Improvement of a TCP Incast Avoidance Method for Data Center Networks

Kazutoshi Kajita†, Shigeyuki Osada†‡, Yukinobu Fukushima† and Tokumi Yokohira†
†The Graduate School of Natural Science and Technology, Okayama University
3-1-1, Tsushima-Naka, Kita-Ku, Okayama, 700-8530, Japan
‡The Japan Research Institute, Limited
2-18-1, Higashi-Gotanda, Shinagawa-Ku, Tokyo, 141-0022, Japan
E-mail: {kajita@net.cne., osada@net.cne., fukusima@, yokohira@}okayama-u.ac.jp

***Abstract*— In distributed file systems, a well-known congestion collapse called TCP Incast occurs because many servers send data to the same client and then many packets overflow the port buffer of the client link. Incast leads to throughput degradation in the network. In our previous work, we have proposed a method to avoid Incast. In the method, we limit the maximum number of simultaneously existing connections to a pre-determined constant value. However, we cannot use the method when the data size is small, and because we have not investigated how to optimize the maximum value, Incast may occur if the maximum value is not appropriate. In this paper, we improve the method so that it is applicable regardless of the data size and the maximum value is optimized. Numerical results show the effectiveness of our proposed method.**

***Keywords*— TCP, Data Center, Distributed File System, TCP Incast**

I. INTRODUCTION

In data center networks, distributed file systems where TCP is used as a communication protocol between a client and a server are very popular. In such distributed systems, a block of data is partitioned into several units called SRUs (Server Request Units) and they are stored into several servers. Thus, when an application on a client tries to read a block, the client sends requests to the servers which have the corresponding SRUs and then the servers almost simultaneously try to send the SRUs to the client. Then, because many packets burstly arrive at the client link, many packets are lost at the port buffer of the client link and consequently some servers have to wait the retransmissions of their lost packets until timeouts occur. In a standard TCP configuration, because the minimum timeout value is too large (the default value is 200 msec) compared to the round trip time (RTT) of data center networks (typically less than a few hundred micro seconds), it leads to serious throughput degradation. Such well-known congestion collapse is called *TCP Incast* [1] (we call it Incast briefly).

In order to avoid Incast, the papers [2] and [3] try to avoid timeouts using some strategies such as reducing the threshold value to trigger the fast retransmission mechanism and disabling the slow start. However, these strategies are not so effective because although Incast is often caused by losses of all packets of the send window size, the papers mainly focus on losses of some packets covered by the send window. The paper [4] proposes a method called ICTCP, which tries to avoid Incast by adjusting the send window size of servers. However, ICTCP is not effective when the number of servers is large as shown later. The paper [5] suggests reducing the minimum timeout value to a microsecond order value to mitigate the impact of timeouts. However, this method needs to modify the server OS's kernel timer into the higher resolution which is very hard task to do and can cause throughput degradation when the number of servers is large [3]. In addition, the paper [6] proposes a TCP protocol for data center networks called DCTCP, which can avoid Incast to some extent. However, because its main purpose is not to avoid Incast but to keep latency time low, Incast can occur when the number of servers is large [6].

Thus, in our previous work [7], in order to avoid Incast, we have proposed a method which limits the maximum number of simultaneously existing connections to a pre-determined constant value. In the method, if the number of currently existing connections is equal to the maximum number, a new connection is allowed to be established only when one of currently existing connections finishes. However, we cannot use the method when the SRU size is small. In addition, we have not investigated how to optimize the maximum value. Therefore, if the maximum value is not appropriate, Incast may occur, and the throughput is small even if Incast does not occur.

In this paper, we improve our previous method so that it is applicable regardless of the SRU size and the maximum value is optimized. In our previous method, each connection is serially established some interval time after its preceding connection is established. For small SRU size, because the interval time is not defined, we cannot use our previous method. In our improved method, we virtually consider some connections as one connection so that the interval time is defined for any SRU size and establish the connections simultaneously. The number of simultaneously established connection is optimized so that the throughput is maximized without Incast. Hereafter, we call our previous method the serialization method, and call our improved method the optimized serialization method.

