유비쿼터스 시스템을 위한 경량화된 결함분석기

현재명° 최창열 김성수
아주대학교 정보통신전문대학원
{jmhyun78°, clchoi, sskim}@ajou.ac.kr

Light-Weight Fault Analyzer for Ubiquitous System

Jaemyung Hyun° Changyeol Choi Sungsoo Kim
Graduate School of Information and Communication, Ajou University

요 약

현재 많은 사람들의 관심을 받고 있는 유비쿼터스 시스템(Ubiquitous System)이 주는 가장 큰 이점은 언제 어디서나 원하는 서비스를 사용할 수 있는 것이다. 따라서 이를 사용가능하게 하기 위해서는 유비쿼터스 시스템이 무선네트워크를 지원하여야 하며 또한 소형의 장치로 사용자가 간편하게 사용할 수 있어야 한다. 그러나 이러한 특성들로 인하여 유비쿼터스 시스템은 네트워크 불량, 자원 부족 등과 같은 문제점을 가지고 있으며 이것은 사용자 서비스의 질과 직접적으로 연결된다. 더욱이 이것으로 인하여 사용자가 서비스 사용에 불편을 겪는다면 유비쿼터스 시스템 사용에 대한 신뢰도는 떨어질 것이다. 이런 문제를 해결하기 위해서는 유비쿼터스 시스템에서 일어날 수 있는 결함을 분석하고 이에 대한 적절한 조치를 취하여 사용자가 서비스를 사용하는데 있어서 불편을 초래하지 않아야 한다. 따라서 본 논문에서는 유비쿼터스 시스템에서 사용될 수 있는 결함 분석기를 개발하였으며 또한 실제로 유비쿼터스 시스템 관리 유틸리티에 적용하였다. 본 결함 분석기에서는 메모리 결함, 배터리 결함, 네트워크 결함, 및 heartbit 결함에 대하여 정의하고 결함을 감지할 수 있도록 하였으며 또한 Java Exception 메시지를 이용하여 조기에 결함을 분석할 수 있도록 개발하였다.

1. 서 론

유비쿼터스에 대한 관심이 점차 증가함에 따라 여러 분야에서의 끊임없는 연구 결과로 유비쿼터스 시스템은 많은 발전을 거두었다. 또한 국가 정책 중 하나로 유비쿼터스에 대한 많은 투자가 이루어지면서 유비쿼터스 신도시 개발 등과 같은 사업으로 인하여 유비쿼터스는 사용자에게 한결 친근한 도구가 되어가고 있다.

유비쿼터스는 "언제 어디서나"라는 뜻으로 그 뜻에 적합하게 소형 디바이스와 무선통신 환경을 기본으로 사용하고 있어 사용자가 손쉽게 어디서나 사용할 수 있는 편리함을 제공하고 있다[1]. 그러나 이러한 유비쿼터스 시스템의 특징들이 사용자에게 편리함을 주는 반면 이로 인한 단점들 또한 나타나고 있다. 무선네트워크 환경은 일반적인 유선네크워크와 비교하여 접속 불량에 대한 많은 부담을 가지고 있으며 소형 디바이스는 작은 저장용량 및 처리 능력에 대한 문제점을 가지고 있다.

유비쿼터스 시스템을 사용하는 사용자의 목적은 언제 어디서나 정상적인 서비스를 원하는 시간만큼 지속적으로 제공받는 것이다. 사용자는 자신의 디바이스 상태 또는 서비스를 받기 위한 어떠한 메커니즘 보다 현재 정상적인 서비스를 받을 수 있는지를 중요히 여기며, 이러한 요구가 만족되지 못할 때 사용자의 시스템에 대한 신뢰되는 낮아진다. 이러한 상황에서 유비쿼터스 시스템이 가지는 문제점들로 인하여 잦은 결함이 발생하고 이것이 정상적인 서비스의 문제가 된다면 유비쿼터스 시스템에 대한 사용자 신뢰도는 떨어질 것이므로 이에 대한 처리가 필요하다.

