A Precise Bandwidth Control Arbitration Algorithm for Hard Real-Time SoC Buses

Bu-Ching Lin, Geeng-Wei Lee, Juinn-Dar Huang, Jing-Yang Jou

Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan

Advanced Design Automation Research

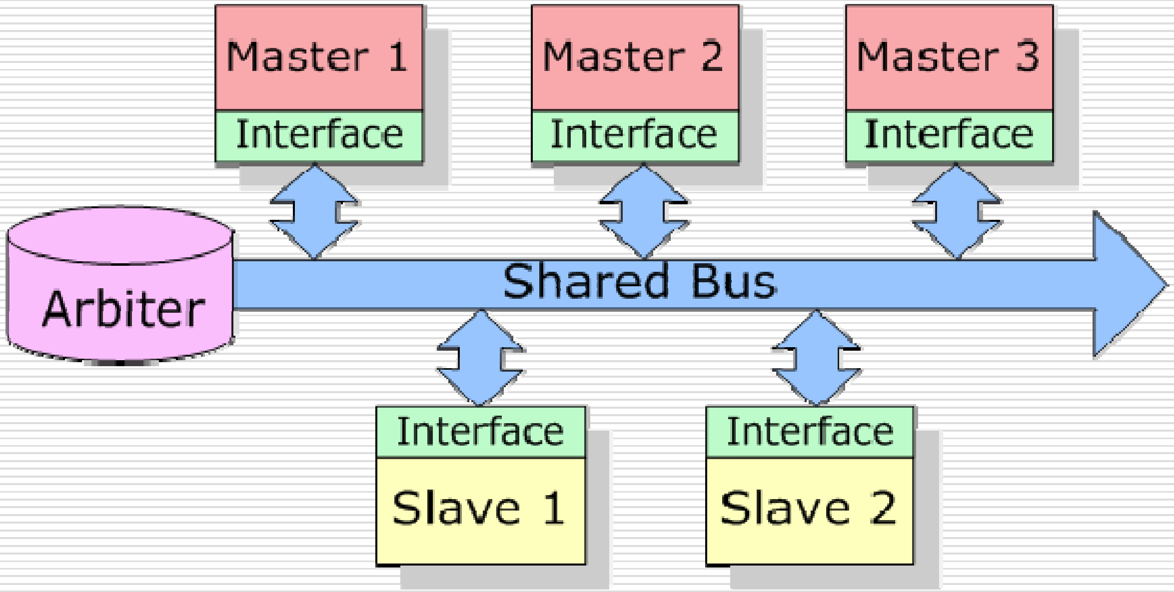
2007/01/24

Outline

- Introduction
- Previous Works
  - fixed priority
  - time division multiple access (TDMA)
  - Lottery
  - RT_lottery
- Proposed Arbitration Architecture
- Experimental Results
- Conclusions

Introduction(1/2)

- Shared bus is widely used in current SoC designs
  - master – initiate communication transactions
  - slave – respond to transactions initiated by masters
  - arbiter – manage the usage of bus



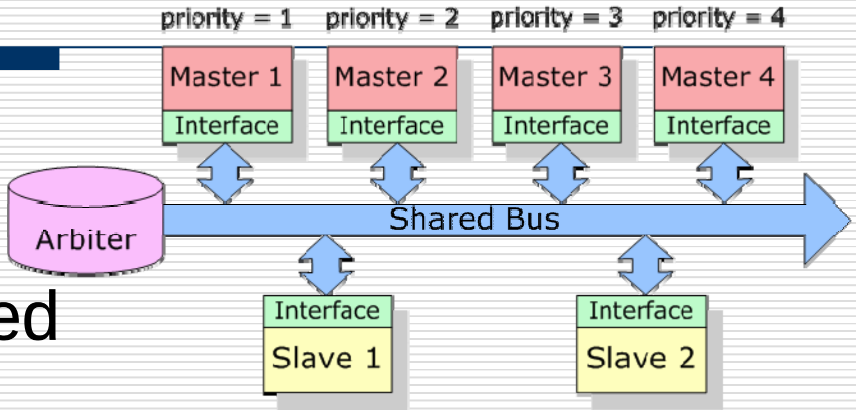
Introduction(2/2)

- Requirements in different applications
  - complete transactions of all requests before the corresponding deadlines in real-time applications
  - take at least a fixed fraction of total bandwidth in multimedia applications
- Difficult to satisfy both real-time and bandwidth requirements simultaneously
  - an innovative arbitration algorithm is required

Previous Works

- Existing arbitration algorithms
  - fixed priority
  - time division multiple access (TDMA)
  - Lottery
  - RT_lottery