The rest of the paper is organizes as follows. Section II describes a target data center network model, the cause of Incast and the serialization method. Section III describes the optimized method in detail. Section IV shows some numerical results to show the effectiveness of the optimized serialization method. Section V gives conclusion.

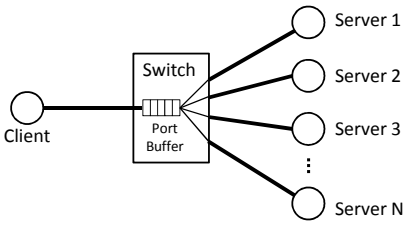


Fig. 1. Network Model

II. PREVIOUS TCP INCAST AVOIDANCE METHOD

A. Network Model

When we focus on the Incast problem, we can simplify data center networks to a network model shown in Fig. 1. There are one client and several servers. The client and servers are connected to one Ethernet switch. A port buffer is equipped in the client link, and some packets can be temporarily buffered when the link is busy. We can consider a network model where there are some switches and some servers and the client are connected to different switches. However, as long as the client link is a bottleneck, such model can be simplified to the model shown in Fig. 1.

B. Cause of TCP Incast

In distributed file systems such as HDFS (Hadoop Distributed File System) and pNFS (parallel Network File System), a block of data is partitioned into several units called SRU and they are stored in several servers. The default SRU size is 65536 KBytes ($K=2^{10}$) in HDFS and is 32 KBytes in pNFS.

When an application on a client tries to read a block, the client sends requests to all the servers which have the corresponding SRUs. Then every server sends the corresponding SRU to the client. The sendings from the servers are easy to occur almost simultaneously. Thus, because many packets from the servers almost simultaneously arrive at the port buffer of the client link, the port buffer is easy to overflow and some packets may be lost. When the communication between the client and each server is being done using TCP, almost all packets of the send window size of a server may be lost due to the huge burst arrival of packets to the port buffer. Thus, the fast retransmission and fast recovery mechanisms of TCP are not triggered because three duplicated ACKs are not returned to the server, and consequently such packet losses are recovered by the timeout mechanism only.

Although the minimum timeout value in a standard TCP configuration may be reasonable in normal network environments (its default value is 200 msec), it is too large in data center network environments, where the bandwidth is the order of Gbps ($G=10^9$) and the round trip time is less than a few hundred micro seconds. Thus, once a timeout occurs, retransmissions of the lost packets occur after a long waiting time, and consequently the delay until all the SRUs belonging to the block are completely received by the client becomes large. On the other hand, a new block read operation from the application does not occur until the current block read operation completely finishes, that is, until SRUs from all connections are received by the client (such application is called *a barrier synchronized application* [4]). Therefore once a timeout occurs, a long idle period appears in the client link, and consequently the average throughput over the period between the time of the sendings of the request from the client to the servers and the time of the finish of the receipts of all the SRUs is small.

C. TCP Incast Avoidance Based on Connection Serialization

As described in the previous section, the cause of Incast is that transmissions from many servers to a client simultaneously occur. Thus, in our previous work, we have proposed the serialization method, which limits the maximum number of simultaneously existing connections to a pre-determined constant value. In the method, if the number of currently existing connections is equal to the maximum number, a new connection is allowed to be established only when one of currently existing connections finishes. By the serialization, because packets from only a few connections arrive at the port buffer, Incast can be avoided. On the other hand, we allow any TCP connection which does not belong to such application to be established and there may be some traffic from UDP, ICMP, routing protocol and so on. However the total amount of such traffic (we call it background traffic) can be considered to be small compared to the traffic from barrier synchronized applications because background traffic is used for system control and management (in other words, we should avoid such system design that generates large amount of control and management traffic).

In the serialization method, we call the maximum number of simultaneously existing connections the overlapping parameter. When the parameter is one, we call the serialization method the complete serialization method, and when the parameter is two or larger value, we call the serialization method nearly complete serialization method. Next, we explain the two serialization methods.