시스템 결함 분석에 관한 많은 연구들이 이루어져 왔지만 유비쿼터스 시스템에 대한 결함 분석은 전통적인 시스템 결함 분석에 비하여 해당 연구진행이 미비하다. 기존의 Armor 미들웨어[2]는 서로 연결된 노드 사이의 중복된 자원을 관리하고, 사용자 어플리케이션(application)과 기본적인 컴포넌트(component) 사이의 에러를 탐지하고, 실패한 서비스 재시작과 같은 작업을 수행 하지만 어플리케이션 자원에 관한 세부적인 결함 분석을 하고 있지 않으며 MEAD[3]는 Crash 결함, 통신 결함, Malicious 결함, Omission 결함을 정의하고 이를 감지하도록 하고 있지만 결함을 분석하기 위한 컴퍼넌트 자체가 사용자 시스템에 삽입되어 있으므로 유비쿼터스 시스템에서의 사용은 전체 시스템에 오버헤드(overhead)를 가져올 수 있다.

따라서 본 논문에서는 유비쿼터스 시스템에서 발생할 수 있는 여러 문제점들을 해결하기 위한 결함 분석기를 개발하였다. 본 결함 분석기는 메모리 결함, 배터리 결함, 네트워크 결함 및 Heartbit 결함을 분석하며 또한 Java를 기반으로 개발되었으므로 Java에서 발생하는 exception 메시지를 이용하여 exception 메시지 발생 후 결함을 추적하여 exception 메시지와 결함에 대한 관계를 분석하여 추후 또다시 같은 exception 메시지가 발생할 경우 기존에 분석한 exception 메시지를 기반으로 결

본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅 및 네트워크 원천기반기술개발 사업의 지원에 의한 것임.

함을 예상할 수 있도록 하였다. 또한 클라이언트-서버 관계를 기반으로 결함분석기를 개발하여 서버에서 결함 분석 작업을 하도록 하여 유비쿼터스 시스템에서의 오버헤드를 줄이도록 개발하였다.

본 논문의 2장에서는 개발한 결함 분석기를 사용하기 위한 전체 유틸리티를 소개하였으며 3장에서는 유비쿼터스 시스템에서의 결함을 감지하기 위한 상태도를 소개하였다, 4장에서는 각 결함 감지를 위한 자세한 메커니즘을 소개하였으며 5장에서 결론을 맺는다.

2. 전체 시스템

그림 1은 본 논문에서 개발한 결함 분석기를 적용하기 위한 유틸리티(AHU : Autonomic Healing Uitilies) 구조로, 크게 AHUServer, AHUClient 그리고 AHUBackup으로 구성된다. AHUServer는 사용자의 시스템과 서비스 어플리케이션을 감시하고 결함이 발생할 경우 이를 감지한다. 본 논문에서 개발한 결함 분석기는 AHUServer에 삽입되어 있으며 결함을 감지하면 주기적으로 저장된 사용자 어플리케이션의 쓰레드(Thread) 정보를 AHUBackup으로 이동시켜 사용자가 계속적으로 서비스를 사용할 수 있도록 한다. AHUClient는 사용자 시스템에서 사용되며 사용자 시스템과 어플리케이션의 정보를 수집하여 주기적으로 AHUServer로 전송하는 역할을 한다. AHUBackup은 추가적인 서버로 사용자의 시스템이나 어플리케이션에 결함이 발생할 경우 사용자는 AHUBackup를 통해 계속적으로 서비스를 사용할 수 있다.

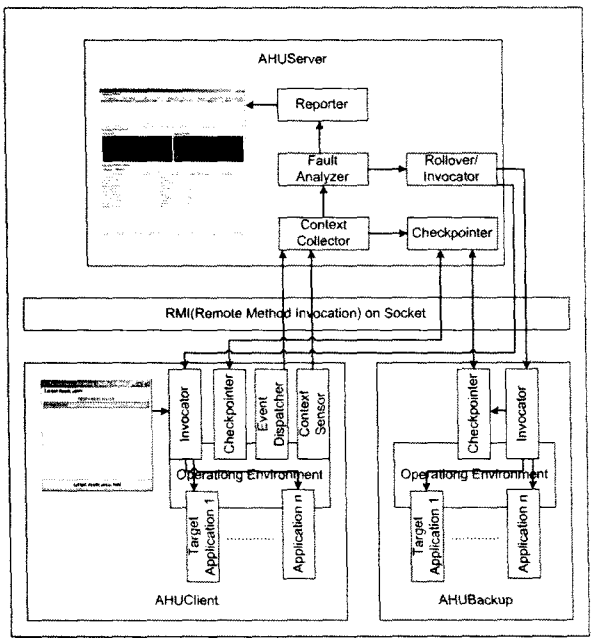


그림 1. AHU 구조도

3. 결함 분석기 상태도

유비쿼터스 시스템의 결함 감지를 위하여 주기적으로 사용자 시스템의 정보를 받아 다음과 같이 분석한다. 그림 2는 결함 감지를 위한 상태도이며 각 단계의 특징 및 상태전이는 다음과 같다.

