사건 기반 세션 Timeout 관리 정책

최창석*, 조학봉*, 문종욱*, 정기현*, 최경희**
* 아주대학교 전자공학부
** 아주대학교 정보 통신 대학원
e-mail : dada@madang.ajou.ac.kr

Event based session timeout management policy

Chang-seok Choi, Hak-bong Cho, Jong-Wook Moon*,
Gihyun Jung*, Kyunghee Choi**
*Dept. of Electronics Engineering, Ajou University
**A Professional Graduate School
for Information and Communication Engineering, Ajou University

요 약

세션 관리 시스템에서 많은 세션에 대한 timeout 처리시 발생하는 오버헤드를 줄이기 위한 방법으로 기존의 time based timeout 관리 방식이 아닌, 세션 관리 시스템의 성격에 적합한, event based timeout 관리 방법을 제시하고 실험을 통해 확인해 본다.

1. 서론

요즘 널리 사용되는 TCP/IP 프로토콜 스택은 상당히 많은 테이블들을 동적으로 처리하고 있다. 이처럼 TCP/IP 프로토콜 스택에 구현된 arp cache, dns cache, session state table 과 같이 동적으로 증가하는 테이블들은 필히 timeout 관리를 필요로 한다. 이들 테이블들이 증가함에 따라 검색 시간이 증가하여 시스템의 성능이 떨어지고 메모리 소모량이 증가하여서 메모리 이용 율이 떨어지기 때문이다. 그래서 각종 timeout 관리 기법을 통해서 테이블의 크기를 줄여주어야 한다.

Timeout 관리는 여러 응용분야에 사용되는 기법으로 각 용도에 따라 적절한 방법이 요구되고 또한 그 Timeout 관리 방법이 얼마나 효율적인가에 따라 시스템의 성능에 큰 영향을 미친다.[1]

현재 보안 분야의 대표적인 장비인 IDS 와 파이어월에서는 과도한 트래픽에 따른 세션 관리가 큰 이슈가 되고 있다. 이러한 세션 관리에 있어서 timeout 관리 기법이 중요한 역할을 하고 있다. 내부 망의 사용자가 500~5000 명이 넘어가는 중, 대형 규모의 사이트에서는 평균 세션의 수가 10 만개에 육박하므로 적절한 timeout 정책을 통해서 테이블의 무한 증식을 막을 수 있어야 한다. 이러한 보안 시스템에서 세션 timeout 관리는 기존에 주로 연구 되어 오던 timeout 관리와는 성격 면에서 차이가 난다. 기존의 연구들은 얼마나 세밀한 단위로 성능의 저하를 최소화 하면서 timeout 처리를 해낼 수 있느냐에 대한 것이다. 하지만 보안 시스템에서의 세션 timeout 관리는 timeout event 가 발생하더라도 경우에 따라서는 즉시 동작을 요하지는 않는다. 이러한 성격을 이용해서 본 논문에서는 기존의 time based 방식을 일부 이용한 event based 방식의 timeout 관리 기법을 제안한다.

실험에 따르면 세션의 수가 많아지는 경우 event based 방식을 사용하게 되면 성능 면에서 예측성이 뛰어나고 통신 지연이 time based 방식이 경우 보다 좋은 결과를 보인다.

2. Background

2.1 Application 에 따른 고려사항

Timeout 을 관리하는 방법은 사용되는 어플리케이션의 성격에 따라 적절한 방법이 선택되어야 한다. 어플리케이션의 성격에 맞는 timeout 관리 방법을 선택을 하기 위해서는 아래의 3 가지 항목에 대한 고려를 해야 한다.

2.1.1 Event granularity

Event Granularity 는 timeout 값의 만료를 검사하는 간격을 조정하기 위해 변경하는, 타이머의 resolution

에 밀접한 관련이 있는 요소이다. 세밀한 resolution 의 검사 주기는 높은 정확성을 보장하지만 시스템의 성능에 오버헤드로 작용한다. OS Kernel 에서의 process scheduling[1], TCP Reno[5] 등의 응용이 fine granularity 를 요구하는 대표적인 어플리케이션들이다. 그리고 TCP Vegas[5] 및 firewall 과 IDS 에서 세션 timeout 관리 때 사용되는 응용들이 coarse granularity 를 요하는 대표적인 어플리케이션이다.