In the complete serialization method, we completely serialize establishments of all the connections belonging to each application. The client first establishes one connection and receives the SRU. Then, the client repeats the same behavior for the second and the succeeding connections serially. In each connection, the client advertises BaseBDP [bits] as its receive window size to the corresponding server, where BaseBDP is the product of the client link bandwidth V [Gbps] and BaseRTT which is defined as the round trip time under the condition that there is no other traffic and TCP receive window size is one MSS. Note that we can easily obtain BaseRTT by a prior experiment for the target system. Because the client advertises BaseBDP as its receive window size, the send window size (sendwin) of each server is as follows.

$$\text{sendwin} = \min(\text{cwnd}, \text{BaseBDP}) \quad (1)$$

Where cwnd is the congestion window size of the server. Therefore in each connection, sendwin increases exponentially in the initial stage because cwnd increases exponentially due to the slow start mechanism of TCP, and then it reaches BaseBDP as shown in Fig. 2(a). The length (T_1) of the initial stage, that is, the period between the time origin and the time sendwin reaches BaseBDP, depends on V, BaseRTT and MSS, and it is derived as follows.

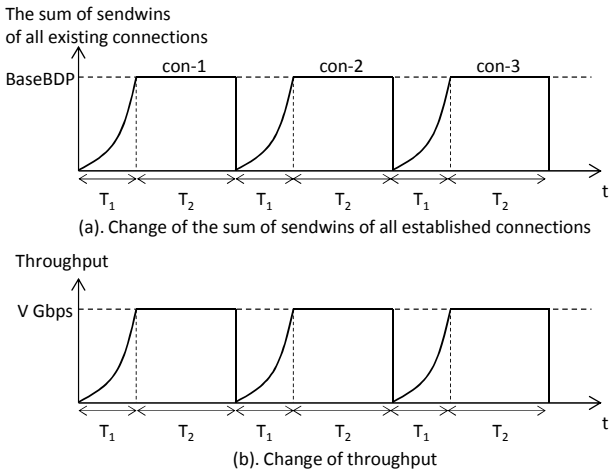


Fig. 2. Complete serialization

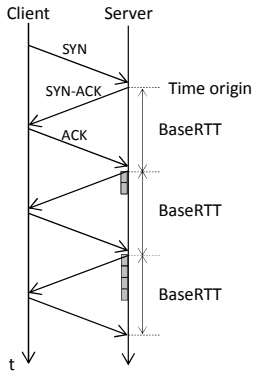


Fig. 3. Timing chart in client-server communication

We assume that the client starts connection establishment (the client sends a SYN segment) as shown in Fig. 3 and the initial cwnd value is two, and we assume that the time origin is the time when the server returns the SYN-ACK segment. Then, T_1 is given as follows.

$$\text{BaseBDP} = V \times \text{BaseRTT} \tag{2}$$

$$\text{BaseBDP\_Pkt} = \frac{\text{BaseBDP}}{(\text{MSS} + 40) \times 8} \tag{3}$$

$$m = \lceil \log_2 \text{BaseBDP\_Pkt} \rceil + 1 \tag{4}$$

$$T_1 = (m-1) \times \text{BaseRTT} + (\lceil \text{BaseBDP\_Pkt} \rceil - 2^{m-2}) \times \frac{(\text{MSS} + 40) \times 8}{V} \tag{5}$$

The length (T_2) of the period between the time when sendwin reaches BaseBDP and the time when the connection finishes depends on V, BaseRTT, MSS and SRU size S and is calculated as follows.

$$T_2 = \frac{\left\lceil \frac{S - \sum_{i=2}^{m-1} 2^{i-1} \times \text{MSS}}{\text{MSS}} \right\rceil \times (\text{MSS} + 40) \times 8}{V} \tag{6}$$

If the SRU size is too small, a connection finishes before its sendwin reaches BaseBDP and T_1 and T_2 are not defined.

