2014년 3월 전자공학회 논문지 제 51권 제 3 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 3, March 2014 http://dx.doi.org/10.5573/ieie.2014.51.3.089

논문 2014-51-3-10

항공관제용 현시시스템을 위한 패턴매칭 기반의 ASTERIX 파싱 모듈 설계 및 구현

(Design and Implementation of ASTERIX Parsing Module Based on Pattern Matching for Air Traffic Control Display System)

김 강 희*, 김 호 중*, 은 윤 동*, 최 상 방**

(Kanghee Kim, Hojoong Kim, Yin Run Dong, and SangBang Choi©)

요 약

최근 국내 항공교통량이 급증함에 따라 안전하고 효율적인 항공교통관리를 위한 항공관제 시스템의 필요성이 커지고 있다. 특히 원활한 항공교통관제를 위해 비행정보구역(FIR : Flight Information Region) 내의 모든 항공상황을 추가지연 없이 보여주어야 하는 현시시스템의 성능 보장이 무엇보다 중요하다. 본 논문에서는 표준 레이더 감시자료 포맷인 ASTERIX(All purpose STructured Eurocontrol suRveillance Information eXchange) 메시지의 파싱 과정에서 발생하는 오버헤드를 줄여 시스템 부하를 최소화함으로써 안정적인 관제운영을 도모할 수 있는 패턴매칭 기반의 ASTERIX 파싱 모듈을 설계하였다. 설계한 패턴매칭 기반의 ASTERIX 파싱 모듈은 수신 ASTERIX 데이터를 분석하여 패턴을 생성하며, 이후 수신되는 ASTERIX는 패턴을 통해 정의된 프로시저로 파싱한다. 기존 비트 수준 파싱 모듈의 불필요한 파싱 과정을 줄여 현시에 필요한 정보만을 빠르게 추출함으로써 현시 오류를 최소화하고 안정적인 항공관제를 가능하게 한다. 설계한 패턴매칭 기반의 ASTERIX 파싱 모듈의 성능을 비교하기 위하여 일반적인 비트 수준 ASTERIX 파싱 모듈과 비교한 결과, 짧은 처리지연시간, 높은 처리량, 낮은 CPU 사용률을 보이는 것을 확인하였다.

Abstract

Recently, as domestic air traffic dramatically increases, the need of ATC(air traffic control) systems has grown for safe and efficient ATM(air traffic management). Especially, for smooth ATC, it is far more important that performance of display system which should show all air traffic situation in FIR(Flight Information Region) without additional latency is guaranteed. In this paper, we design a ASTERIX(All purpose STructured Eurocontrol suRveillance Information eXchange) parsing module to promote stable ATC by minimizing system loads, which is connected with reducing overheads arisen when we parse ASTERIX message. Our ASTERIX parsing module based on pattern matching creates patterns by analyzing received ASTERIX data, and handles following received ASTERIX data using pre-defined procedure through patterns. This module minimizes display errors by rapidly extracting only necessary information for display different from existing parsing module containing unnecessary parsing procedure. Therefore, this designed module is to enable controllers to operate stable ATC. The comparison with existing general bit level ASTERIX parsing module shows that ASTERIX parsing module based on pattern matching has shorter processing delay, higher throughput, and lower CPU usage.

Keywords : ASTERIX, ATC, pattern matching, binary search tree

Ⅰ. 서 론

국내 항공교통량은 한류 열풍, 개별자유여행객의 증가, 저가 항공사의 국내외 노선 공급력 확대, 여가 문화 확산으로 인하여 매년 증가 추세를 기록하고 있다[1]. 국

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering, Inha University)
© Corresponding Author(E-mail: sangbang@inha.ac.kr)
※ 본 연구는 국토해양부 항공선진화사업의 연구비 지원(10항공-항행01)에 의해 수행되었습니다.
접수일자: 2014년1월13일, 수정완료일: 2014년3월3일

토해양부 자료에 따르면 2013년 3분기 항공교통량은 총 151,924대로 전년도 대비 8.55% 증가하였으며, 이 중 국내선 교통량은 12.85%로 크게 증가하였다. 특히 교통량이 집중되는 시간대인 오전 10~11시 사이 시간당 평균 131대의 항공기가 FIR(Flight Information Region : 비행정보구역) 내를 운항하였으며, 인천공항 관제탑에서 통제한 최대 항공기 대수는 평균 52대로 저녁 7시~8시 사이에 집계되었다[2].

이처럼 해마다 늘어나는 항공교통량으로 인해 관제권 내의 모든 항공상황을 보여주어야 하는 현시시스템의 과부하가 발생하며, 현시시스템의 구성요소 중에서 감시자료를 처리하는 모듈과 이를 화면에 현시하는 모듈의 부하가 가장 크다. 시스템 부하 증가로 인한 장시간의 처리 지연은 현시화면 상에서 항적의 정지, 사라짐 또는 중첩으로 나타나며, 이러한 현상이 빈번히 발생할 경우 관제업무에 혼란을 야기하게 된다. 따라서 오류 증상 없이 관제사에게 모든 항공상황을 보여줄 수 있는 현시시스템을 구현하는 것이 매우 중요하다.

시스템 간에 항공기의 항적 데이터를 교환하기 위한 표준 레이더 감시자료 포맷으로 ASTERIX(All purpose STructured Eurocontrol suRveillance Information eXchange)가 있다[3-4]. ASTERIX 데이터는 비트 단위의 인코딩 또는 무손실의 블록 단위의 스트링이 혼합된 구조를 가진다. 따라서 수신 ASTERIX 데이터로부터 현시에 필요한 정보를 추출하여 화면에 현시하기 위해서는 데이터 파싱 과정이 추가로 필요하다. 일반적으로 사용되는 ASTERIX 데이터 파싱 방법은 비트 수준 파싱 방법으로, MSB(Most Significant Bit)부터 차례로 비트 값을 검사하면서 데이터를 파싱하고 필요한 정보를 추출하는 방식을 말한다. 시간에 따라 항공기의 위치, 속도, 고도 등이 변하므로 이를 표현하는 ASTERIX 데이터도 다른 값을 가지게 되며, 이러한 ASTERIX 데이터의 가변성 때문에 비트 수준 파싱 방법은 일반적으로 가장 안전한 ASTERIX 데이터 파싱 방법으로 여겨진다. 그러나 항적 현시에 반드시 필요한 데이터(위치, 속도, 고도, 진행방향 등) 뿐만 아니라, 그렇지 않은 데이터(데이터 갱신 신뢰기간, 추정 정확도 등)도 포함하여 반복적으로 파싱하기 때문에 시스템 처리가 지연되는 문제가 있으며, 기능이 다양화되고 복잡도가 커지고 있는 항공관제용 현시시스템의 요구 사항에도 적합하지 않다는 단점이 있다.

본 논문에서는 ASTERIX 파싱에 소모되는 오버헤드를 최소화하여 시스템 처리 지연을 줄임으로써 급격히 증가하는 항공교통량을 효과적으로 수용할 수 있는 패턴매칭 기반의 ASTERIX 파싱 모듈을 설계 및 구현하였다. 설계한 ATERIX 파싱 모듈은 SDP(Surveillance Data Processor : 감시자료처리)[5] 서버로부터 수신한 ASTERIX 데이터를 학습하여 패턴을 추출하고 패턴 테이블을 생성한다. 그 후 수신된 ASTERIX 데이터가 생성한 패턴 정보와 일치하면 해당 ASTERIX를 미리 정의된 루틴으로 파싱함으로써, 기존의 비트 수준 파싱 모듈에서 발생하는 불필요한 중복을 제거하고 신속하게 항적 현시에 필요한 데이터를 추출할 수 있다. 또한 수신 확률이 높은 패턴이 먼저 검색될 수 있도록 우선순위 기반의 패턴 탐색 트리를 구성함으로써 패턴탐색에 걸리는 시간을 최소화하였다.

