이기종 SOAP 노드의 실시간 성능 모니터링 시스템

(A Performance Monitoring System for Heterogeneous SOAP Nodes)

이 우 중 † 김 정 선 ††

(Woo-Joong Lee) (Jungsun Kim)

요 약 본 논문에서는 Apache Axis나 .Net과 같은 SOAP 기반 미들웨어 상에 존재하는 SOAP 노드에 대한 실시간 성능 모니터링을 수행하는 방법으로써 네트워크 패킷 필터링을 통한 SOAP 오퍼레이션 검출 방법인 "TCP 흐름을 이용한 SOAP 오퍼레이션 검출 방법"을 제시하였다. 네트워크 패킷 필터링에 의한 SOAP 오퍼레이션 검출 방법은 Raw 패킷 내부에 단편화 되어서 전송되는 SOAP 메시지를 직접 분석하기 때문에 다양한 SOAP 기반 미들웨어에 독립적으로 SOAP 노드를 모니터링 할 수 있게 한다. 그러나 Raw 패킷들로부터 SOAP 메시지를 추출하여 분석하는 과정은 시스템의 많은 자원을 필요로 한다. 이러한 문제점을 해결하기 위하여 본 논문에서는 "TCP 플래그를 이용한 선별적인 TCP 흐름에서의 SOAP 오퍼레이션 검출 방법"을 제시하고 첫 번째 방법과의 성능을 비교하였다.

본 논문에서는 제시한 검출 방법을 바탕으로, 패킷 필터링을 통하여 SOAP 오퍼레이션을 검출하는 SOAP Sniffer 컴포넌트와 이를 이용한 SOAP 모니터링 시스템을 구현하였다. 본 논문에서 구현한 SOAP 모니터링 시스템은 SOAP 기반 미들웨어에 독립적인 모니터링 방법을 제공하므로 서로 다른 SOAP 기반 미들웨어 상에 존재하는 SOAP 노드 간 트랜잭션 모니터링이나 로드밸런싱을 위한 모니터링 등의 다양한 활용이 가능할 것이다.

키워드 : SOAP 모니터링 시스템, 패킷 필터링, SOAP 오퍼레이션 탐지

Abstract In this paper, we propose a novel performance monitoring scheme for heterogeneous SOAP nodes. The scheme is basically based on two-level (kernel-level and user-level) packet filtering of TCP flows. By TCP flow, we mean a sequence of raw packet streams on a TCP transaction. In this scheme, we detect and extract SOAP operations embedded in SOAP messages from TCP flows. Therefore, it becomes possible to monitor heterogeneous SOAP nodes deployed on diverse SOAP-based middlewares such as .Net and Apache AXIS.

We present two implementation mechanisms for the proposed scheme. The first mechanism tries to identify SOAP operations by analyzing all fragmented SOAP messages on TCP flows. However, a naive policy would incur untolerable overhead since it needs to copy all packets from kernel to user space. The second mechanism overcomes this problem by selectively copying packets from kernel to user space. For selective copying, we use a kernel-level packet filtering method that makes use of some representative TCP flags.(e.g. SIN, FIN and PSH). In this mechanism, we can detect SOAP operations only from the last fragment of SOAP messages in most cases.

Finally, we implement a SOAP monitoring system using a component called SOAP Sniffer that realizes our proposed scheme, and show experimental results. We strongly believe that our system will play a vital role as a tool for various services such as transaction monitoring and load balancing among heterogeneous SOAP nodes.

Key words : SOAP monitoring system/Packet filtering/SOAP operation detection

---

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10133-0)지원으로 수행되었음

† 학생회원 : 포항공과대학교 컴퓨터공학과
wjlee@postech.ac.kr

†† 종신회원 : 한양대학교 전자컴퓨터공학부 교수
jskim@cse.hanyang.ac.kr

논문접수 : 2004년 1월 13일
심사완료 : 2004년 9월 13일

1. 서 론

SOAP(Simple Object Access Protocol)[1-3]은 분산 컴퓨팅 환경에서 데이타를 교환하기 위한 XML 기반 프로토콜로써 분산 컴퓨팅 환경에서 상호 교환되는 데이타의 직렬화 방식을 표준화함으로써 다양한 분산 컴퓨팅 기반 기술을 통합 가능하게 한다. SOAP은 메시지 전송을 위하여 HTTP, SMTP, CORBA IIOP(Internet Inter-Orb Protocol), JMS(Java Messaging System) 등의 다양한 프로토콜에 바인딩 될 수 있기 때문에 다양한 플랫폼과 컴퓨팅 환경에서 활용 되고 있을 뿐만 아니라 점차적으로 임베디드 환경으로 영역을 넓혀가고 있다.

현재 SOAP은 웹 서비스[4]의 기반이 되는 기술로써 이를 지원하기 위한 SOAP 기반 미들웨어로는 Apache Axis, IBM WSTK, IONA XMLBus, BEA Weblogic 그리고 Microsoft .Net 등이 있다. 이러한 SOAP 기반 미들웨어들이 자체적으로 제공하는 모니터링 도구들은 단순히 SOAP 노드 사이에서 교환되는 메시지나 이들 메시지의 송수신 시간을 보여주는 기초적인 도구들이 대부분이다. 이들은 주로 SOAP을 이용한 프로그램을 작성 시 프로그램의 디버깅을 위한 목적으로 사용되며 SOAP 노드의 실시간 성능 분석을 위한 모니터링에 적용하기는 어렵다.

SOAP 노드의 성능 분석을 위한 일반적인 모니터링 방법은 SOAP 기반 미들웨어에서 제공하는 이벤트 로깅이나 SOAP 노드에서의 직접적인 이벤트 로깅을 통해 이루어진다. 그러나 서로 다른 SOAP 기반 미들웨어 상에 존재하는 SOAP 노드들 사이에서 일어나는 트랜잭션을 모니터링 하거나 로드밸런싱을 위한 모니터링을 하고자 한다면 SOAP 기반 미들웨어에 독립적인 모니터링 방법이 필요하다.

SOAP 기반 미들웨어에 독립적으로 SOAP 노드의 성능 모니터링을 수행하기 위한 방법으로 분산 객체 컴퓨팅을 위한 모니터링 툴킷 인 NetLogger[5]를 사용하는 것은 하나의 대안이 될 수 있다. NetLogger 툴킷은 응용 프로그램 코드에 이벤트 발생 시 이벤트를 원격지 로그 서버에 전송하기 위한 코드를 추가함으로써 다양한 환경의 분산 미들웨어 상에 존재하는 응용 프로그램이나 분산 객체의 모니터링을 가능하게 한다. 그러나 NetLogger는 기존에 동작하고 있는 프로그램 코드를 수정해야하는 단점을 가지고 있다.

본 논문에서는 네트워크 패킷 필터링을 이용하여 SOAP 노드의 오퍼레이션의 호출과 응답을 검출하고 이를 이용하여 SOAP 노드의 오퍼레이션 수행 시간과 같은 성능 요소에 대한 분석을 실시간으로 수행하는 방법을 제시한다. 이러한 방법은 네트워크를 통하여 전송되는 SOAP 메시지를 직접 분석하기 때문에 Apache Axis나 .Net 등과 같은 SOAP 기반 미들웨어에 독립적으로 각각의 미들웨어 상에서 동작하는 SOAP 노드에 대한 성능 모니터링을 가능하게 한다.

본 논문에서는 SOAP 오퍼레이션 검출을 위한 패킷 필터링 방법으로써 두 가지 방법을 제시하고 있다. 패킷 필터링은 커널 계층과 사용자 계층 필터링의 두 단계로 나누어서 고려할 수 있는데, 본 논문에서 제시하고 있는 SOAP 오퍼레이션 검출 방법 중 첫 번째는 커널 계층으로부터 지정된 포트로 들어오는 모든 패킷을 사용자 계층으로 복사하여 이를 순차적으로 사용자 계층에서 필터링을 통하여 오퍼레이션을 검출하는 방법이다. 이러한 방법은 SOAP 오퍼레이션 검출의 정확성은 높지만 SOAP 오퍼레이션 검출을 위하여 불필요한 패킷을 커널 계층으로부터 사용자 계층으로 복사 하여야 하는 문제점을 가지고 있다. 두 번째 방법은 커널 계층으로부터 불필요한 패킷 복사를 줄이기 위하여 커널 계층 패킷 필터링을 수행할 때 TCP 플래그를 활용하는 방안을 제시하고 있다.

한편, 본 논문에서는 제안된 검출 방법을 바탕으로 패킷 필터링을 통하여 SOAP 오퍼레이션을 검출하는 SOAP Sniffer 컴포넌트와 이를 이용한 SOAP 모니터링 시스템을 구현하였다. SOAP Sniffer 컴포넌트의 구현은 이식성을 위하여 플랫폼 독립적인 패킷 캡쳐를 제공하는 libpcap[8] 라이브러리를 사용하였으며 SOAP 프로토콜 바인딩에 따라 패킷 필터링 방법을 동적으로 변경 가능하게 하기 위하여 Strategy 패턴[9]을 사용함으로써 실질적으로 SOAP 기반 미들웨어에 독립적인 SOAP 노드의 모니터링을 가능하게 하였다.

본 논문의 2장에서는 패킷 필터링에 대한 개요를, 3장에서는 SOAP 메시지의 구조와 프로토콜 바인딩을 다루고 있으며 4장에서는 본 논문에서 제안하는 두 가지의 SOAP 오퍼레이션 검출을 위한 패킷 필터링 방법을 제시하고 5장에서는 이를 이용한 실질적인 SOAP 모니터링 시스템의 구현을 기술하고 6장과 7장에서는 제안된 두 가지 SOAP 오퍼레이션 검출 방법에 대한 성능 분석과 본 논문의 결론을 기술하였다.