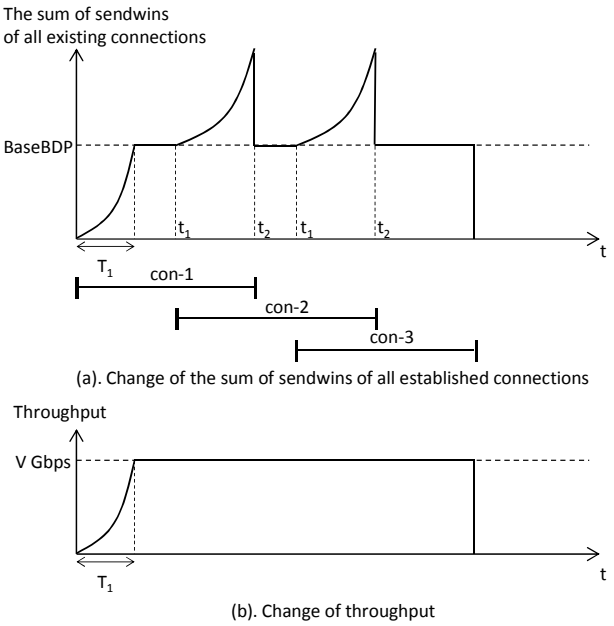


Fig. 4. Nearly complete serialization

Specifically if and only if the following inequality is not satisfied, T_1 and T_2 are not defined.

$$\left\lceil \frac{S}{\text{MSS}} \right\rceil - \sum_{i=2}^{m-2} 2^{i-1} + 2^{m-2} \geq \lceil \text{BaseBDP\_Pkt} \rceil \tag{7}$$

For example, when $V = 1$ Gbps, $\text{BaseRTT} = 100$ μsec, $\text{MSS} = 1460$ Bytes, S of 8 KBytes does not satisfy the inequality and S of 9 KBytes satisfies it. For larger bandwidth case, when V=10 Gbps, S of 115 KBytes does not satisfy the inequality and S of 116 KBytes satisfies it.

In the complete serialization method, because each connection does not overlap with the other connections, the sum of sendwins of all established connections is obtained as shown in Fig. 2(a). Although we cannot fully use the bandwidth of the client link for each period T_1, we can fully use it for each period T_2. Thus, as shown in Fig. 2(b), although we cannot obtain throughput of V Gbps for each period T_1, we can obtain throughput of V Gbps for each period T_2. Because T_1 does not depend on the SRU size and T_2 becomes larger for larger SRU size from Eqs. (5) and (6), the average throughput reaches close to V Gbps for larger SRU size. However, when the SRU size is small, because T_2 becomes small, the complete serialization method cannot fully use the bandwidth.

Thus, in the nearly complete serialization method, in order to avoid underutilized period as much as possible, we overlap two connections partially as shown in Fig. 4. In the method, we try to start the next connection at a time (t_1 in the figure) in advance so that sendwin of the connection reaches BaseBDP at the time (time t_2 in the figure) when the current established connection finishes. By overlapping two connections partially, the sum of sendwins of all established connections is obtained as shown in Fig. 4(a) and we cannot obtain throughput of V Gbps only first period T_1. Although the SRU size is small, the average throughput reaches close to V Gbps. However, it is

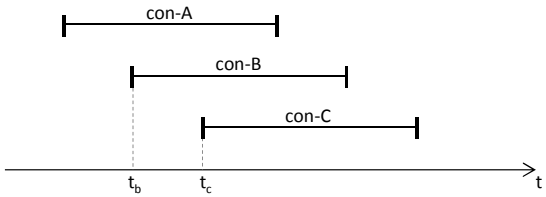


Fig. 5. Overlap of more than two connections

hard to derive time t_1 correctly because the two connections share the bandwidth of the client link. We determine time t_1 approximately as follows.

