國立清華大學

碩士論文

適用於高可用度叢集錯誤轉移事件之 TCP 效能增進機制

TCP Performance Enhancement Schemes in Stateful HA Cluster for Failover Events

系 所 別：　資訊工程研究所

姓名學號：　吳彥敏　(Yen-Min Wu)　9562564

指導教授：　黃能富 博士　(Prof. Nen-Fu Huang)

中 華 民 國　九十七 年　八 月

ABSTRACT

An HA cluster is a clustering system which provides critical network services with high level of availability by redundant hardware and failover procedure. When the active cluster node fails, the backup node would take over the responsibility of active node through failover process. In this way, single point of failure is removed, and overall system downtime is then effectively reduced.

TCP composes more than 90% of Internet traffic [1], and is most widely used to carry critical service. However, the behaviors of TCP flows are not considered within the design of HA cluster. During failover period, the packet losses experienced by active connections are treated as congestion signals by TCP congestion control, and thus reducing the flow sending rate. Also, follow the design of TCP, after failover, the interrupted flow could not resume its transmission until another retransmission timeout.

In this thesis, we propose a local recovery mechanism for high availability cluster (HALR). At ingress/egress switches of HA cluster, the pass-through packets are selectively cached and locally resent after failover process. In this way, the interrupted flows could be restarted upon failover completion. On the other hand, for the active/active cluster, or the clusters with strings of unequal bandwidth, to address the potential bandwidth bottleneck problem after string failure, the post-failover rate control (PFRC) mechanism is further proposed. The mechanism could keep active connections transfer in a steady manner even with insufficient available bandwidth.

From our simulation results, when HALR is deployed, the unused flow time could be eliminated. Moreover, when system bandwidth bottleneck appears after failover, utilizing the rate control mechanism could achieve steady system throughput, fair bandwidth allocation among flows, and bottleneck buffer usage reduction.

中文摘要

高可用度叢集(HA Cluster)，是一種針對關鍵性的服務保障其高可用性的叢集系統，其主要保護的對象，為提供關鍵服務的伺服器，稱為叢集節點。藉由備援軟硬體的提供，叢集節點可以組成一個群組。當群組中任何一個關鍵元件故障時，服務並不會就此中斷，而是經由故障轉移機制來啟動備援。這樣的備援機制可有效移除單點失敗，並降低系統停止提供服務的時間。

TCP 傳輸控制協議，在網路上是最普遍，也是最常使用在關鍵服務的通訊協定。但是，TCP 連線的行為特質並未被考慮在高可用度叢集的設計中。故障轉移期間，TCP 封包將會漏失，而這些掉落的封包則被擁塞控制機制當成網路擁塞的訊號，並錯誤的啟動擁塞控制機制，降低傳輸速率。也使得故障轉移後，被中斷的連線無法立即重新開始傳輸。

在本篇論文中，我們提出一種近端回復的機制。此機制乃是藉由位於高可用度叢集兩端的負載平衡交換機來對通過其上的封包做選擇性近端快取，並在故障轉移完成後立即送出快取的封包，如此一來，即可達成有效率的近端回復機制。另外，對於雙作業模式的叢集，或是擁有不對稱鏈路頻寬的叢集，考量到故障轉移後可能發生頻寬不足的問題，我們接著提出了故障轉移後的速率控制機制，來保證錯誤轉移後 TCP 流量可持續穩定的傳輸。

模擬結果顯示，採用近端回復機制的叢集，能夠在故障轉移事件發生時，有效地避免浪費系統可用卻未被使用的時間。若進一步考慮故障轉移後，頻寬不足的情況，在加上速率控制元件之後，網路效能即可在僅受到輕微之影響的狀態下，達成故障轉移後 TCP 連線之間公平的頻寬利用，較穩定的系統吞吐率，並有效降低位於瓶頸的緩衝區使用量。

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

From users’ perspective, there is no difference between the service outages due to the networks and due to the servers. As a consequence, any near-source or near-destination single-point failure would hinder the service quality. According to [2], when network failures are considered, service availability is often as low as 99%, meaning that a server is out of service for about 15 min a day in average. To increase service availability, a High-availability (HA) cluster solution is often deployed. An HA cluster is a cluster of network devices or servers which could eliminate single-point of failure (SPOF) by redundant nodes and failover procedures. Thus, HA clusters provide better levels of availability fro critical services, such as four-9s or five-9s.

A typical structure of an active/backup HA cluster is illustrated in Figure 1. In order to maintain states of active connections in the system, state replication messages are exchanged between cluster nodes which provide the same function during normal operations. Heartbeat messages are periodically sent between ingress/egress switches in order to probe link/node failure within cluster. Whenever a failure is detected, the backup node would take over the responsibility of previous active node through the failover process. The failover process shorten the total system down time from node repair time, i.e., minutes to hours, to failure detection plus failover time, i.e., less than 10 seconds. This could strongly improve the availability of protected services.

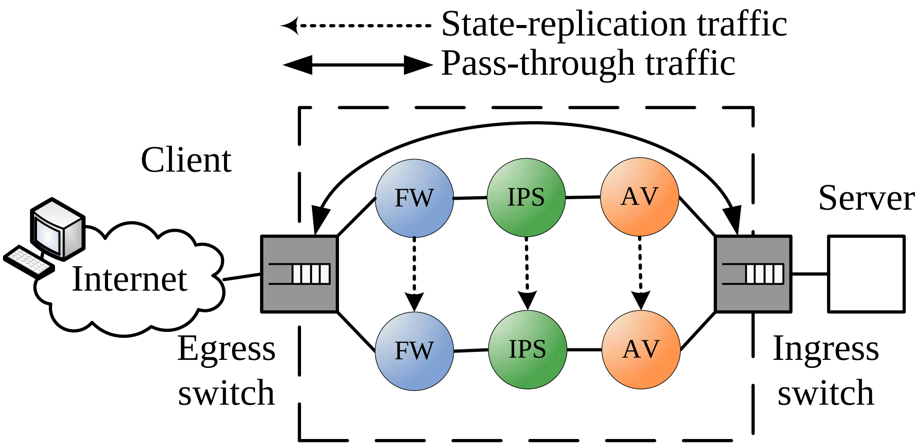

Figure 1. The structure of a high availability cluster

When failure happens on a cluster node, the failover process is required. Since the occurrence of cluster failovers could not be avoided, this results in temporary service unavailability. During this short period of system down time, packet drops are inevitable. Since the protocol used for critical services are mainly TCP flows, they are the target of failover protection. However, TCP treats packet drops as signals of congestion, and therefore lower the flow's sending rate to mitigate the level of congestion in the Internet [3]. This may falsely impact the performance of active flows interrupted by failover events. Observed from our previous experiments, for short-lived TCP flows, it is the series of retransmission timeouts which dominate the failover-affected flow completion time. In this way, the effects on TCP connections caused by failover process shouldn't be ignored.

Moreover, observed from the fact that the failover process between cluster nodes usually means total system bandwidth reduction after failover events, unless for each active string, the cluster has a fully dedicated backup string with equal bandwidth. When the bandwidth capacity is reduced, the flows may face a bandwidth capacity bottleneck, which would result in a saturated buffer, unsteady throughput, and unfairness between flows competing for the same queue. To address this problem, an adaptive rate control mechanism is required upon failover events to improve flow fairness.

In this thesis, we focus on performance enhancement of TCP flows interrupted by the failover process of an HA cluster. From the experiment results in our work, we have a better understanding of TCP congestion control behaviors under failover events with different failover times and packet drop types, such as doubling retransmission timeout value after timeout events and reduced window size. The enhancing mechanism is then designed through analyzing these behaviors.

In the following discussion, we consider HA clusters with two cluster nodes, which is the minimum requirement to provide redundancy. However, please be noted that the proposed mechanisms could also be applied to HA clusters with N (N>2) nodes.

The rest of this work is organized as follows: Chapter 2 introduces the backgrounds and related works, such as enhancing TCP flow performance by local recovery or rate control mechanisms. In Chapter 3, we propose the HALR design and explain it in detail; while in Chapter 4, we describe the characteristics of TCP flows during failover events. By extending current TCP flow models, in chapter 5, we capture the effects of failovers and local recovery mechanism in terms of flow completion time. Furthermore, Chapter 6 shows the rate control mechanism which protects post-failover flow performance. Then, the simulation results are presented in Chapter 7. Finally, the conclusion and future work are given in Chapter 8.

Chapter 2

BACKGROUND AND RELATED WORK

This chapter reviewed the related works of protecting TCP flows during failover events of HA Clusters. First, the background of HA Clusters and the characteristics of TCP congestion control are provided for better understanding the behavior of TCP flows during failover. Also, TCP enhancing mechanisms, such as explicit local recovery and rate enforcement in different network environments, are reviewed. These works are described in the following sections, respectively.

2.1 High Availability Cluster Modes

There are various HA techniques for different applications and network infrastructures. In this work, two major modes of standard HA clusters, active/backup (AB) mode and active/active (AA) mode [4], are considered and shown in Figure 2 and Figure 3. AB mode provides an idle node for redundancy, while AA mode is for load balancing as well as redundancy.

A cluster in AB mode is composed of one primary node (PN) and at least one idle backup node (BN). Through the dedicated link (the *replication link*) between the cluster nodes, the replicated states are used to reflect the active flows on the *pass-through link*. A cluster in AA mode is composed of two or more active nodes (ANs) sharing the pass-through flows, both playing the role of message sender and receiver in replication management. The configuration in AA mode removes a possible throughput bottleneck and is superior in the areas of load balancing capability with better resource utilization and a lower throughput penalty in the case of failure.

In general, an AN/PN can be backed up by a number of cluster nodes. However, in this work, we consider only the *single backup configuration* which is often used in practice.

In Figure 2 and Figure 3, on the boundary of clustering nodes, there is a pair of load-balancing switches (LBSes). In AA mode, the LBSes are responsible of distributing traffic loads evenly across two pass-through strings as well as ensuring that a connection passes through the same string in bi-direction. On the other hand, in AB mode, LBSes direct the traffic to a dedicated string. If a string failure is detected, LBSes redirect the traffic to the other string.

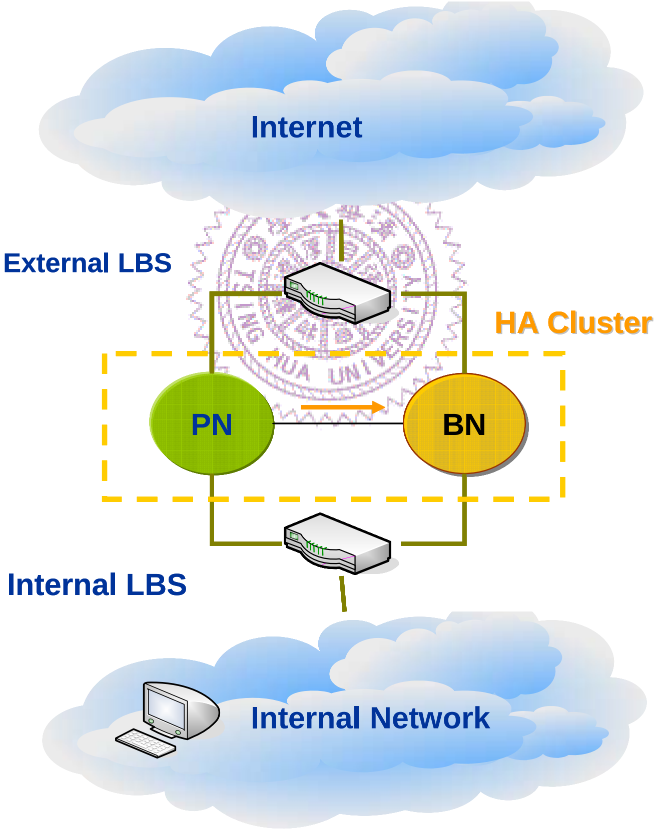

Figure 2. AB mode of Dual-String Stateful HA Cluster

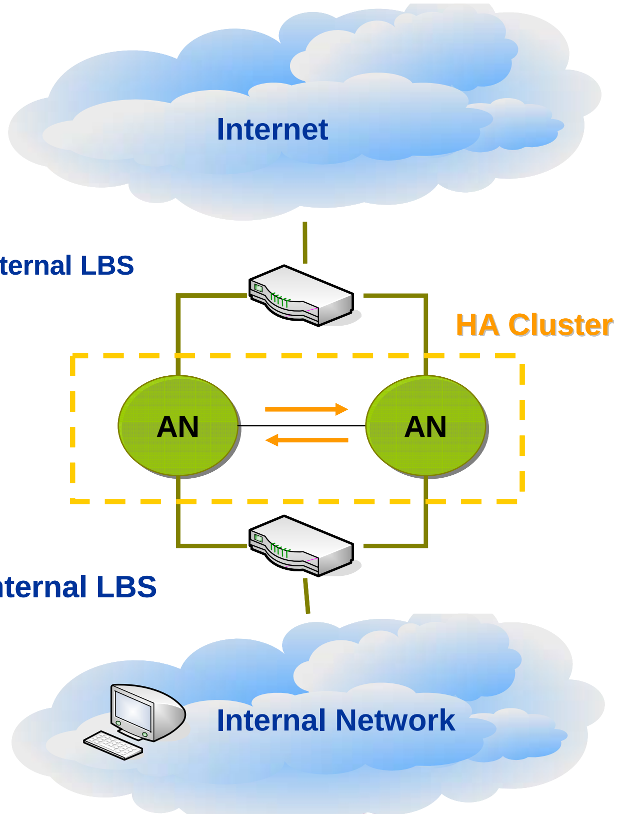

Figure 3. AA mode of Dual-String Stateful HA Cluster

2.2 TCP Congestion Control

TCP is a reliable transport protocol and it treats a packet loss as a signal of network congestion. In TCP, packet loss recovery is integrated into the mechanism of congestion control. For a TCP sender, a packet retransmission can be triggered by a retransmission timeout, 3-duplicate ACKs for fast retransmission, or a partial ACK during fast recovery.