2. 패킷 필터링

2.1 일반적인 패킷 필터의 구조

패킷 필터링[6,7]이란 네트워크 인터페이스를 통과하는 패킷들을 지정한 조건에 의하여 검사하고 조건에 해당하는 패킷을 검출하는 것을 말한다. 사용자 계층에서 응용 프로그램이 소켓을 열고 데이타를 송신하거나 수

신 받을 때 패킷 필터링 기능을 활성화 하면 링크 계층 드라이버는 패킷 필터에게 자신을 통과하는 패킷들을 전송한다. 이러한 구조를 통하여 네트워크 응용 프로그램 간의 데이타 송수신에 영향을 주지 않고 네트워크 인터페이스를 통과하는 패킷을 엿볼 수 있게 한다. 패킷 필터는 일반적으로 커널 계층 패킷 필터와 사용자 계층 패킷 필터로 구분 할 수 있으며 커널 계층 패킷 필터는 사용자 계층으로부터 패킷 필터링 규칙을 지정 받아서 규칙에 부합되는 패킷들을 사용자 계층으로 전달한다. 이렇게 사용자 계층으로 전달된 패킷은 다시 사용자 계층 패킷 필터에 의하여 필터링 될 수 있다. 커널 계층 패킷 필터는 일반적으로 OSI 참조 모델에서 전송 계층 이하의 하부 프로토콜에 대한 고정 크기의 패킷 헤더의 내용을 보고 패킷 필터링 규칙에 부합하는지 비교하여 패킷을 사용자 계층으로 전달하며 전송 계층 이상의 단편화된 상위 응용 프로토콜 데이타에 대한 필터링을 할 수 없다. 즉 이러한 상위 응용 프로토콜 데이타에 대한 필터링을 위하여 사용자 계층 패킷 필터링이 사용된다.

사용자 계층에서 바라보았을 때 커널 계층 패킷 필터를 활성화 하고 규칙을 지정한 뒤 규칙에 부합하는 패킷을 검출하는 것을 "패킷 캡처"라 한다. 이렇게 캡처된 패킷은 사용자 계층의 패킷 필터에 의하여 분석 후 네트워크 모니터링 정보로 활용될 수 있다. 그림 1은 일반적인 패킷 필터의 구조를 개략적으로 나타낸 것이다.

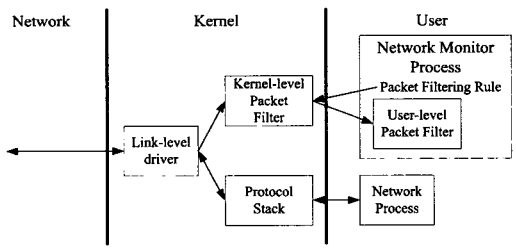


그림 1 일반적인 패킷 필터의 구조

2.2 Libpcap(Portable Packet Capture Library)

위에서 살펴본 커널 계층 패킷 필터의 구조는 플랫폼마다 서로 다르다. 이러한 패킷 필터는 BPF(Berkeley Packet Filter)[7], LSF(Linux Socket Filter), DLPI, NIT, SNOOP, SNIT, SOCK_PACKET 등이 있으며 Libpcap[8]은 다양한 플랫폼에서 동작하는 패킷 필터들의 상위에서 단일한 인터페이스에 의한 접근을 가능하게 해주는 라이브러리이다.

Libpcap은 TCPDump[10]를 개발하는 과정에서 이식성 있는 패킷 캡쳐를 지원하기 위하여 만들어진 라이브러리로써 HTTPDUMP[11], WebTrafMon[12] 등의 네트워크 모니터링 도구 뿐 아니라 Snort[13]와 같은 침입탐지 시스템에서도 사용되고 있다.

2.3 TCP/IP 에서 libpcap을 이용한 패킷 필터링

이제 좀더 구체적으로 TCP/IP에서 libpcap을 이용한 실질적인 패킷 필터링을 살펴보자. 그림 2에서는 TCP/IP에서의 패킷 필터링을 나타내었다.

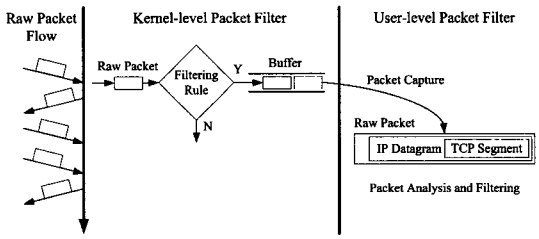


그림 2 TCP/IP에서의 패킷 필터링

위의 그림 2와 같이 네트워크 인터페이스를 통과하는 Raw 패킷들이 커널 계층 패킷 필터로 전달되면 커널 계층 패킷 필터는 Raw 패킷의 TCP/IP 헤더의 필드 값에 대하여 필터링 규칙을 적용하고 규칙에 부합하는 패킷을 검출하여 버퍼링 한다. 일반적으로 TCP/IP 헤더의 각각의 필드의 위치는 Raw 패킷의 시작점으로부터 고정적인 거리만큼 떨어져 있으므로 헤더의 필드 값을 비교하는 것은 간단한 작업이다. 커널 계층 패킷 필터에 의하여 버퍼링 된 Raw 패킷은 Blocking 혹은 Non-blocking I/O 방식에 의하여 사용자 계층 패킷 필터로 캡쳐 될 수 있으며 이후 사용자 계층 패킷 필터에 의하여 Raw 패킷에 대한 분석과 필터링이 수행 된다.

Raw 패킷은 그림 2에서 보인 것처럼 IP 데이타 그램을 포함하고 있다. 사용자 계층 필터는 응용 계층 프로토콜에 대한 필터링을 위하여 이러한 Raw 패킷으로부터 IP 데이타 그램과 IP 데이타 그램 내부의 TCP 세그먼트, 그리고 최종적으로 TCP 세그먼트의 데이타 필드에서 단편화 된 응용 계층 프로토콜의 헤더와 데이타를 추출할 수 있다.

3. SOAP 메시지와 프로토콜 바인딩

2장의 패킷 필터링 방법을 통해서 TCP 세그먼트로부터 단편화된 응용 계층의 헤더와 데이타를 검출하는 방법을 알아보았다. 이제 본 논문에서 구현 하고자 하는 패킷 필터링을 이용한 SOAP 오퍼레이션의 검출을 위하여 3장에서는 SOAP 노드 사이에서 교환되는 SOAP 메시지의 구조와 SOAP 메시지 전송을 위한 프로토콜 바인딩에 대해서 알아보고 4장에서는 TCP 세그먼트를 통해서 전달되는 단편화된 응용 계층의 헤더와 SOAP

메시지를 통하여 SOAP 오퍼레이션을 검출 하는 방법에 대하여 기술 하고자 한다.

3.1 SOAP 메시지의 구조와 프로토콜 바인딩

SOAP 메시지는 XML을 이용하여 정의 되며 그림 3과 같이 "Envelope" 요소 내부에 "Header"와 "Body" 요소를 가지고 있다. "Header"는 선택적 요소이며 보안이나 트랜잭션 등 부가적인 정보를 가지고 있으며 "Body" 요소는 필수 요소로서 오퍼레이션과 SOAP 프로토콜에 의하여 XML로 직렬화 된 오퍼레이션의 인자나 리턴 값을 포함하고 있다. 그림 4에서는 SOAP 메시지 중 오퍼레이션과 관련된 "Body" 요소의 실질적인 예를 보이고 있다.

그림 3의 SOAP 메시지의 예는 "GetPrice" 오퍼레이션을 호출하는 SOAP 메시지의 예이다. "GetPrice" 오퍼레이션은 인자로 문자열 타입의 "itemcode"를 가지고 호출되는 것을 알 수 있다. SOAP은 인자나 리턴 값으로 위의 예에서처럼 기본 타입의 데이타를 전송할 수 있을 뿐 아니라 구조체나 배열과 같은 복합 데이타 타입이나 XML 문서 자체를 전송할 수도 있다.

이제 위에서 소개한 SOAP 메시지의 전송을 위한 프로토콜 바인딩에 대해서 살펴보자. 서론에서 언급한 것처럼 SOAP 메시지는 전송을 위해서 다른 프로토콜과의 바인딩이 필요하다. SOAP 메시지는 기본적으로 어떠한 프로토콜에도 바인딩 가능하다. 실례로 HTTP, SMTP를 비롯하여 CORBA의 IIOP(Internet Inter-Orb Protocol)나 JMS(Java Messaging System)와 같은 프로토콜에 바인딩 될 수 있으며 SOAP 명세서에는 이러한 프로토콜 중 HTTP 프로토콜과 SMTP 프로토콜 바인딩을 기술하고 있다. 이중 HTTP 프로토콜 바인딩은 SOAP 프로토콜 바인딩 중에서 가장 널리 사용되고 있으며 RPC(Remote Procedure Call) 메커니즘을 지원한다. 이러한 HTTP 프로토콜 바인딩의 예를 살펴보면 그림 4와 같다.

