Proceedings of the 12th
KACC, October 1997

효과적 데이터처리를 위한 필드버스의 온라인 토큰제어
On-Line Token Control of Fieldbus for Effective Data Service

°이석원, 남부희

강원대학교 제어계측공학과
(Tel:0361-250-6341; Fax:0361-242-2059; E-mail:boonam@cc.kangwon.ac.kr)
:seokwon@pine.kangwon.ac.kr

Abstract Fieldbus system should be designed by considering the number of sensors/actors(actuators+ controllers) and their sampling times. But the number of sensors/actors and which are connected to the network bus can be varied by addition and/or fault of the field devices. That variation decreases the efficiency of system which is off-line designed optimally. This paper proposes an algorithm that ensures transmission of cyclic data even though there is the number change in the field devices. We assign the different weight values to cyclic data and acyclic data. By simulation and experiment, the cyclic and acyclic data are processed via the proposed network, and the results are compared with those of the Profibus system. It is shown that the proposed algorithm is more robust with the uncertainties of the field devices of the control system.

Keywords cyclic/acyclic data, weight value, on-line scheduling

1. 서 론

1.1 필드버스의 등장배경 및 특징

필드버스 네트워크는 CIM의 계층구조상 가장 하위 계층인 생산 field의 분산 공정 제어(distributed process control)에 있어서 센서, 공정제어기, 액츄에이터, 계측기기, 스위치, 분석기 등과 같은 제어기기들 간의 통신을 위하여 비교적 저가이고 통신에 필요한 최소한도의 기능을 수행하는 단순한 구조를 가지면서도 빠른 응답시간으로 실시간 처리에 적용될 수 있는 네트워크 시스템이다. 종래의 생산현장에서 많이 사용되던RS-232, RS422와 같은 점대점 (point-to-point)의 1:1방식의 통신은 센서, 액츄에이터 등의 장치들과 통신을 위해서 많은 배선이 필요하게 되고 또한 유지보수측면에서 보았을 때 어려움이 있고, 새로운 장비를 추가 할 때마다 새로운 배선이 필요하다. 그러나 필드버스 시스템에서는 하나의 버스(bus)를 공동사용하는 네트워크시스템이므로 배선량을 상당히 줄일 수 있고 새로운 시스템의 확장이나 장비 등의 추가가 용이하고 유지보수면에서도 용이하다.

1.2 기존의 연구 및 연구방향

필드버스 시스템의 기존의 연구들을 보면 버스의 효율을 높이기 위하여 대역폭(bandwidth)할당에 관한 연구[2], FDL에 우선순위(Priority) 개념[3]의 도입 등 많은 연구들이 이루어져 왔다. 특히 대역폭을 할당하기 위해서는 버스를 사용하는 스테이션의 수, 데이터의 도착률, 데이터프레임의 크기, 전송속도 등 모든 시스템의 구성요소를 알아야만 폴링주기, 슬롯타임 등의 시스템을 설계할 수 있다. 또한 같은 버스를 사용하고 있는 마스터 스테이션 중에 장애를 일으킨 마스터 스테이션이 발생하면, 설계될 당시의 스테이션 수보다 스테이션이 적어질 경우가 발생하게 되므로 장애를 일으킨 스테이션이 사용할 수 있도록 할당된 것만큼 시스템 전체적으로 버스이용률이 저하되게 된다. 설계된 스테이션의 수를 초과하여 스테이션이 버스를 사용하게 될 경우 기존의 시스템에서는 데이터를 처리 못할 수 있다. 기존의 시스템들은 버스에 참여하는 스테이션의 수의 변화에 대해 적절히 대처할 능력이 없다. 이에 주기적 데이터와 비주기적 데이터를 갖는 필드버스시스템에서, 비주기적 데이터의 처리 량을 제어함으로서 주기적 데이터의 처리를 보장하면서도 버스의 이용 효율을 높일 수 있고, 논리적 토큰 링에 다른 스테이션의 가입을 쉽게 결정할 수 있는 알고리즘을 연구하고자 한다.