TCP Reno and NewReno are the most popular implementations in the Internet [5, 6]. The fast recovery algorithm of Reno is optimized for the case when a single packet is dropped. However, Reno still suffer from unnecessary timeouts when multiple packet losses within a window of data [7, 8]. As the result, NewReno [9] is proposed to address this problem by modifying the mechanism of fast recovery. Moreover, to address the problem of consecutive losses when the dropped segments are non-contiguous, SACK [10] is developed. The receiver uses SACK option fields in TCP packets to notify the sender which contiguous blocks of packets are received successfully. The sender utilizes the block information to determine which segments are lost. Therefore, only the missing packets are retransmitted, and they could be retransmitted within a single RTT. In the case of a system failure and failover process, multiple packet losses are likely to occur. However, Reno, NewReno, and SACK perform exactly the same slow-start algorithm after a retransmission timeout. A timeout has a significant impact on TCP performance because it indicates a period of idle time; and the congestion window size (*cwnd*) is set to one segment when a timeout event is triggered. Finally, note that the value of retransmission timeout (RTO) should be no less than 1 sec [11]. Thus, when a failover process forces connections to timeout, the connection idle time will be at least 1 sec.

2.3 Local Loss Recovery Mechanism

In order to address the problem of packet loss due to lossy links and handoff, various TCP enhancement schemes have been proposed for wireless networks [12-15]. The group of Snoop-like protocols does not attempt to change or modify the TCP protocol itself. Based on link-layer knowledge of disconnection and duplicate ACKs for packet losses, TCP packets are buffered, removed, or delayed by a wire-wireless gateway.

In wireless environments, the disconnections shorter than RTO are considered as a less frequent case. In contrast, our works focus on the influences of a range of small failover time (less than 12 sec) and the memory requirements in high-speed networks. Furthermore, these Snoop-like solutions relied primarily on simulation and testbed experiments. Also, analytical modeling for the impact of handoff events on TCP performance was not presented in the previous works.

2.4 Rate Control Scheme

To control the sending rate of TCP flows by edge devices in different network environments, several mechanisms are proposed [16-22]. The main purpose of rate control is to make each active flow obtain their fair share of available bandwidth. To determine a proper sending rate under current network condition, these schemes require bandwidth monitors on input rate and target rate. Then, available bandwidth is fairly allocated to active connections.

There are two major approaches on allocating bandwidth: window sizing [16, 19, 20] and ACK pacing [18, 21]. The former controls the TCP sender’s sending rate by modifying receiver’s advertised window size in TCP acknowledgments, while the latter takes effect by buffering ACK packets in middle devices and release them according to the proper rate.

In this thesis, in order to address the potential bandwidth bottleneck after cluster failover, a window sizing approach is adopted at LBSes to control the flow rate. We select the approach of window sizing for its simplicity, and it does not require any additional queue to buffer ACK packets.

Chapter 3

HIGH AVAILABILITY LOCAL RECOVERY MECHANISM DESIGN

3.1 Overview

The main purpose of HALR is to eliminate idle time for flow transfers after failover. To achieve this goal, local kick-start mechanism should be deployed. We place this mechanism on the load balancing switches, because those switches keep track of the status of active node/string. HALR has the following two key operations: the first is to selectively cache the incoming packets during the normal status; the second is to locally resend the cached packets after the failover is completed.

For loss recovery, TCP regards all packet losses as notifications of traffic congestions, and then retransmits the unacknowledged segments for end-to-end reliability. However, when cluster failure happens between two TCP endpoints, retransmissions are not useful until the failover is finished. If the failover time, denoted as $T_{failover}$, is quite long, the TCP back-off algorithm may continues to double RTO until a threshold is reached or an ACK packet arrives. This mechanism causes a severe unused flow time (UFT) problem even the cluster is ready after a failover, the retransmission of the TCP sender is not performed until the scheduled time. This would result in temporarily low link utilization. In this work, we define an UFT as the time between the failover termination and the scheduled time for the next packet retransmission as shown in Figure 4.

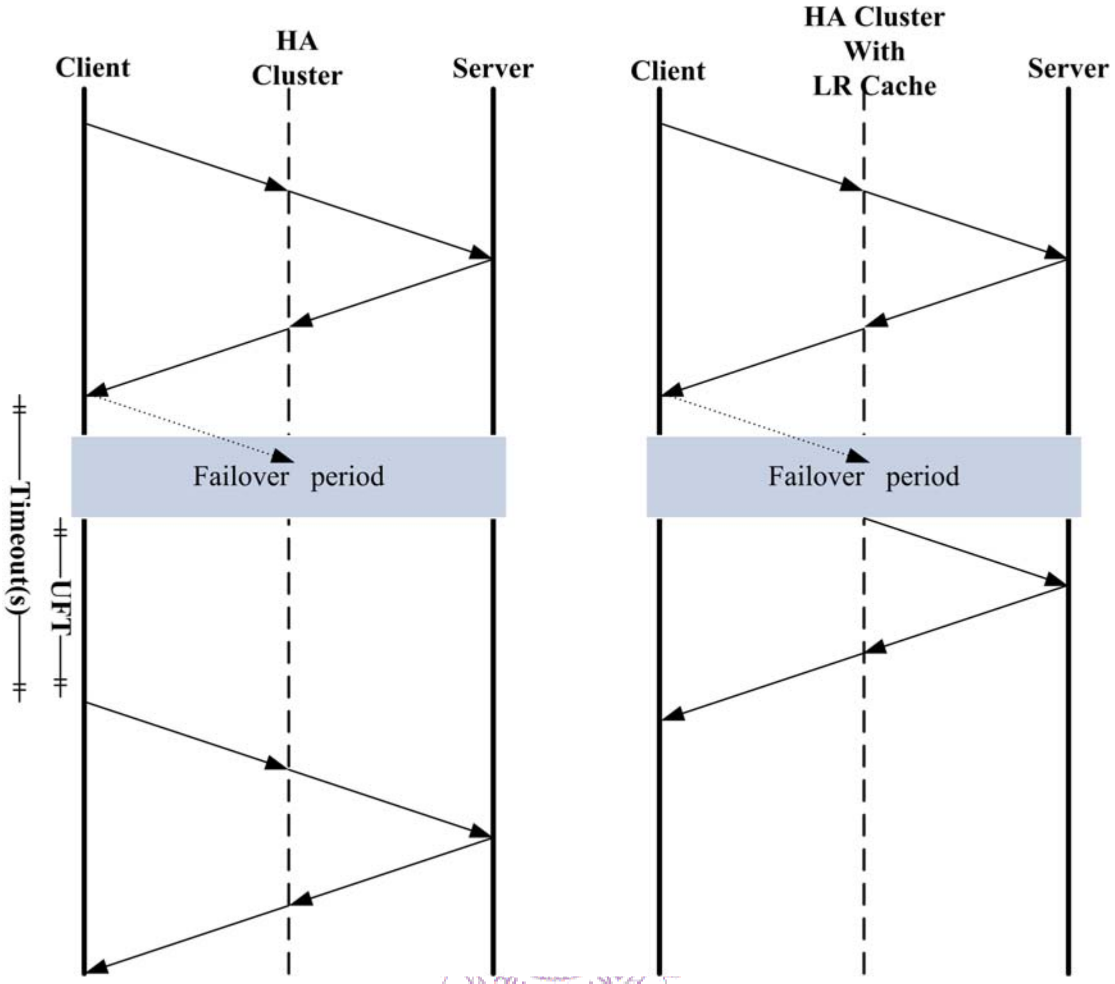

Figure 4. Unused Flow Time

For a short flow (say less than 500 ms), a failover-related TCP timeout may significantly lengthen the flow completion time. As a basic performance metric from end-user point of view, the flow completion time of a connection is its total transmission time. Obviously, lower latencies indicate better TCP performance. On the other hand, for a long flow with high-bandwidth, a failover-related TCP timeout causes a throughput degradation because the connection would take several round trip times (RTTs) for achieving the original throughput. For minimizing these failover-related effects, besides a small $T_{failover}$, it is necessary to minimize the possibility of TCP timeouts and the time delay caused by a failover. Thus, we propose the HALR scheme to enhance TCP performance over a failover event.

3.2 Design of HALR Switch

The basic idea behind the HALR scheme is that if packets are lost due to the transmissions across the cluster string during a failover, the edge switches (i.e., HALR switches) take the responsibility to recover these lost packets. In Figure 5, the HALR switch keeps track of state information from every pass-through TCP packets and saves certain numbers of unacknowledged data packets and ACKs into Local Cache (LC). Once the failover is finished, the switch performs local recovery by resending immediately the cached packets to the TCP endpoints that may be waiting for timeouts. After resending, cached packets could be released.

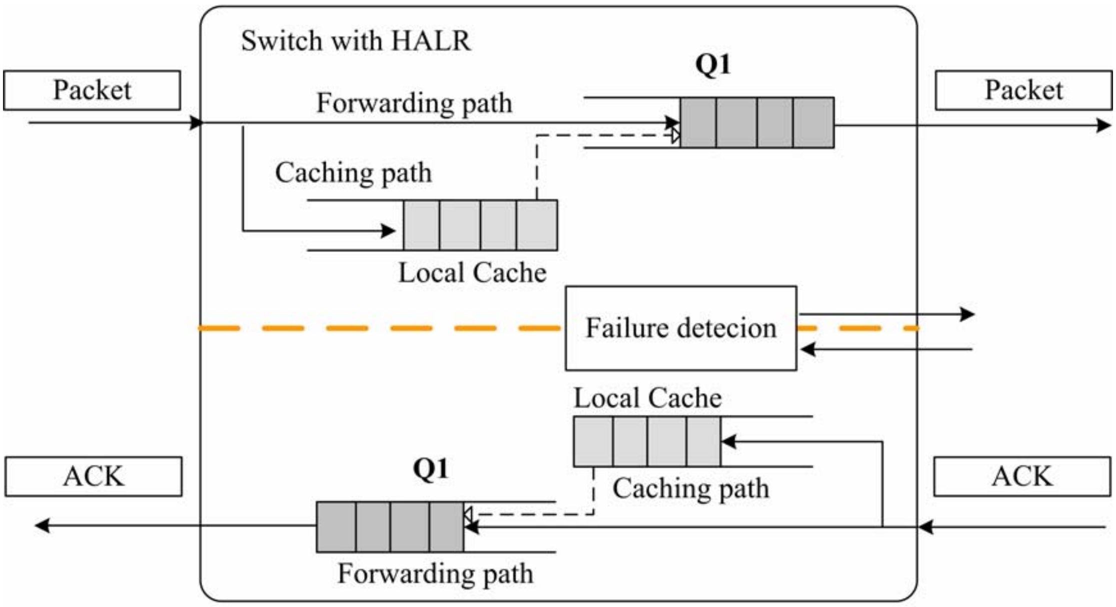

Figure 5. HALR Switch Design

As a result of resending, TCP implementations simply regard the resent data packet as a delayed segment of the connection and the resent ACKs fire up a quick retransmission immediately. Ideally, the resent packets minimize the possibility of issuing a timeout signal at the TCP sender and eliminate an unnecessary UFT. On the other hand, if the $T_{failover}$ was too long (e.g., over 9 min) and the connection was already closed, the TCP sender would ignore the resent packets.

3.3 Packet Caching Policies

The caching policies for the HALR scheme can be classified into two types according to whether or not the packet is going to traverse the cluster in the rest transmission.

First, for the packets leaving the cluster, the HALR switch replaces the buffered packet if the new packet has a higher acknowledgement number. When performing local recovery, each (ACK) packet is resent for at least 3 times for triggering the fast retransmission at TCP sender. If the fast retransmission is triggered before the scheduled RTO (i.e., a short $T_{failover}$), the TCP sender can thus retransmit immediately the lost packet indicated by 3-duplicate ACKs. On the other hand, if a timeout has already occurred in the TCP sender due to a longer $T_{failover}$, *cwnd* has also been set to one segment and then 3-duplicate ACKs are not useful to keep a larger *cwnd*.

Second, for the packets not yet entering the cluster string, the basic idea is to save unacknowledged packets into LC for resending them by local recovery. These resent packets will be acknowledged by the receivers and then the connections are woken up as normal. The first caching policy is to buffer all unacknowledged packets. Thus, an HALR switch may buffer a full window of packets (e.g., 64 Kbytes) in maximum for a single direction of one connection. Apparently, this policy may require lots of memory space in the HALR switch and result in high network bandwidth consumption when resending all cached packets. More importantly, it requires a packet-copy operation for each unacknowledged segment. It is well known that the memory copying should be performed as little as possible, especially in elementary procedures for every packet. This policy may be acceptable in wireless networks, but it incurs very large overheads in high-bandwidth networks. Thus, another policy is to save only one unacknowledged data packet (with the lowest sequence number). Though this policy may force the sender to retransmit a full window of data packets in worst case, the memory requirements and memory operations in the switch can be greatly reduced. Finally, the switch can only save SYNs received during the failover when $T_{failover}$ is less than the initial RTO for SYN loss. As discussed in Section 4.2, this policy resumes the transmission without waiting for the initial RTO and it further minimizes the memory requirement by ignoring the unacknowledged data packets and ACKs.

About caching timing, the HALR switches can save the pass-through packets when the cluster works well. However, the operations of memory copying can be significantly reduced if the caching is started before the failover process. Realize that a failover is usually declared after the heartbeat message losses in a row and exceed a small threshold (e.g., three losses). The purpose of this strategy is to make sure that the failover procedure is taken place only when it is necessary; we do not know whether a heartbeat loss is caused by a system crash or just a temporary high CPU load that makes the heartbeat unable to be delivered in time. Therefore, we propose to start the caching before a failure declaration but after at least two heartbeat losses in a row. Another advantage of this policy is to avoid the caching and resending for idle connections because only the active packets during the failover are saved.

In summary, in order to achieve the smallest memory requirement and memory operations, the combination of caching policies need to: 1) start to cache before a failure declaration, 2) for the packets not passing through the cluster, only save the packets with the highest acknowledgement number, and 3) for the packets passing through the cluster, save the lowest unacknowledged packets or save the SYNs only.