We call the current connection and the next established connection con-A and con-B, respectively. If we assume that only con-B exists in the network, sendwin of con-B can reach BaseBDP after the period of length T_1 in Eq. (5). However, con-A and con-B coexist in the network. Thus, we assume the two connections equally share the bandwidth of the client link and we consider that $2T_1$ is needed for sendwin of con-B to reach BaseBDP. Also from the equal sharing assumption, we consider that con-A can use half the bandwidth ($V/2$) of the client link averagely. Thus, con-A can send the data of T_1V ($2T_1 \times V/2$) [bits] during the period between t_1 and t_2. Therefore we can obtain time t_1 as the time when the number of bits which have been already sent in con-A reaches $S - T_1V$ (Recall that S is SRU size).

From the above discussion, we count up the number of bits which have been already sent in the current connection, and when the number reaches $S - T_1V$, we start the next connection (we call the condition that the number of bits in the current connection reaches $S - T_1V$ *the condition-I*). However, because $S - T_1V$ is an approximate value, if we start the next connection based on the condition-I only, the situation that there exist more than two connections may occur. For example, in Fig. 5, assume that because the condition-I is satisfied for connection con-A, connection con-B starts at time t_b. Then if the condition-I is satisfied for con-B at time t_c, a new connection con-C starts. But at the time t_c, con-A may have not finished yet and remain alive. Such situation is easier to occur when S is smaller. If we allow such multiple overlapping without any restriction, Incast may have occurred. Thus, we introduce the overlapping parameter K and we do not allow establishment of a new connection when the number of already existing connections is K, and when one of them finishes, we establish a new connection.

III. Improvement of the Previous Method

A. Idea for Improvement

In the nearly complete serialization method, we start the next connection when the number of bits which have been already sent in the current connection reaches $S - T_1V$. Thus, when the inequality (7) is not satisfied because the SRU size is too small, T_1 is not defined, and consequently we cannot use the method. The smallest value of the SRU size for the applicability of the method becomes larger for larger bandwidth of the client link. For example, when $\mathrm{BaseRTT} = 100\ \mu\mathrm{sec}$ and $\mathrm{MSS} = 1460$ Bytes, the smallest values for $V = 1$ Gbps and $V = 10$ Gbps are about 9 KBytes and about 116 KBytes, respectively as shown in the preceding section II-C. Even if the SRU size satisfies the inequality, when it is small, the method cannot fully utilize the client bandwidth for a small value of K and the method may cause Incast for a large value of K. Thus, although K should be optimized, the optimization of K has not been investigated in our previous work.

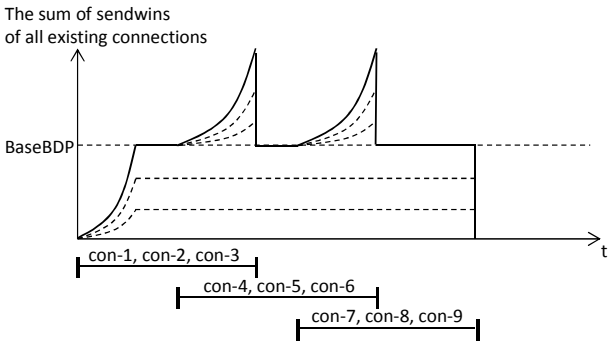


Fig. 6. Optimized serialization

In order to resolve these problems described above, when the SRU size is smaller than a threshold value, we adopt a strategy which establishes a set of several connections simultaneously while serializing such sets like the nearly complete serialization method as shown in Fig. 6. In the figure, the client establishes the set of three connections (con-1, con-2 and con-3) simultaneously. Then, the client starts the second set of the next three connections (con-4, con-5 and con-6) some interval time after the first set is started, and then the client repeats the same behavior.

In the strategy, the number (n) of connections in each set is selected so that each set can fully utilize the client link's bandwidth and at most two sets are overlapped as described later. The client advertises $\mathrm{BaseBDP}/n$ (it is rounded up in a unit of the packet size). Thus, the number of bits which queue in the client port buffer is at most about BaseBDP. Thus, using such buffer size, we can fully utilize the client link's bandwidth without causing Incast.

We call such connection serialization method *the optimized serialization method*. In the next section, we describe the threshold value to use the optimized method and how to derive the value of n.