본 논문에서는 설계한 패턴매칭 기반의 ASTERIX 파싱 모듈과 비트 수준 파싱 모듈의 성능을 비교하기 위해 인천공항 서울접근관제소 내의 개발실에 설치된 현시시스템 2대에 각 모듈을 적용하고 SDP로부터 ASTERIX를 수신하면서 각 현시시스템의 처리지연시간, 처리량, CPU 및 메모리 사용률을 측정하였다. 그 결과 설계한 패턴매칭 기반의 ASTERIX 파싱 모듈은 비트 수준 파싱 모듈에 비해 짧은 처리지연시간과 높은 처리량, 낮은 CPU 사용률을 보이는 것을 확인하였다.

본 논문은 다음과 같이 구성된다. Ⅱ장에서는 ASTERIX 파싱 모듈 구현과 관련된 연구를 소개하고, Ⅲ장에서는 본 논문에서 제안한 패턴매칭 기반의 ASTERIX 파싱 모듈의 구조와 동작 과정을 설명한다. Ⅳ장에서는 실험을 통해 제안한 모듈의 성능을 평가하고, Ⅴ장에서는 결론으로 마무리한다.

Ⅱ. 관련연구

이 장에서는 본 논문에서 제안하는 패턴매칭 기반 ASTERIX 파싱 모듈의 성능을 평가하기 위해 항공관제용 현시시스템과 관련 ASTERIX에 대하여 설명한다. 그리고, ASTERIX 구조에 대하여 설명한 후, 비트 수준 ASTERIX 파싱 방법에 대하여 설명한다.

2.1 항공관제용 현시시스템과 ASTERIX

안전한 항공관제 업무를 위하여 현시시스템은 관제

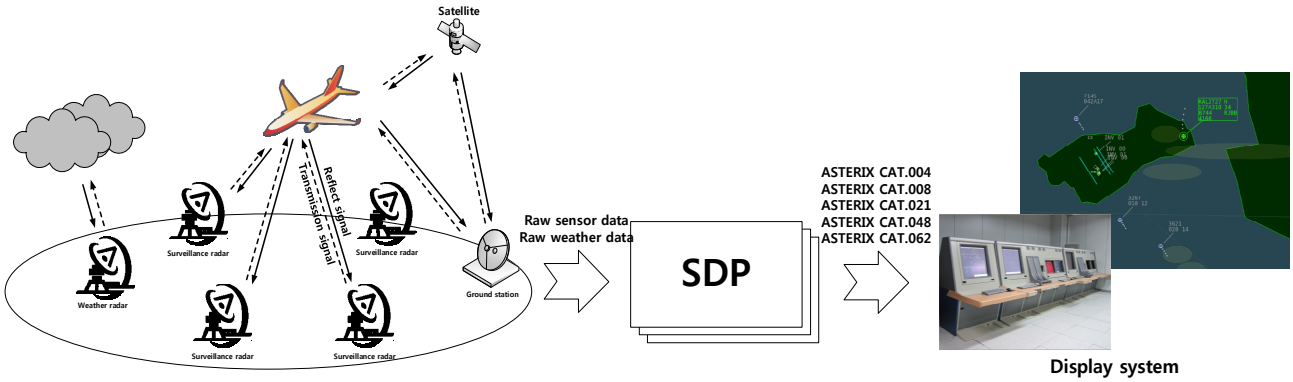


그림 1. 항공관제시스템의 감시자료 및 기상자료의 흐름
Fig. 1. Flow of surveillance data and weather data in ATC system.

구역에서 운항하는 모든 항공기의 정보와 기상 상황을 관제사에게 보여주어야 한다. 또한, 항공기 간 또는 항공기와 지형 간에 충돌 위험을 시청각적으로 알려주어야 한다.[6-7]항공관제시스템에서는 이와 같은 정보를 관제시스템 간에 교환하기 위하여 ASTERIX를 사용한다. ASTERIX는 EUROCONTROL에 의해 개발되고 표준화된 ATM(Air Traffic Management) 감시자료 이진 메시지 포맷이며, 시스템 간에 감시자료의 교환을 쉽게 하기 위해 설계되었다.

그림 1은 항공관제시스템의 감시 자료와 기상 자료의 흐름을 나타낸다. 감시 레이더는 항공기의 위치를 알아내기 위해 주기적으로 요청 신호를 전송하며 항공기에 의해 수신되는 응답 신호를 이용하여 위치, 속도, 고도, ID 정보 등을 포함한 Raw sensor data를 생성한다. 기상레이더도 감시 레이더와 동일하게 구름에 반사된 신호를 이용하여 강수 강도 등을 포함한 Raw weather data를 생성한다. 감시 레이더와 기상 레이더가 생성한 raw data는 네트워크를 통해 SDP로 전송된다. SDP는 다수의 Raw sensor data를 이용하여 항공기의 위치를 추정하고 이를 하나의 항적으로 만든다. 이후 항적은 ASTERIX CAT.062, 항적 생성과 연관된 Raw sensor data들은 ASTERIX CAT.048로 인코딩하여 현시시스템으로 전송한다. 생성한 항적 중에서 항공기 간의 충돌 또는 지형과 충돌 위험이 예상되면 정보를 ASTERIX CAT.004로 인코딩하여 현시시스템에 전송한다. 그리고 기상 자료는 ASTERIX CAT.008로 인코딩하여 현시시스템에 전송한다. SDP는 레이더를 이용한 감시시스템 뿐만 아니라 위성을 이용한 감시시스템인 ADS-B(Automatic Dependant Suveillance – Broadcast)에 의해 생성된 ASTERIX CAT.021도 현시시스템으로 전달한다.

항공교통량과 감시시스템의 수에 따라 현시시스템에 수신되는 ASTERIX의 데이터양도 비례하여 증가하기 때문에, 비트 수준 ASTERIX 파싱 모듈은 급증하는 ASTERIX 데이터를 처리하는 데 한계가 있다. 따라서 수신 ASTERIX 데이터를 효과적으로 처리할 수 있는 ASTERIX 파싱 모듈의 설계가 필요하다.

2.2 ASTERIX 구조

그림 2는 ASTERIX 구조를 설명하기 위한 샘플 데이터이다. 설명의 편이를 위해 바이너리 데이터인 샘플 데이터를 1 octet씩 나누고, 해당 데이터 값을 Hexadecimal로 표기하였다. 샘플 데이터의 길이는 78 octets이며, 시작점으로부터의 Offset값을 Data 하단에 기록하였다. 본 절에서는 이 샘플 데이터를 활용하여 ASTERIX 구조에 대하여 설명한다.

그림 3은 ASTERIX 개략적인 구조를 나타낸다. CAT(CATegory)는 카테고리 번호이며, 1 octet의 크기를 가진다. 그림 2의 샘플 데이터에서 CAT에 해당하는 값은 Offset 0에 위치한 3e(decimal : 62)라는 것을 알 수 있다. ASTERIX 표준에서 카테고리 번호를 004, 008, 048, 062와 같이 decimal로 표기하여 구분하고 있기 때문에, 가독성을 높이려면 수신 데이터의 CAT를 decimal로 변환해야 한다.