그림 4의 요청에서는 HTTP 헤더의 "POST" 키워드

Request

```
POST /Stock HTTP/1.1
HOST : www.kerbung.org
Content-Type: application/soap+xml charset="utf-8"
Content-Length: nnnn

<?xml version="1.0">
<SOAP-ENV:Envelope
 xmlns:SOAP-Env="http://www.w3.org/2001/12/soap-envelope"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 ...
  <SOAP-ENV:Body>
    <m:GetPrice xmlns:m="www.kerbung.org/Stock">
      <itemcode xsi:type="xsd:string">n12345</itemcode>
    </m:GetPrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

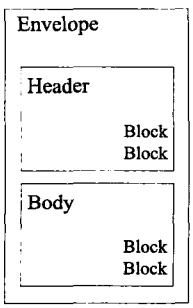
Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml charset="utf-8"
Content-Length: nnnn

<?xml version="1.0">
<SOAP-ENV:Envelope
 xmlns:SOAP-Env="http://www.w3.org/2001/12/soap-envelope"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 ...
  <SOAP-ENV:Body>
    <m:GetPriceResponse xmlns:m="www.kerbung.org/Stock">
      <itemcode xsi:type="xsd:int">10000</itemcode>
    </m:GetPriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

그림 4 SOAP HTTP 바인딩

를 통하여 오퍼레이션을 호출하기 위한 SOAP 노드의 경로인 "/Stock"을 지정하고 있으며 응답에서는 "HTTP 1.1/ 200 OK" 메시지를 이용하여 오퍼레이션의 성공여부를 함께 전송한다. 일반적으로 SOAP 노드의 경로는 SOAP을 바인딩 하고 있는 프로토콜에 의존한다. 예를 들어 SMTP 바인딩의 경우에는 "Stock@kerbung. org"와 같은 형식으로써 오퍼레이션의 경로를 지정한다. 4장에서는 이러한 정보를 토대로 SOAP 오퍼레이션을 검출하고 모니터링 하기 위한 패킷 필터링 방법에 대해서 기술 할 것이다.

4. SOAP 오퍼레이션 검출을 위한 사용자 계층 패킷 필터링



Simple Example

```
<?xml version="1.0">
<SOAP-ENV:Envelope
 xmlns:SOAP-Env="http://www.w3.org/2001/12/soap-envelope"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 ...
  <SOAP-ENV:Body>
    <m:GetPrice xmlns:m="www.kerbung.org/Stock">
      <itemcode xsi:type="xsd:string">n12345</itemcode>
    </m:GetPrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

그림 3 SOAP 메시지의 구조와 예

4.1 개요

네트워크 인터페이스를 통과하는 Raw 패킷으로부터 SOAP 오퍼레이션을 검출하기 위해서는 SOAP 메시지를 전송하기 위해 바인딩 된 HTTP나 SMTP와 같은 응용 프로토콜의 헤더와 SOAP 메시지의 분석이 필요한데 네트워크를 통해서 전송되는 과정에서 이들은 Raw 패킷 내부의 TCP 세그먼트에 단편화 되어서 전달된다. 그 이유는 일반적으로 SOAP 메시지 전송을 위한 프로토콜의 헤더와 SOAP 메시지의 크기가 MSS (Maximum Segment Size)보다는 크기 때문이다. 전송을 위한 프로토콜의 헤더와 SOAP 메시지가 하나의 패킷 안에 존재하지 않으므로 하나의 SOAP 오퍼레이션을 검출하기 위해서는 이들 패킷들을 "TCP 흐름"으로 묶어서 고려할 필요가 있다. 본 논문에서 언급하고 있는 TCP 흐름은 TCP 프로토콜에 의한 클라이언트와 서버 간의 통신에서 TCP 연결부터 종료까지를 하나의 TCP 트랜잭션이라고 할 때 하나의 TCP 트랜잭션에 속한 모든 패킷들의 집합이다.

그림 5는 패킷 필터링 상에서 단일 SOAP 송수신 노드 사이의 TCP 흐름과 복수 SOAP 송신 노드와 단일 수신 노드 사이의 중첩된 TCP 흐름을 개략적으로 나타낸 것이다.

그림 5(a)는 단일 SOAP 송수신 노드 사이의 TCP 흐름을 나타내고 있으며 5(b)는 복수 SOAP 송신 노드와 단일 수신 노드 사이의 중첩된 TCP 흐름을 보여주고 있다. 그림 5(b)와 같이 복수개의 TCP 흐름이 존재할 때 A와 B의 흐름을 구성하는 모든 패킷이 패킷 필터의 입력으로 들어오게 될 것이다. 이때 SOAP 송신 노드 A가 호출하는 SOAP 오퍼레이션을 검출하는 과정을 생각해보자. 단 그림 5(b)의 TCP 흐름 내에서 SOAP 송신 노드로부터 수신 노드로 전송되는 흐름내의 모든 패킷들(A1, A2, A3, A4, A5)은 모두 TCP 세그먼트의 데이타 필드에 SOAP 메시지와 메시지 전송을 위한 하부 계층 프로토콜의 헤더가 포함된 패킷이라고 가정하자. SOAP 오퍼레이션을 검출하기 위하여 이들 모두를 분석하는 것은 비효율적이다. SOAP 메시지의 크기가 크면 클수록 하나의 흐름을 구성하는 패킷의 수는 증가한다. 하나의 TCP 흐름 중 SOAP 오퍼레이션의 검출이 끝난 이후의 패킷은 더 이상 TCP 데이타 필드를 분석할 필요가 없다. 예를 들어 A1 과 A3 패킷의 TCP 세그먼트의 데이타 필드에 SOAP 오퍼레이션 검출을 위한 데이타가 들어있다고 가정하고 A3 패킷을 통하여 오퍼레이션의 검출이 끝났다면 A4와 A5패킷은 TCP 세그먼트의 데이타 필드 분석을 하지 않아도 충분하다. 이러한 방식은 SOAP 메시지가 크면 클수록 효율성을 가지게 될 것이다. 4.3절과 4.4절에서는 HTTP와

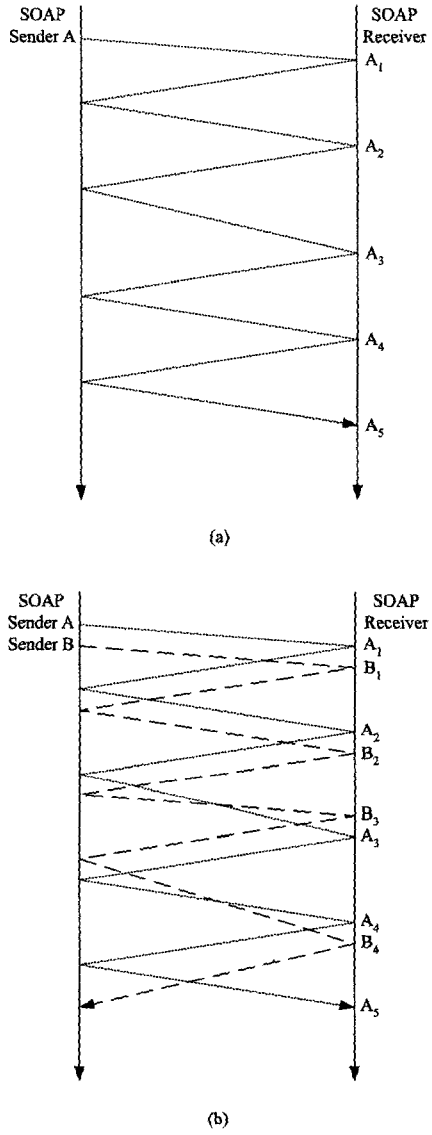


그림 5 SOAP 노드 사이의 TCP 흐름

SMTP상에서 TCP 흐름을 이용하여 효율적으로 SOAP 오퍼레이션을 검출하는 방법을 제시한다. 그러나 하나의 TCP 흐름 내에서 순차적으로 패킷을 분석하는 SOAP 오퍼레이션 검출 방법은 여전히 문제점을 가지고 있다. 그 이유는 TCP 세그먼트의 데이타 필드를 분석할 필요가 없는 패킷을 커널 계층 패킷 필터로부터 사용자 계층으로 캡쳐 하여야 하기 때문이다. 위의 예에서 A1과 A3를 통하여 SOAP 오퍼레이션이 검출 되었다면 A2와 A4 그리고 A5는 사용자 계층으로 불필요하게 캡쳐 된 패킷이다. 이러한 문제점을 해결하기 위한 방안으로 4.5절에서는 TCP 플래그를 이용한 선별적인 TCP 흐름에

서의 SOAP 오퍼레이션 검출 방법을 제시한다.

4.2 SOAP의 메시지 교환 패턴과 SOAP 오퍼레이션

SOAP의 메시징 메커니즘은 기본적으로 단 방향메시지 교환 패러다임을 제공하지만 이를 이용하여 request/response나 request/multiple response와 같은 메시지 교환 패턴을 만들어 낼 수 있다. 이러한 메시지 교환 패턴들을 구성하는 SOAP 오퍼레이션은 Synchronous 오퍼레이션(SOAP HTTP 바인딩)과 Asynchronous 오퍼레이션(SOAP SMTP 바인딩)으로 나눌 수 있다. 이러한 구분은 SOAP 오퍼레이션의 호출과 응답을 검출하는 방법에 있어서 차이를 가진다. Synchronous SOAP 오퍼레이션은 요청 이후에 데이타를 전송한 소켓 인터페이스를 닫지 않고 응답을 기다리며 이후 응답이 전송된 이후에 소켓 인터페이스를 닫는다. 즉 하나의 TCP 흐름에서 SOAP 오퍼레이션의 요청과 응답에 대한 검출을 모두 처리 할 수 있다. 그러나 Asynchronous SOAP 오퍼레이션은 SOAP 요청을 수신 측 SMTP 서버에 전송한 이후 연결을 종료하고 이에 대한 응답은 SOAP 요청을 송신한 쪽의 SMTP 서버를 통하여 받는다. 즉 SOAP 요청과 응답이 서로 다른 TCP 흐름을 통하여 이루어지므로 요청과 응답을 하나의 TCP 흐름에서 검출할 수 없다.

SOAP 오퍼레이션에서 요청과 응답의 검출은 중요하다. 그 이유는 SOAP 오퍼레이션의 성능 분석에서 오퍼레이션의 수행시간을 계산하기 위한 요소이기 때문이다. 그러나 Synchronous 또는 Asynchronous SOAP 오퍼레이션을 통하여 수행되는 one-way 메시지 교환 패턴의 응용 중 하나인 방송 서비스와 같은 경우에는 오퍼레이션의 수행 시간 보다는 기준 시구간 사이에서 호출된 오퍼레이션의 수가 성능 분석의 중요한 요소로 작용할 것이다.