B. Optimization of the Number of Simultaneously Established Connections

In the nearly complete serialization method, the period which is needed for sendwin of the next connection to reach BaseBDP is assumed to be $2T_1$. Therefore if $T_2 \geq 2T_1$, we can fully utilize the client link's bandwidth by overlapping at most two connections. From Eqs. (5) and (6), the inequality $T_2 \geq 2T_1$ is expressed as follows.

$$\left\lceil \frac{S}{\mathrm{MSS}} \right\rceil \geq 2 \times \mathrm{BaseBDP\_Pkt} \times \lceil \log_2 \mathrm{BaseBDP\_Pkt} \rceil + 2 \times \lceil \mathrm{BaseBDP\_Pkt} \rceil - 2 \quad (8)$$

Therefore, when the SRU size (S) satisfies the inequality, we simply use the nearly complete serialization method. On the other hand, when the SRU size does not satisfy inequality, we

consider each set of n connections as one connection in the nearly complete serialization method and define T_1 and T_2 of each set in the same way. Then, if $T_2 \geq 2T_1$, we can fully utilize the client link's bandwidth by overlapping at most two sets. The inequality $T_2 \geq 2T_1$ is expressed as follows.

$$n \times \left\lceil \frac{S}{\mathrm{MSS}} \right\rceil \geq 2 \times \mathrm{BaseBDP\_Pkt} \times \lceil \log_2 \mathrm{BaseBDP\_Pkt} \rceil + 2 \times \lceil \mathrm{BaseBDP\_Pkt} \rceil - 2 \quad (9)$$

Thus, we select the minimum value ($n_{\min}$) of n which satisfies the inequality (9) to decrease the number of bits which queue in the client port buffer as much as possible.

After $n_{\min}$ is derived, although the client advertises $\mathrm{BaseBDP}/n_{\min}$ (it is rounded up in a unit of the packet size), when $\mathrm{BaseBDP}/n_{\min}$ is smaller than four packets size and if one packet loss occurs, the fast retransmission and fast recovery mechanisms are not triggered because three duplicated ACKs are not returned to each server for small send window size. As a result, Incast may occur. To avoid such situation, the client advertises at least four packets size, and when $\mathrm{BaseBDP}/n_{\min}$ is smaller than four packets size, we adjust $n_{\min}$ as follows again.

$$n_{\min} = \left\lfloor \frac{\mathrm{BaseBDP\_Pkt}}{4} \right\rfloor \quad (10)$$

As described above, although at least $n_{\min}$ connections are established in the almost entire period of an application, the number of remaining connections may smaller than $n_{\min}$ in the final period of an application. In the final period, if we continue to use the advertised window of $\mathrm{BaseBDP}/n_{\min}$, we cannot fully use the client's link bandwidth. To avoid such situation, when the number of remaining connections is smaller than $n_{\min}$, the client increases the advertised window sizes of remaining connections so that the sum of sendwins becomes BaseBDP to fully use the bandwidth.

IV. Performance Evaluation

We incorporated the optimized serialization method into the NS2 simulator [8] and performed extensive simulation runs for every combination of the parameters shown in Table I. In the simulation, we assume that there are no bit errors in packets and there is UDP traffic with a constant bit rate from each of N servers (for example, when the rate is x Mbps, each server generates UDP traffic with the constant bit rate of x/N Mbps) as a background traffic.

Fig. 7 shows example results when the client's link bandwidth is 1 Gbps, the SRU size is 32 KBytes, the port buffer size is 40 packets and background traffic of 10 Mbps (0.1%). In Fig. 7(a), goodput is the value of throughput (application level throughput) when the sum (40 Bytes) of the TCP and IP headers are not included in throughput calculation. We also set such simulation parameters that MSS is 1460 KBytes, MTU is 1500 KBytes and the overhead of layer 2 or lower is zero. An active server means a server which has an SRU of a requested block. That is, when the number of active servers is 20 and the SRU size is 32 KBytes, we randomly select 20 active servers from N servers and we assume that the client requests the SRU of 32 KBytes to each active server (the size of the request block is $32\ \mathrm{KBytes} \times 20$).