LEN(LENgth)은 데이터 포맷의 길이로 CAT, LEN, FSPEC, Items를 포함한 길이이며 2 octets의 크기를 가진다. 그림 2의 샘플 데이터에서 LEN은 Offset 1, 2에 위치하고 있으며, 값 004e(decimal : 78)은 수신 데이

Data (hex)	b5	df	30	00	81	01	01	10	71	bf	40	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	da	01
Offset (octet)	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

Data (hex)	3e	00	4e	9b	fd	2d	04	c8	c8	39	16	16	00	6a	e4	2e	01	67	44	f8	fe	2a	ff	f9	00	00	08	63	00	04	14
Offset (octet)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Data (hex)	01	01	10	00	e8	80	e6	01	78	c0	00	40	00	55	00	47
Offset (octet)	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77

그림 2. 샘플 데이터
Fig. 2. Sample data.

Octets : 1	2	variable	variable
CAT	LEN	FSPEC	Items

그림 3. ASTERIX의 개략적인 구조
Fig. 3. Rough structure of ASTERIX.

터의 길이와 같다. ASTERIX 인코딩 시 LEN 값은 반드시 데이터의 길이와 일치해야 하며, 이러한 규칙 때문에 수신 ASTERIX 데이터를 처리하는 파싱 모듈은 수신 데이터의 LEN 값을 통해 길이 오류를 쉽게 찾을 수 있다.

FSPEC(Field SPECification)은 어떤 데이터 필드들이 Items 안에 존재하는지를 정의하는 플래그 비트들로 구성되어 있다. FSPEC 파싱 시, FSPEC의 크기와 값, 그리고 FSPEC의 플래그 비트들과 데이터 필드의 관계를 알아내려면 추가적인 정보가 필요하며, 이에 ASTERIX 표준은 이러한 정보를 UAP(User Application Profile)로 정의하고 있다. UAP란 데이터를 데이터 필드에 할당하고 표준화된 메시지 인코딩과 디코딩을 통하여 필요한 모든 정보를 담아내기 위한 매커니즘을 말한다. 카테고리마다 UAP가 포함하고 있는 데이터 필드 정보가 다르므로 반드시 해당 카테고리의 UAP를 이용하여 수신 ASTERIX 데이터를 파싱해야 한다.

표 1은 샘플 데이터의 카테고리인 062의 UAP다. UAP는 FRN, Data Item, Information, Length로 구성되어 있다. FRN(Field Reference Number)는 데이터 필드의 숫자 인덱스 값이다. 인덱스 값은 1씩 증가하며 8배수에서는 숫자 인덱스가 아닌 FX(Field eXtenstion indicator)가 할당된다. FX는 FSPEC을 확장하기 위한 플래그로 1이면 뒤이어 FSPEC이 존재함을 가리키며, 0이면 더 이상 FSPEC이 존재하지 않음을 가리킨다. Data Item은 FRN와 같이 숫자 인덱스 값이지만 카테고리 번호와 고유 필드 번호를 결합하여 표기하기 때문에 FRN보다 가독성이 높다. Information은 해당 데이터 필드에 어떤 정보가 포함되어 있는지를 보여준다. Length는 해당 데이터 필드의 길이를 정의하며, “+”가 붙지 않은 것은 길이가 고정이며 해당 octet만큼의 크기를 가지고 있음을 가리키고, “1+”는 길이가 1 octet 이상으로 가변될 수 있음을 가리킨다. FRN 34, 35에 위치한 RE(Reserved Expansion Field)와 SP(Reserved For Special Purpose Indicator)는 UAP에 정의되지 않은 데이터 필드를 추가할 수 있도록 예약된 필드로 거의 사용되지 않기 때문에 본 논문에서는 다루지 않는다.

그림 4는 표 1의 UAP를 통해 파싱한 샘플 데이터의 FSPEC 구조를 나타낸다. 샘플 데이터의 Offset 3에서부터 1 octet씩 순차적으로 UAP에 대입함으로써 그림 4와 같은 FSPEC을 얻을 수 있다. 샘플 데이터의 FSPEC은 4 octets의 크기를 가지며 그 이상 증가하지 않는다. Offset 6에서 LSB(Least Significant Bit)의 위치는 UAP의 FRN 28 다음의 FX이며, 값이 0이라는 것은 추가 FSPEC가 없음을 가리키기 때문이다. FSPEC를 통해 샘플 데이터는 총 14개의 데이터 필드를 포함하고 있으며, 그 중 고정 길이 데이터 필드가 11개, 가변 길이 데이터 필드가 3개(I062/380, I062/080, I062/0500)가 있음을 알 수 있다. 이처럼 FSPEC은 ASTERIX에 포함된 데이터 필드들을 정의하는 중요한

표 1. UAP(CAT = 062)
Table 1. UAP(CAT = 062).

FRN	Data Item	Information	Length
1	I062/010	Data Source Identifier	2
2	-	Spare	-
3	I062/015	Service Identification	1
4	I062/070	Time Of Track Information	3
5	I062/105	Calculated Track Position (WGS-84)	8
6	I062/100	Calculated Track Position (Cartesian)	6
7	I062/185	Calculated Track Velocity (Cartesian)	4
FX	-	Field extension indicator	-
8	I062/210	Calculated Acceleration (Cartesian)	2
9	I062/060	Track Mode 3/A Code	2
10	I062/245	Target Identification	7
11	I062/380	Aircraft Derived Data	1+
12	I062/040	Track Number	2
13	I062/080	Track Status	1+
14	I062/290	System Track Update Ages	1+
FX	-	Field extension indicator	-
15	I062/200	Mode of Movement	1
16	I062/295	Track Data Ages	1+
17	I062/136	Measured Flight Level	2
18	I062/130	Calculated Track Geometric Altitude	2
19	I062/135	Calculated Track Barometric Altitude	2
20	I062/220	Calculated Rate Of Climb/Descent	2
21	I062/390	Flight Plan Related Data	1+
FX	-	Field extension indicator	-
22	I062/270	Target Size & Orientation	1+
23	I062/300	Vehicle Fleet Identification	1
24	I062/110	Mode 5 Data reports & Extended Mode 1 Code	1+
25	I062/120	Track Mode 2 Code	2
26	I062/510	Composed Track Number	3+
27	I062/500	Estimated Accuracies	1+
28	I062/340	Measured Information	1+
FX	-	Field extension indicator	-
29	-	Spare	-
30	-	Spare	-
31	-	Spare	-
32	-	Spare	-
33	-	Spare	-
34	RE	Reserved Expension Field	1+
35	SP	Reserved For Special Purpose Indicator	1+
FX	-	Field extension indicator	-

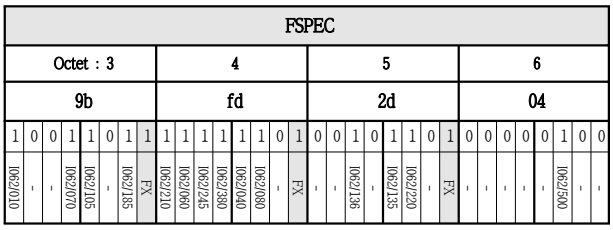


그림 4. 샘플 데이터의 FSPEC 구조
Fig. 4. Structure of FSPEC of sample data

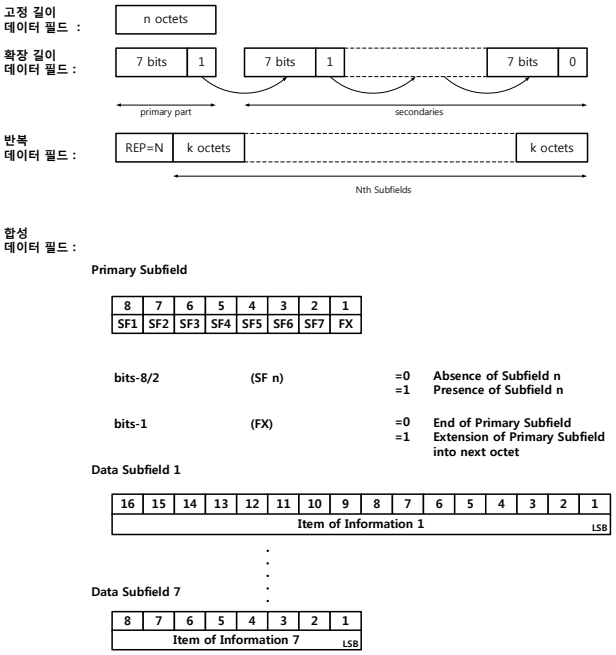


그림 5. 데이터 필드의 구조
Fig. 5. Structure of data field

역할을 한다.