요약하면 SOAP 오퍼레이션 검출 방법에서는 이러한 오퍼레이션의 형태에 대한 고려가 필요하다. 단순히 TCP 흐름을 유지 하는 것으로는 SOAP 오퍼레이션의 요청과 응답을 검출하기 어렵다. 본 논문에서는 SOAP 오퍼레이션을 검출하기 위하여 TCP 흐름과 더불어 SOAP 세션을 유지하기 위한 방법을 제시하고 있다. 구체적인 사항은 4.3절에서 다룬다.

4.3 SOAP HTTP 바인딩에서의 TCP 흐름을 이용한 SOAP 오퍼레이션의 검출

TCP 흐름에서 SOAP 오퍼레이션을 검출하기 위해서 먼저 HTTP SOAP 바인딩에서의 TCP 흐름의 예를 살펴보면 아래의 그림 6과 같다. 그림 6의 예에서 굵은 선은 단편화 된 HTTP 헤더와 SOAP 메시지를 포함한

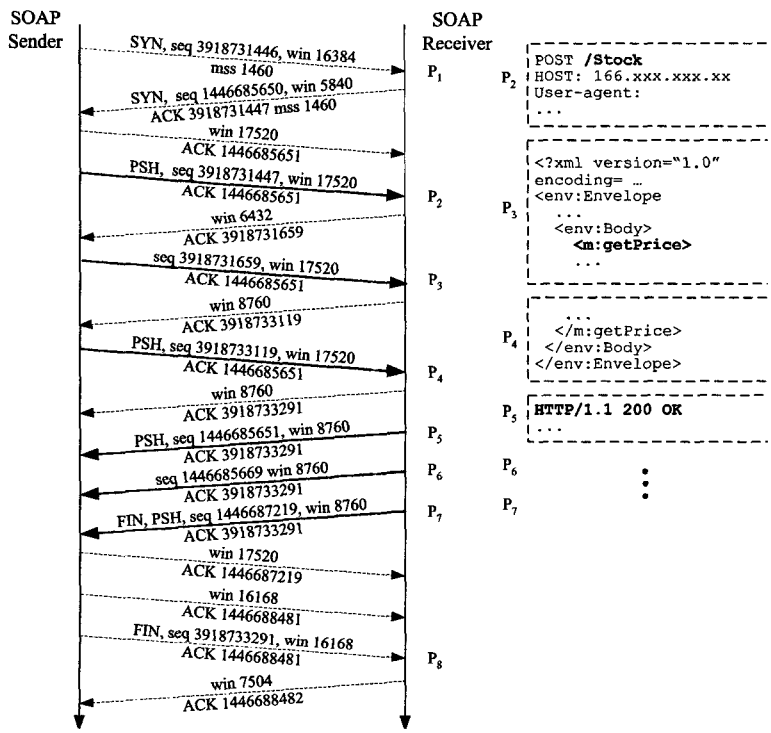


그림 6 SOAP HTTP 바인딩에서의 TCP 흐름

TCP 세그먼트를 나타내고 있다.

그림 6에서의 TCP 흐름을 살펴보면 SOAP 송신 노드가 P1을 통하여 접속을 요청한 이후 접속이 성립되면 P2, P3, P4를 통하여 SOAP 오퍼레이션의 요청이 전송된다. SOAP 수신 노드 측에서 요청에 대한 처리가 끝나면 P5, P6, P7을 통하여 SOAP 오퍼레이션의 응답이 전송되고 이후 P8에 의하여 접속을 종료한다. 이러한 TCP 흐름은 앞에서 기술한 것처럼 임의의 시 구간에서 중첩되며 패킷 필터로 입력되는 패킷들 사이에서 패킷이 속한 TCP 흐름을 유지하여야 SOAP 오퍼레이션의 요청과 응답을 검출 할 수 있다. 그림 6과 같은 SOAP HTTP 바인딩에서의 TCP 흐름을 유지하기 위해 사용할 수 있는 방법 중 하나는 해시 테이블을 이용하여 TCP 흐름 테이블을 만들고 SOAP 송신 노드의 IP 주소와 포트의 쌍을 TCP 흐름 키 값으로 사용하는 것이다. 이는 IP 헤더와 TCP 헤더의 Source Address 와 Source Port 또는 Destination Address와 Destination Port의 값을 이용하여 검출 할 수 있다. 패킷 필터에서 패킷의 Destination Address가 SOAP 수신 노드의 IP와 동일한지 비교하고 만일 동일하다면 패킷의 흐름 키 값을 Source Address, Source Port의 쌍으로 생성하고 만일 수신 노드의 IP가 아니라면 Destination Address, Destination Port의 쌍으로 생성 할 수 있다. 이는 중첩될 수 있는 TCP 흐름 내에서 유일하게 식별 될 수 있는 키 값이 될 수 있다. 그 이유는 동일한 호스트에서 호출 되는 여러 개의 SOAP 오퍼레이션도 서로 다른 포트를 이용할 수밖에 없기 때문이다.

SOAP HTTP 바인딩에서의 Synchronous 오퍼레이션의 경우는 위에서 설명한 TCP 흐름 테이블의 유지만으로 오퍼레이션의 호출과 응답을 해시테이블을 이용하여 검출 할 수 있다. 그러나 Asynchronous 오퍼레이션을 고려하여 독립적으로 SOAP 세션 테이블을 유지 하고 오퍼레이션 검출과 관련된 내용은 세션 테이블을 통하여 유지할 필요가 있다. 이때 SOAP 세션 테이블 역시 해시 테이블로 구성하여 빠른 검색이 가능하도록 하여야 할 것이다. 아래의 그림 7은 TCP 흐름테이블과 SOAP 세션 테이블의 구조와 각각의 필드의 값들에 대한 예를 보이고 있다.

그림 7의 TCP 흐름 테이블에서 세션 키는 HTTP 바인딩인 경우 흐름 키를 그대로 사용할 수도 있지만 UUID를 이용하여 유지할 수도 있을 것이다. 또한 Asynchronous 오퍼레이션(SOAP SMTP 바인딩)에서는 요청일 경우 Message-ID를 이용하여 세션 키를 만들고 이후 응답에서는 Reply-ID를 이용하여 기존의 세션에 접근 할 수 있을 것이다.

SOAP 세션 테이블은 TCP 흐름 테이블에 기록된 세션 키로 각각의 세션에 접근할 수 있다. 이때 Session flag 필드는 세션의 진행 단계를 구분하기 위하여 사용된다. 만일 처리된 내용이 하나도 없으면 0, 오퍼레이션 경로가 검출되었으면 1, 마지막으로 오퍼레이션 이름이 검출되었으면 2가 된다. Session flag 필드는 경우에 따라 확장을 할 수도 있다. 만일 오퍼레이션의 성공이나 실패여부까지 모니터링 하고자 한다면 Session flag에 3과 같은 새로운 값을 지정할 수도 있다. 이러한 Session flag는 오퍼레이션의 검출이 끝난 TCP 흐름에 속하는 TCP 데이타 필드에 데이타 값을 가진 TCP 세그먼트를 무시할 수 있게 함으로써 패킷 필터링의 효율을 높일 수 있게 한다.

이제 위의 TCP 흐름 테이블과 SOAP 세션 테이블을 이용하여 그림 6의 TCP 흐름을 처리하는 과정을 구체적으로 살펴보자. P_1과 같이 새로운 TCP 흐름을 생성하는 패킷을 검출하면 - TCP 헤더의 플래그 중 SYN 비트가 켜져 있다. - Source Address와 Source Port의 쌍으로 TCP 흐름 키 값을 생성하고 TCP 흐름 테이블과 SOAP 세션 테이블에 해당 키 값을 이용하여 TCP 흐름과 SOAP 세션을 생성한다. 이후 P_2, P_3, P_4에서는

TCP Flow table

Flow Key	Protocol	Session Key
166.104.xxx.xxx:36047	HTTP	166.104.xxx.xxx:36047 (Flow Key or UUID)
166.104.xxx.xxx:36007	SMTP	Id-1110 (Message ID or UUID)

SOAP Session table

Session Key	Start time	Finish time	Operation name	Operation path	Session flag
166.104.xxx.xxx:36047	2003-10-21 09:20:00	2003-10-21 09:20:10	GetPrice	/Stock	2
Id-1110	Null	Null	Null	Stock@xxx.com	1

그림 7 TCP 흐름 테이블과 SOAP 세션 테이블

Source Address와 Source Port를 이용하여 TCP 흐름 키 값을 생성하고 키 값으로 TCP 흐름테이블에서 TCP 흐름을 찾고 해당하는 세션 키를 찾아서 세션 테이블의 플래그를 보고 오퍼레이션을 검출한다. P_5, P_6, P_7과 같이 Destination Address가 서버의 주소가 아닌 패킷에 대해서는 Destination Address와 Destination Port를 이용하여 TCP 흐름 키를 생성하고 이를 이용하여 TCP 흐름 테이블에서 해당하는 세션 키를 찾아서 SOAP 세션테이블에 오퍼레이션의 응답을 처리하며 이후 P_8과 같이 TCP 흐름을 종료하는 패킷을 검출하면 - TCP 헤더의 플래그 중 FIN 비트가 켜져 있는 패킷이다. - Destination Address Destination Port를 이용하여 TCP 흐름 키를 구하고 이를 이용하여 해당하는 SOAP 세션과 TCP 흐름을 각각의 테이블 상에서 제거한다.