3.4 Memory Requirements for Packet Caching

We estimate the memory requirement for HALR by the trace-based simulation, which gives us a practical picture of memory requirement. The caching policy is to buffer two packets (one for data and the other for ACK) per connection in maximum. TCP stateful tracking is applied to the bi-directional 24-hour packet trace (denoted as AUCK-4) from NLANR [23] that was captured at the University of Auckland in 2001. By tracking the sequence/acknowledgement numbers and packet sizes of every active TCP connections, the maximum memory requirement of HALR can be estimated. In AUCK-4, the maximum buffer size is 792 Kbytes and the average size is 305 bytes per connection.

In practice, network traffic is time and link dependant and it is impossible to measure memory overheads by all traffic mixes. For example, suppose that the packet sizes for data and for ACK are 1,514 bytes and 64 bytes on a single connection. For caching 10,000 active flows, HALR will need to save 15.78 Mbytes. However, we believe that the memory requirement is quite likely to be much smaller when the HALR switch only saves the packets of specific connections, like the Top-100 heavy hitters. Finally, consider a high-end HA cluster for protecting a web site whose setup connection rate is 10,000 conn/sec and $T_{failover}$ is 100 ms. If only the SYNs received during the failover are cached, the memory requirement in this case is roughly equal to 64 Kbytes, which is not a concern for modern network equipments.

Chapter 4

OBSERVATION: INFLUENCES OF HALR ON TCP BEHAVIOR

In this section, we discuss the influences of introducing HALR into an HA cluster. Besides eliminating unused flow time (UFT), HALR may result in different behaviors of congestion control under some particular circumstances and thus affect the performance significantly.

4.1 Timeline around a Failover Event

Figure 6 illustrates the connections arrive around a failover process. Let t be the time instance of the timeline in Figure 6. Consider the following scenario: when $t = t_2$, a failover event occurs, and the process is finished at $t = t_3$, resulting in a $T_{failover}$ of $(t_3 - t_2)$. Also, t_1 is defined as "*one normal flow completion time* before the failover begins". On the timeline, the flows getting through the HA cluster could be categorized by the arrival time of their first SYN packet, $t_{arrival}$, as following:

- ***Case 1: Flow ended without affected by failover.*** If $t_{arrival} < t_1$ or $t_{arrival} > t_3$, the completion of the flows would not be postponed, that is, all flows arriving before t_1 or after t_3 will finish sending all segments within a normal completion time.

- ***Case 2: Data-packet loss due to failover.*** For the flows whose SYNs arrive between the time t_1 and t_2 ($t_1 < t_{arrival} < t_2$), i.e. less than one flow completion time before the failover event happens. The packet transmission would be interrupted before their termination.

- ***Case 3: SYN loss due to failover.*** For the flows that their SYNs arrive between the time t_2 and t_3 ($t_2 < t_{arrival} < t_3$), i.e. during failover period, the SYNs would be lost.

Based on whether the flow completion is delayed by the failover or not, the flows belonging to *Case 1* are called “*normal flows*” and the flows in *Case 2* and *Case 3* are called “*abnormal flows*”. We use L and L' to indicate “*normal flow completion time*” and “*abnormal flow completion time*”, respectively.

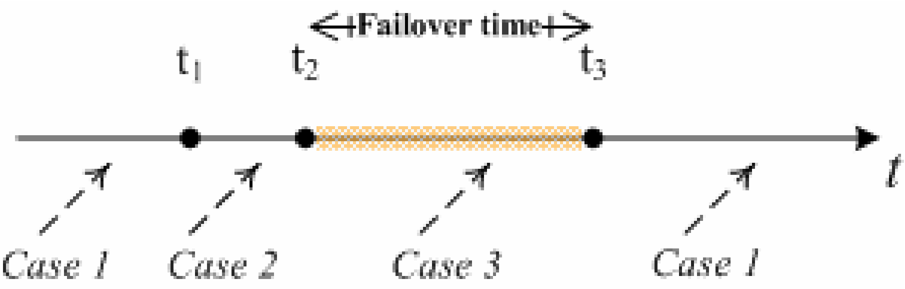

Figure 6. Timeline around a failover event

4.2 Analysis

First, we discuss the influences of introducing HALR on the flows belonging to *Case 2*. When $T_{failover}$ > RTO (e.g., 1.5 sec), TCP flows using TCP Reno, NewReno, and SACK could not avoid a timeout event. And, all the three variants employ the slow-start algorithm after timeout events. In this case, HALR can eliminate UFT by triggering the packet transmission before a timeout. Furthermore, note that two consecutive timeouts can result in performance degradation significantly and should be avoided. As shown in Figure 7, besides the idle period from the exponential back-off algorithm, after two consecutive timeouts, *cwnd* will be set as one and slow-start threshold (*ssthresh*) is set as two segments by the equation: *ssthresh* = max(*cwnd*/2, 2). As a result, after two packet transmissions, the sender enters the congestion avoidance mode and *cwnd* is limited to grow linearly, i.e., roughly one packet per RTT. As our evaluation results shown in Chapter 7, when HALR is enabled, TCP can avoid some of the retransmission timeouts that would otherwise be experienced. On the other hand, if $T_{failover}$ < RTO, even HALR resends 3-duplicate ACKs to the Reno sender, multiple packet losses within a window of data could still result in a timeout. Instead, for NewReno, HALR can trigger a fast retransmission and then partial ACKs received during fast recovery recover the lost packets without a timeout. Furthermore, HALR resending the unacknowledged packet to the NewReno receiver can accelerate the recovery from multiple packet losses.

Next, we discuss the impact of HALR on the flows belonging to *Case 3*. The initial *cwnd* is one and the initial RTO for SYN is usually set to 3 sec. Therefore, different from the flows of *Case 2*, when a single timeout occurs for a SYN loss, *ssthresh* is set as two. Furthermore, note that even with a small $T_{failover}$ (say 100 ms), the retransmission of a SYN loss must be triggered by a lengthy timeout (say 3 sec). Figure 8 shows that, with HALR, the flow resumes its transmission without waiting for a lengthy timeout. More importantly, the sender still remains in the slow-start mode that increases *cwnd* exponentially. In contrast, the flow without HALR increases *cwnd* linearly. Thus, if possible, the initial timeout for a lost SYN should be avoided, especially for short flows.

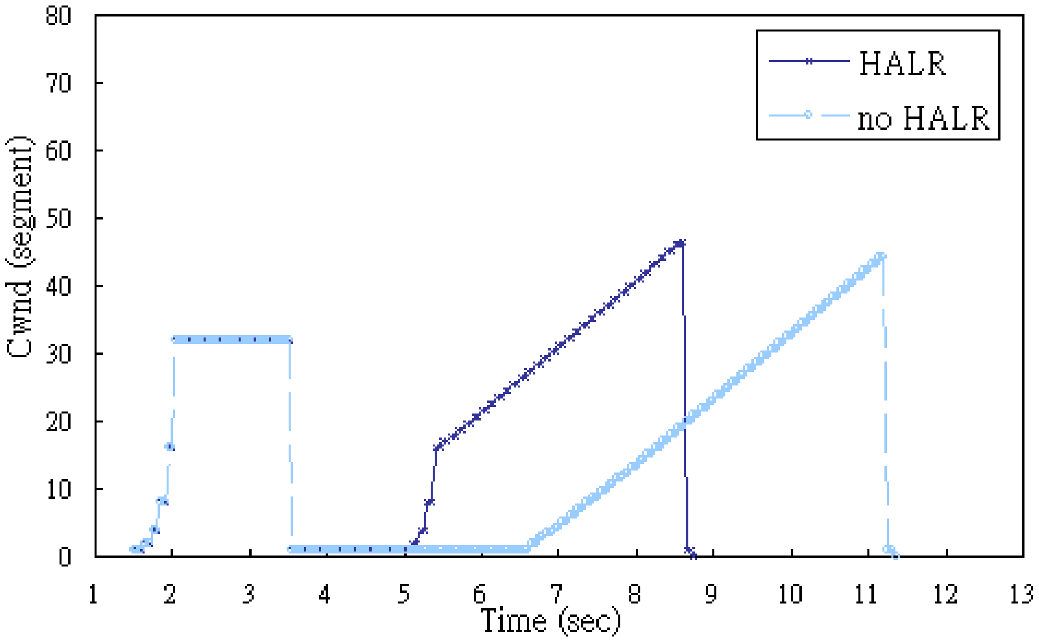

Figure 7. Congestion Window Evolution for Case 2: Data Drop Flows

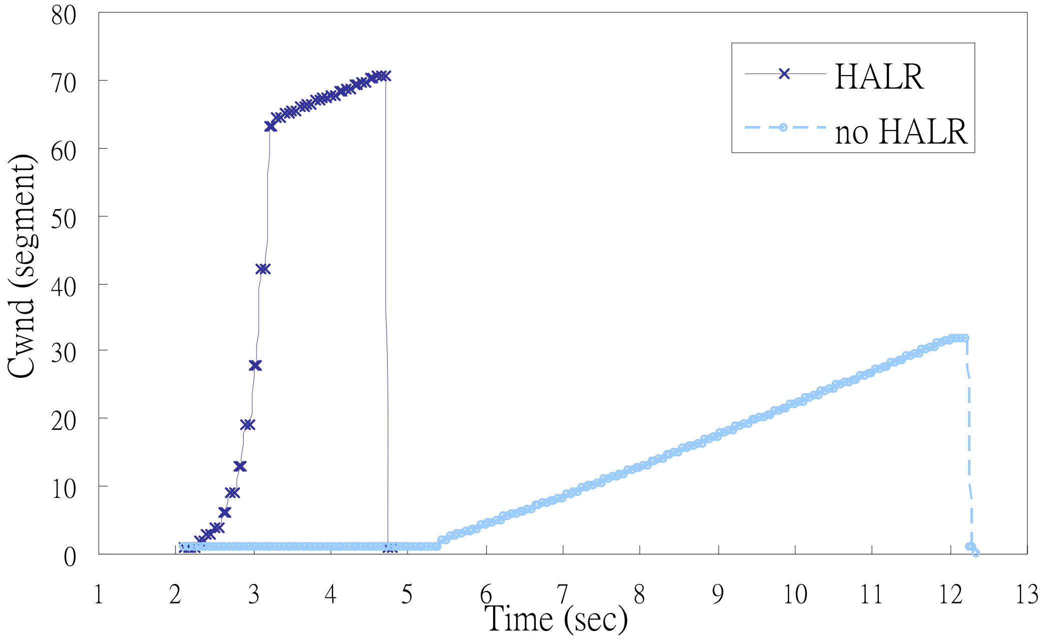

Figure 8. Congestion Window Evolution for Case 3: SYN Drop Flows

Chapter 5

MODELING FAILOVER FLOW COMPLETION TIME

In this section, we extend existing analytical TCP Reno flow models on transfer latency for both short flows [24] and long flows [25] to predict the effects of different failover times and the improvement by the HALR scheme. Table I lists the parameters used in the following discussions on the flow models.

Table I. Parameter list for modeling

Notation	Description	Value
L_{sim}	The length of evaluation	15 sec (t = 0–15)
$T_{failover}$	Failover time	50 ms–12 sec
p	Data segment loss rate	0.1%
p_s	SYN packet loss rate	0.1%
T_o	Initial timeout for data segments	1.5 sec
T_s	Initial timeout for SYN packets	3 sec
R	Average RTT	100 ms
D	Data segments in one flow	D = 9 for short, 1000 for long
W	Initial congestion window size	1
W_{max}	Max congestion window size	64

5.1 Additional Time Delay from a Failover

Through extending the modeling in [24, 25], we are capable of obtaining the abnormal latency, L'. The main idea is that when a connection encounters a failover event, it would be postponed by an additional delay. In *Case 2*, the additional delay is a series of timeouts T_o, while in *Case 3*, the additional delay is a series of T_s. The notation $AD(T)$ is used to denote the *Additional Delay* of a specific flow incurred from the failure event, given the failover period encountered by a specific flow, T_{down}. Note that *T* is the RTO value when flow interruption happens. On the other hand, T_{down} could be computed as the duration between the flow interruption time point (first packet drop due to failover) and the time point of failover termination.

$$T_{down} = (link\_failure\_time\_point) + T_{failover} - (1st\_blocked\_pkt\_arrival\_time\_point)$$

To compute the $AD(T)$ of a connection, assume that there are totally *x* times of timeouts triggered by a failover event, then the time period of the additional delay is equal to a series of x timeouts. $AD(T) = T + 2T + \cdots + (2^{x-1} \times T)$, while x is the minimum number satisfying: $\sum_{n=0}^{x-1} 2^n(T) \geq T_{down}$. However, if HALR is deployed, the delay incurred by a failover is not $AD(T)$ any more. Instead, the flow protected by HALR is simply delayed by T_{down}, which is shorter than $AD(T)$; namely, an UFT is eliminated.

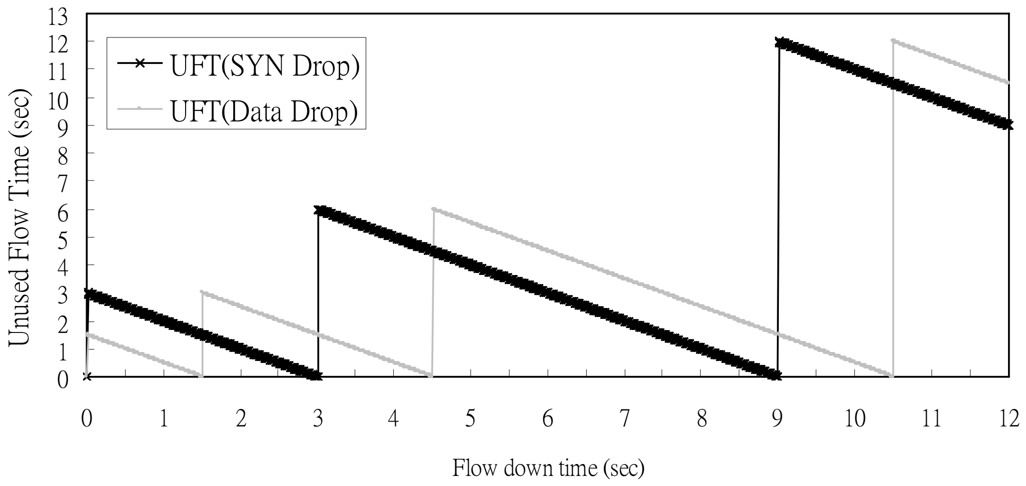

Figure 9. Unused Flow Time vs. Flow Down Time

Moreover, when HALR is not deployed, UFT can be directly computed from the timeouts and the down time experienced by the flow: $UFT = AD(T) - T_{down}$.