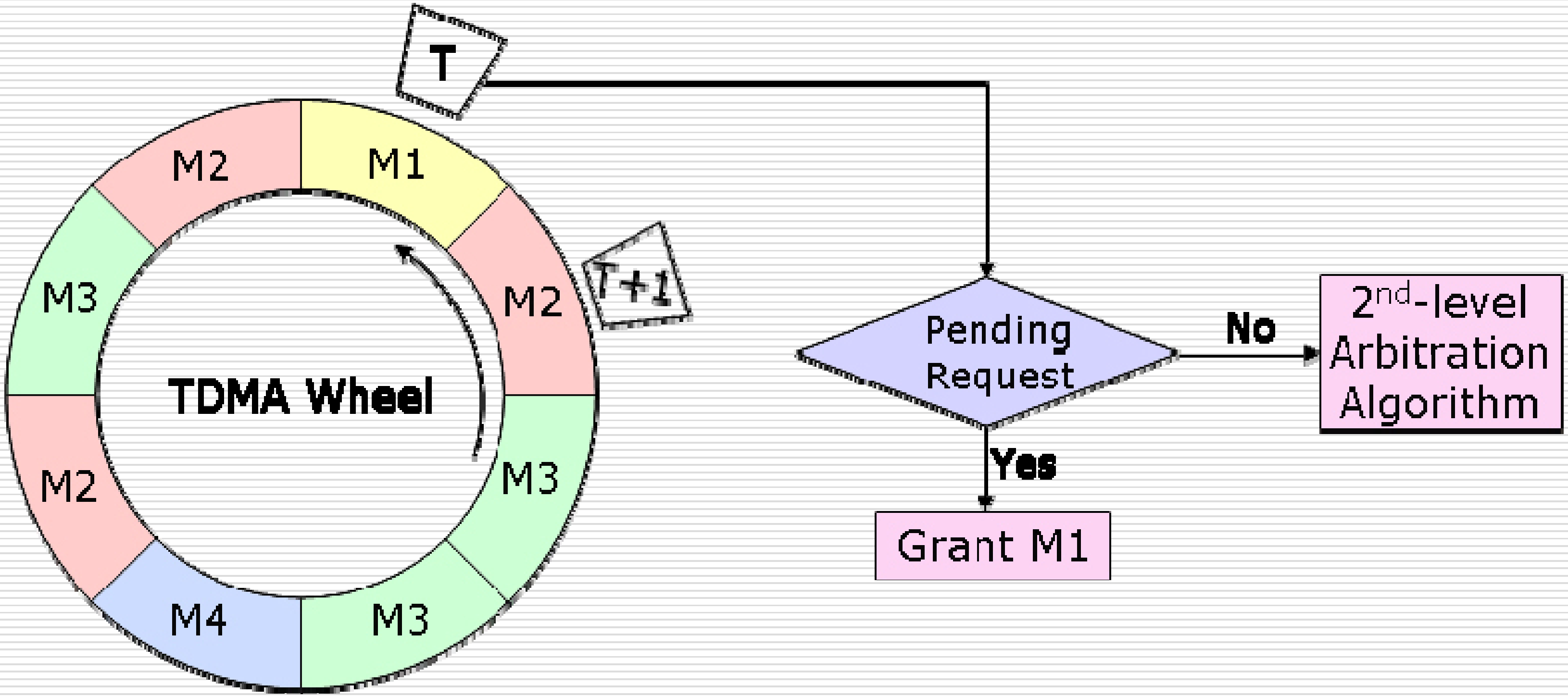
Fixed Priority



- Among the requesting masters, the one with the highest priority gets granted
- Pros
  - simple, low hardware cost and easy to implement
- Cons
  - starvation problem – the masters with lower priority hardly get the service
  - lack of control over real-time and bandwidth requirements

TDMA (1/2)

- Execution time is divided into time slots which are statically assigned to masters
- 2nd-level of arbitration is usually adopted to alleviate the wasted time slots