그림 5는 Items를 구성하는 고정 길이 데이터 필드, 확장 길이 데이터 필드, 반복 데이터 필드, 합성 데이터 필드의 구조를 나타낸다. 고정 길이 데이터 필드는 Data Source Identifier(2 octets), Time Of Track Information(3 octets), Calculated Track Position(8 octets), Calculated Track Velocity(4 octets), Track Mode 3/A Code(2 octets) 등과 같이 길이가 변하지 않는 데이터를 기록할 수 있는 필드이다. 확장 길이 데이터 필드는 LSB 값에 따라 데이터 필드가 추가되는 데이터 필드로 LSB가 1이면 뒤이어 같은 길이의 데이터 필드가 추가되며 LSB가 0이면 더 이상 데이터 필드가 증가하지 않는다. 반복 데이터 필드는 지정된 숫자만큼의 Subfield가 존재하는 데이터 필드로 REP(field REPetition indicator)값이 N이라고 할 때, 길이 k

Field	CAT	LEN		FSPEC				I062/010		I062/070			I062/105								I062/185				I062/210		I062/060		I062/245		
Data (hex)	3e	00	4e	9b	fd	2d	04	c8	c8	39	16	16	00	6a	e4	2e	01	67	44	f8	fe	2a	ff	f9	00	00	08	63	00	04	14
Offset (octet)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Field	I062/245				I062/380																						I062/040		I062/080		
Data (hex)	b5	df	30	00	81	01	01	10	71	bf	40	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	da	01
Offset (octet)	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

Field	I062/080			I062/136		I062/135		I062/220		I062/500							
Data (hex)	01	01	10	00	e8	80	e6	01	78	c0	00	40	00	55	00	47	
Offset (octet)	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	

그림 6. 샘플 데이터 파싱 결과
Fig. 6. Parsing result of sample data.

octets을 가지는 서브필드들이 N번 반복하여 존재한다. 합성 데이터 필드는 FSPEC과 데이터 필드들의 관계와 유사하게 Primary Subfield와 Data Subfield들로 구성되며, Primary Subfield의 플래그 비트들의 값에 따라 뒤에 추가되는 Data Subfield의 존재 유무가 결정된다.

이와 같이 ASTERIX 데이터는 FSPEC 값에 따라 길이가 가변적이며, Item에서는 확장 데이터 필드, 반복 데이터 필드, 합성 데이터 필드에 따라 그 길이가 유동적이다. 따라서 수신한 ASTERIX 데이터로부터 현시시스템 화면에 하나의 항적을 정확하게 현시하기 위해서는 데이터 포맷을 변화시키는 요소들의 존재 유무를 파악하여 순차적으로 데이터를 파싱해야 한다. 그림 6은 그림 2의 샘플 데이터를 파싱한 결과를 나타낸다.

2.3 비트 수준 ASTERIX 파싱 방법

ASTERIX로부터 정보를 추출하기 위한 기본적인 방향을 제시하는 OpenSource로는 AsterixInspector[8], vog의 asterix[9]가 있다. AsterixInspector는 Qt framework를 사용하며, ASTERIX 데이터가 기록된 바이너리 데이터 파일을 분석하여 표와 HTML로 추출한 데이터를 자세하게 볼 수 있는 기능을 제공한다. C/C++ API를 사용하기 때문에 ASTERIX 파싱 모듈을 설계하는 개발자들에게 높이 평가된다. 특히 ASTERIX 데이터 포맷 정보를 모두 포함하고 있는 xml 파일을 사용하기 때문에 ASTERIX 데이터 포맷이 변경되거나 추가될 경우 xml을 수정하여 프로그램에 적용될 수 있다. 따라서 ASTERIX 를 분석하기를 원하는 사용자에 대한 편의성이 높은 프로그램이라고 할 수 있다.

vog의 asterix는 Volker Grabsch의 개인 Git 레포지토리에서 공개한 ASTERIX 파싱 프로그램으로 python을 사용한다. AsterixInspector와 유사한 포맷의 xml을 사용하며, 단순히 데이터 필드 이름과 스트링 값만을 출력할 수 있는 프로그램으로 정밀한 데이터 처리 모듈을 제공하고 있지 않다.

이 두 프로그램에서 사용하는 파싱 방법은 MSB부터 차례로 데이터를 파싱하는 비트 수준 파싱 방식이며, 항공관제 현시시스템의 ASTERIX 파싱 모듈에 적용 가능하다. 그러나 xml 파일에 정의된 모든 CAT의 데이터 필드 정보를 한 번에 메모리에 적재한 후 파싱에 이용하므로 메모리 사용률이 높으며, ASTERIX에 포함된 모든 데이터 필드의 값을 추출하기 때문에 불필요한 파싱 과정이 반복되어 처리 시간이 지연된다는 단점을 가지고 있다.

Ⅲ. 패턴매칭 기반 ASTERIX 파싱 모듈

3.1. ASTERIX 데이터 특징 정의

SDP는 현시에 필요한 데이터들을 중심으로 ASTERIX 데이터를 인코딩하여, 네트워크를 통해 현시 시스템으로 전송한다. 따라서 현시시스템에 수신되는 ASTERIX 데이터 필드들의 offset은 일정한 값을 가지게 되며 반복되는 비트값들을 추출하여 패턴으로 사용할 수 있다. 본 논문에서는 패턴을 형성하는 특징을 ASTERIX 포맷의 CAT, LEN, FSPEC, FX(Field eXtension indicator), REP로 정의한다. CAT은 ASTERIX 카테고리 번호로, ASTERIX 구조 및 각 데이터 필드가 의미하는 값을 변경한다. LEN은 가변적인 ASTERIX 데이터의 전체 길이를 정의할 수 있다. FSPEC은 ASTERIX 포맷 내부의 데이터 필드 구조를 정의하며 전체 데이터 길이를 가변시킨다. FX는 확장 길이 데이터 필드 또는 합성 데이터 필드에서 데이터 필드의 확장 시 사용되는 값으로 Subfield의 구조와 전체 데이터 길이를 가변시킨다. 그리고 REP는 반복 데이터 필드에서 Subfield의 반복 횟수를 나타내므로 전체 데이터 길이를 가변시킨다.

3.2. ASTERIX 패턴 생성

3.1에서 정의한 특징을 기반으로 패턴을 생성하기 위해 ASTERIX 데이터를 수집한다. 이때 데이터는 주·야간, 일일 정기편 등 일 단위로 반복되는 항공교통상황을 반영하기 위해 24시간동안 수집한다. ASTERIX 데이터를 수신하면 비트 수준 데이터 파싱을 수행하며, 그 결과를 패턴 학습 모듈로 전송한다. 패턴 학습 모듈은 파싱된 데이터로부터 특징들을 추출하여 패턴을 생성한다.

Octets : 1	2	variable	variable	54
CAT	LEN	FSPEC	Mask	Offset

그림 7. 생성 패턴 구조 (CAT = 062)
Fig. 7. Structure of generated pattern (CAT = 062).