TABLE I. Simulation parameters

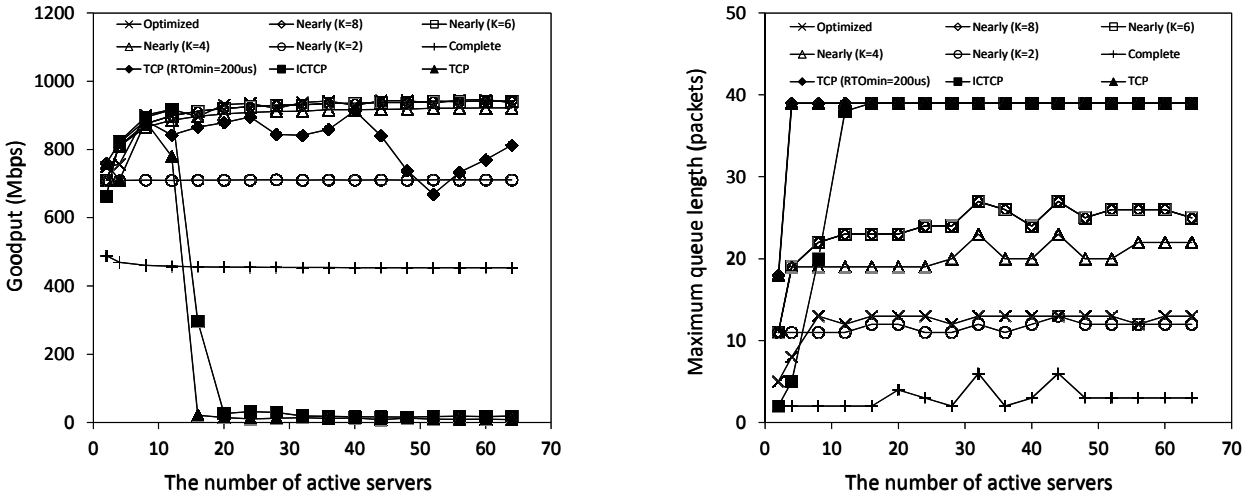
Parameter	V=1Gbps	V=10Gbps
BaseRTT (μsec)	100	
SRU (KBytes)	32, 64, 128, 256, 512, 1024	
Port buffer size (packets)	40, 80	200, 400
Cluster size (N)	64	256
The number of active servers	1～64	1～256
TCP RTOmin	200msec, 200μsec (fine-granularity timer)	
UDP background traffic rate (%)	0, 0.1, 1, 10	

In Fig. 7(a), we can see the highest goodput in the optimized serialization method, and in Fig. 7(b), the method attains a small maximum queue length (about BaseBDP_Pkt) because the method overlaps only two sets of connections. In the complete serialization method, as we suggested before, we cannot fully use the bandwidth when the SRU size is small. In the nearly complete serialization method, although goodput becomes larger when we set the large parameter ($K = 4, K = 6$ and $K = 8$), maximum queue length becomes large. Thus, Incast may occur when the port buffer size is small. In TCP and ICTCP, we can see the drastic performance degradation when the numbers of active servers exceed about 16 by Incast. Although ICTCP decreases the send window size to avoid Incast, its minimum value is 2 MSS. Thus, when the number of active servers is larger than about 20 (the port buffer size of 40 packets / the minimum send window size of 2 MSS), Incast can occur with high probability. By contrast, TCP with a fine-granularity timer (200μsec) largely improves goodput. However, when the number of active servers is large, we can see goodput degradation because timeouts repeat in a short span due to a heavy congestion.

Fig. 8 shows larger bandwidth (10 Gbps), when the SRU size is 512 KBytes, the port buffer size is 200 packets and background traffic of 100 Mbps (0.1%). In the figures, the optimized serialization method obtains the highest goodput and attains a small maximum queue length. In the nearly complete serialization method, as we suggested before, although goodput becomes large when we set the appropriate parameter ($K = 4$), Incast occurs when we set larger parameter ($K = 6$ and $K = 8$). In the other methods, we can see almost the similar performance as Fig. 7.