2.1.2 Reaction granularity

Reaction granularity 는 해당 event 가 발생했을 때 해당 event 에 반응하는데 있어서의 granularity 를 의미한다. 즉 Reaction Granularity 는 timeout 만료 시의 reaction 의 성향을 말한다. 이 reaction 은 크게 passive reaction 과 active reaction 으로 나뉠 수 가 있다. Passive reaction 은 응용의 성격에 따라 어느 정도의 지연을 허용하는 soft realtime 성격을 지니는데 반해서 active reaction 은 지연을 거의 허용하지 않는 hard realtime 성격을 지닌다. 예를 들면 패킷의 재전송을 위한 timeout expiry 는 즉각적인 반응(active reaction)을 요구하는 application 이다. 왜냐하면 패킷의 재전송이 느려지게 되면 전체적인 성능에 큰 영향을 미치기 때문이다. 이러한 reaction 의 성향이 응용 마다 틀린 이유로 timeout 관리하는 방법에 많은 유연성이 있다. 이러한 reaction 의 성향 중 passive reaction 의 경우는 즉각적인 반응을 요구하지 않으므로 event based method 를 사용한 timeout 관리를 가능하게 한다.

2.1.3 Event quantity

Event Quantity 란 특정 시점에서 timeout expiry 를 관리해야 하는 대상의 양을 말한다. 여기서 특정 시점은 timeout expiry 를 검사하는 시점을 말한다. 이 양은 어플리케이션에 따라 불과 수 십 개 에서 수 만개에 이르기 까지 다양하다. 이러한 양이 많으면 많을수록 일반 적으로는 Event Granularity 를 세밀하게 하기가 힘들어 진다.

2.2 timeout 관리 방법의 영향

Timeout 관리가 시스템에 미치는 오버헤드 요소는 다음과 같이 정의한다.

Space: timeout 관리 에서 사용하는 data structure 의 메모리 요구량을 말한다. 관리대상의 양과 expiry action 의 성격에 영향을 받는다.

Latency: timeout 값의 설정과 갱신, timeout expiry processing, timeout expiry 검사 의 각각에 소요되는 시간을 말한다.

3. 관련연구

3.1 Time based approach

일반적으로 timeout 관리는 타이머를 사용하여 관리하게 된다. 일반적으로 사용되는 타이머의 구성요소는 다음과 같다.[1],[2]

- STARTTIMER (Interval, RequestID, ExpiryAction)
  : "Interval" 후에 만료되는 타이머를 "RequestID"로 구분하여 메모리의 적당한 위치에 등록하고 만료 시의 동작을 "ExpiryAction"으로 지정한다.
- STOPTIMER(RequestID)
  : "RequestID"를 사용하여 해당 타이머를 찾아 정지시킨다.
- PERTICKBOOKEEPING
  : 타이머의 세분성 단위를 T 라 고 하면 매 T 단위 시간 마다 만기된 타이머가 있는지 검사한다. 만기된 타이머가 있으면 STOPTIMER 를 호출한다.
- EXPIRYPROCESSING
  : STARTTIMER 에 명시된 "ExpiryAction"을 행한다.

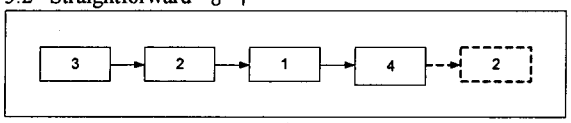
[1]에서 Space 의 오버헤드는 작다고 가정 했을 때 Latency 에 따른 critical 한 parameter 는 표 1 과 같다.

Routine	Critical Parameter
STARTTIMER	Latency (average and worst-case)
STOPTIMER	Latency (average and worst-case)
PERTICKBOOKKEEPING	Latency (average)
EXPIRYPROCESSING	None

<표 1>

다음에 위의 타이머 요소를 사용하여 기존의 타이머 방식에 따른 오버헤드를 알아본다.

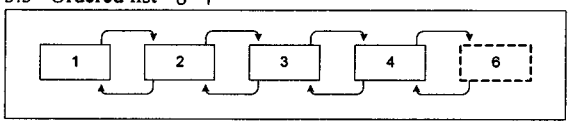
3.2 Straightforward 방식



<그림 1>

[2]를 참고하면 STARTTIMER 는 timeout 값(만료시간 또는 interval)을 등록하고 PERTICKBOOKKEEPING 은 주기적(T)으로 interval 값을 감소하거나 만료를 검사한다. 예를 들어 <그림 1>의 경우 타이머의 등록은 점선의 상자같이 리스트의 제일 뒤에 위치시키고 만료시간을 기록하게 된다. timeout 검사는 주기적으로 리스트를 따라 만료시간을 감소하면 된다. 절대시간을 기록했을 경우는 현재의 시간과 비교하게 된다. 이 방식은 관리대상의 양이 적을 경우와 Timeout 값이 작은 경우 좋은 성능을 보인다.