	SF1	SF2	SF3	SF4	SF5	SF6	SF7	FX	SF8	SF9	SF10	SF11	SF12	SF13	SF14	FX	Data Subfield 1	
Data :	1	1	0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	1	· · ·
Mask :	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	· · ·

							REP		first Subfield		
Data :	0	0	0	0	0	0	1	1	1	1	· · ·
Mask :	0	0	0	0	0	0	1	1	0	0	· · ·

그림 8. Mask의 구조
Fig. 8. Structure of Mask.

그림 7은 CAT 값이 062일 경우 생성된 패턴 구조를 나타낸다. CAT, LEN, FSPEC은 수신 데이터의 CAT, LEN, FSPEC과 동일하며, 패턴 생성 시 수신 데이터로부터 각 필드를 그대로 복사하여 사용한다.

그림 8은 Mask의 구조를 나타낸다. 길이는 수신 ASTERIX에서 CAT, LEN, FSPEC을 제외한 나머지인 Items의 길이와 같다. 음영 부분은 Mask가 기록되는 부분이며, 나머지 부분은 기록되지 않는 부분이다. Mask 기록 시 FX가 1일 때는 해당 위치의 비트를 1로 기록하고, REP가 존재할 때는 해당 위치의 REP 값을 그대로 기록하며, 나머지 부분은 zero padding 한다. FX와 REP을 기록하는 이유는 FX와 REP를 제외한 나머지 데이터 필드들과 Subfield들은 수신 데이터마다 다른 값을 가질 수 있기 때문이다. Mask는 수신 데이터와 AND Operation을 수행하여 결과값이 Mask와 동일한지를 검증함으로써 잘못된 데이터 수신으로 인한 ASTERIX 파싱 오류로부터 시스템을 보호하기 위해 사용한다.

Offset은 데이터 필드가 존재하는 경우 해당 데이터 필드의 offset값을 기록하는 부분으로, 패턴기반 ASTERIX 파싱 시 Offset 값을 이용하여 미리 지정된 데이터 필드 위치를 찾아가 빠르게 데이터를 추출하기 위해 사용한다. CAT가 가진 총 FRN 개수만큼 2 octets 크기의 블록이 추가된다. CAT가 062일 때 FRN은 총 27개이므로 Offset의 길이는 그림 7과 같이 54 octets가 되며, CAT가 021일 때 FRN은 총 26개이므로 Offset의 길이는 52 octets가 된다.

그림 9는 CAT가 062일 때, Offset의 구조를 나타낸다. 예를 들어 Offset of I062/010의 값이 0x0007이면 데이터의 7번째 octet부터 Data Source Identifier 정보가 있음을 나타내며, Offset of I062/185가 0x001c이면 데이터의 28번째 octet부터 Calculated Track Velocity 정보가 존재함을 나타낸다.

패턴을 생성한 후에는 패턴이 *PatternList*에 존재하는지 검사한다. 만약 *PatternList*에 패턴이 존재하면 패

Octets : 2	2	2	2	2	· · ·	2
Offset of I062/010	Offset of I062/015	Offset of I062/040	Offset of I062/060	Offset of I062/070	· · ·	Offset of I062/510

그림 9. Offset의 구조 (CAT = 062)
Fig. 9. Structure of Offset (CAT = 062).

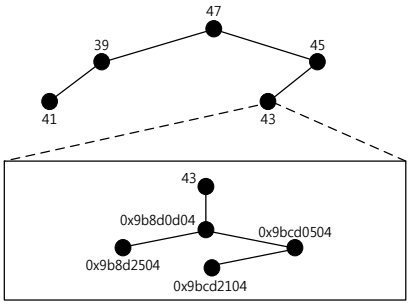
턴 발생빈도수를 1 증가시키며, 그렇지 않으면 생성 패턴을 *PatternList*에 저장한다.

3.3. 패턴매칭 기반 ASTERIX 파싱

수신한 ASTERIX의 데이터 패턴을 찾기 위해 패턴 검색 트리를 생성한다. 구현한 ASTERIX 파싱 모듈의 패턴 검색 트리는 구현이 간단하고 많은 응용분야에서 활용되는 이진 검색 트리를 기반으로 한다[10~11]. 그림 10은 패턴 테이블에 기록된 LEN 값이 39, 41, 43, 45, 47이고, 그 중 LEN값 43을 가지는 패턴의 FSPEC이 0x9b8d0d04, 0x9bcd0504, 0x9b8d2504, 0x9bcd2104라고 가정할 때, ASTERIX 패턴 검색 트리 구조와 패턴 테이블을 나타낸다.

ASTERIX 패턴 검색 트리에서 길이 검색 트리는 생성 패턴의 LEN 값으로 구성되어 있으며 각 노드는 LEN 값을 가지는 패턴의 FSPEC 값들로 구성된 또 하나의 서브트리인 FSPEC 검색 트리를 가진다. LEN table과 FSPEC table은 각각 LEN과 FSPEC를 발생 빈도수별로 내림차순 정렬한 것이다. LEN table은 길이 검색 트리를 구성할 때 사용하며, FSPEC table은 FSPEC 검색 트리를 구성할 때 사용한다.

LEN table에서 빈도수가 68.55%로 가장 높은 LEN 값 47이 길이 검색 트리의 최상위 *root*에 위치하게 되며, 하위 자식 트리의 *root*는 최상위 root값을 제외한 나머지 값들 중 빈도수가 각각 19.99%, 4.44%로 높은 39, 45가 위치하게 된다. 그리고 이진 검색 트리의 균형을 맞추기 위해 자식 트리를 형성하는 LEN들의 집합을 평균값으로 양분한다. 최상위 root값을 제외한 나머지 LEN 집합 {39, 41, 43, 45}의 평균값은 42이므로 42보다



LEN	빈도수
47	68.55%
45	19.99%
39	4.44%
41	3.74%
43	3.28%

<LEN table>

LEN	FSPEC	빈도수
43	0x9b8d0d04	71.53%
	0x9bcd0504	27.35%
	0x9b8d2504	0.95%
	0x9bcd2104	0.17%

<FSPEC table of LEN 43>

그림 10. ASTERIX 패턴 검색 트리 구조와 패턴 테이블
Fig. 10. Structure of ASTERIX pattern search tree and pattern table.

작거나 같은 {39, 41}은 좌측 자식 트리를 형성하며 42보다 큰 {43, 45}는 우측 자식 트리를 형성한다. 길이 검색 트리의 node값이 결정되면 바로 서브트리인 FSPEC 검색 트리를 형성하게 된다. node값이 길이 43인 FSPEC 검색 트리도 길이 검색 트리와 동일하게 가장 빈도수가 71.53%로 가장 높은 0x9b8d0d04가 트리의 최상위 root에 위치한다. 이후 나머지 {0x9b8d2504, 0x9bcd0504, 0x9bcd2104}의 평균인 9bb7c3af로 양분함으로써 {0x9b8d2504}는 좌측 자식 트리를 형성하며 {0x9bcd0504, 0x9bcd2104}는 우측 자식 트리를 형성한다. 만약 수신 ASTERIX의 LEN 값이 43과 일치하면 그림 10 LEN 43의 FSPEC 검색 트리를 통해 수신 ASTREIX의 FSPEC 일치 여부를 찾게 된다.