본 논문의 구성을 보면 2장에서는 제안된 알고리즘에 대하여 설명하고, 3장에서는 주기적 데이터만 발생 할 때 토큰의 폴링에 따른 주기적 데이터의 발생 빈도를 보이고, 제안된 알고리즘과 기존의 알고리즘을 이용한 데이터처리량을 비교하여 결과에 대해서 설명하고, 마지막으로 4장에서 결론을 맺는다.

2. 제안된 알고리즘

2.1 제안된 알고리즘

기존의 필드버스(Profibus) 시스템에서는 마스터 스테이션들 간에 logical ring을 구성한 후 스테이션에 토큰이 오면 무조건 1개의 data를 처리한 후 자신에게 할당된 시간이 남아 있으면 남

은 시간동안 데이터를 처리하고 토큰을 다음 스테이션으로 넘겨준다. 이런 대역폭을 할당하기 위해서는 버스를 사용하는 스테이션의 수, 데이터의 도착률, 데이터프레임의 크기, 전 송속도 등 모든 시스템의 구성요소를 알아야만 폴링주기, 슬롯타임 등의 시스템을 설계할 수 있다. 또한 같은 버스를 사용하고 있는 마스터 스테이션 중에 장애를 일으킨 마스터 스테이션이 발생하면, 설계될 당시의 스테이션 수보다 스테이션이 적어질 경우가 발생하게 되므로 장애를 일으킨 스테이 션이 사용할 수 있도록 할당된 것만큼 시스템 전체적으로 버스 이용률이 저하되게 된다. 설계된 스테이션의 수를 초과하여 스테이션이 버스를 사용하게 될 경우 기존의 시스템에서는 데이터를 처리 못할 수 있다. 기존의 시스템들은 버스에 참여하는 스테이션의 수의 변화에 대해 적절히 대처할 능력이 없다. 뿐만 아니라 비주기적 데이터의 과부하 상태하에서 주기적 데이터의 도착률이 큰 스테이션은 주기적 데이터를 거의 처리 못하는 단점이 있다. 이에 주기적 데이터와 비주기적 데이터를 갖는 필드버스 시스템에서, 비주기적 데이터의 처리 량을 제어함으로서 주기적 데이터의 처리를 보장 하 면서도 버스의 이용 효율을 높일 수 있고, 논리적 토큰 링에 다른 스테이션의 가입여부를 쉽게 결정할 수 있는 알고리즘을 제안한다. 이 알고리즘은 주기적 data를 모두 처리하고, 추가적으로 acyclic data 처리 량을 늘릴 수 있다. 기존의 알고리즘이 논리적 토큰 링에 참여하는 스테이션의 수가 줄었을 때 walk time시간이 늘어나지만 제안된 알고리즘에서는 walk time에 소비되는 시간을 줄이고 data처리량을 늘림으로서 버스의 효율을 높인다.

2.2 제안된 알고리즘 단계

1) cyclic데이터의 weight value는 스테이션의 갯수 N 으로 하고 acyclic data의 weight value는 1로 한다.

2) 각 스테이션은 자신에게서 발생한 주기적 데이터를 처리하면 앞 스테이션으로터 받은 데이터 서비스 값 Sv에 '0'을, 주기적 데이터가 발생하지 않아 데이터를 처리하지 않고 바로 다음 스테이션으로 토큰을 전달하면 Sv에 '+1'을 더하여 다음 스테션에 토큰과 함께 전달한다.

3) 주기적 데이터가 발생하지 않고 전송큐에 대기 중이던 비주기적 데이터를 하나 처리하면 Sv에서 '1'을 뺀 나머지 값을 토큰과 함께 다음 스테이션으로 전달한다.

4) 스테이션이 토큰을 받았을 때 처리해야 할 데이터weight value의 합이 N이상이면 데이터를 처리한다. 이때 처리해야하는 데이터의 수는 앞 스테이션들이 사용하지 않고 넘겨준 데이터 처리시간과 자신의 시간을 사용할 수 있다.