Namely, UFT is equal to a series of timeouts minus T_{down} . Thus, $0 \leq UFT \leq 2^{(x-1)}T$. Figure 9 further shows that UFTs have a “sawtooth” effect.

5.2 TCP Short Flow Models

In [24], the latency (flow completion time) of a single *normal flow* was modeled. The completion time is composed of two parts, C_s and C_m^w. Namely, $L = C_s + C_m^w$. C_s represents the connection establish time, while C_m^w is the average time needed for sending *m* data segments with an *initial* congestion window size *w*. The L could be computed recursively by the equation, $C_n^1 = C_1^1 + C_{n-1}^2$. For example, consider sending 3 data segments (D = 3) and *w* is equal to 1, the latency L would be $C_s + C_3^1 = C_s + (C_1^1 + C_2^2)$ and each component is computed as listed below.

$$C_s = R + T_s \tfrac{p_s}{1-2p_s}$$

$$C_1^1 = R + T_o \tfrac{p}{1-2p}$$

$$C_2^2 = Rq^2 + qp(T_o + R + C_1^1) + pq(T_o + C_1^1) + p^2(T_o + C_2^1)$$

Where R denotes the average RTT, T_o and T_s represent the initial timeout value for data segment and SYN packet, respectively. The p (p_s) is the drop/loss rate of data (SYN) packets and $q = 1 - p$.

For *Case 2* in Figure 6, consider a flow with D segments to transfer, assume that there are *i* packets arrive on a specific flow before the failover begins, but the flow hasn’t been completed yet, that is, *i* < D. In this case, subsequent non-SYN packets will be lost continuously until the failover succeeds. The L' belonging to *Case 2* could be computed through Eq. (1) for not deploying HALR and Eq. (2) for deploying HALR.

$$L' = C_s + C_i^1 + AD(T_o) + C_{D-i}^1 \tag{1}$$

$$L' = C_s + C_i^1 + T_{down} + T_{lr} + C_{D-i}^1 \tag{2}$$

In Eq. (2), we define the time required to locally resend the cached packets by an HALR switch as T_{lr}. Assume T_{lr} is a small delta when compared to L and L' and it can be ignored.

For *Case 3*, when there are flows trying to send SYNs to the servers during the failover, the SYNs would be lost continuously. After the failover, the retransmitted SYNs will be accepted and pass through the cluster. L' of *Case 3* could be computed through Eq. (3) for not deploying HALR and Eq. (4) for deploying HALR.

$$L' = AD(T_s) + C_s + C_D^1 \tag{3}$$

$$L' = T_{down} + T_{lr} + C_s + C_D^1 \tag{4}$$

5.3 TCP Long Flow Models

Note that the flow models mentioned above only cover the slow start phase of TCP connection. Also, the complexity grows exponentially when the number of total segments increases since all loss conditions need to be considered. Thus, the long-flow modeling is necessary to complement the short flow model. In [25], the total latency L of a TCP long flow is evaluated as a function of packet loss rate p, transfer size D, and average RTT, R. The basic equations of long flow modeling are listed below.

$$L = E[L_h] + E[T]$$

$$E[L_h] = R + 2T_s(\frac{1-p}{1-2p} - 1)$$

$$E[T] = E[T_{ss}] + T_{loss} + E[T_{ca}] + E[T_{delack}]$$

Similar to the above short flow models, the expected latency for long flows is composed of the expected time of connection establishment phase $E[L_h]$ and the time to transfer D data segments $E[T]$. The data transfer part of long flow, $E[T]$, is composed of four components:

I) $E[T_{ss}]$, the expected latency in slow start.

II) In the end of the slow start, packet losses occurs, T_{loss} is the cost for fast recovery or timeout losses.

III) After loss recovery, the connection enters congestion avoidance phase, the expected time period in congestion avoidance, $E[T_{ca}]$, is computed as the expected segments transferred in congestion avoidance, $E[d_{ca}]$ = $D - E[d_{ss}]$, divided by a steady state throughput equation in [26].

IV) At last, a cost $T[delack]$ is added to reflect the effect caused by delayed ack mechanism if *w* is 1. The cost is equal to 150 ms for MS Windows platforms and 100 ms for BSD systems.

For *Case 2* in Figure 6, if there is no HALR mechanism, we define the affected data transfer time as $E[T']$. Since we assume that the packet loss always happens in congestion avoidance phase, the affected flow latency L' could be modeled by modifying the expected time $E[T_{ca}]$ to $E[T^*_{ca}]$. Thus, the equations for computing L' of long flows can be written as following:

$$L' = E[L_h] + E[T'] + AD(T_o) \tag{5}$$

$$E[T'] = E[T_{ss}] + T_{loss} + E[T^*_{ca}] + E[T_{delack}]$$

After a data-packet loss, the connection would enter slow start phase again. The modified $E[T^*_{ca}]$ could be evaluated as below.

Assume that there are *j* packets arrive before the failover begins (*j* < D), there would be $D_{left} = D - j$ segments left to transfer after a failover, and the method to obtain the corresponding data segments sent in different phases after failover is quite straightforward.

Before failover, the number of segments transferred in congestion avoidance is $E[d'_{ca}] = j - E[d_{ss}]$.

After failover, depending on the total segments left to transfer, there are two possible conditions of an interrupted long flow: First, the flow finally ends in slow start; second, the flow leaves slow start phase and ends in congestion avoidance phase. In the previous case, the number of segments transferred in re-entered slow start phase is $E[d''_{ss}] = D_{left}$; and for the latter case, the number of segments transferred in re-entered congestion avoidance phase is $E[d''_{ca}] = D_{left} - E[d_{ss}]$. Similar to the relationship between $E[T_{ca}]$ and $E[d_{ca}]$, $E[T''_{ca}]$ could be computed as $E[d''_{ca}]$ divided by steady-state throughput, and so on. Without HALR, assume that $T_{failover}$ is longer than the flow packet inter-arrival time. The modified data transfer latency $E[T^*_{ca}]$ is given here:

$$E[T^*_{ca}] = \begin{cases} E[T'_{ca}] + E[T_{ss}] + E[T_{ca}''] & \text{for } D_{left} \geq E[d_{ss}] \\ E[T'_{ca}] + E[T_{ss}''] & \text{otherwise} \end{cases}$$

On the other hand, if HALR is deployed, the latency could be modeled as below. Note that although HALR could reduce the probability of timeouts, if $T_{failover}$ > RTO, the flow with data segment losses still enters slow start. However, if $T_{failover}$ < RTO, a flow could be restarted by 3-duplicate ACKs or the resent unacknowledged packets immediately. If HALR is deployed and the flow is restarted by 3-duplicate ACKs, $E[d'''_{ca}] = D_{left}$ is the segments transferred in congestion avoidance phase after failover. Below, two situations are considered separately. Note that HALR reduces the failover-related time delay to the flow from $AD(T)$ to T_{down}.

1. If $T_{failover} \geq T_o$

$$L' = E[L_h] + E[T'] + T_{down} + T_{lr} \tag{6-1}$$

2. If $T_{failover} < T_o$

$$L' = E[L_h] + E[T''] + T_{down} + T_{lr} \tag{6-2}$$

$$E[T''] = E[T_{ss}] + T_{loss} + E[T'_{ca}] + E[T'''_{ca}] + E[T_{delack}]$$

For *Case 3*, the affected latency for long flows is relatively simple. If HALR is not deployed, the latency could be computed through Eq.(7). Otherwise, if HALR is deployed, the affected latency equals to Eq. (8).

$$L' = AD(T_s) + E[L_h] + E[T] \tag{7}$$

$$L' = T_{down} + T_{lr} + E[L_h] + E[T] \tag{8}$$

Chapter 6

EMPLOYING RATE CONTROL AFTER FAILOVER EVENTS

6.1 Rate Control Motivation

Throughout previous discussions, we consider the case of active/backup clusters with strings of equal bandwidth. Using this specific setting, the system bandwidth remains unchanged after failover events. Therefore, the system would have enough bandwidth to process post-failover flows if a failover is happened. However, in practice, it is possible to have different bandwidth consumption after failover. For example, take a look at an active/active cluster, assuming two active strings have equal bandwidth. Then the capacity of system processing is halved upon link/node failure. This kind of situation would become even worse for the case of unbalanced link bandwidth, for example, the bandwidth of active (failure) string is twice as much of the backup (remaining) string. In these cases, after a failover event, when the failover flows restart transmitting packets, if the connection establishing rate is at the same level, it may face a post-failover bottleneck. When a bandwidth bottleneck exists, the buffer usage would grow up and finally being saturated, resulting in buffer overflows and therefore packet drops. Those packet drops due to buffer overflow would cause throughput fluctuations, longer flow completion times, etc. To address this problem, we employ an adaptive rate control mechanism to allocate a fair share of aggregate flow rate to each active flow.

When the bottleneck locates at the post-failover cluster node link, an adaptive rate control mechanism is vital. Such a mechanism could stabilize the transfer rate of connections passing through the cluster. It could not only reduce the buffer requirement and ensure flow fairness, but also achieve better user-perceived service quality.

6.2 Design of Rate Control Scheme

In this work, the window adjustment approach is utilized. According to the description in [27], the sending rate of TCP flows is determined by their window sizes, the actual window size is the minimum of congestion window and receiver advertised window(*awnd*), min(*cwnd*, *awnd*). Therefore, the flow rate could be effectively controlled by adjusting advertised window. Same as the local recovery mechanism, the rate control scheme could be employed on LBSes since each LBS has the full knowledge of when failover period begins and ends. As Figure 10 shows, a rate control module is attached to LBSes.

With the rate control module in hand, the next question would be: when to trigger the rate control? In order to deal with the potential bandwidth bottleneck after failover, the rate control module is activated upon receiving failure alert, until receive the link recovery notification of previous failed string. During this time period, a modified window value is computed periodically and used as an upper limit of receiver’s advertised window size for active pass-through flows. At the end of each pre-defined time interval, the feedback value is generated. For each incoming ACK packets, if the advertised window size exceeds the value computed by LBS, the receiver's advertised window would be reduced to the computed feedback value by modifying receiver advertised window field in this ACK packet.

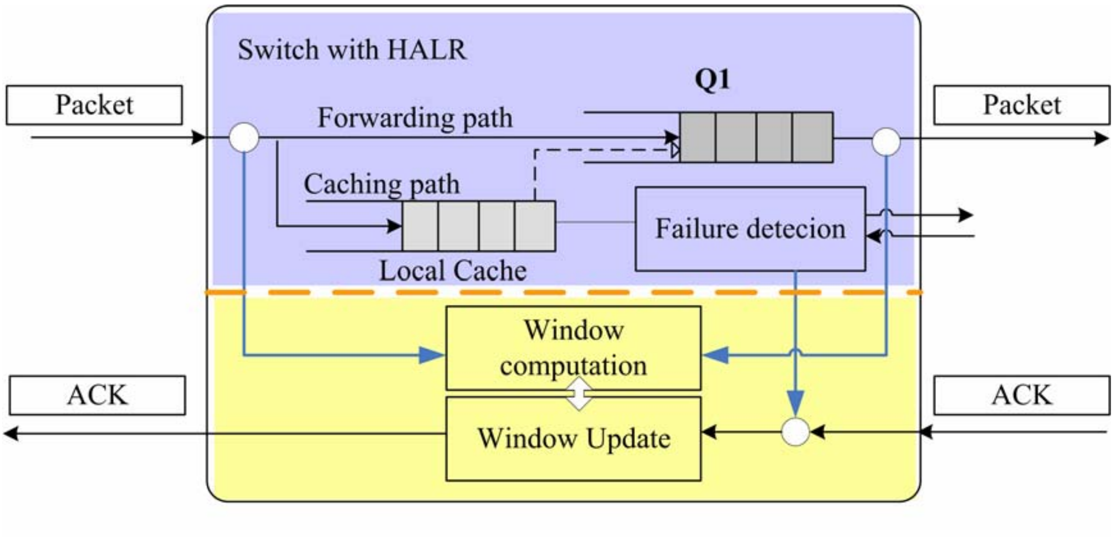

Figure 10. The integration of Rate Control Module on LBS

In the rate control module, the main operations could be divided into *window computation* and *window update*, the detail of window computation and update are described in the following section.

6.3 Rate Enforcement Method

When rate control is activated, a window modification function is carried out to pose a limitation on the pass-through flows. The window adjustment function is shown as below:

$$win_{controlled}(n+1) = win_{controlled}(n) - \alpha\left[\frac{In_{sampled} - \delta \times Out_{estimate}}{N_c \times (T_{gran} \div RTT)}\right] \qquad (9)$$

$$awnd(n+1) = max\{awnd_{min}, win_{controlled}(n+1)\} \qquad (10)$$

Where $win_{controlled}(n+1)$ is the computed window feedback value calculated at the end of nth interval, and will take effect during n+1th interval. In Eq(9),T_{gran} means the granularity of time interval, RTT is round-trip time, α is an adjustable parameter to add a weight to the monitored data during the latest monitoring interval, which is set to 1 by default, and N_c is the total number of currently active connections. $In_{sampled}$ is the total number of bytes received during the estimating time interval T_{gran}, and $Out_{estimate}$ represents total number of bytes that the cluster string is capable of processing over the interval T_{gran}. Which could be computed as link bandwidth $\times$ T_{gran}. The utilization factor δ is used to represent the target bandwidth utilization ratio (e.g. 95%).