그림 11은 그림 10과 같은 우선순위 기반의 이진 검색 트리를 생성하기 위하여 사용하는 BuildSearchTree 프로시저의 의사코드이다. 최초 프로시저 수행 시 리스트가 존재하지 않으면 트리를 생성없이 프로시저를 종료한다. 만약 리스트가 존재하면 2번을 통해 발생빈도가 가장 많은 패턴의 LEN 또는 FSPEC 값을 *root*로 설정한다. 3의 조건을 통해 *root*값이 LEN 값이면 같은 LEN 값을 가지는 패턴들의 *fspecList*를 4번을 통해 추출하고 이 리스트를 5번을 통해 BuildSearchTree 프로시저에 입력값으로 사용하여 FSPEC 검색을 위한 자식 트리를 생성한다. 6의 조건을 통해 *root*값이 FSPEC이면 7, 8을 통해 *PatternList* 안에서 동일한 LEN과

```
    BuildSearchTree(List)
1   if List is empty then return
2   Set root = the maximum frequent value in the List
3   if root is LEN then
4     Let fspecList contains all fspecs of root
5     fspecTree(root) = BuildSearchTree(fspecList)
6   elseif root is FSPEC then
7     Set mask = getMask(root)
8     Set caseNum = getCaseNum(root)
9   endif
10  Set m = the average value of List except root
11  Let leftList and rightList contains all elements in the
    left and right of m except root
12  leftChild(root) = BuildSearchTree(leftList)
13  rightChild(root) = BuildSearchTree(rightList)
14  end BuildSearchTree
```

그림 11. BuildSearchTree 프로시저의 의사코드
Fig. 11. Psuedo code for BuildSearchTree procedure.

FSPEC을 가지는 패턴의 *mask*와 인덱스 번호를 찾아 *mask*와 *caseNum*에 각각 저장한다. *root*값으로 사용되지 않은 나머지 패턴들로 구성된 자식 트리를 생성하기 위해 10번에서 리스트 값들의 평균 값인 *m*을 설정하고 11을 통해 *leftList*와 *rightList* 둘로 나눈다. 나눈 후에는 12, 13을 통해 각각의 리스트를 BuildSearchTree 프로시저의 입력값으로 사용하여 자식 트리를 생성한다.

BuildSearchTree 프로시저로 생성한 ASTERIX 패턴 검색 트리를 이용하여 수신 ASTERIX 데이터의 패턴의 일치 여부를 확인하기 위해 그림 12의 SearchPattern 프로시저를 사용한다. 자식 트리가 존재하지 않으면 1을 통해 (-1)을 반환한다. Value 값이 LEN 값일 때 *root*값과 일치하면 뒤이어 FSPEC 검색 트리를 통해 FSPEC을 검색한다. 만약 value값과 *root*

```
    SearchPattern(tree,value,data)
1   if tree = NIL then, return -1

    // search LEN
2   if value is LEN then
3    if value = root then
4     return SearchPattern(fspecTree,data.FSPEC,data)
5    endif
6    if value < m then
7     return SearchPattern(leftChild(tree),value,data)
8    else then
9     return SearchPattern(rightChild(tree),value,data)
10   endif
11  endif

    // search FSPEC
12  if value is FSPEC then
13   if value = root then
14    if (mask & data) = mask then
15     return caseNum
16    else then
17     return -1
18    endif
19   if value < m then
20    return SearchPattern(leftChild(tree),value,data)
21   else then
22    return SearchPattern(rightChild(tree),value,data)
23   endif
24  endif

25  end SearchPattern
```

그림 12. SearchPattern 프로시저의 의사코드
Fig. 12. Psuedo code for SearchPattern procedure.

값이 일치하지 않으면 *m*과 비교를 통해 좌측 자식 트리 또는 우측 자식 트리로 추가 검색한다. Value값이 FSPEC일 때 value값이 *root*값과 일치하면 추가로 14를 통해 *mask*와의 AND Operation하여 데이터 패턴의 매칭 여부를 최종 확인 후 *caseNum*을 반환한다. 14의 조건을 만족하지 않을 경우 17을 통해 (-1)을 반환함으로써 패턴을 이용한 파싱을 사용하지 않도록 한다. 그 이유는 CAT, LEN, FSPEC이 모두 일치하였음에도 mask가 다른 것은 수신 ASTERIX가 학습 패턴과는 전혀 다른 데이터 필드의 구성을 이루고 있거나 포맷에 맞지 않는 잘못된 데이터이기 때문이다. 이러한 데이터를 그대로 활용할 경우 비정상적인 항적의 현시로 인해 관제 업무에 혼란을 줄 수 있다. Value 값이 *root*값과 일치하지 않을 경우 LEN 검색과 동일하게 *m*과 value의 비교를 통해 좌우측 자식 트리를 추가 검색한다.

BuildSearchTree 프로시저로 생성한 패턴 검색 트리를 이용하여 ASTERIX 데이터를 파싱하는 과정은 그

```
    // Bulid ASTERIX Pattern Search Tree
1   BuildSearchTree(PatternList)

    // ASTERIX Parsing Procedure
2   while do
3    Set IncomingData = IP Datagram Payload
4    Set ExistData = false

    // Pattern Matching Inspection
5    if CatList.contain(IncomingData[0]) then
6     i = SearchPattern(IncomingData)
7      if (i >= 0) then
8       AsterixInfo = PatternList[i].ParsingProcess()
9       ExistData = true
10      break
11     endif
12   else then
13    return NULL
14   endif

    // Exception Procedure
15   if ExistData == false then
16    AsterixInfo = DefaultParsingProcess()
17   endif
18   return AsterixInfo
19  endwhile
```

그림 13. 패턴매칭 기반 ASTERIX 파싱 과정의 의사코드
Fig. 13. Pseudo code for ASTERIX parsing procedure based on pattern matching.

림 13과 같다. *IncomingData*는 네트워크를 통해 수신한 ASTERIX 데이터를 의미하며, *ExistData*는 *IncomingData*와 매칭되는 패턴이 존재하는지를 가리키는 플래그이다. *IncomingData*의 패턴을 검사하기 전 5를 통해 *IncomingData*의 CAT가 현시시스템에서 처리할 수 있는 CAT들을 포함한 CatList에 존재하는지 검사한다. 처리할 수 없는 CAT인 경우 13을 통해 사용할 수 없는 데이터임을 알리고 데이터를 폐기한다. 처리할 수 있는 CAT이면 패턴 테이블 *PatternList*에 저장되어 있는 패턴과 수신 데이터의 패턴이 일치하는지를 6을 통해 확인한다. LEN과 FSPEC의 경우 값이 정확히 일치하는지를 확인하며, Mask의 경우 *IncomingData*와 AND Operation을 수행한 후 그 값이 Mask와 동일한지 확인한다. 패턴이 매칭되면 해당 패턴의 인덱스 값 *i*를 반환하며 패턴 리스트 *i*번째 패턴에 정의된 Offset에 따라 데이터의 파싱을 수행한다. 패턴 테이블과 일치하는 패턴을 찾지 못하는 경우 *ExistData* 값은 false이므로 16을 통해 비트 수준 파싱 과정을 거쳐 데이터 파싱을 수행한다.

Ⅳ. 성능 측정 및 분석

이 장에서는 제안한 패턴매칭 기반의 ASTERIX 파싱 모듈을 적용한 현시시스템의 성능을 측정하기 위한 실험환경을 정의하고, 수신 ASTERIX 데이터양이 증가함에 따른 처리지연시간, Throughput 및 CPU/메모리 사용률을 측정하고, 이에 대한 분석 결과를 기술한다.

4.1 실험 환경

항공관제 시스템 개발 테스트 베드가 설치된 AICC (Airport Integrated Communication Center)에서 패턴 기반 ASTERIX 파싱 모듈이 적용된 현시시스템의 성능을 측정 및 평가하였다.