4.4 SOAP SMTP 바인딩에서의 TCP 흐름을 이용한 SOAP 오퍼레이션의 검출

SOAP SMTP 바인딩에서의 Asynchronous SOAP 오퍼레이션의 경우는 SOAP 세션이 하나의 TCP 흐름에서 끝나지 않는 것과 세션 키로서 SMTP 바인딩에서 정의하고 있는 Message-ID와 Reply-ID를 이용하여 세션 유지를 하는 것을 제외하고는 HTTP 바인딩과 차이가 없다. 3절과 유사한 방법으로 오퍼레이션의 요청과 응답을 검출해 낼 수 있으므로 자세한 알고리즘은 생략한다.

4.5 TCP 플래그를 이용한 선별적인 TCP 흐름에서의 SOAP 오퍼레이션 검출

3절과 4절에서는 TCP 흐름 유지 방법을 이용한 SOAP 오퍼레이션의 요청과 응답을 검출하는 방법을 보였다. 그러나 이러한 방법은 커널 계층으로부터 지정된 포트로 통신하는 모든 패킷을 검출하여 순차적으로 패킷의 TCP 데이타 필드를 분석하고 오퍼레이션의 검출이 모두 끝난 TCP 흐름에 대해서는 TCP 데이타 필드의 분석을 하지 않고 버리는 방법을 통해서 약간의 효율성을 꾀한다. 그러나 이는 근본적으로 커널 계층 패킷 필터로부터 사용자 계층 패킷 필터로 패킷을 복사하는 작업에서 발생하는 오버헤드를 줄이지 못하고 있다. 또한 SOAP 메시지에서 SOAP 헤더가 크면 클수록 SOAP 오퍼레이션 검출의 오버헤드가 증가한다.

본 논문에서는 이러한 문제점을 극복하는 방법으로서 "TCP 플래그를 이용한 선별적인 TCP 흐름에서의 SOAP 오퍼레이션 검출 방법"을 제시한다. 이 방법은 TCP 플래그 중 PSH 플래그를 활용하는 것이다. PSH 플래그는 가능한 한 빨리 수신자 측 TCP 버퍼의 내용을 수신자 측의 응용 프로그램에게 전달하라는 신호이다. PSH 플래그는 일반적으로 송신 측 응용 프로그램에서 지정할 수는 없지만 TCP/IP 스택의 구현상에서 송신자 측의 버퍼가 비거나 송신자 측에서 특정한 명령어를 수신자 측에 전달할 경우에 1로 설정되어 수신자 측 TCP/IP 스택으로 전달된다. 여기에서 주목해야 할 점은 HTTP나 SMTP의 헤더와 SOAP 메시지가 하나의 TCP 세그먼트 안에 포함되지 않아서 단편화 되는 경우 가장 마지막 TCP 세그먼트에는 반드시 PSH 플래그가 켜져서 전송된다는 점이다.

그림 8(a)는 그림 6에서 보인 TCP 흐름이며 그림 8(b)는 커널 계층 패킷 필터에서 TCP 플래그가 SYN, FIN, PUSH인 TCP 세그먼트를 포함하는 패킷을 사용자 계층 패킷 필터의 입력으로 전달했을 경우의 TCP 흐름이다. 그림 8(a)에서 볼 수 있듯이 P_2와 P_4 그리고 P_5와 P_7에서 PSH 비트가 켜져서 전송되었음을 알 수 있다. 이때 P_2와 P_5는 HTTP 헤더 정보를 포함한 패킷이고 P_4와 P_7은 SOAP 메시지의 마지막 부분이 TCP 세그먼트에 포함된 패킷이다.

이때 SOAP 메시지의 마지막 부분으로도 SOAP 오퍼레이션을 검출 할 수 있다는 점이 중요하다. 그림 9의 P_4의 경우를 살펴보면 데이타 필드의 끝으로부터 검색을 시작하여 "Body" 요소의 닫는 태그까지만 이동하면 SOAP 오퍼레이션을 지정하고 있는 요소의 닫는 태그로 접근할 수 있으며 이를 통하여 어떠한 오퍼레이션이 호출 되었는지 분석가능하다. 이는 TCP 세그먼트의 데이타 필드에 단편화 되어 전송되는 SOAP 메시지를 순차적으로 검색하는 방법보다 훨씬 더 빠르게 오퍼레이션을 검출해 낼 수 있으며 SOAP 헤더의 길이에 무관하게 일정한 속도를 가진다.

물론 SOAP 메시지가 매우 거대하여 HTTP나 SMTP헤더와 SOAP 메시지의 마지막 부분이 포함된 TCP 세그먼트를 포함하는 패킷 사이에 몇 번의 PSH 패킷이 더 존재 할 경우도 있다. 그러나 SOAP 메시지의 마지막인지 아닌지를 검출하는 것은 너무도 간단하다. 단지 TCP 데이타 필드의 마지막 부분에 "Envelope" 요소의 닫는 태그가 존재하는지만 살펴보면 이것이 마지막 필드인지 아닌지를 검출 해 낼 수 있다.

이러한 특성을 활용하기 위하여 커널 계층 패킷 필터의 규칙에서 특정한 TCP 플래그가 켜진 패킷만 사용자 계층 패킷 필터의 입력으로 복사되게 만들 수 있다. 즉 TCP흐름의 생성과 삭제를 위하여 SYN, FIN 플래그와 위에서 언급한 PSH 플래그가 켜져 있는 패킷만을 검출해 내도록 하게 하면 그림 8(b)의 경우에서처럼 불필요한 패킷의 복사를 막을 수 있다.

그림 9에서는 TCP 플래그를 활용한 선별적 TCP 흐름으로부터 SOAP 오퍼레이션을 검출 하는 방법을 보이고 있다. 그림에서 볼드체로 표기한 부분을 통하여 좀 더 명확하게 설명하면 P_4의 </m:getPrice>, P_7의 </m:

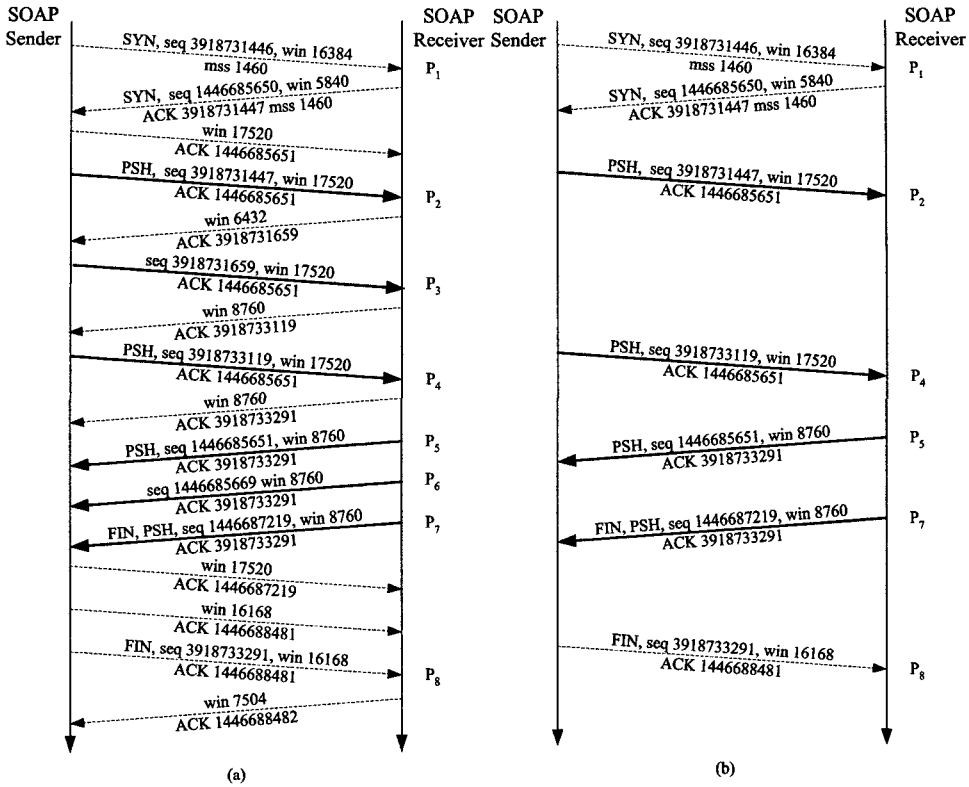


그림 8 TCP 플래그를 이용한 커널 계층 패킷 필터링에 의한 사용자 계층 패킷 필터의 입력 변화

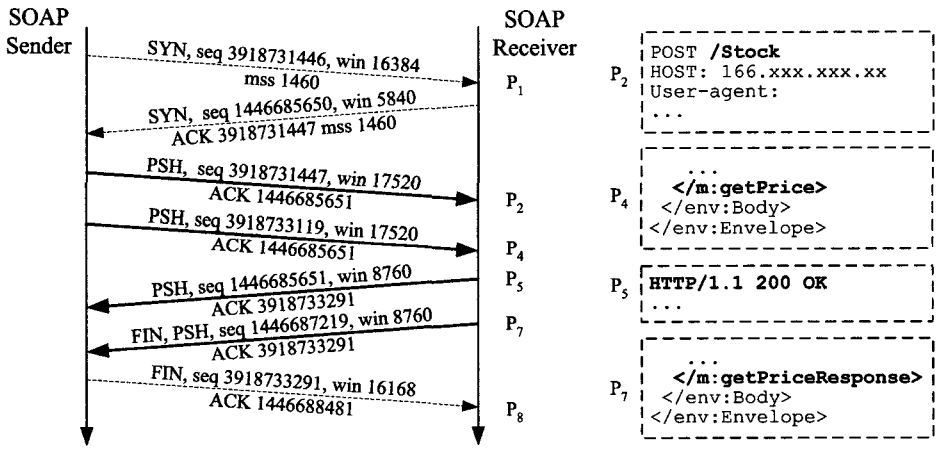


그림 9 SYN, PUSH, FIN 플래그를 활용한 오퍼레이션 검출 방법

getPrice Response> 태그와 같이 XML의 닫는 태그를 활용하여 SOAP 오퍼레이션의 검출을 수행할 수 있다.