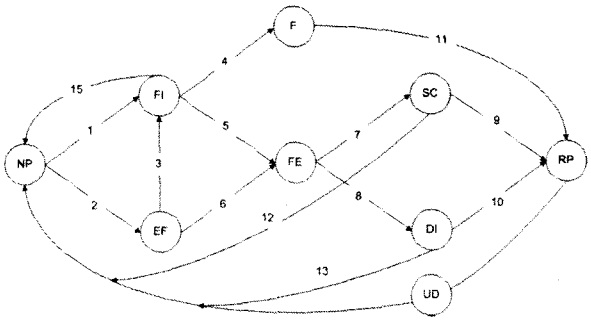


그림 2. 결함 감지를 위한 상태도

상태도

상 태	설 명
NP (Normal Processing)	사용자 정상 사용 상태
FI (Fault Inspection)	1단계 결함 분석 상태
EF (Exception Fault)	Exception 메시지 감시 상태
FE (Fault Expection)	1단계 검사 후 결함 예상 상태
SC (Server Comparison)	네트워크 및 Heartbit 결함의 2단계 결함 분석 상태
DI (Detail Inspection)	메모리 및 배터리 결함의 2단계 결함 분석 상태
RP (Recovery Processing)	2단계 검사 후 최종 결함으로 감지된 후 결함 복구 상태
UD (Update)	Exception 메시지로 인한 결함 발생시, 관련 결함 분석 방법 업데이트 상태
F (Fault)	결함 발생 상태

상태전이

상태전이	설 명
NP→FI	일정 시간이 경과 하였을 때
NP→EF	Exception 메시지를 감지한 경우
EF→FI	등록되어 있지 않는 Exception 메시지를 감지한 경우
FI→F	결함이 발생한 경우
FI→FE	1단계에서 예상 결함을 발견한 경우
EF→FE	이미 정의되어 있는 Exception 메시지를 감지한 경우
FE→SC	예상 결함이 네트워크 또는 Heartbit 결함일 경우
FE→DI	예상 결함이 메모리 또는 배터리 결함일 경우
SC→RP DI→RP	2단계 결함 분석 결과 결함으로 인식한 경우
F→RP	1단계에서 결함을 인식한 경우
SC→NP DI→NP UD→NP	결함 분석에서 정상으로 인식한 경우

4. 결함 감지 및 복구 방법

유비쿼터스 시스템에서 발생할 수 있는 결함을 메모리 결함, 배터리 결함, 네트워크 결함, Heartbit 결함, Java Exception 결함과 같이 다섯 가지로 정의하였으며 각각의 결함 감지 부분은 1단계와 2단계로 나누어져 있으며, 1단계에서는 결함을 감지하였을 경우 사용자 어플리케이션의 상태를 체크포인트[4]하여 결함 복구 시 사용자가 가장 최근의 자료를 사용할 수 있게 하였다.

4.1 메모리 결함 감지

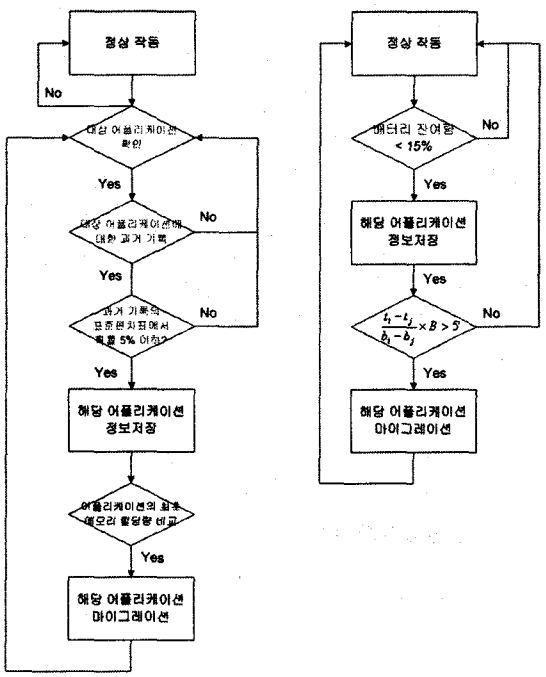
메모리 결함은 사용자 어플리케이션에서 가장 흔하게 일어날 수 있는 결함으로 최초 어플리케이션이 접속한 순간부터 계속적으로 사용자 어플리케이션의 메모리 정보를 분석하며 축적한다. 순서도 1(좌)는 메모리 결함분석 순서도를 나타낸다.