3.3 Ordered list 방식

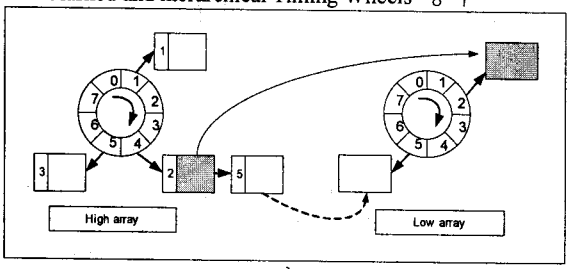


<그림 2>

[1]과 [2]를 참고하면 PERTICKBOOKKEEPING 에 대한 latency 를 줄일 수 있는 방법이다. 이 방법은 STARTTIMER 에서 타이머를 등록할 때 해당 타이머의 만료시각을 저장한다. 저장 시 만료시각이 빠른 순서대로 정렬하여 저장한다. PERTICKBOOKEEPING 은 단지 현재시각을 증가하다가 주기 T 마다 리스트를

앞에서부터 현재 시각보다 큰 항목을 만날 때까지 검사해 나간다. 검사하면서 현재 시각보다 작은 시각을 가진 항목을 만나면 그 항목을 지우고 EXPIRYPROCESSING 을 호출한다. 예를 들어 <그림 2>의 경우 만료 시간이 3 인 점선의 타이머를 등록시 리스트에서 만료 시간이 2 와 5 인 항목의 사이에 등록하게 된다. 또 timeout 검사는 현재 시각이 4 인 시점에서 검사를 하게 되면 리스트의 만료시간이 3 인 타이머 까지만 검사를 하고 timeout 만료 처리를 하게 된다. 이러한 Ordered list 방식은 정렬하여 메모리에 기록하는 STARTTIMER 부분에 오버헤드가 발생한다. 동일한 간격의 timeout 값을 갖는 응용에서는 정렬의 오버헤드가 줄어서 좋은 성능을 보이게 된다. 또한 <그림 2>의 linked list 방식이 아니라 tree-based data structure 를 사용하는 unbalanced binary trees, heaps, post-order and end-order trees 또는 leftist-trees 등의 방법 [3][4]을 사용하면 보다 성능을 높일 수 있다.

3.4 Hashed and hierarchical Timing Wheels 방식



<그림 3>

시간을 계층적으로 나누어 각각의 timing wheel 을 두는 것이다. 타이머의 등록은 만료시각에 따라 계층적으로 나누어 등록된다. 또 PERTICKBOOKEEPING 의 경우는 각 계층의 time tick 단위를 증가시키다가 등록된 타이머가 존재하면 해당 리스트를 만료시킨다. 예를 들어 <그림 3>과 같은 두 개의 계층을 생각해보자. 높은 쪽의 단위를 분으로 하고 낮은 쪽의 단위를 초라고 하면 4 분 2 초에 만료되는 타이머는 <그림 3>의 음영 표시된 부분처럼 등록된다. 즉 현재시각이 4 분이 되면 High array 의 4 번에 등록된 타이머들은 Low array 에 등록하고 2 초 후에 만료되는 것이다. 이 방법은 계층의 수만큼 STARTTIMER 의 오버헤드가 증가한다. 이 방법은 BSD UNIX implementation 과 여러 상용시스템에서 사용 되고 있다.

지금까지 여러 가지 타이머 관리 방식에 대해 알아보았다. 각 방식에 따른 오버헤드 요소를 표 2 에 나타내었다. [1]

	STRARTIMER	PERTICK BOOKEEPING	비고
STRAIGHT FWD	O(1)	O(n)	
ORDERED LIST	O(n)	O(1)	
HIERARCHICAL WHEEL	O(m)	O(1)	m 은 계층의 종류의 수

<표 2>

표에서 n 은 특정 시점에서 관리되는 타이머 의 수를 나타낸다.