Note that for all active connections, the same window feedback value is computed. Therefore, LBSes do not have to maintain per-flow state information. Also, in Eq(10), the term $awnd_{min}$ is used as a lower bound of window feedback. If the computed feedback value $win_{controlled}(n+1)$ is lower then the configured lower bound, the actual window feedback would be set as $awnd_{min}$. The main purpose of this control mechanism is to avoid a computed tiny window from taking effect, which may be severely harmful to the sending rate.

Chapter 7

SIMULATION RESULTS

7.1 Simulation Configurations

To prove the effectiveness of proposed mechanisms, the Berkeley's network simulator ns-2 [28] is used to simulate the different scenarios under active/active and active/backup HA Cluster modes. The general topology is depicted in Figure 11, which is the dumb-bell topology.

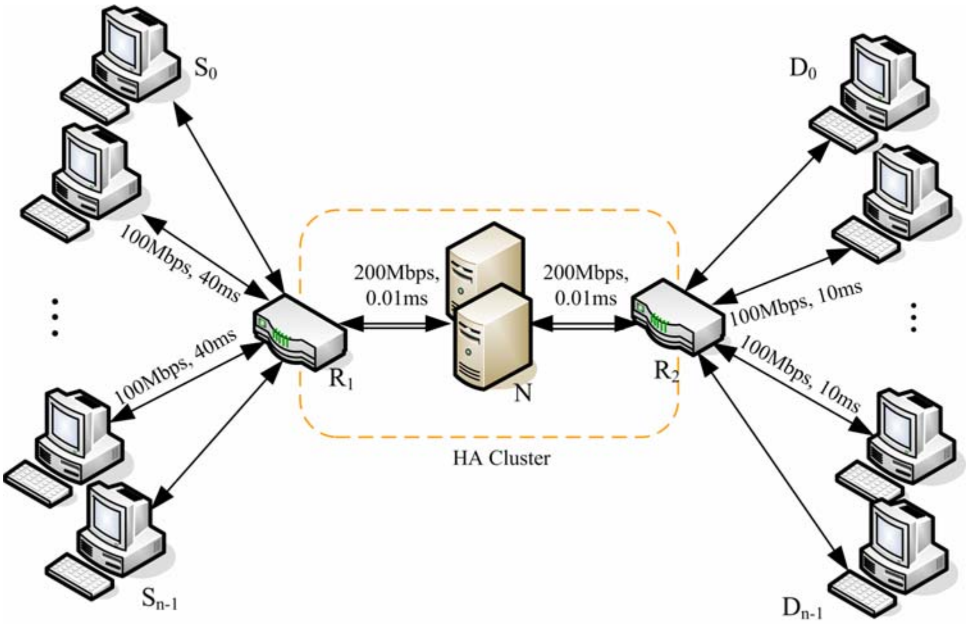

Figure 11. Simulation topology

We use 3 nodes, R_1, N, and R_2, to simulate an HA Cluster. In the HA cluster, R_1 and R_2 are boundary LBSes, while N represents cluster nodes. Local recovery and window control modules are implemented by extending ns-2 queue object in LBS nodes. The dropping policy of all buffers is DropTail. Note that the HA cluster is deployed close to the destination nodes, D_0 to D_{n-1}. For simplicity, single direction data transfer (left to right) is assumed.

7.2 Analysis on Improvements of HALR

In this section, we will use simulations to illustrate some of the influences and validate our analysis in Section 4.2. Note that an active/backup cluster with equal bandwidth strings is considered throughout this section.

7.2.1 Simulation Topology of HALR

In the topology depicted in Figure 11, the HA cluster link is shared by TCP flows originated from TCP source nodes, S_0 to S_{n-1}. The experiments are carried for short flow and long flow transfers. The flow transfer sizes are 10 and 1,000 segments for short flows and long flows, respectively. The maximum segment size is set to 1,460 bytes. TCP Reno, NewReno, and SACK with delayed ACK mechanism are adopted in our evaluations. The dropping policy of all buffers is DropTail. The buffer size of the gateway is set to be large enough to prevent buffer overflow.

The failovers on the topology are simulated by taking down the pass-through string during failover periods. At time = 2 sec, two links in the HA cluster are brought down to simulate the cluster failure and the failover procedure. $T_{failover}$ is the experiment parameter, which ranges from 50 ms to 12 sec.

7.2.2 Failover Time vs. Flow Completion Time

Figure 12, Figure 13, and Figure 14 shows the flow completion times for transferring 10 segments under different flow arrival times. Since L for 10 segments is about 0.6 sec, the flows arriving between 1.4 sec and 2 sec belong to *Case 2*. For the flows arriving between 2 sec and (2 +$T_{failover}$) sec, they belong to *Case 3*.

From Figure 12 to Figure 14, we can see that the flows protected by HALR are less harmed by a failover, while raw TCP flows have to pay the cost of UFTs. As expected, Figure 14 shows that without HALR, the flows of *Case 3* have to wait for lengthy timeouts for retransmissions. It indicates even a small $T_{failover}$ is achieved, without HALR, the flows belonging to *Case 3* increase its latency from 0.6 sec to 4.01 sec. On the other hand, the HALR-protected flows resume transmission after a 500-ms failover. The resulting L' values are between 800 ms and 1.9 sec.

In Figure 13, a small group of flows (arriving at between 1.8 sec and 2 sec) without HALR have latencies about 10 sec, (namely, the $AD(T_s)$ derived from T_{down} is $T_s + 2T_s$ = 9 sec), while the flows with HALR keep the completion times around 4 sec. The improvement of HALR becomes more obvious in Figure 14 that 40% of the incoming flows during the failover wait for an additional 9 sec to retransmit and the other flows wait for 3 sec. By contrast, L' with HALR only depends on T_{down}. Furthermore, in Section 7.3, through modeling analysis, we will show that $T_{failover}$ less than or close to 3 sec is a boundary for a good failover time. The test results here are consistent with our analysis.

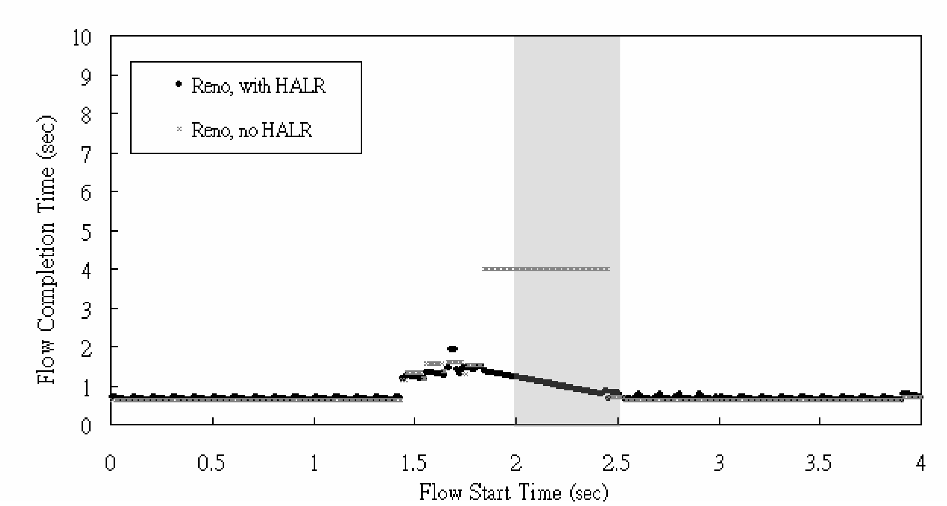

Figure 12. Flow completion times vs. flow arriving time (failover = 0.5 sec)

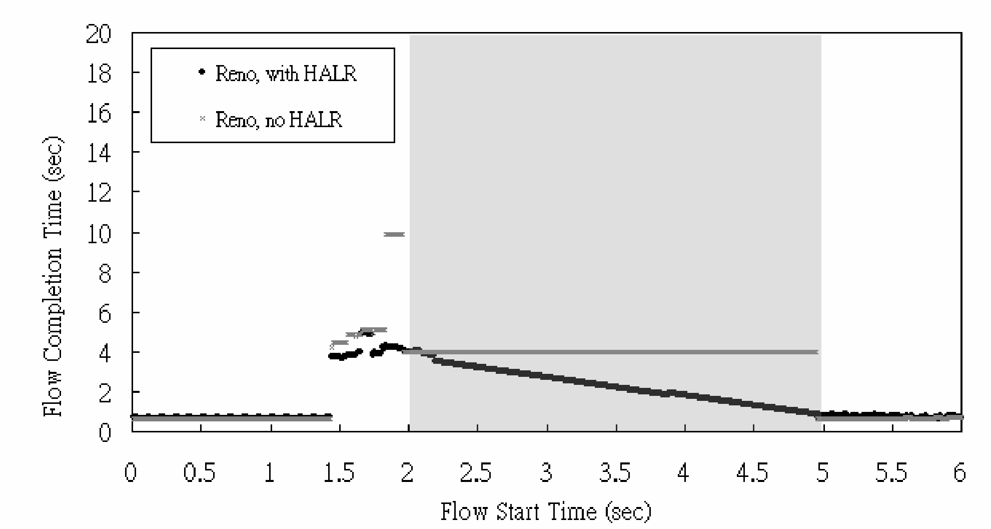

Figure 13. Flow completion times vs. flow arriving time (failover = 3 sec)

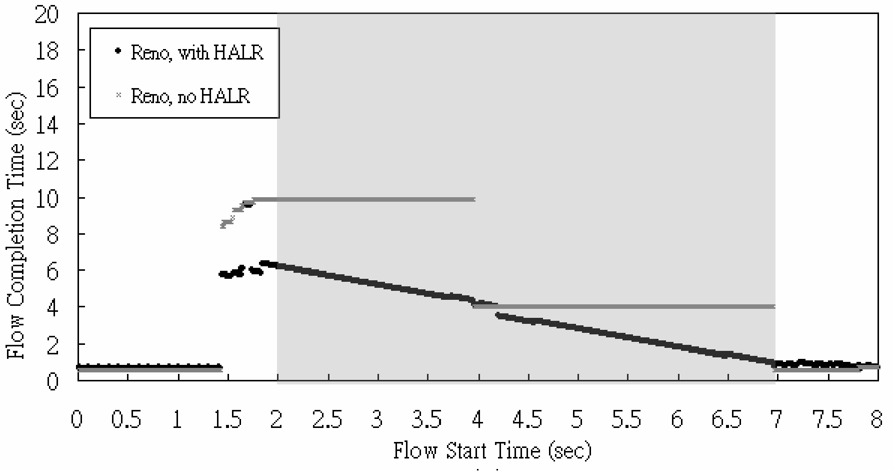

Figure 14. Flow completion times vs. flow arriving time (failover = 5 sec)

7.2.3 Failover Time vs. Retransmission Timeout

Figure 15, Figure 16, and Figure 17 shows the retransmission timeout distributions of the flows (1000-segment) belonging to *Case 2* under a range of failover times (50 ms – 12 sec) with different TCP implementations. In each pair of bar with the same failover time, the left one is the result with HALR and the one on the right side is without HALR. First, we find that when HALR is enabled, TCP can indeed avoid some of the timeouts that would be experienced if without HALR, especially in NewReno and SACK with $T_{failover}$ < 1 sec. HALR also avoids some of consequent timeouts which make *ssthresh* become two segments when $T_{failover}$ = 3 sec. We notice that the improvement of HALR for NewReno is more than that for Reno. This is because if multiple packets are lost in one congestion window, Reno has a higher probability to cause sender timeout which can not be eliminated by HALR. Moreover, with SACK options, the number of timeouts could be further reduced when $T_{failover}$ < 1 sec. The reason is that upon receiving out-of-order packets, SACK sender is notified with missing segments and resends them without waiting for a timeout. Finally, when $T_{failover}$ > RTO, Reno, NewReno, and SACK have the same behaviors on the timeout distributions because the major difference between Reno, NewReno, and SACK only lies on the modification of fast recovery; the three TCP implementations all employ slow-start after a retransmission timeout.

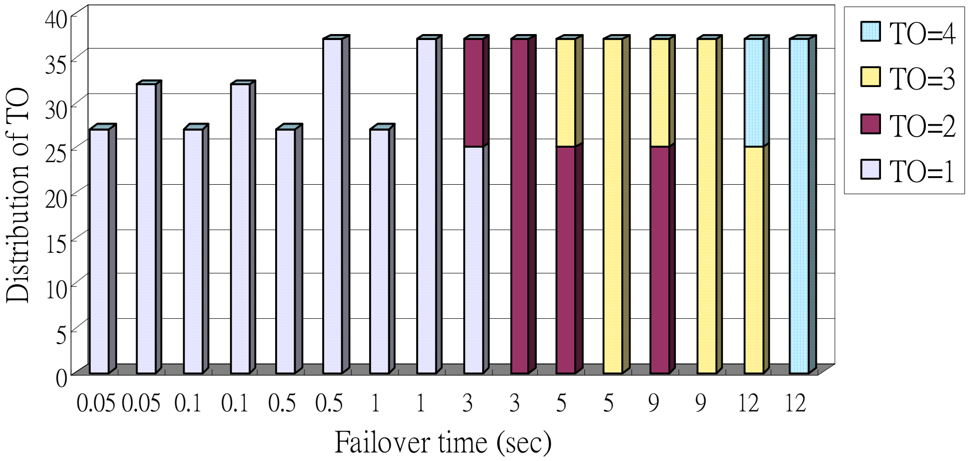

Figure 15. Number of timeouts of the flows in Case 2: TCP Reno.