5) 논리적 토큰 링의 마지막 스테이션은 자신의 데이터를 모두 처리한 후 데이터 서비스데이터 값을 '0'로 설정한 후 논리적 링의 맨 앞 스테이션으로 토큰을 전달한다. Sv값을 '0'으로 설정하기 전에 서비스 값이 '+'이면 논리적 링에 다른 스테이션을 추가할 수 있고 이 값이 '0'이면 논리적 링에 스테이션을 추가할 수 없다.

3 모의실험 및 결과

3.1 전송속도

제안된 알고리즘을 실험하기 위하여 구성한 시스템은 6개의 스테이션으로 이루어져 있다. 각 스테이션은 주기적 데이터와 비주기적 데이터 큐를 갖으며 주기적 데이터와 비주기적 데이터 프레임의 크기는 같다. 각 스테이션의 주기적 데이터 도착율 은 λ_1 =1.66, λ_2 =2.5, λ_3 =5, λ_4 = 12.5 λ_5 =16.7, λ_6 =25이며 시스템이 안정화 될 조건은 식(1)을 만족시켜야하고, 각 스테이션

$$\sum_{i=1}^{N} \lambda_i b_i < 1 \qquad (1)$$

에서 갖는 주기적 데이터의 크기는 같으므로 b_i (i=1,2,...N)는 모두 같다. $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6$ = 63.36이므로 63.36 b_i < 1 에서 b_i < 15.78 msec가 되어야 한다. 그러나 폴링 시스템에서 최악의 경우인 모든 스테이션에 동시에 데이터가 도착했을 때를 고려하여 시스템을 설계해야한다.

$$\lambda_M = \max[\lambda_i] \ (i=1, 2, \cdots, N) \qquad (2)$$

식(2)에서, 스테이션이 갖는 데이터 도착률 중의 가장 큰 값은 25이다. 전체 시스템에서 동시에 데이터가 발생했을 경우에 대비하여 25 • N의 값을 전체 시스템의 데이터 도착률로 결정한다. 여기서는 N이 6이므로,

$$25 \times N = 150 \qquad (3)$$

그러므로 식(1)에 대입하면 150 b_i < 1에서 스테이션i에서의 서비스시간은

$$b_i < 6.67 \text{ msec} \qquad (4)$$

과 같이 나타낼 수 있다. 하나의 주기적 데이터를 처리하는 과정은 총 20 byte의 데이터를 전송하는 시간으로 볼 수 있고, 20byte를 6.67msec 안에 전송하기 위해서는 속도가 29.98 kb/s 이상이 되어야한다.[6]

3.2 비주기적 데이터의 생성

비주기적 데이터의 생성은 랜덤함수를 사용하여 생성하였다. 여기서 발생되는 비주기적 데이터는 모두 처리되는 것으로 간주하였으므로 발생량은 처리량과 같다. 랜덤함수 $Rand(S_i)$ 는 스테이션 i에서 발생하는 0에서 1사이의 값이며, 이 값에 데이터의 생성 비율을 조정할 수 있는 값인 Rr($0 \le Rr < 1$)을 더하여 1보다 큰 값이 나왔을 때 비주기적 데이터 가 발생한 것으로 한다. Rr이 큰 값을 갖게 되면 비주기적 데이터의 생성비율이 크며, Rr이 작은 값을 갖게 되면 비주기적 데이터의 생성비율이 낮아진다. 이렇게 생성된 비주기적 데이터의 각 스테이션에 대한 분포는 균일하게 분포되어진다.

$$Rand(S_i) + Rr \ge 1 \ (i=1, 2, \cdots N) \qquad (5)$$
$$0 \le Rr < 1$$

식(5)는 스테이션 i 에서 비주기적 데이터의 생성을 나타내며, 이 랜덤 함수는 스테이션이 토큰을 받을 때마다 실행하여 비주기적 데이터의 서비스 여부를 결정한다. Rr 값의 초기치는 0이며 토큰이 시스템을 폴링하면서 주기적 데이터가 발생하지 않아서 처리할 데이터가 없을 때 Rr 값을 0.001 증가시키고, 주기적 데이터가 발생되어 비주기적 데이터를 처리할 시간이 없을때는 Rr 값을 줄임으로서 비주기적 데이터의 처리량을 줄인다. 즉 주기적 데이터의 양에 따라 비주기적 데이터의 처리량을 제어하여 모든 스테이션에서의 안정성을 요구하는 범위 내에서 버스이용 효율을 최대로 할 수 있도록 한다.