그림 14는 실험환경 구성도이다. 신불 레이더, 왕산 레이더, 김포 레이더, 동광 레이더가 생성한 항공기 plot은 네트워크를 통해 인천공항에 위치한 ACC로 전달된다. ACC는 FIR 내에서 운항하는 모든 항공기의 항행안전을 위해 관제업무를 수행하는 곳이다. ACC에서는 국내에 설치된 모든 레이더 감시자료를 수신하고 있으나 보안상의 문제로 개발 환경에서는 현재 4개의 레이더만을 제공하고 있다. ACC에서 수신한 레이더 감시자

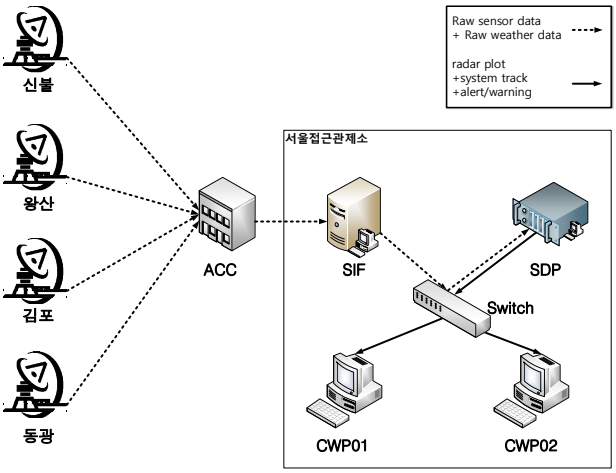


그림 14. 실험환경 구성도
Fig. 14. Diagram of test environment.

표 2. 개발 현시시스템의 제원
Table 2. Specification of developing display system.

모델	HP Z800 Workstation
CPU	Intel Xeon processor E5640 (Qurd-Core)
Memory	8GB DDR3 (4GB X 2)
HDD	250GB (RAID 10)
VGA	NVIDIA Quadro 2000
OS	RHEL 5
Software framework	Qt 4.7.1

료는 서울접근관제소 내 개발실에 위치한 SIF (Surveillance data InterFace) 시스템으로 Raw sensor data와 Raw weather data를 전송한다. SIF는 CAT와 LEN 검증을 통과한 raw data를 내부망을 통해 SDP로 전달한다. SDP는 raw data를 이용하여 항적/경고·경보/기상 정보를 포함한 ASTERIX 데이터를 생성하여 스위치로 연결된 내부망을 통해 현시시스템(CWP : Controller Working Position)으로 전달한다. 현시시스템은 SDP로부터 수신한 모든 ASTERIX를 파싱하여 필요한 정보를 추출한 후 화면에 현시한다. 실험을 위해 CWP01에는 비트 수준 파싱 모듈이 적용된 현시시스템을 CWP02에는 패턴매칭 기반의 파싱 모듈이 적용된 현시시스템을 설치하였다. 개발 현시시스템이 갖는 제원은 표 2와 같다.

4.2 처리지연시간 및 Throughput 비교

그림 15는 하나의 수신 ASTERIX 데이터을 파싱하

는데 걸리는 평균 처리지연시간을 비교한 것이다. 현시되는 항공기 대수를 150대씩 증가시키면서 처리지연시간을 측정한 결과, 패턴매칭 기반의 ASTERIX 파싱 모듈의 평균 처리지연시간은 비트 수준 ASTERIX 파싱 모듈의 평균 처리지연시간에 비해 1.7~2us 단축되었다. 비트 수준 ASTERIX 파싱 모듈은 데이터의 시작점으로부터 데이터 필드 구조의 변화를 일으키는 요인들을 순차적으로 찾으면서 데이터 파싱을 하므로 이에 따른 오버헤드가 발생한다. 반면, 패턴매칭 기반의 ASTERIX 파싱 모듈은 이진 검색 트리를 이용하여 패턴을 찾는 과정에서 오버헤드가 발생하지만 현시에 필요한 데이터 필드만을 신속하게 추출함으로써 비트 수준 ASTERIX 파싱 모듈보다 평균 처리지연시간이 단축되는 것을 알 수 있다. 항공기 대수가 증가함에 따라

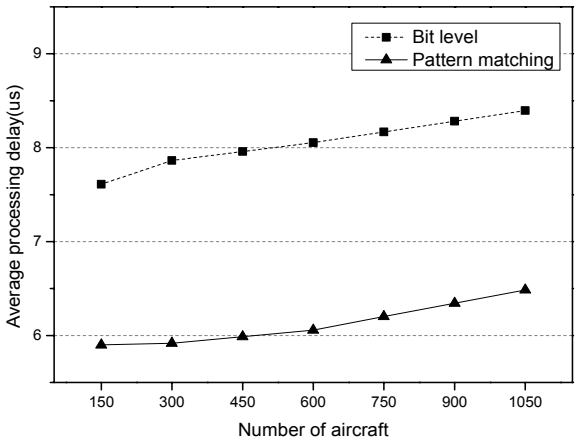


그림 15. 평균 처리지연시간 비교
Fig. 15. The comparison with average processing delay.

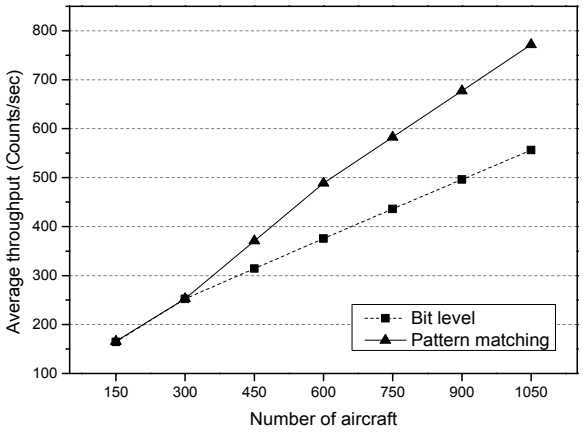


그림 16. 평균 처리량 비교
Fig. 16. The comparison with average throughput.

평균 처리지연시간이 증가하는 것은 운항 항공기의 대수만큼 생성되는 ASTERIX 데이터가 증가하여, 파싱 모듈 외에도 현시 모듈 및 수신 데이터를 기록하는 로깅 모듈에도 부하가 커지면서 시스템 자원 할당에 걸리는 시간이 늘어나기 때문이다.

그림 16은 초당 처리할 수 있는 평균 ASTERIX 데이터 처리량을 비교한 것이다. 항공기 약 300대까지 동일한 평균 처리량을 보이는 것은 ASTERIX 데이터 양이 많지 않아 파싱 모듈이 빠르게 데이터를 처리하더라도 단위 시간 내에 처리되는 ASTERIX 데이터 개수가 같기 때문이다. 그러나 300대 이상 항공기의 수가 증가하면 패턴매칭 기반의 ASTERIX 파싱 모듈의 평균 처리량이 크게 증가하는 것을 알 수 있다. 특히, 항공기의 수가 약 1050대일 경우 평균 처리량은 패턴매칭 기반의 ASTRERIX 파싱 모듈이 771.59개/s, 비트 수준 ASTERIX 파싱 모듈이 556.285개/s로 약 215개/s 차이를 보인다. 이는 패턴매칭 기반의 ASTERIX 파싱 모듈이 평균 처리지연시간을 단축시킴으로써 비트 수준 ASTERIX 파싱 모듈보다 네트워크로부터 수신되는 다수의 ASTERIX 데이터를 신속하게 처리하기 때문이다.

4.3 CPU 및 메모리 사용률 비교

그림 17은 각 파싱 모듈의 평균 CPU 사용률을 비교한 것이다. 항공기 대수 약 300대까지 4% 이하의 낮은 CPU 사용률을 보이지만 항공기 대수가 많아짐에 따라 CPU 사용률이 크게 증가하는 것을 알 수 있다. 이는 수신 ASTERIX 데이터의 파싱 횟수가 증가함에 따라 CPU의 단위시간 사용률도 많아지기 때문이다. 패턴매칭 기반의 ASTERIX 파싱 모듈은 비트 수준 ASTERIX 파싱 모듈에 비해 평균 11.28% 낮은 CPU 사용률을 보인다. 이것은 패턴매칭 기반의 ASTERIX 파싱 모듈이 비트 수준 ASTERIX 파싱 모듈과는 달리 파싱 과정을 최소화하고 필요한 정보만을 추출하는 방식을 사용하면서 CPU의 연산량이 크게 줄었기 때문이다.