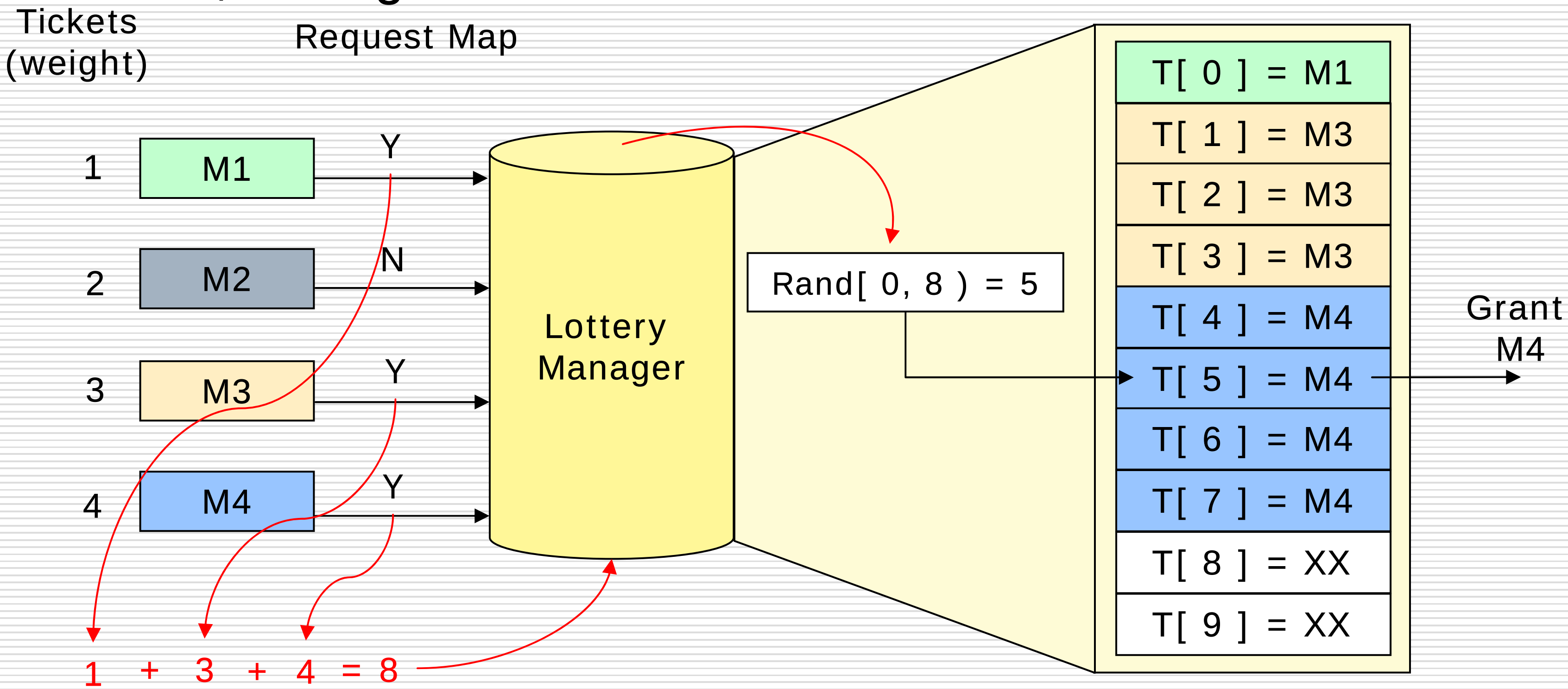


TDMA (2/2)

- Pros
  - deterministic worst-case response latency
  - reserved bandwidth for each master
- Cons
  - difficult to design time slot sequences in an unpredictable system
  - more slots ➔ more bandwidth and shorter latency
    - what if a master with LOW bandwidth requirement but needs SHORT response latency?

Lottery

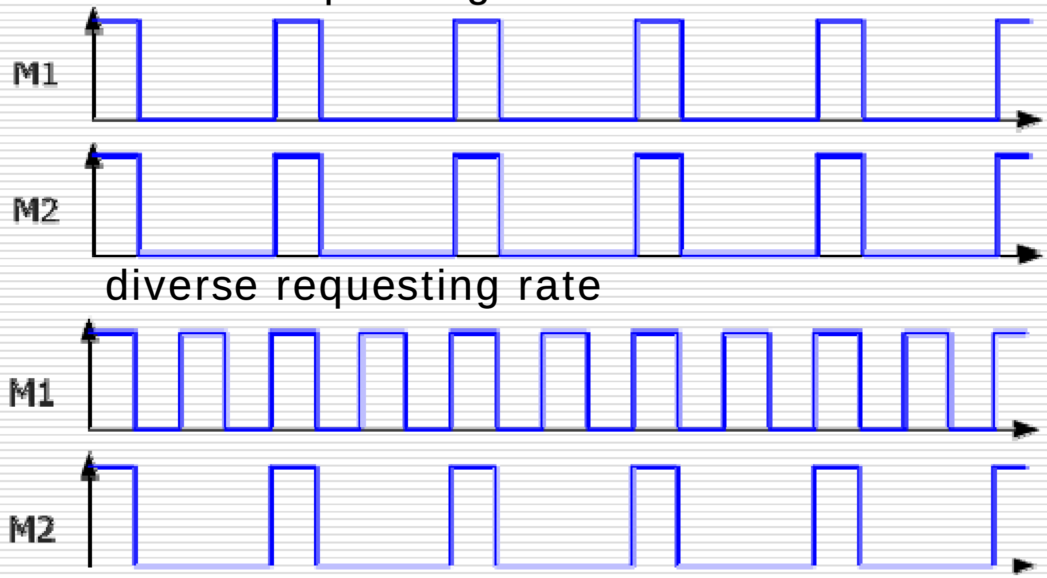
- Arbiter grants a master stochastically from contending requests
  - i.e., a weighted random mechanism



Why Lottery is not suitable in SoC ?