- 1단계 : 새로 측정된 사용자 어플리케이션의 메모리 사용량 패턴이 과거 축적된 메모리 사용량 패턴과 일정 부분 이상 틀릴 경우 이를 결함이 곧 발생할 결함 예상 상태로 인식한다. 과거 통계 자료를 기반으로 계산한 표준편차에 대하여 현재 측정된 자료가 상·하위 각각 5%에 해당될 때를 메모리 결함 예상 상태라 정의하며 계산식은 아래와 같다.

$$P(1.645 > Z, \ -1.654 > Z)$$

- 2단계 : 사용자 어플리케이션의 최초 할당된 메모리 사용량과 현재 사용자 어플리케이션의 메모리 사용량을 비교하여 현재의 메모리 사용량이 일정 비율 이상 높을 경우 이를 결함으로 인식한다.

- 복구방법 : 1단계 이후 저장된 정보를 이용하여 다른 시스템으로 현재 어플리케이션을 마이그레션(migration)한다[5].



순서도 1. (좌) 메모리 결함, (우) 배터리 결함

4.2 배터리 결함 감지

유비쿼터스 시스템의 이동성으로 인하여 거의 모든 시스템이 배터리로부터 전원을 공급받아 작업을 처리한다. 이런 환경에서 배터리 사용량에 대한 결함 분석 또한 사용자에게 서비스를 지속적으로 제공하기 위한 하나의 기준이 된다. 순서도 1(우) 배터리 결함분석에 관한 순서도를 나타낸다.

- 1단계 : 현재 사용자 시스템의 잔여 배터리 사용량이 약 15%가 되었을 경우 이것을 결함 예상 상태로 인식한다. 이것은 일반적으로 노트북에서 배터리 잔여량 경고에 사용되는 하한선을 사용하였다.

- 2단계 : 사용자 시스템이 1단계 결함 예상 상태 인식 후 사용자에게 경고를 했음에도 사용자 시스템이 AC 전원으로부터 전원을 공급받지 못할 경우 과거 30분 동안 배터리 사용량에 대한 패턴을 바탕으로 현재 남은 잔여 배터리 사용량으로 사용할 수 있는 시간이 5분(일반적인 reboot 시간) 미만일 경우 이를 결함으로 판단하며, 이를 위한 계산식은 다음과 같다.

$$\frac{T_0-T_1}{B_0-B_1}\times B_1 \le 5' \quad B = \text{배터리 잔여량}, T = B \text{ 측정시간}$$

복구방법 : 1단계 이후 저장된 정보를 이용하여 다른 시스템으로 현재 어플리케이션을 마이그레션한다[5].

4.3 네트워크 결함 감지

유비쿼터스 시스템은 무선네트워크 환경을 기반으로 하고 있으므로 이러한 상황에서 네트워크 상태는 사용자 서비스에 상당한 영향을 준다. 본 논문에서의 시스템은 사용자에게 서비스를 제공하는 입장이 아닌 사용자 시스템을 모니터링 하고 결함 발생 시 복구를 시행하는 것에 초점을 맞추고 있으므로 네트워크 결함 발생을 감지하였을 경우 가장 최근의 정보를 저장하여 추후 네트워크 결함으로 인하여 사용자가 서비스를 받지 못 하면 저장된 정보를 이용하여 가장 최근의 정보로부터 서비스를 다시 시작할 수 있도록 한다. 순서도 2(좌)는 네트워크 결함 분석에 관한 순서도를 나타낸다.