3.3 실험결과

3.3.1 주기적 데이터만 존재하는 경우

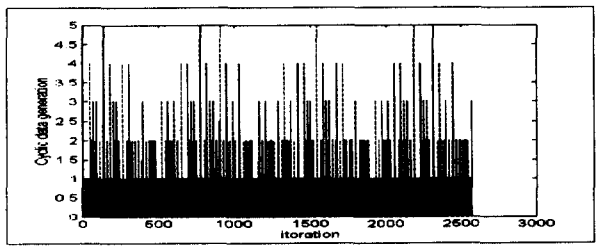

그림 1. 실험시스템에서의 주기적 데이터의 발생빈도

그림1은 여섯 개의 스테이션이 논리적 링을 구성하고 있을 때 주기적 데이터의 발생빈도를 나타내는 것이다. 이 때 각 스테이션의 주기적 데이터의 도착률은 $\lambda_1 = 1.66$, $\lambda_2 = 2.5$, $\lambda_3 = 5$, $\lambda_4 = 12.5$, $\lambda_5 = 16.7$, $\lambda_6 = 25$ 이며 토큰이 6개의 스테이션 모두를 폴링하는 동안에 발생되는 주기적 데이터의 수를 나타낸다. 앞 절에서 전체 시스템의 안정성을 고려하여 전송속도는 6개의 스테이션 모두에서 주기적 데이터가 발생했을 경우를 고려하여 설정하였지만 실제로는 6개의 스테이션에서 동시에 데이터가 발생한 경우는 약 2500여번의 순환동안 한번도 일어나지 않았다. 실제로 한번 순환하는데 하나의 주기적 데이터만이 발생되는 경우가 그림 1에서와 같이 제일 많이 발생하였다.

3.3.2 비주기적 데이터가 랜덤하게 발생할 경우의 Rr변화

1.그림11은 6개의 스테이션이 모두 논리적 링에 가입하였을 때

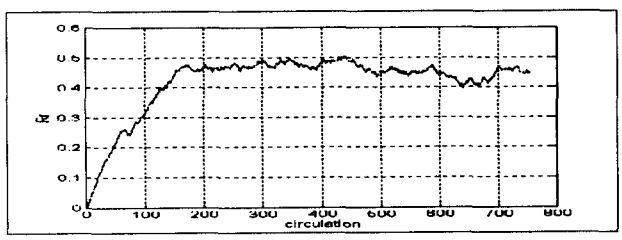

그림2. 6개의 스테이션이 논리적링에 참여하였을때의 Rr의 변화

Rr값의 변화이다. 주기적 데이터의 도착률은 앞 절과 같고 비주기적 데이터의 발생은 식(5)와 같이 발생된다. 그림 1에서와 같이 전체 시스템을 한번 폴링하는 동안 주기적 데이터의 발생빈도는 낮다. 그러나 시스템은 전체 시스템을 한번 폴링하는 동안