이러한 기존의 방법의 각각의 장단점을 생각하여 세션 관리 시스템에 적합한 timeout 관리 방법을 다음 장에 제시한다.

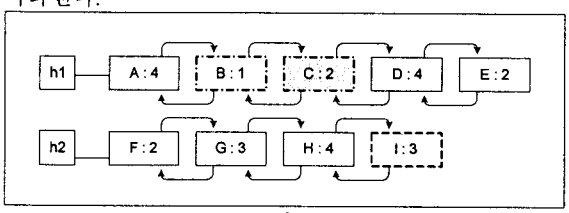
4. Event based timeout management

세션 관리는 firewall 이나 IDS 같은 장비에서 사용되는 응용으로 패킷 하나하나의 connection state 를 검사하고 세션 테이블을 유지하는 일을 한다. 이러한 세션 관리 시스템의 특징은 다음과 같다.

1) 관리 대상의 양이 많다.(50000 개 이상이다.)
2) timeout 값의 갱신이 빈번하다.
3) timeout expiry action 의 신속성이 불필요하다.

약 50000~100000 개 세션 의 timeout 만료를 관리해야 하는 세션 관리 시스템에서 기존의 타이머를 사용한 방법을 사용시 많은 관리대상으로 인한 PERTICKBOOKKEEPING 의 오버헤드가 발생한다. 또한 100M 네트워크인 경우 최대 초당 약 150000 개의 패킷 이 지나가게 된다. 이런 상황에서 세션 테이블에 존재하는 패킷 마다 timeout 값을 갱신하게 되는데 이는 STARTTIMER 의 오버헤드가 매우 큼을 알 수 있다. 이러한 세션 관리 시스템에 적당한 timeout 관리 방법을 다음에 제안한다.

세션 관리 시스템은 매 패킷마다 connection state 를 검사 하여 세션 테이블에 state 와 timeout 값을 갱신하게 된다. 이처럼 세션 관리 시스템이 필수적으로 행하게 되는 세션 테이블 검색 루틴을 이용하여 timeout 관리 의 오버헤드를 줄이고자 하는 것이다. 여기서 제안하는 방법은 기존의 타이머를 사용한 time-based-timeout 관리 방식이 아닌 event-based-timeout 관리 방법이다. 즉 패킷이 도착하여 행하는 세션 테이블을 검색하는 과정 동안 거치게 되는 세션들 만의 timeout 을 관리하는 것이다. 위의 세션 관리 시스템의 특성 중 세 번째 항목인 "timeout expiry action 의 신속성이 불필요하다." 때문에 timeout 이 발생해도 즉시 제거할 필요가 없다. 후에 참조되는 시점에 만료를 확인 후 제거해도 되는 것이다. <그림 4> 에 제안하는 방법을 나타낸다.



<그림 4>

1) event 에 대한 hash 를 사용하여 분산되어 있는 세션들을 검색하는 동안에 거치게 되는 루틴 상의 세션 만 timeout 를 처리한다.
2) 검색 루틴 중에 만료된 세션 은 제거한다.
3) 새로 들어온 세션 은 hash 리스트의 제일 뒤에

위치시킨다.

4) 메모리의 효율성을 위해 세션의 개수가 기준치(S)에 다다르면 전체 list 를 체크하여 만료된 세션을 제거한다.

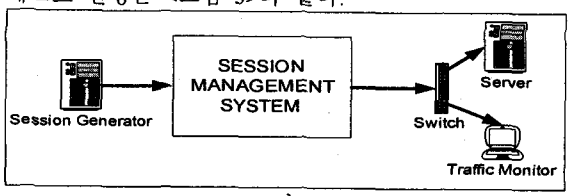
예를 들어 현재 시간이 3 인 시점에서 D 세션의 패킷이 도착하면 D 까지 검색하는 루틴 중에서 시간 3 보다 적은 값을 가지는 항목들(B,C)은 제거한다.