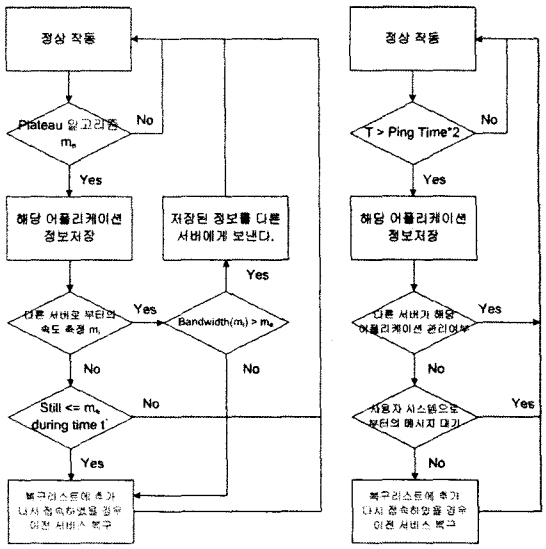
- 1단계 : Plateau 알고리즘[6]을 사용하여 현재의 네트워크 속도가 일정 시간동안 과거의 자료보다 낮은 분포를 나타날 때 이를 결함 예상 상태로 인식한다.

- 2단계 : 결함을 인식한 서버는 다른 서버에게 현 사용자 시스템의 네트워크 속도를 측정하여 전송하도록 지시한다. 이렇게 측정된 속도와 현 서버에서의 속도를 비교하여 다른 서버에서의 측정값이 현재 서버보다 높을 경우 이를 결함으로 인식한다. 본 연구에서 개발하는 시스템은 단 하나의 서버로만 운용되는 것이 아니라 많은 수의 유비쿼터스 시스템을 관리하기 위하여 지리적으로 분산되어 있는 다수 서버로 구성될 수 있다.

- 복구방법 : 다른 서버에서의 측정값이 낮을 경우 현 서버의 해당 어플리케이션을 복구리스트에 추가하여 네트워크가 단절된 이후 다시 서버로의 연결이 생성되었을 경우 저장된 정보를 이용하여 사용자가 계속적인 서비스를 할 수 있으며, 다른 서버에서 측정된 속도가 현재 서버의 측정값 보다 높을 경우 저장된 정보를 다른 서버에게 전달한다(클라이언트 프로그램은 속도가 가장 빠른 서버로 접속한다는 것을 전제).

4.4 Hearbit을 이용한 결함 감지

결함을 감지하기 위한 방법 중 하나로 사용자 시스템으로부터 일정 간격으로 전송되는 메시지를 이용하여 메시지가 일정 시간 이상 오지 않을 경우 결함으로 예상한다. 순서도 2(우)는 Heartbit 결함 분석에 관한 순서도를 나타낸다.



순서도 2. (좌) 네트워크 결함, (우) Heartbit 결함

- 1단계 : 메시지를 받아야 하는 시간이 현 네트워크 속도의 2배(현재 네트워크 상태에서 패킷이 왕복하기 위해 필요한 시간을 기준) 시간이상 지연되었을 경우 이를 결함 예상 상태로 인식한다.

$$T(\text{현재시간}) - T_0(\text{마지막 메시지 도착시간}) \times 2 > RTT(\text{현재 네트워크의 } RTT)$$

- 2단계 : 네트워크 결함 감지에서와 같은 방법으로 다른 서버들에게 현재 사용자 어플리케이션을 모니터링하고 있는지 확인한다. 이것은 사용자 어플리케이션이 어떤 이유로 인하여 접속을 종료하지 않고 다른 서버에 다시 접속할 수 있기 때문이다.

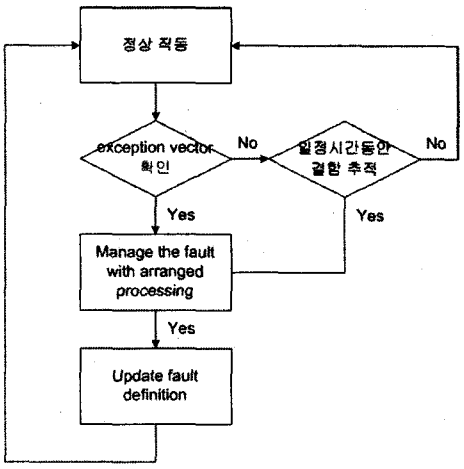
• 복구방법 : 다른 서버에서 대상 어플리케이션을 관리하고 있을 경우 서버는 대상 어플리케이션에 대한 모니터링을 종료하며, 그렇지 않을 경우 복구리스트에 해당 어플리케이션을 추가하여 이후 같은 사용자 시스템으로부터 접속이 생성되었을 경우 이전 어플리케이션에 대한 서비스를 계속적으로 할 수 있게 한다.