- Similar requesting rate for each master is assumed
- What if the requesting rates are not similar?
  - e.g., 3 masters with the same tickets assignment, one with the requesting rate varies from 80% to 40%, the rate of the other 2 is fixed at 40%

similar requesting rate



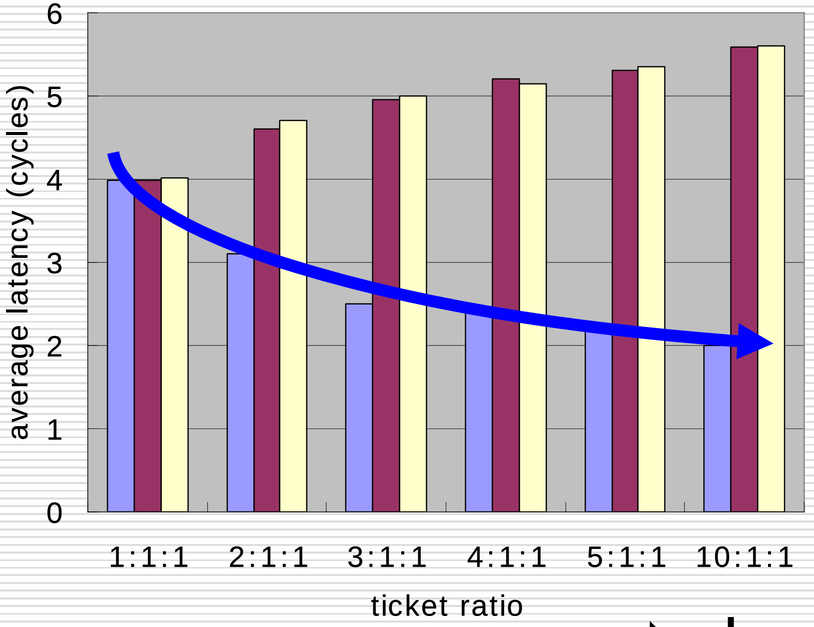
	Requesting Rate of M1				
	80%	70%	60%	50%	40%
M1	49%	46%	42%	38%	33%
M2	24%	26%	28%	30%	33%
M3	24%	26%	28%	30%	33%

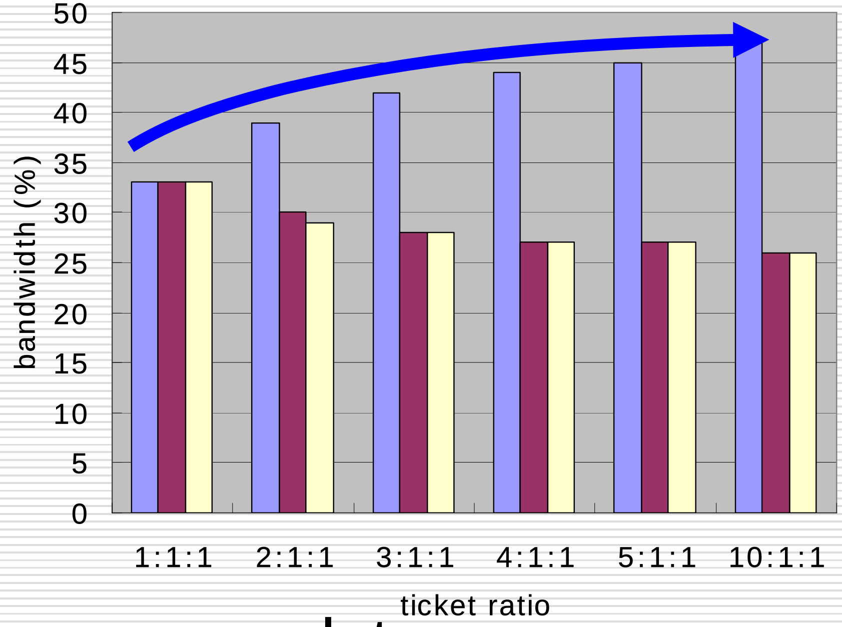
⇨ ticket ratio ≠ bandwidth ratio

⇨ weight tuning is required

Latency and Bandwidth in Lottery

- Response latency and bandwidth allocation both controlled by the number of tickets
  - e.g., 3 masters have similar traffic behaviors