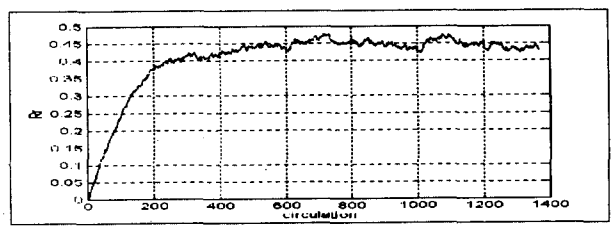

그림3 스테이션6이 논리적 링에서 제외 되었을 때의 Rr 값의 변화

각 스테이션은 각각 하나의 데이터프레임을 처리할 수 있도록 설계되어 있으므로 주기적 데이터가 발생하지 않는 동안은 비주기적 데이터를 처리하는 것이 효율적이다. 그림2의 Rr값에 비하여 그림3의 Rr값을 전체적으로 높게 나타난 것을 볼 수 있다. 그림2는 6개의 스테이션이 모두 논리적 링에 가입하여 있는 상태이고, 그림3은 주기적 데이터의 도착률이 25로 제일 큰 스테이션이 논리적 링에서 제외되어 있는 시스템임으로 상대적으로 주기적 데이터의 발생빈도가 낮다. 그러므로 효율을 높이기 위해 더 많은 비주기적 데이터의 발생을 위해 Rr값이 커진 것이다. 비주기적 데이터들이 랜덤하게 발생되므로 Rr값이 비선형적인 모양으로 나타나지만 Rr값을 제어하여 전체 시스템의 효율을 높이는 것을 볼 수 있다.

3.3.3 Profibus와 제안된 알고리즘의 성능비교

아래 표들은 논리적 토큰링에 참여하는 스테이션의 변화에 따른 주기적, 비주기적 데이터의 처리량을 비교한 것이다.

[표 1. 6개의 스테이션이 논리적링에 참여하였을때]

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	60	120	300	401	601	0
처리된비주기적데이터	562	542	482	302	201	1	0
폴링횟수	602						
처리된총데이터수	3612						

(a) Profibus

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	60	120	300	400	601	0
처리된비주기적데이터	400	470	374	321	257	165	0
폴링횟수	883						
처리된 총 데이터수	3508						

(b) 제안된 알고리즘

[표2.스테이션5가 논리적링에서 제외되었을때]

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	60	120	300	0	601	0
처리된비주기적데이터	682	662	602	422	0	121	0
폴링횟수	722						
처리한 총데이터수	3610						

(a) Profibus

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	60	120	300	0	601	0
처리된비주기적데이터	517	592	506	434	0	289	0
폴링횟수	1113						
처리한 총데이터수	3459						

(b) 제안된 알고리즘

[표 3. 스테이션7이 논리적링에 추가 되었을 때]
( ): 처리못한 데이터수

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	61	122	305	407	523 (88)	524 (87)
처리된비주기적데이터	484	463	402	219	117	1	0
폴링횟수	524						
처리한 총데이터수	3668						

(a) Profibus

스테이션번호	1	2	3	4	5	6	7
처리된주기적데이터	40	61	122	305	407	611	611
처리된비주기적데이터	317	391	288	247	173	68	68
폴링횟수	732						
처리한 총데이터수	3718						