4.5 Java Exception 메시지를 이용한 결함 감지

본 연구에서 개발하는 시스템은 Java를 기반으로 하고 있으므로 거의 모든 부분에서 Java exception 메시지가 발생할 수 있다. 그러므로 이를 이용하여 결함을 분석하고자 한다. exception 메시지 발생 이후 일정 시간내에 메모리 결함, 네트워크 결함 또는 Heartbit 결함이 감지되었을 경우 이것을 하나의 결함이 일어나기 위한 예비 과정으로 판단하고 이후 같은 exception 메시지가 감지될 경우 이를 해당 결함의 결함 예상 상태로 판단하고 해당 결함에서 정의된 결함 복구 과정을 수행한다. 순서도 3은 Java Exception 메시지를 이용한 결함 분석에 관한 순서도를 나타낸다.

• 1단계 : 과거 정의된 exception 메시지 중 결함을 발생 시킨 exception 메시지가 있는지 비교하고 해당 메시지가 존재할 경우 해당 결함에 대한 복구 과정을 수행한다.

• 2단계 : 해당하는 exception 메시지가 존재하지 않을 경우 일정시간동안 대기하여 일정시간 내에 결함이 일어나는지 추적하며 결함이 발생할 경우, 해당 exception 메시지와 현재 발생한 결함을 하나의 벡터로 묶어 관리한다.



순서도 3. Exception 결함

5. 결 론

본 논문에서는 유비쿼터스 시스템이 가지는 특징 때문에 발생하는 결함과 이로 인하여 일어 날 수 있는 사용자 서비스의 중단을 방지하기 위하여 결함을 조기에 분석하여 발견함으로써 이에 대한 적절한 조치를 미리 취하여 사용자의 서비스를 계속적으로 제공할 수 있도록 하는 결함 분석기를 개발하였다. 또한 결함분석기는 서버에서 동작하며 사용자 시스템으로 부터 결함 분석에 필요한 정보만을 수집하여 사용자 시스템에서의 오버헤드를 최대한 줄일 수 있도록 개발하였다.

현재 개발된 AHU는 경량화된 유틸리티로 시스템에 오버헤드를 만들지 않으므로 유비쿼터스 환경에서 많은 수의 사용자가 동시 접속하여도 이를 처리하기 위하여 많은 자원을 필요로 하지 않으며 또한 결함이 발생하더라고 빠르게 사용자의 서비스를 다시 사용할 수 있게 한다. 유비쿼터스 환경이 점차 현대 생활 속에 필수적인 존재로 자리 잡아가는 현 시점에서 AHU는 사용자의 서비스를 보장하므로 많은 이익을 줄 수 있을 것이라 기대한다.

참고문헌

[1] M. Brugnoli, et al., "User Expectations for Simple Mobile Ubiquitous Computing Environments," Proceedings of the 2nd Workshop on Mobile Commerce and Services, pp. 2-10, July 2005.

[2] Z. Kalbarczyk, et al., "Application Fault Tolerance with Armor Middlewar," Proceedings of the Internet Computing, pp. 28-37, March/April 2005.

[3] P. Narasimhan, et al., "Middleware for Embedded Adaptive Dependability," Proceddings of the Workshop on Large Scale Real-Time and Embedded System, December 2002.

[4] Jon Howell, et al., "Straightforward Java Persistence through Checkpointing in Advances in Persistent Object Systems," Proceedings of the 3rd International Workshop on Persistence and Java, pp. 322-334, January 1999.

[5] D. Milojicic, et al., "Process Migration Survey," ACM Computing Surveys, Vol. 32 No. 8, pp. 241-299, September 2000.

[6] C. Logg, et al., "Experiences in Traceroute and Available Bandwidth Change Analysis," Proceedings of the Workshops on SIGCOMM, September 2004.