5. SOAP 모니터링 시스템의 구현

본 논문에서의 구현은 시스템의 이식성을 위하여 Python[18]언어를 사용하였으며 libpcap 라이브러리의 Python모듈인 pylibpcap[19] 모듈과 Python 웹 서비스 모듈인 SOAPpy[20] 모듈을 이용하여 모니터링 시스템을 구현하였다. 또한 SOAP 프로토콜 바인딩에 따라 패킷 필터링 방법을 동적으로 변경 가능하게 하기 위하여 Strategy 패턴[9]을 사용하여 사용자 계층 패킷 필터를 구현하였다.

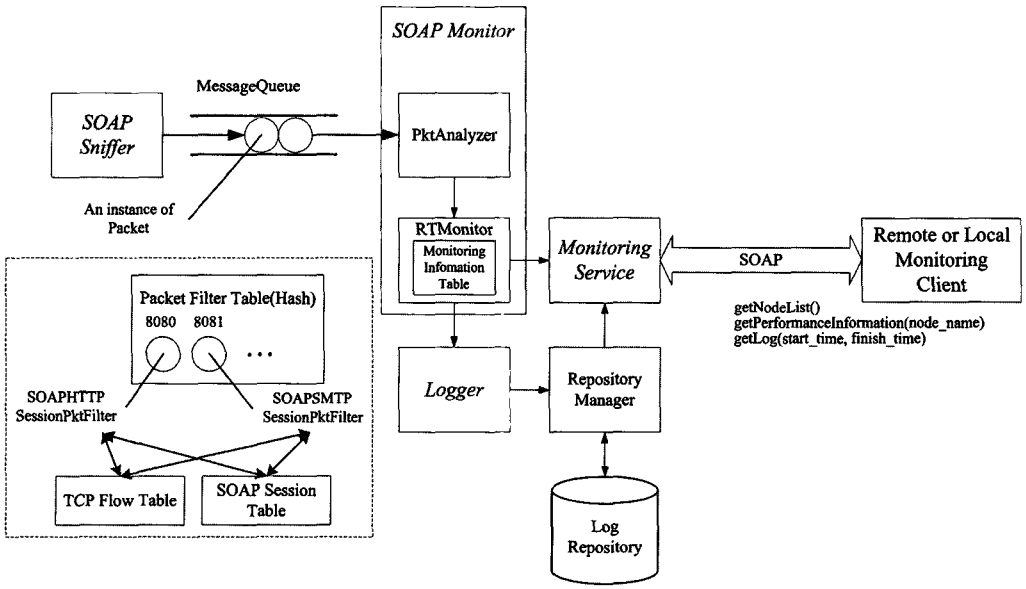


그림 10 모니터링 시스템의 구조

Nodelist.xml

```
<NodeList>
        <Node>
                <Name>Weather</Name>
                <Port>8080</Port>
                <Path>/WeatherService</Path>
                <ProtocolBinding>HTTP</ProtocolBinding>
                <Operation>getTemperature</Operation>
                <Operation>getWind</Operation>
                <Operation>getSummary</Operation>
        </Node>
        <Node>
                <Name>Stock</Name>
                <Port>8081</Port>
                <Path>Stock@kerbung.org</Path>
                <ProtocolBinding>SMTP</ProtocolBinding>
                <Operation>getPrice</Operation>
                ...
        </Node>
        ...
</NodeList>
```

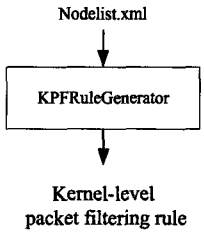


그림 11 커널 계층 패킷 필터링 규칙의 생성

그림 10은 본 논문에서 구현한 SOAP 모니터링 시스템의 구조이다. SOAP 모니터링 시스템은 크게 SOAP Sniffer, SOAP Monitor, Logger, 그리고 Monitoring Service 컴포넌트로 나눌 수 있으며 이들 각각의 컴포넌트는 독립적인 쓰레드로 동작한다.

5.1 SOAP Sniffer 컴포넌트

SOAP Sniffer 컴포넌트는 SOAP 오퍼레이션을 검출하기 위하여 커널 계층 패킷 필터링 규칙을 생성하고 커널 계층 패킷 필터로부터 규칙에 부합하는 Raw 패킷을 캡쳐 하여 이를 메시지 큐에 버퍼링 하는 컴포넌트이다. SOAP Sniffer는 메시지 큐에 패킷을 버퍼링 할 때 Strategy 패턴을 이용하여 패킷 필터 테이블에 등록되어 있는 패킷 필터의 레퍼런스와 함께 이를 버퍼링 함으로써 이후 SOAP Monitor 컴포넌트의 PktAnalyzer가 적합한 패킷 필터링 방법을 사용하여 패킷 필터링을 수행할 수 있게 한다. 이에 대한 구체적인 내용은 5.1.2절에서 기술하고 있다.

5.1.1 커널 계층 패킷 필터링 규칙의 생성

SOAP Sniffer 컴포넌트의 KPFRuleGenerator는 모니터링 하고자 하는 SOAP 노드의 리스트를 전달 받아서 커널 계층 패킷 필터링 규칙을 생성해준다.

그림 11은 Nodelist.xml 파일을 통하여 커널 계층 패킷 필터링 규칙을 생성하는 예를 보이고 있다. Nodelist.xml 파일의 "NodeList" 요소는 "Node" 요소를 가지고 있으며 각각의 "Node" 요소는 모니터링 하고자 하는 SOAP Node의 이름, 사용하는 포트, 오퍼레이션을 호

출하기 위한 SOAP 노드의 경로와 프로토콜 바인딩 및 노드가 가지고 있는 오퍼레이션의 이름을 기술한다. 이러한 Nodelist.xml 파일은 커널 계층 패킷 필터링 규칙의 생성 뿐 아니라 SOAP 프로토콜 바인딩에 따라 동적으로 패킷 필터링 규칙을 변경 가능하게 하기 위한 패킷 프로세서 테이블의 구성과 SOAP 모니터 컴포넌트 내부의 모니터링 정보 테이블의 스키마를 제공한다.

KPFRuleGenerator 위의 그림 11과 같은 Nodelist.xml 파일을 통해 커널 계층 패킷 필터링 규칙을 생성하며 이를 통해 생성된 커널 계층 패킷 필터링 규칙[10]의 예를 들면 그림 12와 같다.

그림 12의 d)에서의 커널 계층 패킷 필터링 규칙은 포트 번호가 8080이거나 8081이면서 TCP 플래그가 SYN, PUSH, FIN인 Raw 패킷을 검출한다. 이때 규칙의 적용의 순서는 효율성을 위하여 무시할 수 없는 요소이다. 본 구현에서는 커널 계층 패킷 필터링 시에 비교 연산의 수를 줄이기 위하여 먼저 포트 정보로 패킷을 검출한 뒤 TCP 플래그를 적용하도록 필터링 규칙을 생성하였다.

a) tcp
b) udp port 80
c) (port 23 or port 25) and (tcp[tcpflags]&(tcp-syn) != 0)
d) (port 8080 or port 8081) and (tcp[tcpflags]&(tcp-syn|tcp-push|tcp-fin) != 0)

그림 12 커널 계층 패킷 필터링 규칙의 예

5.1.2 Strategy 패턴을 이용한 동적인 사용자 계층 패킷 필터링

SOAP 미들웨어에 독립적인 패킷 필터링을 위하여 사용자 계층 패킷 필터링 시에 SOAP 프로토콜 바인딩에 따라서 동적으로 패킷 필터링 규칙을 변경할 수 있어야 한다. 이러한 처리를 위하여 본 논문의 구현에서는 Strategy 패턴을 적용하였다. 이를 이용한 방법을 구체적으로 설명하기 위하여 먼저 패킷 필터 테이블에 대하여 기술하면, 이는 앞에서 소개한 Nodelist.xml 파일의 "Port" 요소와 "ProtocolBinding" 요소를 추출하여 포트와 특정한 프로토콜 바인딩을 처리하기 위한 패킷 필터를 매핑하고 있는 해시 테이블이다. 패킷 필터 테이블의 구조와 사용 예를 살펴보면 그림 13과 같다.

Packet Filter Table

Port	PacketFilter
8080	HTTPSesstionPktFilter
8081	SMTPSessionPktFilter

그림 13 패킷 필터 테이블

패킷 필터 테이블에서 "HTTPSessionPktFilter"나 "SMTPSession PktFilter"와 같은 패킷 필터는 그림 14에서와 같이 PktFilter 클래스를 상속받아서 구현된 실질적인 패킷 필터의 인스턴스로써 이러한 패킷 필터는 앞의 4장에서 기술한 패킷 필터링 방법을 구현한 것이다.