(b) 제안된 알고리즘

표1은 6개의 스테이션이 모두 논리적 링에 참여하였을 때 Profibus 알고리즘과 제안된 알고리즘에 의해 처리된 주기적 데이터 및 비주기적 데이터 수이다. Profibus의 알고리즘은 모든 스테이션은 토큰을 받으면 주기적 데이터를 처리하고 할당된 시간이 남아 있으면 비주기적 데이터를 처리한다. 실험 시스템에서의 할당된 시간은 하나의 데이터 프레임을 처리할 수 있는 시간이다. 그러므로 토큰을 받았을 때 주기적 데이터가 발생하면 주기적 데이터를 처리하고 그렇지 않으면 비주기적 데이터를 처리하는 방법으로 데이터를 처리하게 된다. 제안된 알고리즘은 3.3절에서와 마찬가지로 토큰을 받았을 때 주기적 데이터가 발생하였으면 주기적 데이터를 처리하고 그렇지 않았으면 Sv 값에 +1을 더하여 다음 스테이션으로 이 토큰을 넘겨주고 토큰을 넘겨받은 스테이션은 주기적 데이터가 발생하였으면 주기적 데이터를 처리하고 Sv 값이 기준치 이상이 되면 비 주기적 데이터를 처리한 후 일정 값을 Sv에서 뺀다. 여기서는 Sv값이 4이상일 때 비주기적 데이터를 처리하고 비주기적 데이터를 처리하였을 때는 1.8을 뺀다. 이 두값을 조정함으로서 주기적 데이터는 모두 처리하면서 비 주기적 데이터 처리량을 조정 할 수 있다. 표1 은 6개 스테이션이 모두 참여하였을 때 처리한 데이터 량을 보여주고 표.2.는 스테이션5가 논리적 링에서 제외되었을 때 처리된 데이터 량을 보여준다. 이 두 가지 경우에서 보면 Profibus 알고리즘이나 제안된 알고리즘 모두 주기적 데이터는 다 처리하였다. 그러나 비주기적 데이터 처리량은 다르다. Profibus알고리즘으로 처리한 데이터의 양은 주기적 데이터 와 비주기적 데이터의 양을 합하면 모든 스테이션이 똑같은 개수의 데이터를 처리한 것을 볼 수 있다. 그러나 제안된 알고리즘 에서는 주기적 데이터의 도착률이 큰 스테이션이 전체적으로 더 많은량의 데이터를 처리한 것을 볼 수 있다. 실제로 표1에서 스테이션6은 비주기적 데이터를 1개밖에 처리하지 못하였다. 비주기적 데이터가 시간적인 측면에서는 주기적 데이터 보다 늦게 처리 되어야지만 그렇다고 처리되지 않아도 되는 것은 아니므로 현실적으로 불합리하다. 반면에 제안된 알고리즘은 도착되는 모든 주기적 데이터를 처리하면서도 비주기적 데이터 처리량이 덜 집중되어 있다. 표3은 스테이션6과 똑같이 주기적 데이터 도착율 $\lambda_7 = 25$ 인 스테이션을 추가 하였다. 표3 의 Profibus의 스테이션 6과 7의 주기적 데이터란에 괄호 안의 숫자는 처리하지 못한 데이터의 양을 표시한다. 반면에 제안된 알고리즘은 스테이션 7을 추가하여도 모든 주기적 데이터를 처리하는 것을 보여 주었다. 이것은 비 주기적 데이터의 처리 량을 조절함으로서 추가된 주기적 데이터를 처리할 수 있다는 것을 보여준다.

4.결 론

본 연구에서는 전송속도와 시스템안정성과의 관계를 보였다. 논리적링에 참여하는 스테이션들 사이에 주기적 데이터가 동시에 발생하는 빈도를 보였다. 기존의 Profibus, IEEE 802.4의 토큰버스 방식에서 사용되는 효율적인 네트워크를 이용할 수 있도록 하기 위해, 비 주기적 데이터의 도착률을 제어하여 주기적 데이터의 도착률이 변화하더라도 안정적으로 주기적 데이터를 처리할 수 있는 알고리즘은 제안 하였고, 주기적 데이터의 도착률의 변화에도 전체적인 데이터 처리 량은 크게 변하지 않음을 보였다. 또한 설계시 보다 적은 수의 스테이션이 논리적 토큰 링에 가입하였을 때나 더 많은 스테이션이 논리적 토큰 링에 가입하였을 때도 비주기적 데이터의 처리 량을 조절하여 모든 주기적 데이터를 처리하는 유연성을 보였다.

참 고 문 헌

[1] PROFIBUS standard DIN19 245 part I and II, 1992.

[2] S. H. Hong, "Scheduling Alogorithm of Data Sampling Times in the Integrated Communication and Control Systems," IEEE Tran. on Control System Technology, Vol. 3, No. 2, pp.225-230, June 1995.

[3] S. Cavalieri, A. Di Sefano and O. Mirabella, "Optimization of Acyclic Bandwidth Allocation Exploiting the Priority Mechanism in the Fieldbus Data Link Layer," IEEE Trans. Industrial Electron., Vol. 40, No. 3, pp.297-306, June 1993.

[4] Kang G. Shin, Chih-Che Chou, "Design and Evaluation of Real-Time Communication for Fieldbus-Based Manufacturing Systems", IEEE Trans. Robotics and Automation, Vol., 12, No. 3, June 1996.

[5] O. C. Ibe and X. Cheng, "Stability Condition for Multiqueue Systems with Cycle Service," IEEE Tran. on Automatic Control, Vol.,33, No.1, pp.102-103, January 1988.

[6] 이석원, 장석호, 최형섭, 남부희 "필드버스를 이용한 자동화 네트워크의 구현" '96 자동제어학술회의 논문집 제2권. pp1420-1423