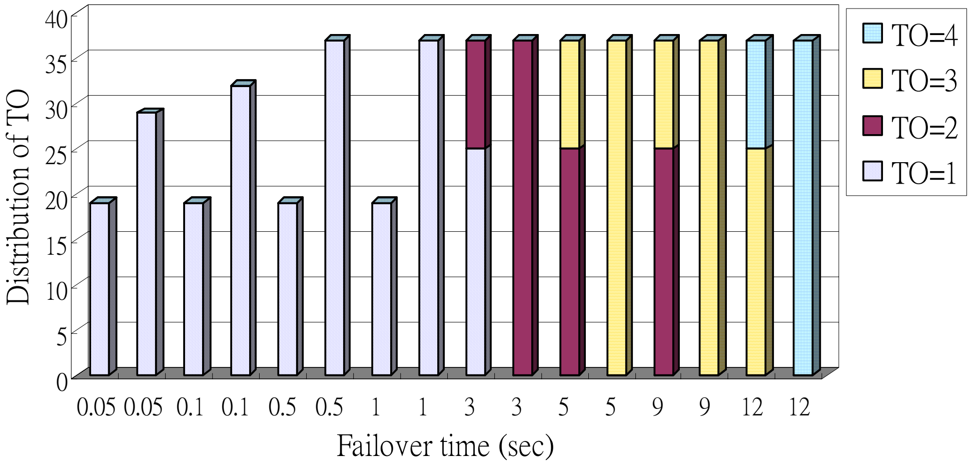

Figure 16. Number of timeouts of the flows in Case 2: TCP NewReno.

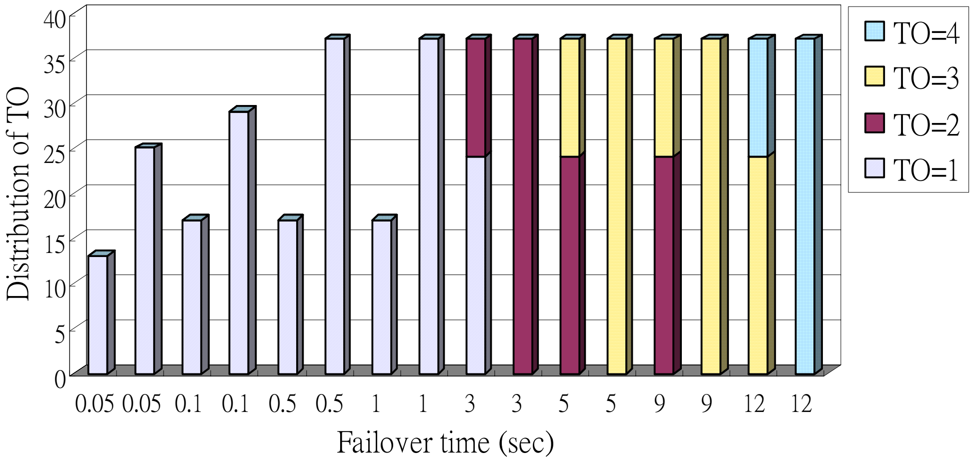

Figure 17. Number of timeouts of the flows in Case 2: TCP SACK.

7.3 Model Validation

The modeling results are compared to ns-2 simulation results to validate the extended models. Figure 18 and Figure 19 present the relationships between the $T_{failover}$ (from 50 ms to 12 sec) and L' measured by ns-2 simulation and estimated based on short flow and long flow models, respectively. The modeling results, including the maximum and minimum L' values, are depicted by lines; while simulated latencies by ns-2, including L' and L (normal flow completion time), are represented by one dot per connection (blue dots for latencies with HALR and red dots for latencies without HALR).

First, for raw TCP without HALR (the general case of HA clusters), a $T_{failover}$ = 3 sec (the default value of the pfsync [29]) seems to be a boundary of the "good-enough" failover times. For example, in Figure 18, when $T_{failover} < 3$ sec, the maximum L' is less than 5.3 sec. However, when $T_{failover}$ = 5 sec, the maximum L' becomes 11.2 sec. Thus, if a false failure declaration is possible, it could be inferred that enforcing a $T_{failover}$ close to or less than 3 sec might be a good balance.

Second, Figure 18 shows that HALR effectively reduces the L' for all failover times. This result is advantageous to the delay-sensitive applications, such as WWW. Note that the maximum L' of raw TCP is affected non-linearly by the characteristics of doubling TCP timeouts. With HALR, the maximum L' grows linearly when $T_{failover}$ increases because the sawtooth effect of UFT is eliminated. In this way, the latency over a failover is no longer bounded by RTO values but T_{down}. Therefore, through HALR, TCP flows could take full advantage of small failover times. Otherwise, for a HA cluster without HALR, the end results of TCP performance around a failover seemingly diminish the efforts of achieving accurate state replication (even for short flows) and a very low failover time.

We find that the L' values from short-flow modeling and from simulation have close agreement with each other. However, in Figure 19, although the results of long flow modeling follow the similar trend as in Figure 18, some inaccuracy is observed between the simulation results and modeling estimation. This is mainly due to the effect of *ssthresh* =2 problem described in Section 4.2. The gap separating ns-2 flow completion times into two major groups, representing the difference between Data drop flows with or without consecutive timeouts, or SYN drop flows with or without single timeout.

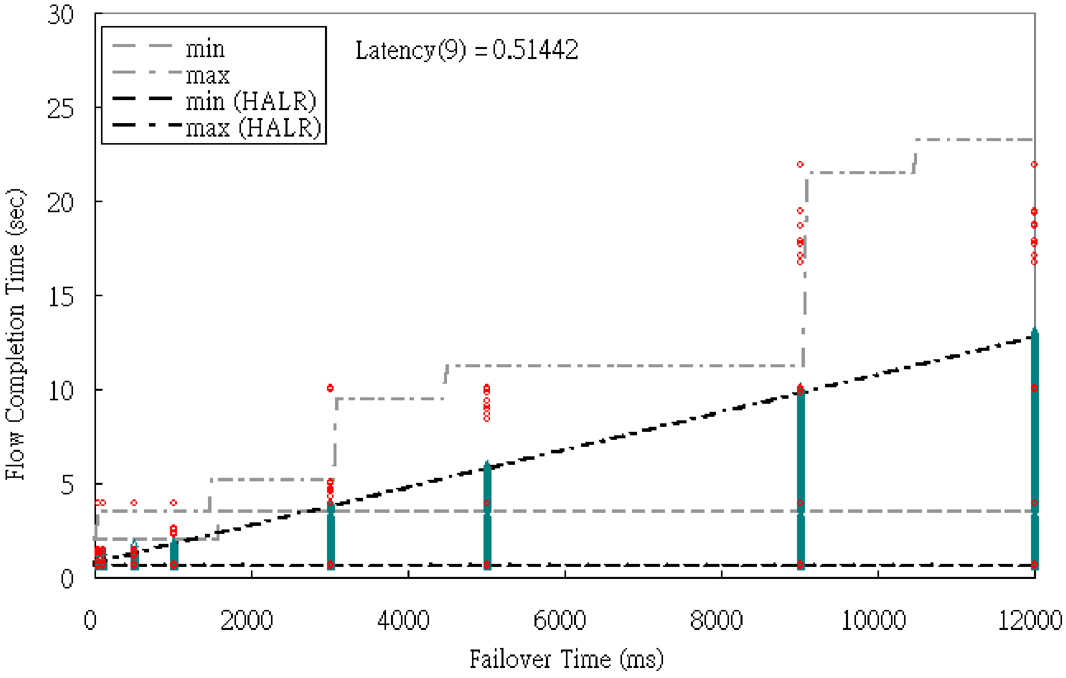

Figure 18. Short flow model validation

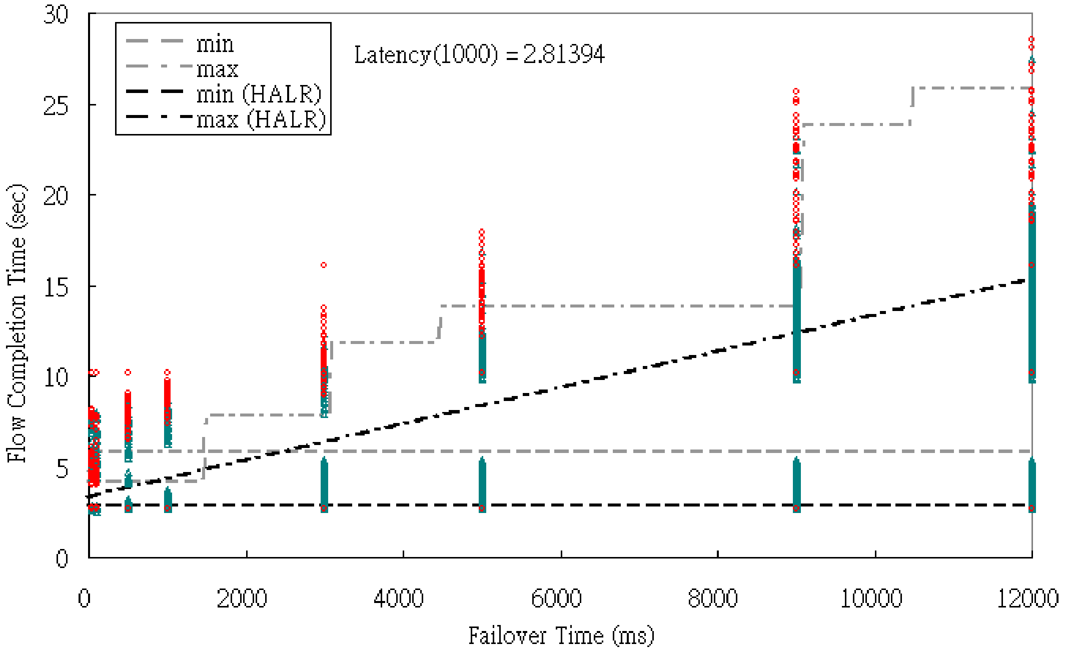

Figure 19. Long flow model validation

7.4 Improvements of Post-Failover Rate Control

During the simulation of rate control mechanism, an active/active cluster is considered. And various string bandwidth ratios are tested. Cluster failover is simulated by halving (or proportionally cutting) system bandwidth and dropping active flows on failed string. We carried an extensive set of experiments with various parameters settings; however, unless stated otherwise, the basic configuration is used as described as the following.

At the beginning of simulation, two strings are both actively processing flows passing through them. The TCP flows are distributed by LBSes into 2 groups; each is processed by one active string. For each group, there are 50 TCP flows; each is transferred between one source-destination pair, that is, S_0 to D_0 or S_{n-1} to D_{n-1}, respectively. From time=0, for each 250msec, a new connection is established from each group. Again, MSS=1460 byte, and each flow transfers 3000 packets in total.

The link capacity is set to 100Mbps for each string and other network links. Therefore, for an active/active cluster with equal string bandwidth, the system capacity would fall from 200Mbps to 100Mbps upon failover. Since the main focus here is the bandwidth bottleneck issue after failover events. In this section, a typical failover period, 3 sec, is chosen for illustration. At time t = 5 sec, a failover starts, and the failover ends at t = 8 sec. It is also validated through other simulation sets that using different failover times will not affect the macroscopic system behavior, such as aggregate throughput evolution, after failover events.

For the bottleneck buffer size, the buffer size limit for router R_1 is set to 750 packets, which is about the size of network bandwidth-delay product (for 100Mbps link) unless stated otherwise.

7.4.1 Throughput Analysis

We define the string experiencing failure as *failure string*, and the other string as *remaining string*. As described in previous section, the failover period is between t = 5 sec and t = 8 sec. During this time period, although newly connections arriving at the cluster would be directly assigned to the remaining string by LBS, the active connections on failure string would experience temporary service interruption.