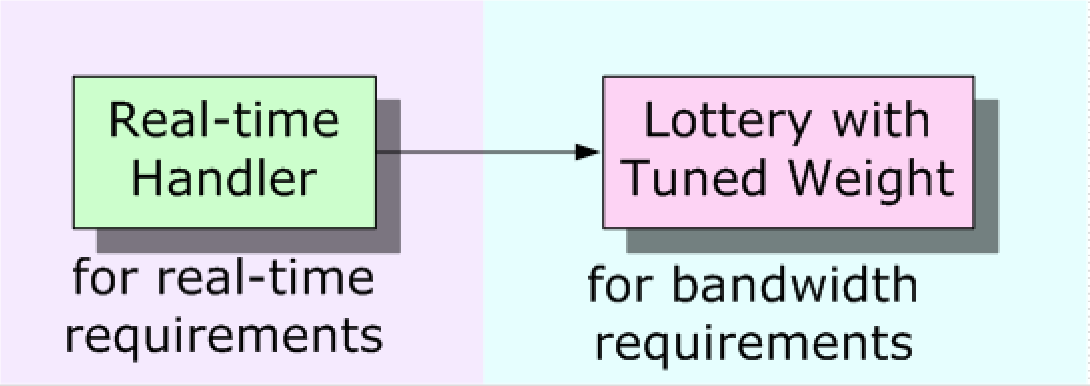
more tickets ⇨ shorter response latency
⇨ more bandwidth allocation

Summary of Lottery

- Pros
  - good control over bandwidth allocation in network switching applications
  - fair average response latency
- Cons
  - no hard real-time consideration
  - no independent controllability over response latency and bandwidth allocation

RT_lottery

- A 2-level arbitration algorithm dealing with real-time and bandwidth requirements simultaneously
- The proposed architecture
  - 1st level – real-time handler
    - handles the hard real-time requirements
  - 2nd level – Lottery with tuned weight
    - reserves the bandwidth allocation for each master

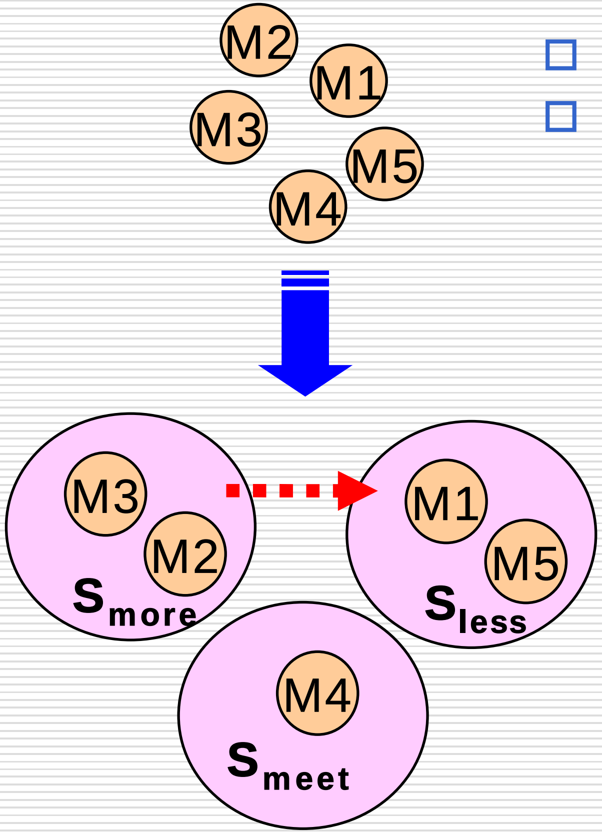


Real-Time Handler

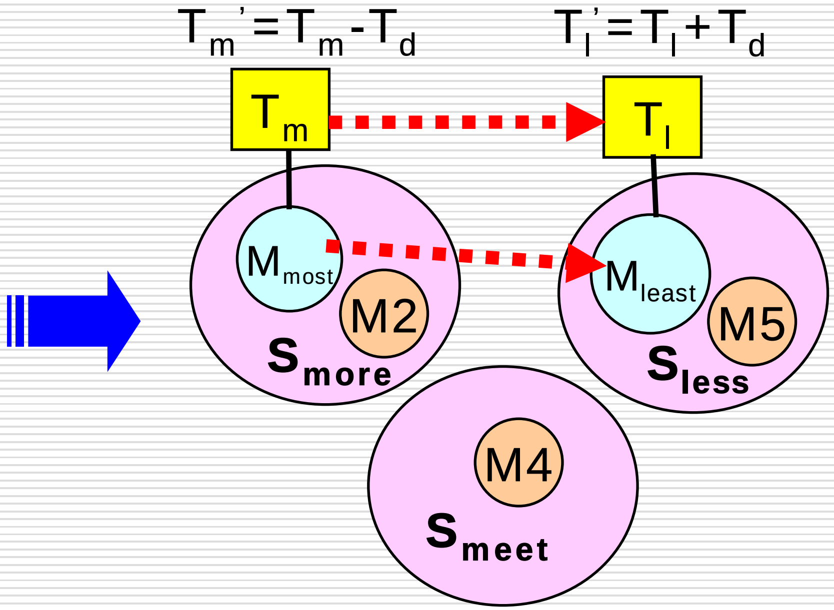
- Similar to earliest deadline first scheduling (EDF)
  - the request with earliest deadline and below the warning line gets granted
- Deadline
  - the time limit for a master to complete a request
  - missing the deadline is regarded as the real-time violation
- Warning line
  - the worst case of scheduling the contending requests