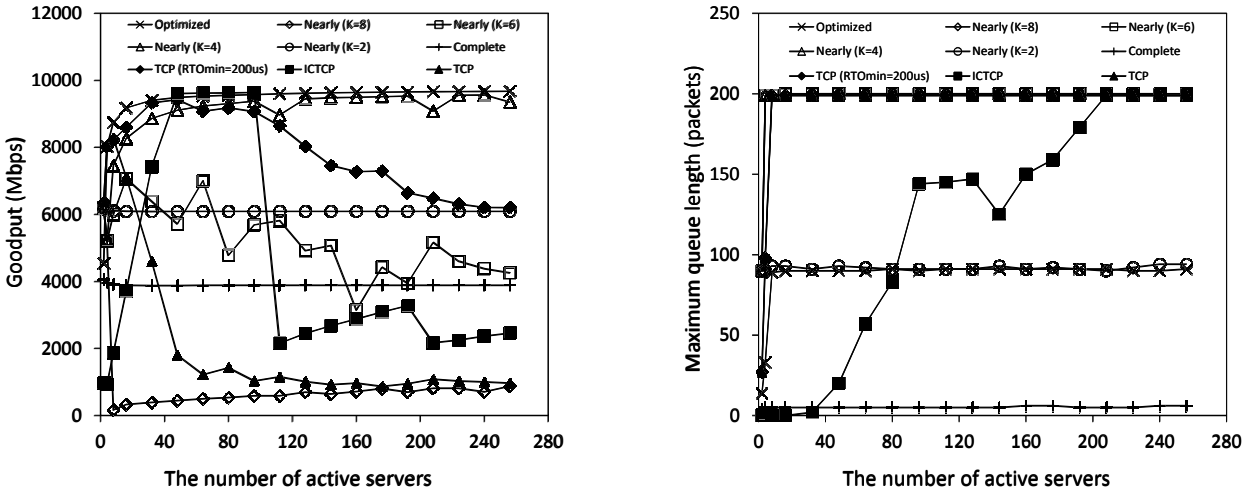
V. Conclusion

We have proposed an improved method of the serialization method to avoid TCP Incast. In evaluation results, the proposed method showed high throughputs and attained small queue lengths of the port buffer while avoiding Incast. In particular, when the bandwidth becomes larger, we show the effectiveness of the method compared to the previous methods. One of our future work is to investigate the effectiveness of the method in real systems.



(a). Goodputs (b). Maximum queue lengths

Fig. 7. Goodputs and maximum queue lengths when V=1Gbps



(a). Goodputs (b). Maximum queue lengths

Fig. 8. Goodputs and maximum queue lengths when V=10Gbps

REFERENCES

[1] D. Nagle, D. Serenyi and A. Matthews, "The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage", IEEE/ACM Supercomputing 2004, pp. 53-62.

[2] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems", The 6th USENIX Conference on File and Storage Technologies (FAST 2008), pp. 1-13.

[3] Y. Chen, R. Griffith, J. Liu, R. H. Katz and A. D. Joseph, "Understanding TCP Incast Throughput Collapse in Datacenter Networks", ACM WREN 2009, pp. 73-82.

[4] H. Wu, Z. Feng, C. Guo and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in Data Center Networks", ACM CoNEXT 2010.

[5] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication", SIGCOMM 2009, pp. 303-314.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Parabhakar, S.Sengupta and M. Sridharan, "Data Center TCP (DCTCP)", SIGCOMM 2010, pp. 63-74.

[7] S. Osada, K, Kajita, Y. Fukushima and T. Yokohira, "TCP Incast Avoidance Based on Connection Serialization in Data Center Networks", APCC 2013, pp. 120-125.

[8] The Network Simulator-ns-2. http://www.isi.edu/nsnam/ns/.