이 방법은 기존의 타이머를 이용한 방법 중 Straightforward 방법과 Ordered list 방법 그리고 Hashed and hierarchical Timing Wheels 방법의 장점을 섞어서 사용하였다. 다시 말하면 Straightforward 방법의 STARTTIMER 루틴의 장점을 사용하였고 Ordered list 방법의 PERTICKBOOKKEEPING 루틴의 장점과 세 번째 방법의 hash 를 사용하여 오버헤드를 줄였다. 두 번째와 세 번째에 방법에 있어서 차이점은 time 을 기준으로 한 것이 아니라 event 를 기준으로 검색과 hash 를 한 것이다. Time-based 방식은 별도의 타이머를 사용함으로써 해당 tick 마다 인터럽트를 발생시키게 된다. 이는 시스템의 전체적인 성능 면에서 많은 리소스를 잠식하는 결과를 초래한다. 이에 대해 event-based 방식은 시스템의 필요에 의해 수행되는 필수 루틴의 과정에서 timeout 을 관리하므로 과중한 인터럽트 오버헤드를 줄일 수 있게 된다.

5. 실험

실험은 communication processor 인 MPC8250 을 사용한 세션 관리 시스템을 사용하였다. event based method 를 위해서 위에 제시한 <그림 4>의 방식을 사용하였고 time based method 는 <그림 4>의 방식에서 주기적(T)으로 전체의 세션 리스트의 만료를 검사하였고 timeout 갱신은 <그림 4>의 방식을 그대로 사용하였다.

테스트 환경은 <그림 5>와 같다.



<그림 5>

세션 생성 서버에서 64byte 의 세션이 랜덤 하게 바뀌는 udp 패킷을 내보내고 Traffic Monitor 컴퓨터를 통해 세션 관리 시스템이 처리하는 양을 관찰하였다. 세션이 랜덤하게 바뀌도록 udp 패킷 생성 프로그램인 mgen 의 source code 를 수정하여 사용 하였다. 두 방식을 각각 Event Based Timeout Management (EBTM)과 Time Based Timeout Management (TBTM)이라고 하면 실험 결과는 다음과 같다.

1) TBTM 방식

Timeout expiry 를 검사하는 resolution(T)을 1 초, 0.5 초, 0.1 초, 0.05 초로 바꿔가며 세션 관리 시스템이 처리하는 패킷의 양을 측정하여 나타냈다.

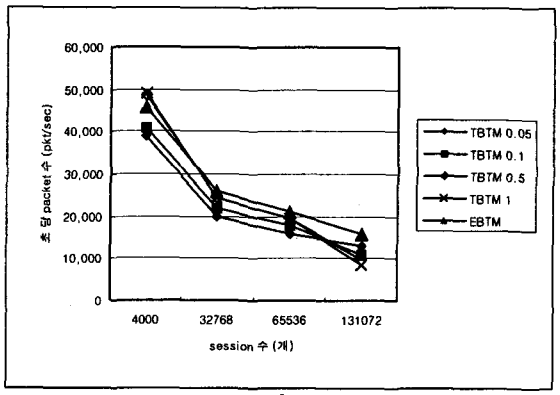
2) EBTM 방식

메모리에 저장되는 세션의 개수 S 를 변경하면서 세션 관리 시스템이 처리하는 패킷의 양을 측정하여 나타냈다. 세션 margin 의 개수만큼 세션 이 memory 에 저장되면 전체 세션 리스트의 timeout expiry 를 검사하여 메모리 사용 양을 조정한다.

전체 세션 의 양을 조정해가면서 실험 하였다.

패킷은 초당 50000 개를 전송하였다.

<그림 6>에서 Y 축은 세션 관리 시스템이 처리한 초당 packet 의 수이다.



<그림 6>

실험결과 EBTM 의 경우 세션 margin(S)의 변화와는 거의 무관한 결과를 보였다. 세션의 수가 30000 개 이상이면 EBTM 의 경우가 더 좋은 성능을 보이고 있다.

6. 결론

실험 결과를 보면 세션의 수가 많아 지면 time based 방식보다 event based 방식의 timeout 관리를 사용한 것이 더 좋은 성능을 보이고 있다. 세션이 적은 경우의 결과도 타이머의 resolution 이 세밀해 지면 event based 방식이 더 좋은 성능을 보이는 것을 알 수 있다.

참고문헌

[1] George Varghese and Anthony Lauck, Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility 1997

[2] A.S. Tanenbaum, Computer Networks 3rd ed. Upper Saddle River, NJ:Prentice-Hall

[3]T.Cormen, C Leiserson, and R.Rivest, Introduction to Algorithms. Cambridge, MA:MIT Press/McGraw-Hill

[4] J. G. Vaucher and P. Duval, "A comparison of simulation event list algorithms," Commun.ACM, vol.18,1975.

[5] L. Brakmo, S. O Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in Proc. ACM SIGCOMM'94, London, England.