이제 위에서 보인 패킷 필터 테이블을 사용하여 Raw 패킷을 처리하기 위한 패킷 필터 인스턴스의 레퍼런스를 함께 버퍼링 하는 과정을 살펴보자. 그림 14에서 PktFilter 클래스는 패킷 필터들을 위한 인터페이스이며 Packet 클래스는 Raw 패킷으로부터 파싱 된 IP 데이타그램과 이를 처리하기 위한 패킷 필터의 인터페이스인 PktFilter의 레퍼런스를 가지고 있다. 이러한 구조는 Packet 인스턴스의 process 메서드를 호출하면 Packet 인스턴스가 가지고 있는 IP 데이타 그램의 처리를 PktFilter 인터페이스를 통하여 Packet 인스턴스에 등록되어 있는 HTTPSessionPktFilter와 같은 패킷 필터에게 위임하게 한다. 즉 SOAP Sniffer 컴포넌트는 이러한 Packet 인스턴스를 메시지 큐에 버퍼링 함으로써 SOAP Monitor 컴포넌트의 PktAnalyzer에 의한 동적인 패킷 필터링의 수행을 가능하게 한다. - Packet-Analyzer는 메시지 큐에 버퍼링 된 Packet 인스턴스를 가져와서 process 메서드만 호출하면 된다. - 그림 15

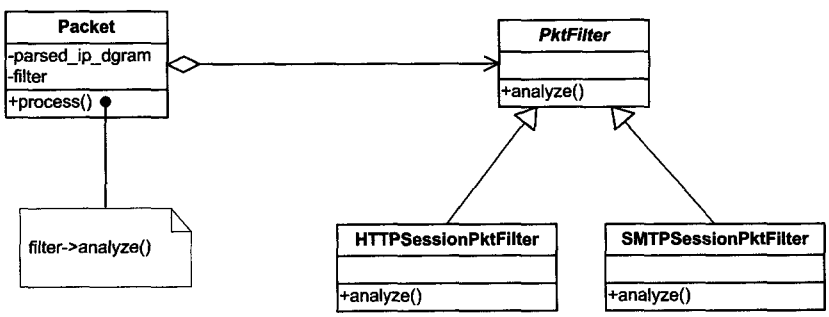


그림 14 Strategy 패턴을 적용한 패킷 필터의 구조

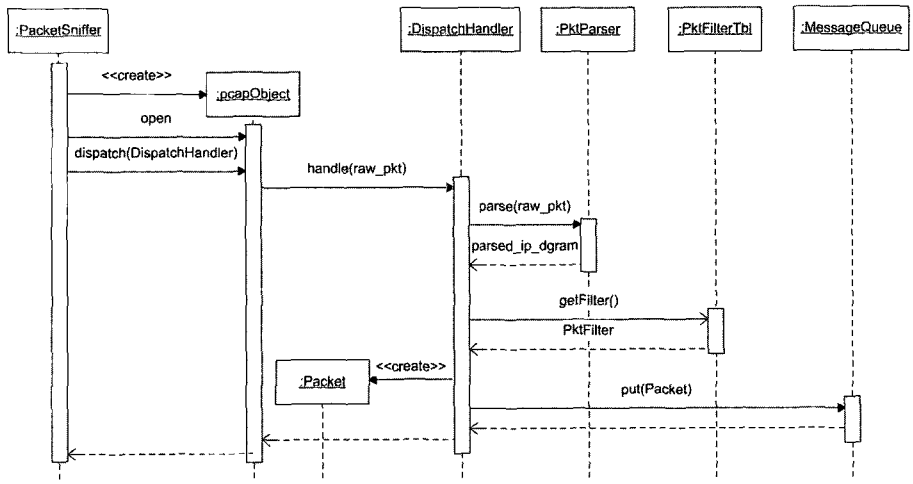


그림 15 SOAP Sniffer 컴포넌트 내에서 Raw 패킷을 Packet의 인스턴스로 만들고 이를 버퍼링 하는 과정

는 SOAP Sniffer 컴포넌트 내에서 Raw 패킷을 Packet의 인스턴스로 만들고 이를 버퍼링 하는 과정을 구체적으로 나타낸 것이다.

5.2 SOAP Monitor 컴포넌트

SOAP Monitor 컴포넌트는 모니터링 정보를 유지하기 위한 RTMonitor와 메시지 큐로부터 Packet의 인스턴스를 꺼내어 패킷 필터링을 수행하기 위한 Packet-Analyzer로 구성된다.

RTMonitor는 Monitor Object 패턴[21]을 사용하여 구현되었으며 SOAP 노드들에 대한 모니터링 정보를 유지하고 있다. RTMonitor는 Nodelist. xml 파일에 의하여 생성된 모니터링 정보 테이블을 가지고 있다. 이는 Python 언어의 내장형 자료구조인 Dictionary를 통하여 유지하고 있으며 이의 구조를 살펴보면 그림 16과 같다.

Structure of the Monitoring Information Table

{"NodeName":{"OperationA":[RequestsPerMin, MeanExTime, ActiveUsers], ...}

Example

{"Weather":{"getTemperature":[11,1,4], "getWind":[2,1,0], "getSummary":[7,10,1]},

"Stock":{"getPrice":[13,3,2]},

"Shop1":{"getList":[20,8,2], "getItem":[30,2,1]},

"Shop2":{"getList":[12,4,1], "getItem":[30,3,1]},

"Calendar":{"getMonth":[11,5,2], "getYear":[13,7,2]}

... }

그림 16 모니터링 테이블의 구조와 예

모니터링 정보테이블에서 모니터링 데이타 리스트는 "[분당 오퍼레이션 요청 수, 평균 실행 시간, 현재 호출 중인 SOAP 노드의 수]"로 구성되며 Nodelist.xml 파일에 의하여 "[0,0,0]" 으로 초기화 되어서 생성된다. 이러한 모니터링 데이타 리스트는 SOAP 세션 테이블의 정보를 이용하여 타이머에 의해 주기적으로 업데이트 된다.

5.3 Logger 컴포넌트

Logger 컴포넌트는 타이머를 이용하여 지정된 주기에 따라 SOAP Monitor 컴포넌트의 RTMonitor를 통하여 유지되고 있는 모니터링 정보 테이블의 데이타를 로깅 한다.

```
<Log time=2003-10-30-20-15-30>
        <NodeA>
                <Operation>
                        <name></name>
                        <ExecutionTime></MeanExTime>
                        <ReqPerMin></ReqPerMin>
                        <ActiveUser></ActiveUser>
                </Operation>
                ...
        </NodeA>
        <NodeB>
                ...
        </NodeB>
        ...
</Log>
```

그림 17 XML을 이용한 로그 데이터의 예

5.4 Monitoring Service 컴포넌트

Monitoring Service 컴포넌트는 SOAPpy모듈을 이용하여 웹 서비스로 구현하였으며 SOAP Monitor 컴포넌트의 RTMonitor와 Repository - Manager를 통하여 다음과 같은 오퍼레이션을 모니터링 클라이언트에게 제공한다.

- getNodeList() - 모니터링 하고 있는 SOAP 노드의 리스트를 제공한다.
- getPerformanceInformation(node_name) - 인자로 받은 SOAP 노드에 대한 실시간 성능 정보를 제공한다.

• getLog(start_time, finish_time) - 지정된 시 구간 내의 로그 기록을 제공한다.

6. 성능 분석

6.1 실험 환경

실험은 Intel® Pentium® 4 프로세서(1700 MHz)와 RAM 크기가 512 Mbyte인 Linux 서버 상에서 수행하였으며 Apache Axis와 SOAPpy를 이용한 Echo-Service와 EchoClient를 각각 구현하였다. 또한 측정의 오차를 줄이기 위하여 Echo Client가 EchoService를 10000번씩 호출하고 이의 평균값을 취하였다.

6.2 SOAP 오퍼레이션 검출 방법의 성능 분석

6.2.1 사용자 계층으로의 패킷 복사량

본 논문에서 제시한 두 가지 SOAP 오퍼레이션 검출 방법의 성능 분석을 위하여 먼저 커널 계층 패킷 필터를 통하여 사용자 계층 패킷 필터로 전달되는 패킷의 수를 실험을 통하여 구하였다. 그림 18은 데이타 크기에 따라서 변화되는 패킷의 수를 보이고 있다.

그림 18에서 볼 수 있듯이 커널 계층 패킷 필터에서 TCP 플래그를 이용하여 필터링 할 경우 패킷의 수의 변화가 거의 없는 것을 볼 수 있다.

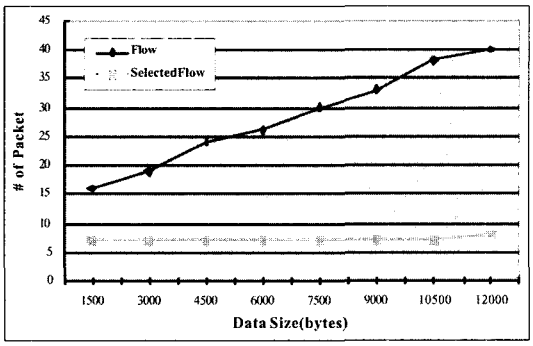


그림 18 사용자 계층으로 복사된 패킷 수

6.2.2 패킷 인스턴스 생성 및 버퍼링 수행시간

6.2.1의 실험에서는 커널 계층 패킷 필터를 통하여 사용자 계층 패킷 필터로 복사되는 패킷의 수를 살펴보았다. 그러나 이 결과에서는 커널 계층 패킷 필터를 통하여 패킷 필터링이 수행되는 시간에 대한 고려가 없다. 이를 위하여 커널 계층 패킷 필터를 통하여 수행되는 필터링 수행 시간과 이를 통하여 사용자 계층으로 전달된 Raw 패킷이 Packet 클래스로 인스턴스 화 되어 메시지 큐에 버퍼링 되기까지에 걸리는 수행시간을 비교하였다.

그림 19의 결과는 커널 계층으로부터 필요한 패킷만을 선별하여 사용자 계층으로 복사하는 데에 따르는 전체적인 효율의 차이를 보여주고 있다. TCP 플래그를 이용하여 커널 계층에서 미리 필터링을 수행하는 것이 사용자 계층으로 패킷을 모두 복사하여 필터링을 수행하는 것보다 훨씬 더 효율적임을 알 수 있다.