Weight Tuning

- A ticket redistribution mechanism to meet the required bandwidth by simulation

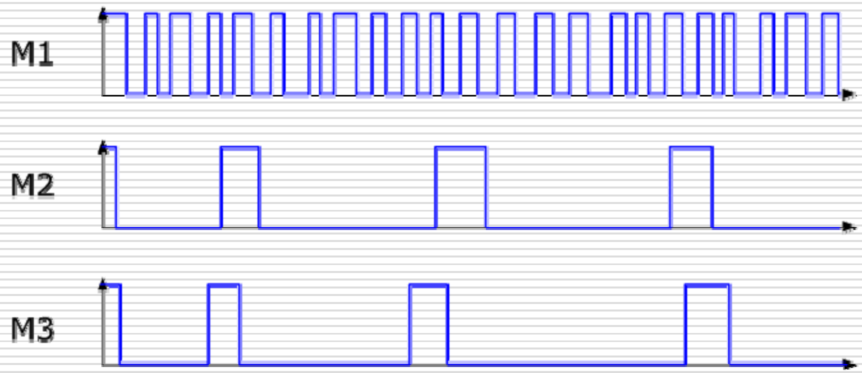


- Initially, $T_d = T_m/2$
- If any master in S_{more} or S_{met} -> S_{less}
  - $T_d = T_d / 2$



Fail Case of Weight Tuning

- Fail to meet bandwidth requirements due to diverse masters' requesting rate
  - e.g., the requesting rate of each master is 80%, 30%, 30%, respectively and each of them requires at least 30% bandwidth



over-allocated bandwidth even with few tickets

	Ticket Assignment of M1, M2 and M3							
	100:100:100	**60**:120:120	**44**:128:128	**34**:133:133	**28**:136:136	**16**:142:142	**8**:146:146	**2**:149:149
M1	60	58	57	56	55	55	54	53
M2	19	20	21	21	21	22	22	23
M3	19	20	21	21	22	22	22	23

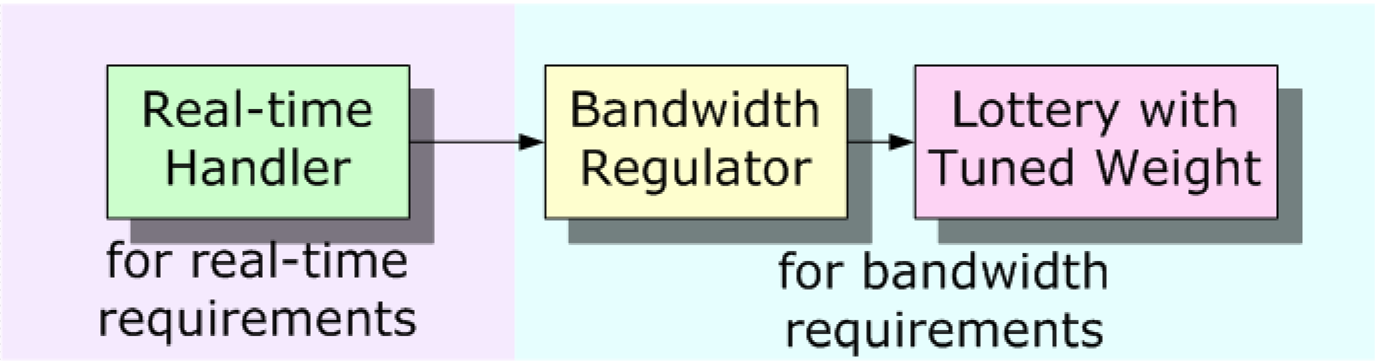
⇨ weight tuning is not a panacea !