그림 18은 각 파싱 모듈의 평균 메모리 사용률을 비교한 것이다. 항공기 대수가 증가함에 따라 다수의 ASTERIX 데이터를 처리하기 위해 더 많은 메모리 공간을 요구하게 되므로 메모리 사용률이 높아짐을 알 수 있다. 그리고 비트 수준 ASTERIX 파싱 모듈과 패턴매칭 기반의 ASTERIX 파싱 모듈의 평균 메모리 사용률

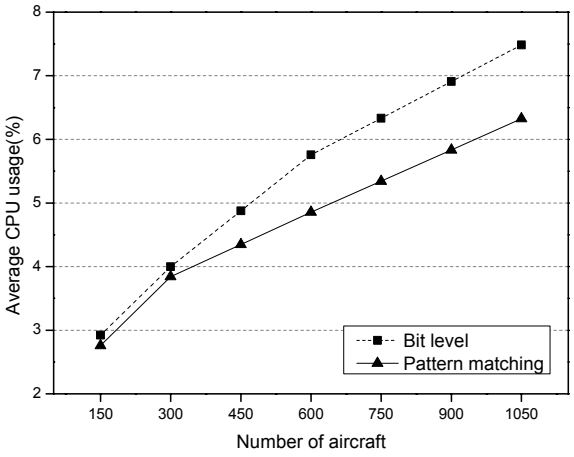


그림 17. 평균 CPU 사용률 비교
Fig. 17. The comparison with average CPU usage.

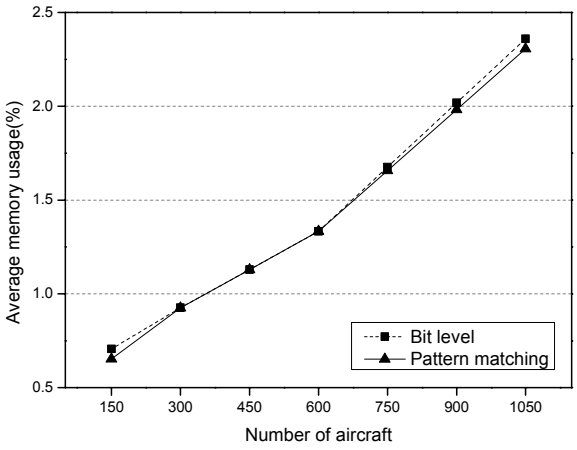


그림 18. 평균 메모리 사용률 비교
Fig. 18. The comparison with average memory usage.

이 동일함을 알 수 있다. 이것은 파싱 방식이 다르더라도 추출한 정보를 저장하는 공간은 동일하기 때문이다.

V. 결 론

본 논문에서 설계한 항공관제용 ASTERIX 파싱 모듈은 일반적인 파싱 방법인 비트 수준 파싱 방법의 단점을 보완한 것으로, 패턴매칭을 통해 불필요한 파싱 과정을 줄이고 신속하게 정보를 추출하는 것이다. 장시간 동안 ASTERIX 데이터를 수집하여 분석한 특성을 기반으로 패턴을 생성하고, 이 패턴을 빠르게 탐색하여 파싱에 적용하기 위해 우선순위 기반의 이진 탐색 트리를 이용한다.

설계한 ASTERIX 파싱 모듈의 성능을 평가하기 위하여 현재 인천공항 관제소 내에서 개발 중인 항공관제 시스템의 실험 환경을 이용하였다. 실험 환경내의 현시 시스템 두 대에 각각 비트 수준 ASTERIX 파싱 모듈과 패턴매칭 기반의 파싱 모듈을 설치하여 성능을 비교분석하였다. 패턴매칭 기반의 ASTERIX 파싱 모듈은 비트 수준 ASTERIX 파싱 모듈보다 평균 1.92us 짧은 처리지연시간을 가졌으며, 평균 22.5% 처리량이 향상하였다. 그리고 동일한 메모리 사용률을 가지면서 비트 수준 ASTERIX 파싱 모듈보다 평균 11.82% 낮은 CPU 사용률을 보였다. 따라서, 해마다 증가하는 항공교통량을 고려하였을 때, 패턴매칭 기반 ASTERIX 파싱 모듈이 효율적이라는 것을 알 수 있다.

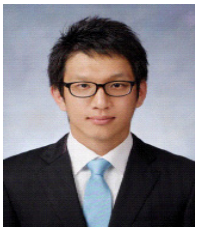
REFERENCES

[1] Ministry of Land, Infrastructure and Transport, “Market trend of air transportation,” Vol. 3, Jan, 2013.
[2] Ministry of Land, Infrastructure and Transport, “Press : Breaking highest air traffic in third quarter ,” Available at : http://www.mltm.go.kr/USR/NEWS/m_71/dtl.jsp?lcmspage=4&id=95073096
[3] Eurocontrol Standard Document for Surveillance Data Exchange Part 1 : All Purpose Structured Eurocontrol Surveillance Information Exchange (ASTERIX), European Organization for The Safety of Air Navigation, SUR.ET1.ST05.2000-STD-01-01, Edition 1.30, Nov, 2007.
[4] V. Manetti and L. M. Petrella, “FITNESS: A Framework for Automatic Testing of ASTERIX Based Software Systems,” JAMAICA 2013 Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation, ACM, Lugano, Switzerland, pp. 71-76, Jul, 2013.
[5] H. Ko. D. K. Jeon, Y. J. Eun and C. H. Yoem, “Establishment of Test Environment for Surveillance Data Processor,” *Conference of Korea Society of Computer Information*, Vol. 18, no. 2, pp 91-94, July, 2010.
[6] S. I. Na, J. W. Lee, I. S. Won, S. B. Choi, H. D. Park and D. S. Jeong, “The research of the Control Work Position for developing ATC,” *Conference of Institute of Electronics Engineers*

of Korea, pp 1197- 1198, Jun, 2008.
[7] U. Ahlstrom, J. Rubinstein, S. Siegel, R. Mogford, and C. Manning, "Display Concepts For EnRoute Air Traffic Control," DOT/FAA/CT-TN01/06, Feb, 2001.
[8] sourceforge, "AsterixInspector," Available at : http://sourceforge.net/projects/asterix/
[9] GitHub, "asterix," Available at : https://github.com/vog/asterix
[10] N. Yazdani and P. S. Min, "Prefix Trees: New Efficient Data Structures for Matching Strings of Different Lengths," Database Engineering and Applications, 2001 International Symposium on, pp. 76-85, Jul, 2001.
[11] C. Yim, H. Lim, and B. Lee, "Weighted Binary Prefix Tree for IP Address Lookup," *Conference of Institute of Electronics Engineers of Korea ISOCC*, pp 374-377, Oct, 2004.

저 자 소 개

김 강 희(학생회원)
2011년 인하대학교 전자공학과 학사 졸업.
2013년 인하대학교 전자공학과 석사 졸업.
2014년~현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 컴퓨터 네트워크, 무선 센서 네트워크, SoC>



김 호 중(학생회원)
2011년 인하대학교 전자공학과 학사 졸업.
2012년~현재 인하대학교 전자공학과 석사과정.
<주관심분야 : 병렬 및 분산 컴퓨팅, 컴퓨터 구조>

은 윤 동(학생회원)
2012년 인하대학교 전자공학과 학사 졸업.
2012년~현재 인하대학교 전자공학과 석사과정
<주관심분야 : 컴퓨터 네트워크, 병렬 및 분산 컴퓨팅>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주).
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>