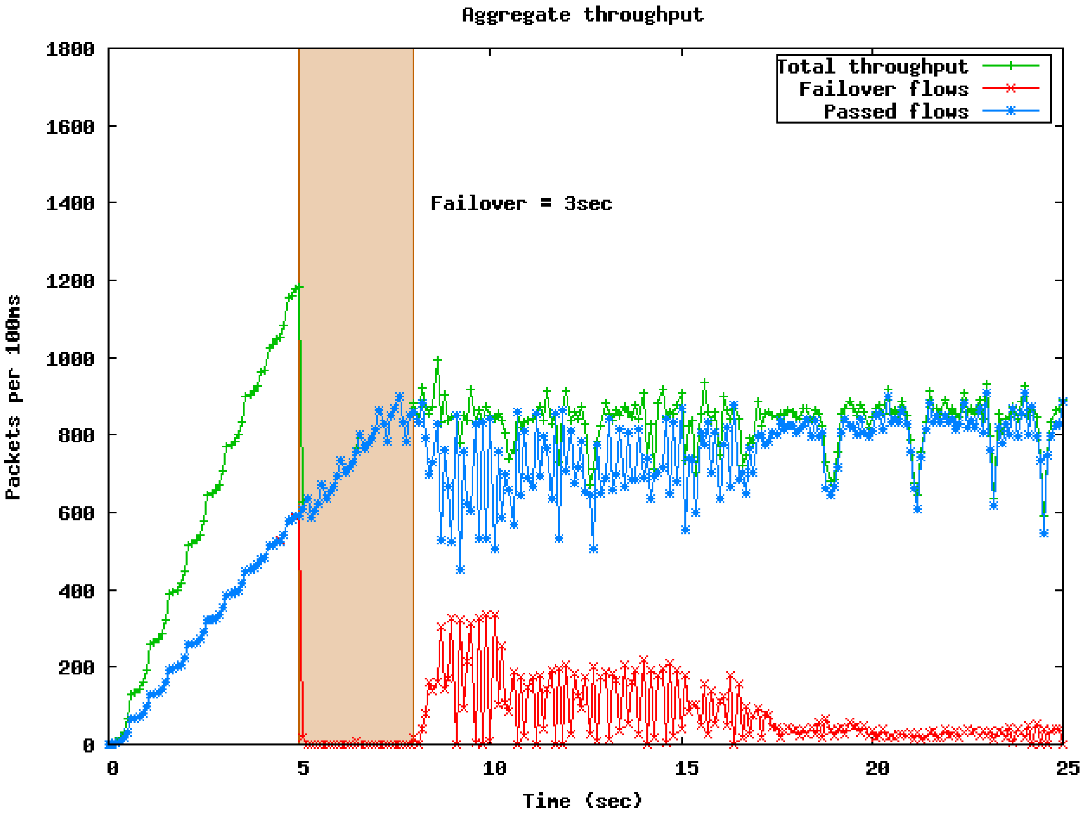

Figure 20. Throughput vs. time: without rate control

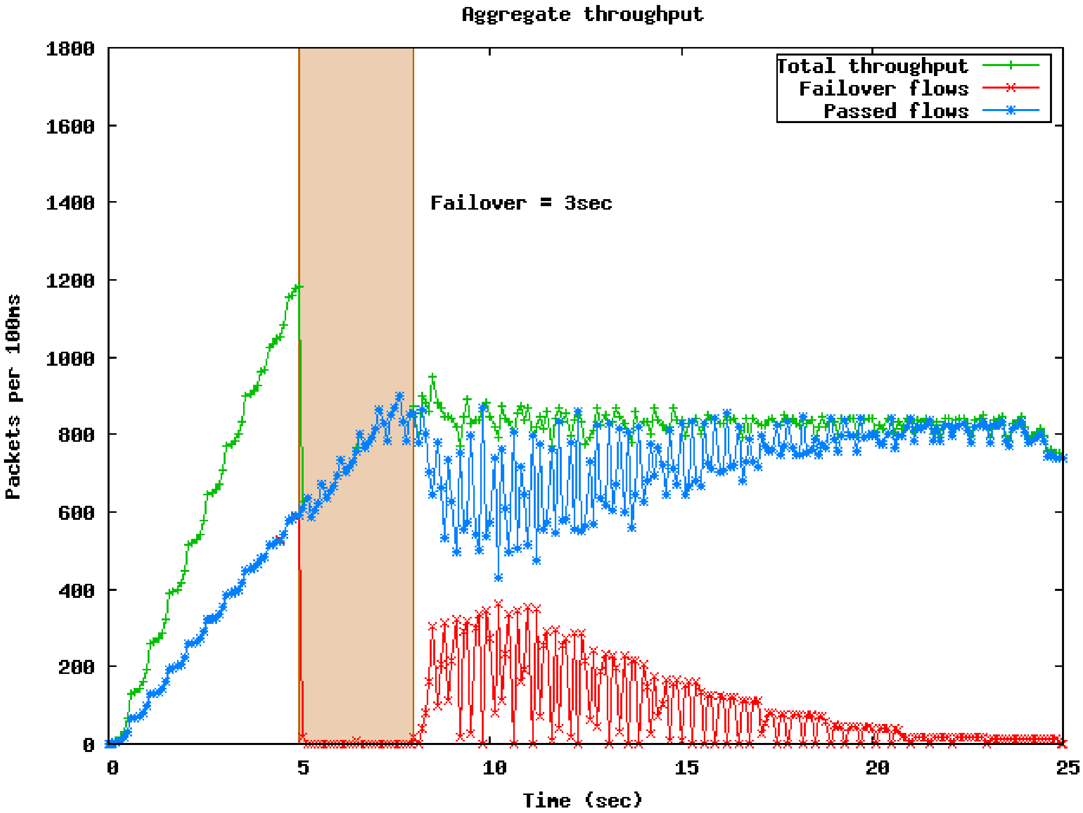

Figure 21. Throughput vs. time: with rate control

In Figure 20 and Figure 21, the aggregate throughputs vs. timeline with and without post-failover rate control are depicted, respectively. The total throughput is composed of those generated from passed flows and failover flows. Passed flows represent the flows without experiencing failover during their lifetime. While the failover flows are those assigned to the failure string, and were interrupted when failover happens. After failover, they are switched over to the remaining string.

If there is no post-failover rate control, although the aggregate throughput after failover remains at the level of available bandwidth, the fluctuations are apparent on total throughput due to the active flows, including those haven’t completed when failover happens and those arrive after failover, compete for insufficient bandwidth. In this case, a proportion of post-failover transfers require longer times for completion.

On the contrary, if post-failover rate control is deployed, although the total throughput after failover may be slightly lower due to the under estimation of window adjustment functions, the aggregate throughput has better smoothness. The smoothness comes from the explicit allocation of bottleneck link bandwidth achieved by post-failover rate control. From Figure 21, it could be inferred that through rate control mechanism, better per-flow fairness is achieved. Also, by mitigating the problem of bandwidth competition, the bottleneck buffer size is reduced, as we will present in later sections.

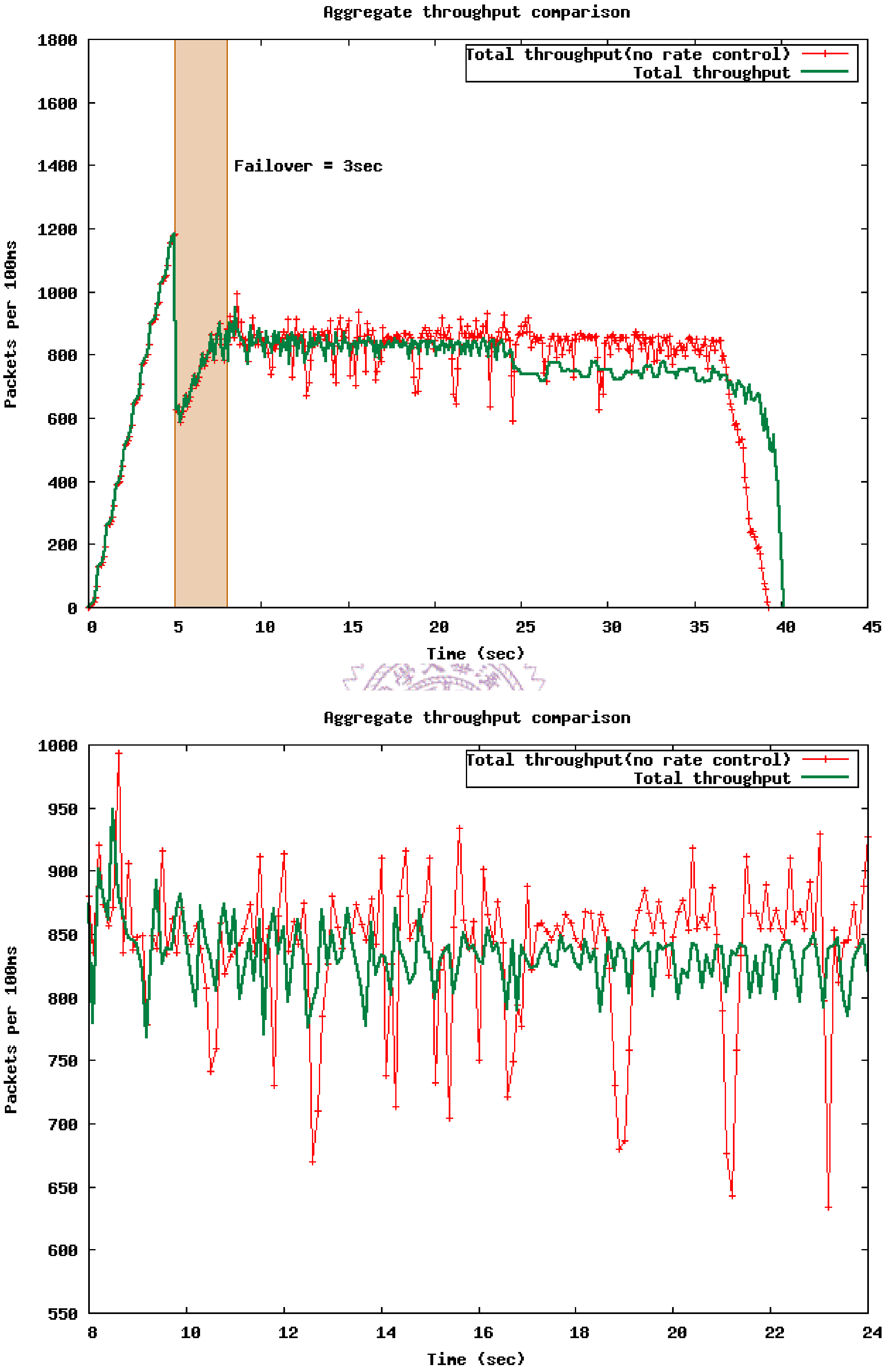

Figure 22. Aggregate throughput comparison (a) during simulation (b) zoomed-in.

7.4.2 Per-flow Fairness

To further illustrate the fairness characteristic, we use the metric “per flow bandwidth utilization ratio” to measure the fairness between each active flow. Since there are number of TCP flows during simulation period, only 10 flows within 100 flows are sampled and plotted. The simulation results with and without post-failover rate control are depicted in Figure 23 and Figure 24, respectively.

From Figure 23, it is observed that due to the resource contention, some flows, especially for those begin after failover, fail to attain their fair share of bandwidth until previous active flows finish their transfers. There are also some throughput drops followed by throughput bursts during the simulation period, representing that some flows experienced packet loss thus reducing their sending rate, therefore flow completion are postponed. On the other hand, in Figure 24, the throughput share after failover distributed by rate control is illustrated. In this figure, it could be seen that the throughput shares among flows are fairly allocated. Even for newly arrived flows after failover event, they would not grow over existing flows. By explicitly allocating available bandwidth among total active flows, a reasonable bandwidth usage is achieved. As a consequent, for every flow, no throughput degradation due to packet loss is observed during simulation period.

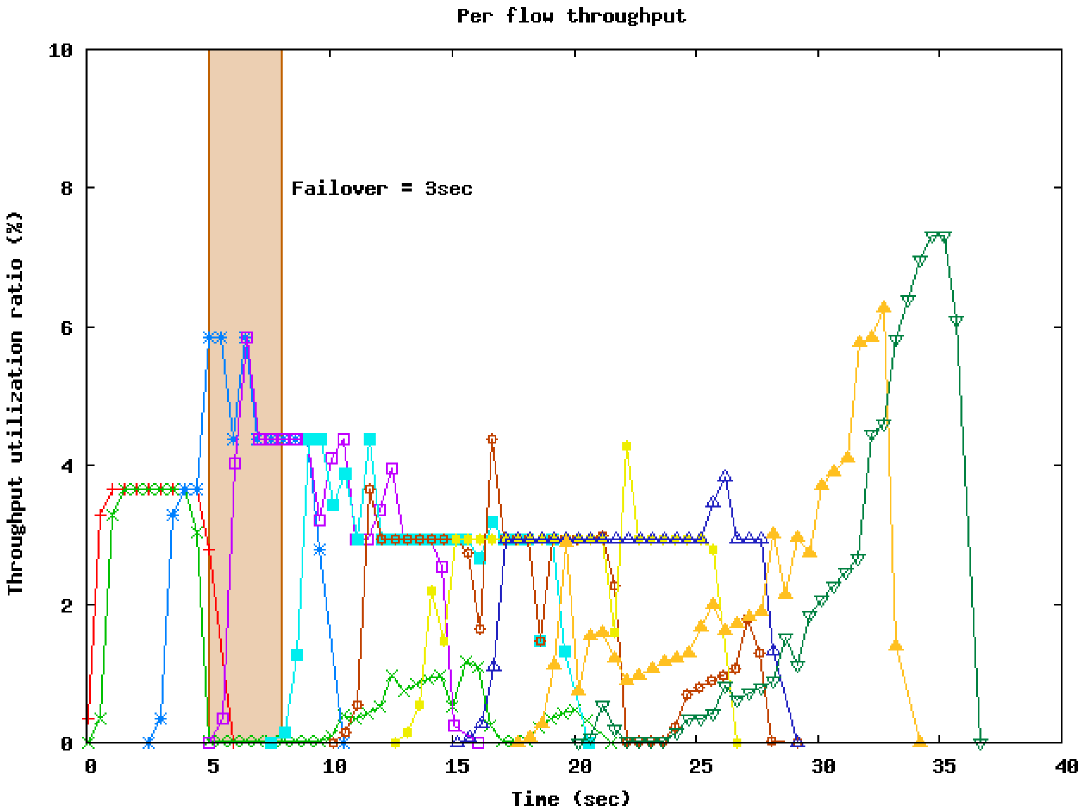

Figure 23. Per flow fairness: without rate control

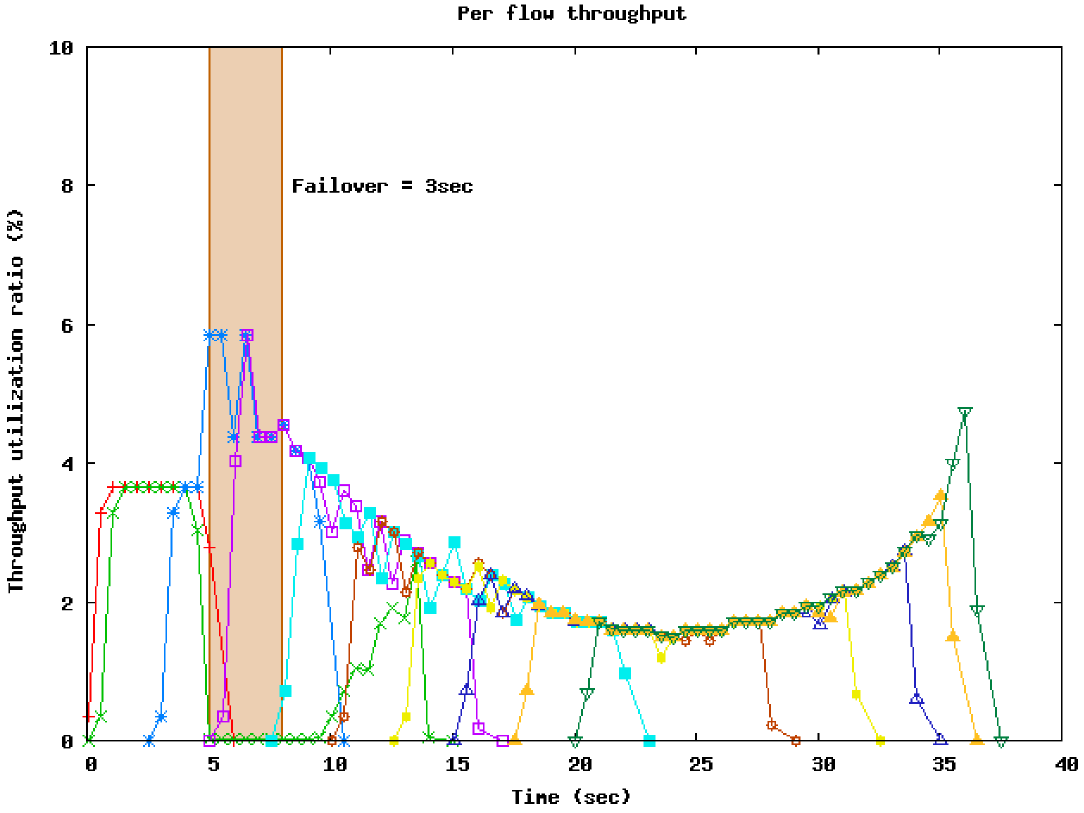

Figure 24. Per flow fairness: with rate control

7.4.3 Bottleneck Buffer Size

From the aspect of router buffer size on bottleneck link (R_1→N), the results with rate control and without rate control are shown below. In general, if post-failover rate control is deployed, the buffer size requirement would be much lower, which is the result of proper bandwidth adjustment. However, if there is no rate control, buffer size could not be effectively controlled and finally being saturated. A buffer with high usage would result in longer queuing delays. Also, packets drops due to buffer overflow are incurred.