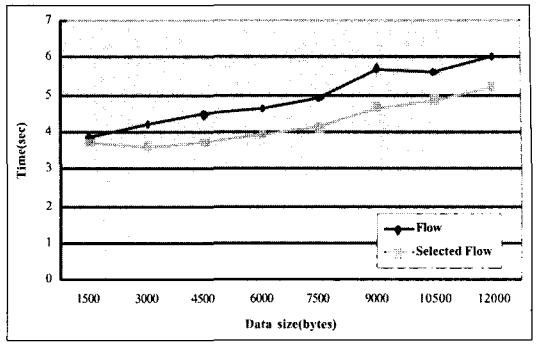


그림 19 Packet 인스턴스 생성 및 버퍼링 수행시간 비교

6.2.3 SOAP 오퍼레이션 검출

패킷으로부터 SOAP 오퍼레이션을 검출 하는 수행시간을 비고 해보면 아래의 그림 20과 같다.

그림 20의 결과는 사용자 계층 패킷 필터를 통하여 필터링을 수행하기 전 버퍼링 된 패킷의 수에 따른 오퍼레이션 검출 시간의 차이를 극명하게 보여주고 있다. 이러한 결과 값은 그림 18에서의 사용자 계층으로 복사된 패킷의 수에 의존함을 알 수 있다.

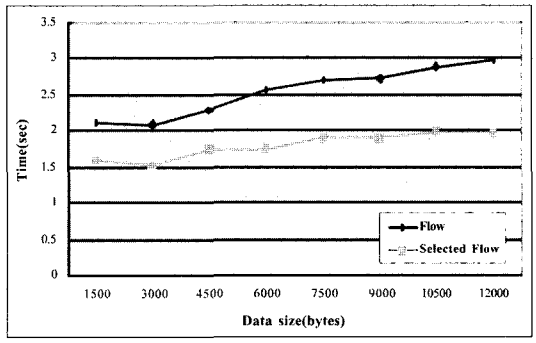


그림 20 SOAP 오퍼레이션 검출 시간 비교

6.2.4 모니터링 성능 평가

그림 19와 20의 결과로부터 선별적인 TCP 흐름에서의 SOAP 오퍼레이션 검출방법이 버퍼링 시간과 SOAP 오퍼레이션 검출에서 훨씬 더 효율적이며 SOAP 노드 간 메시지의 크기의 증가에 대한 영향이 훨씬 더 적음을 알 수 있다. 이를 종합하여 비교하면 그림 21과 같다.

마지막으로, 패킷 필터링을 통한 모니터링 방법이 SOAP 오퍼레이션 수행 성능에 미치는 영향을 알아보기 위하여, 단일 Echo 오퍼레이션의 수행시간을, 모니터

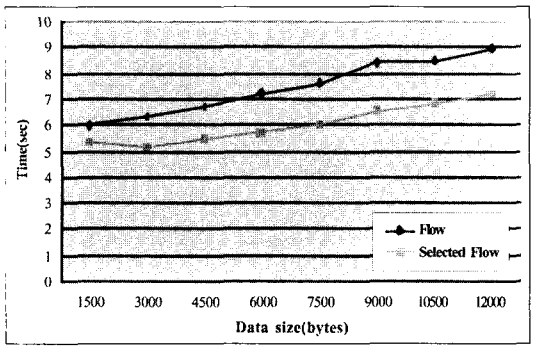


그림 21 모니터링 수행 성능 비교

링을 수행하지 않았을 때, TCP 흐름을 이용한 모니터링 수행, 그리고 선별적인 TCP 흐름을 이용한 모니터링을 수행하였을 때의 SOAP 오퍼레이션 수행 시간(msec)을 비교하여 보았으며. 그 결과는 그림 22와 같다.(수행시간을 구하기 위하여 Echo 오퍼레이션을 10000번 호출하고 평균값을 취하였다.)

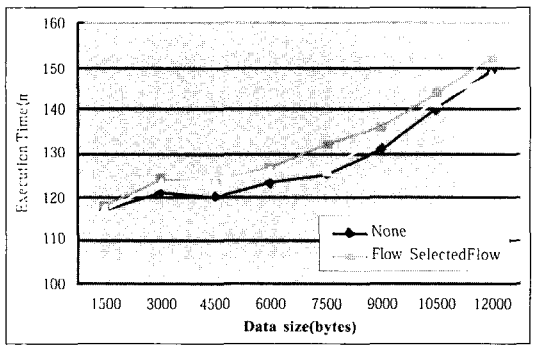


그림 22 SOAP 오퍼레이션 수행 성능 비교

그림 22의 결과를 살펴보면, 전반적으로 패킷 필터링을 이용한 모니터링 방법이 SOAP 오퍼레이션 수행 성능에 미치는 영향이 크지 않음을 알 수 있으며(모니터링 오버헤드 1~3% 내외), 선별적인 TCP 흐름을 이용한 모니터링 방법은 모니터링을 수행하지 않았을 때의 오퍼레이션 수행 시간과 거의 근접함을 알 수 있다.

결론적으로, 그림 21과 22에서 볼 수 있듯이 선별적인 TCP 흐름을 이용한 성능 개선 알고리즘이 전체적으로 좋은 성능을 보임을 알 수 있다. 그러나 본 논문에서 제시한 선별적인 TCP 흐름을 이용한 성능 개선 알고리즘은 한가지의 문제점을 가지고 있다. 그것은 패킷이 단편화 될 때, 검출해야할 오퍼레이션의 닫는 태그의 시작 부분부터 SOAP 메시지의 끝부분 사이에서 단편화가 일어난 경우, 오퍼레이션을 검출 할 수 없다는 점이다.

SOAP 메시지의 단편화는 TCP 계층에서 정의된 MSS(Maximum Segment Size)에 의하여 결정되며, MSS는 네트워크 트래픽과는 무관하게 일정한 크기를 유지한다. 그러므로 위와 같은 경우는 SOAP 메시지의 크기에 의존하는데 - 일반적으로 SOAP 프로토콜에 의하여 직렬화 되어서 전달되는 인자들의 크기에 의하여 결정 된다. - 전체 메시지의 크기에 비하여 오퍼레이션의 닫는 태그부터 SOAP 메시지의 끝부분까지의 크기는 매우 작으므로 검출하지 못하는 경우의 확률은 매우 낮다. 또한, 일반적인 모니터링 응용의 경우는 100%의 정확성을 필요로 하지 않는 경우가 많으므로 대부분의 모니터링 응용에서 선별적인 TCP 흐름을 이용한 성능 개선 알고리즘이 유리할 것이다.

7. 결론 및 추후 연구 과제

본 논문에서는 SOAP 기반 미들웨어에 독립적인 SOAP 노드의 실시간 성능을 모니터링하기 위한 방법으로써 패킷 필터링에 의한 SOAP 오퍼레이션 검출 방법을 제시하였으며 패킷 필터링을 통한 SOAP 오퍼레이션 검출에서의 비효율성을 극복하는 알고리즘을 제안 및 구현하고 이를 이용하여 SOAP 모니터링 시스템을 구현하였다.

본 논문에서 구현한 SOAP 모니터링 시스템은 SOAP 기반 미들웨어에 독립적인 모니터링 방법을 제공하므로 서로 다른 SOAP 기반 미들웨어 상에 존재하는 SOAP 노드 간 트랜잭션 모니터링이나 로드밸런싱을 위한 모니터링 등의 다양한 활용이 가능할 것이다.

참 고 문 헌

[1] W3C, "SOAP Version 1.2 Primer," W3C, June 2003.
[2] W3C, "SOAP Version 1.2 Part 1: Messaging Framework," W3C, June 2003.
[3] W3C, "SOAP Version 1.2 Part 2: Adjuncts," W3C, June 2003.
[4] Eric Newcomer, "Understanding Web Services," Addison Wesley, May 2002.
[5] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee, "NetLogger: A Toolkit for Distributed System Performance Analysis," In Proceedings of the IEEE Mascots 2000 Conference, Aug 2000.
[6] Jeffrey C. Mogul, Richard F. Rashid and Michael J. Accetta, "The Packet Filter : An Efficient Mechanism for User-level Network Code," In Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, pages 39-51. ACM Press, November 1987.
[7] Steven McCanne and Van Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," In Proceedings of the Winter

1993 USENIX Conference, pages 259-269. USENIX Association, January 1993.
[8] "Libpcap(Portable Packet Capture Library)," URL : http://www.tcpdump.org
[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns," Addison Wesley, pp 315-323, December 1998.
[10] "TCPDump," URL : http://www.tcpdump. org
[11] Roland Wooster, Stephen Williams and Patrick Brooks, "HTTPDUMP : Network HTTP Packet Snooper," 1998.
[12] J. Won-Ki Hong, Soon-Sun Kwon and Jae-Young Kim, "WebTrafMon: Web-based Internet/Intranet Network Traffic Monitoring and Analysis System," Computer Communications, Elsevier Science, Vol. 22, No. 14, September 1999, pp. 1333-1342.
[13] Snort, URL : http://www.snort.org
[14] W. Richard Stevens, "TCP/IP Illustrated volume 1," Addison Wesley, March 1996.
[15] W. Richard Stevens, "TCP/IP Illustrated volume 3," Addison Wesley, April 1996.
[16] "RFC 791: Internet Protocol," IETF, September 1981.
[17] "RFC 793: Transmission Control Protocol," IETF, September 1981.
[18] "Python," URL : http://www.python.org
[19] "pylibpcap," URL : http://pylibpcap.sourceforge.net
[20] "SOAPpy," URL : http://pywebsvcs.sourceforge.net
[21] Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, "Pattern-Oriented Soft- ware Architecture volume 2," Wiley, pp. 399-422, 1999.

이 우 중
2002년 한양대학교 화학공학과 학사
2004년 한양대학교 컴퓨터공학과 석사
현재 포항공과대학교 정보통신 연구소 연구원. 관심분야는 분산 객체 컴퓨팅, 임베디드 시스템

김 정 선
정보과학회논문지 : 컴퓨팅의 실제
제 10 권 제 3 호 참조