Summary of Previous Works

	Pros	Cons
fixed priority	simplicity area efficiency	no real-time consideration no means for bandwidth control
TDMA	deterministic worst-case latency reserved bandwidth allocation	no hard real-time guarantee no precise bandwidth control
Lottery	reserved bandwidth allocation fair average latency	no real-time consideration no precise bandwidth control
RT_lottery	hard real-time guarantee	limitation of Weight Tuning

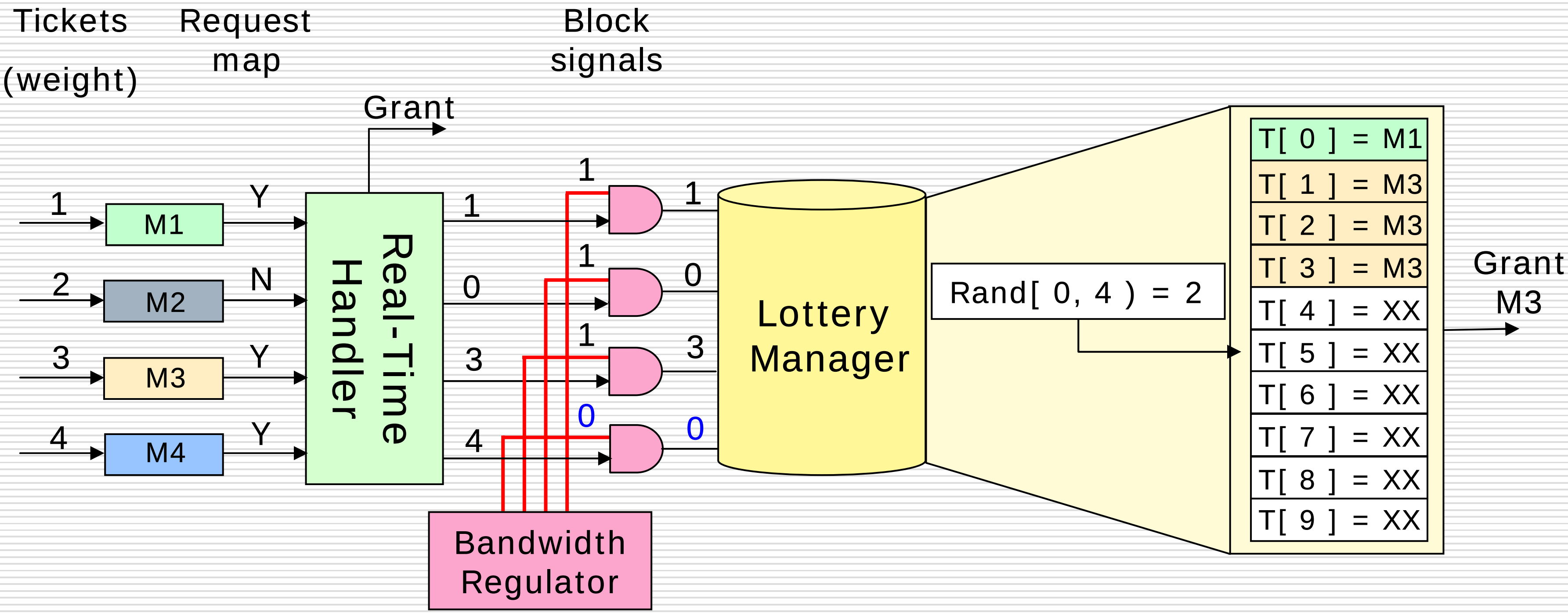
RB_lottery Architecture

- 3-level arbitration algorithm
  - real-time handler – handles the hard real-time requirements
  - Lottery with tuned weight – reserves the bandwidth allocation for each master
  - bandwidth regulator –provides fine-grained control over bandwidth allocation