In Figure 25, the bottleneck buffer limit for LBS node, R_1, is set to 1500 packets, while in Figure 26, the buffer limit is set to 750 packets, which is about twice or the size of network bandwidth-delay product for 100Mbps link. When there is no protection of rate control, because the sending rate exceeds available bandwidth, buffer grows fast and overflows will eventually happen. It is even worse in the case of smaller buffer. For example, when buffer limit = 750 packets, the buffer overflow occurs more frequently than the case of 1500-packet buffer size. There are eight surges (rather than 3 surges) of buffer overflows during a 20-second post-failover transferring period. However, for both figures, by employing rate control, the buffer requirement is restrained well with a maximum level of 600 packets.

Note that, if we utilize even larger buffer size, such as a limit of 3000 packets; although the saturation of the buffer could be postponed, it could not be avoided. Moreover, a large amount of buffered packets would result in even longer queuing delays. And it may possibly trigger retransmissions falsely for packets still buffered in queue, thus aggravating the consumption of the bottleneck buffer.

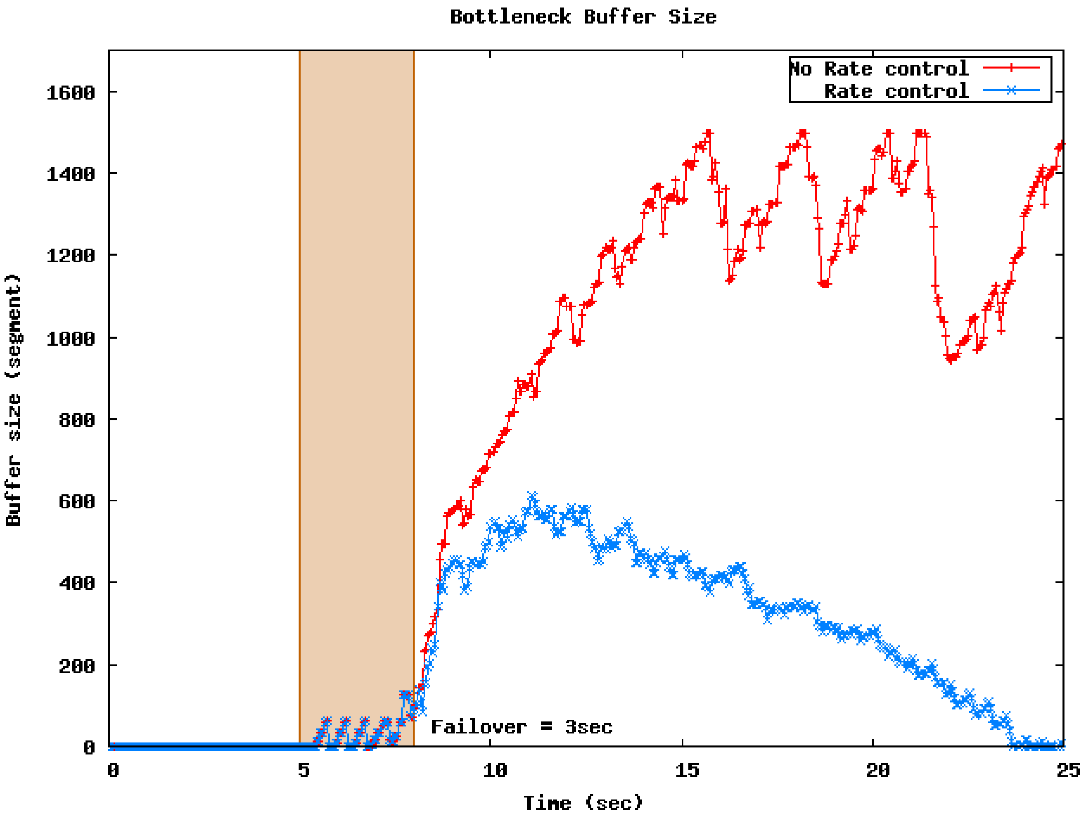

Figure 25. Bottleneck buffer size (buffer limit= 1500 pkts)

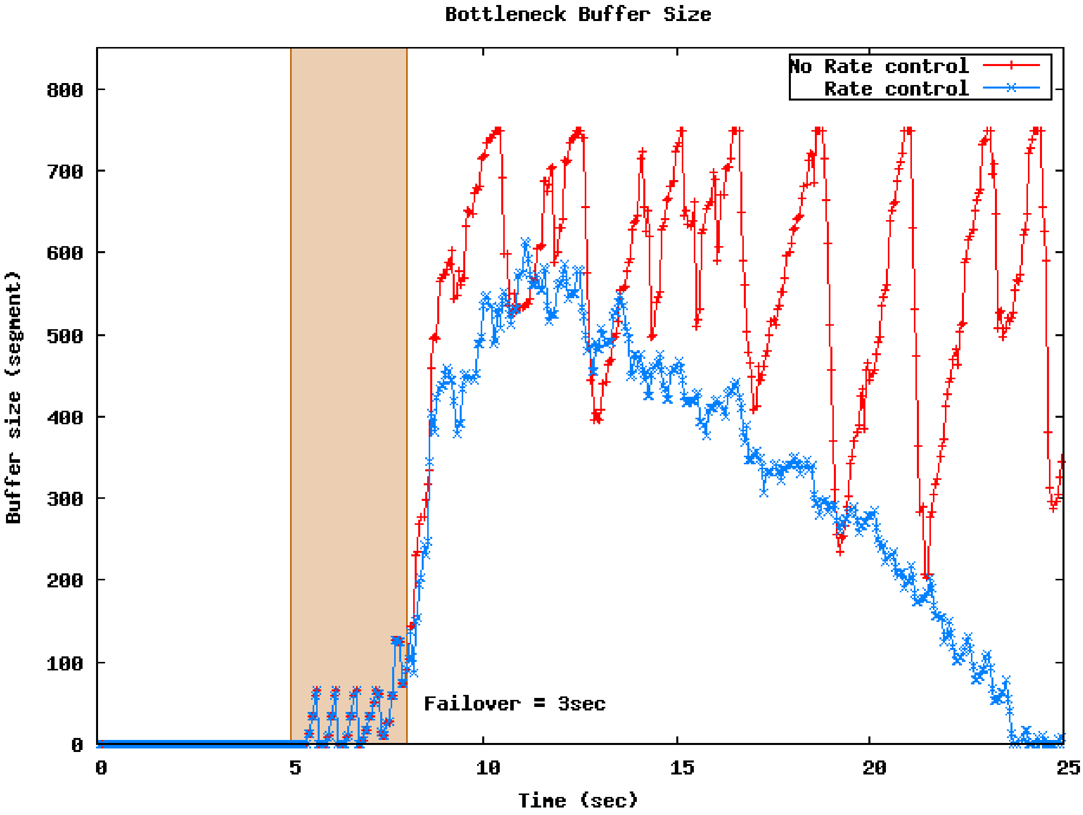

Figure 26. Bottleneck buffer size (buffer limit= 750 pkts)

In summary, combining with the figures in previous sections, the proposed post-failover rate control mechanism not only smoothes the aggregate throughput when a post-failover bottleneck exists, but also achieves stable buffer requirement and per-flow fairness in terms of bandwidth utilization. Meanwhile, compared to the non-rate controlled flows, a similar level of system throughput is preserved.

Chapter 8

DISCUSSION & CONCLUSION

In this thesis, a new scheme, called HALR, has been proposed to improve the TCP performance over the failover events in the HA cluster. HALR caches TCP packets selectively and resends them locally after the failover is finished. The scheme keeps the end-to-end TCP semantics and is compatible with existing TCP implementations. Furthermore, analytic models for TCP completion times were developed to characterize the effects of failover times and the effects of the proposed method. Analytical estimations were validated by ns-2 simulation experiments.

The investigations lead to a number of main conclusions. First, using modeling results and ns-2 simulation, we show that HALR improves the end-to-end TCP performance over a failover by minimizing the unused flow time. Especially, HALR is useful for the short flow arriving during the failover process to resume its transmission after the failover immediately without waiting for a lengthy RTO (3 sec). Second, we find that TCP with HALR can avoid some of the retransmission timeouts that would be experienced if without HALR. Third, we find that if a false failure declaration is possible, a failover time close to or less than 3 sec might be a good balance. Finally, by the simulation on real packet trace, we show that HALR is a lightweight solution on memory requirements. Furthermore, if only the SYN packets received during the failover are cached, the memory requirement can be minimized.

After failover events happen, in order to prevent the flows to suffer from potential system bandwidth bottleneck, the post-failover rate control mechanism is proposed. This mechanism utilizes window adjustment method to allocate fair share bandwidth to all active connections in system. An extensive set of simulations are carried to evaluate the effects of post-failover rate control under different scenarios. Through simulation results, by triggering rate control upon failover, the system bandwidth is fairly allocated to flows. Also, the bottleneck buffer size is reduced, eliminating the occurrence of packet drops. Finally, this mechanism ensures smooth transfer for pass-through flows encountering post-failover bottleneck.

REFERENCES

[1] J. Wolfgang and T. Sven, "Analysis of internet backbone traffic and header anomalies observed," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* San Diego, California, USA: ACM, 2007.

[2] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate, "End-to-end WAN service availability," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, pp. 300-313, 2003.

[3] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, pp. 157-187, 1995.

[4] P. Lewis, "A High Availability Clustering Solution," *Linux Journal*, vol. 1999, 1999.

[5] J. Padhye and S. Floyd, "Identifying the TCP Behavior of Web Servers," *ACM SIGCOMM*, 2001.

[6] V. Paxson, "Automated packet trace analysis of TCP implementations," *Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 167-179, 1997.

[7] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," *Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 270-280, 1996.

[8] F. Kevin and F. Sally, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 26, pp. 5-21, 1996.

[9] S. Floyd and T. Henderson, "The NewReno Modification to TCP’s Fast Recovery Algorithm," RFC 2582, April 1999, 1999.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, October 1996, 1996.

[11] V. Paxson and M. Allman, "Computing TCP’s Retransmission Timer," 2000.

[12] J. H. Hu, G. Feng, and K. L. Yeung, "Hierarchical Cache Design for Enhancing TCP Over Heterogeneous Networks With Wired and Wireless Links," *IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS*, vol. 2, p. 205, 2003.

[13] S. James and M. Glenford, "Link layer-based TCP optimisation for disconnecting networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 31-42, 2003.

[14] M. Ji-Cheol and L. Byeong Gi, "Rate-adaptive snoop: a TCP enhancement scheme over rate-controlled lossy links," *IEEE/ACM Trans. Netw.*, vol. 14, pp.

603-615, 2006.

[15] D. Martin, R. H. Thomas, and M. Jeff, "Experience with "link-up notification" over a mobile satellite link," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 93-104, 2004.

[16] H. J. Byun and J. T. Lim, "Explicit window adaptation algorithm over TCP wireless networks," *Communications, IEE Proceedings-*, vol. 152, pp. 691-696, 2005.

[17] W. Huan-Yun, T. Shih-Chiang, and L. Ying-Dar, "Assessing and improving TCP rate shaping over edge gateways," *Computers, IEEE Transactions on*, vol. 53, pp. 259-275, 2004.

[18] A. James, O. Michel, and Y. M. Delfin, "A self-regulating TCP acknowledgment (ACK) pacing scheme," *Int. J. Netw. Manag.*, vol. 12, pp. 145-163, 2002.

[19] A. James, O. Michel, Y. M. Delfin, and Y. Zhonghui, "Improving network service quality with explicit TCP window control," *Int. J. Netw. Manag.*, vol. 11, pp. 169-188, 2001.

[20] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit window adaptation: a method to enhance TCP performance," *Networking, IEEE/ACM Transactions on*, vol. 10, pp. 338-350, 2002.

[21] Y. Matsushita, T. Matsuda, and M. Yamamoto, "TCP Congestion Control with ACK-Pacing for Vertical Handover," *IEICE TRANSACTIONS on Communications*, vol. 90, pp. 885-893, 2007.

[22] K. Shrikrishna, K. Shivkumar, B. Prasad, and P. Bob, "TCP rate control," *SIGCOMM Comput. Commun. Rev.*, vol. 30, pp. 45-58, 2000.

[23] "NLANR PMA Trace," http://pma.nlanr.net/.

[24] M. Mellia and H. Zhang, "TCP model for short lived flows," *Communications Letters, IEEE*, vol. 6, pp. 85-87, 2002.

[25] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2000, pp. 1742-1751 vol.3.

[26] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empiricalvalidation," *Networking, IEEE/ACM Transactions on*, vol. 8, pp. 133-145, 2000.

[27] W. R. Stevens, "TCP/IP illustrated (vol. 1): the protocols," 1993.

[28] "ns-2," http://www.isi.edu/nsnam/ns/.

[29] R. McBride, "Firewall Failover with pfsync and CARP."