback framework, and the sender sets Limited Slow-Start parameters `max_ssthresh` (pkt) and `ssthresh` (pkt) to appropriate values, respectively. These parameters are set as follows:

$$\begin{aligned} \text{max_ssthresh} &= 1/2 \times \text{Queue Size}; \\ \text{ssthresh} &= 3/4 \times \text{maximum cwnd}; \end{aligned}$$

where

$$\text{maximum cwnd} = \text{Available Bandwidth} \times \text{RTT}/2 / \text{MSS}$$

Thus, TCP NewReno with Extended Limited Slow-Start can take the advantages both of Limited Slow-Start and TCP Quick-Start.

5.5 Evaluations of Extended Limited Slow-Start

Figure 5 shows throughput of TCP NewReno with Extended Limited Slow-Start. In this experiment, `max_ssthresh` and `ssthresh` is calculated and set to 100 and 5000, respectively.

We also show throughput of original TCP NewReno implemented on NetBSD 2.0 and TCP NewReno that is fixed the issue of neglecting decimal place of `MSS/K` as shown in 5.1.

6. CONCLUSIONS

In this paper, we study congestion control with explicit router feedback using the hardware-based high-performance network

emulator, where new explicit router feedback functions can be easily implemented and evaluated. We proposed the unified fine-grained explicit router feedback framework. Through implementation and preliminary experiments of the network emulator, we had an insight that the framework is feasible to an agreeable degree.

7. REFERENCES

- [1] S. Floyd. HighSpeed TCP for Large Congestion Windows. *RFC 3649*, Dec. 2003.
- [2] S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. *RFC 3742*, Mar. 2004.
- [3] A. Jain, S. Floyd, M. Allman, and P. Sarolahti. Quick-Start for TCP and IP. *Internet Draft, draft-amit-quick-start-03.txt*, Mar. 2004.
- [4] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. *Proc. IEEE INFOCOM '04*, Mar. 2004.
- [5] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. *Proc. ACM SIGCOMM '02*, Aug. 2002.
- [6] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM Computer Communications Review*, 33(2):83–91, Apr. 2003.
- [7] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). *Internet Draft, draft-ietf-dccp-spec-07.txt*, July 2004.
- [8] R. Krishnan, J. Sterbenz, W. M. Eddy, C. Partridge, and M. Allman. Explicit transport error notification (eten) for error-prone wireless and satellite networks. *Computer Networks*, 46.
- [9] S. S. Kunniyur. AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections. *Proc. ICC'03*, May 2003.
- [10] K. K. Ramakrishnan, S. Floyd, and D. L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. *RFC3168*, Sept. 2001.
- [11] M. Welzl. PTP: Better Feedback for Adaptive Distributed Multimedia Applications on the Internet. *Proc. IEEE IPCCC 2000*, Feb. 2000.
- [12] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control for Fast Long-Distance Networks. *Proc. IEEE INFOCOM '04*, Mar. 2004.