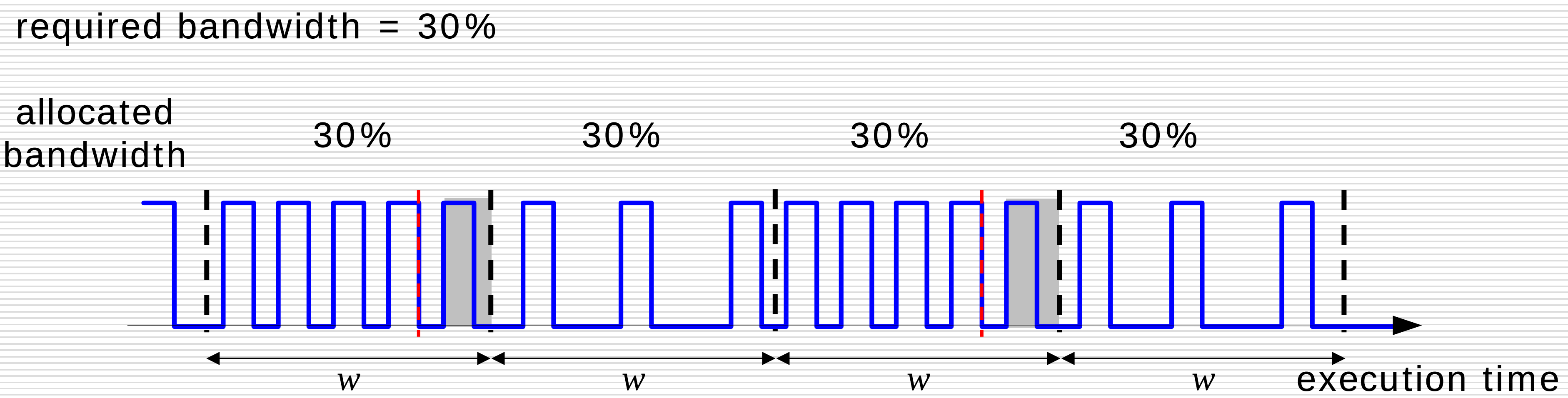
An Example of RB_lottery

- Bandwidth regulator monitors the bus traffic
  - record the transactions of each master
  - temporarily block the requests from masters that have already got the required bandwidth in a period

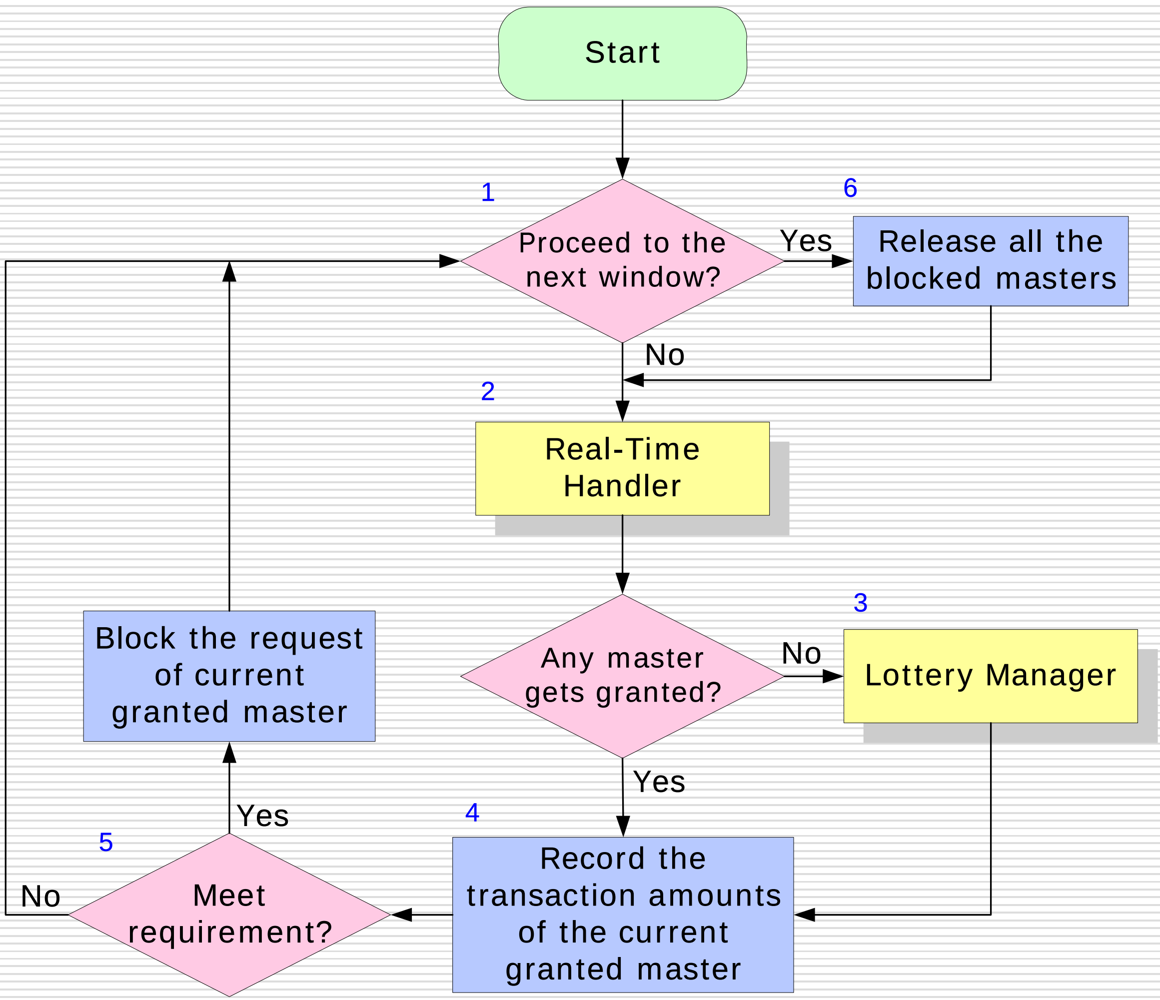


The Implementation

- Observation window (*w*) — the execution time is divided into windows of size *w* cycles
  - block the requests of over-served masters temporarily
- Bandwidth register — the allocated bandwidth in the current window



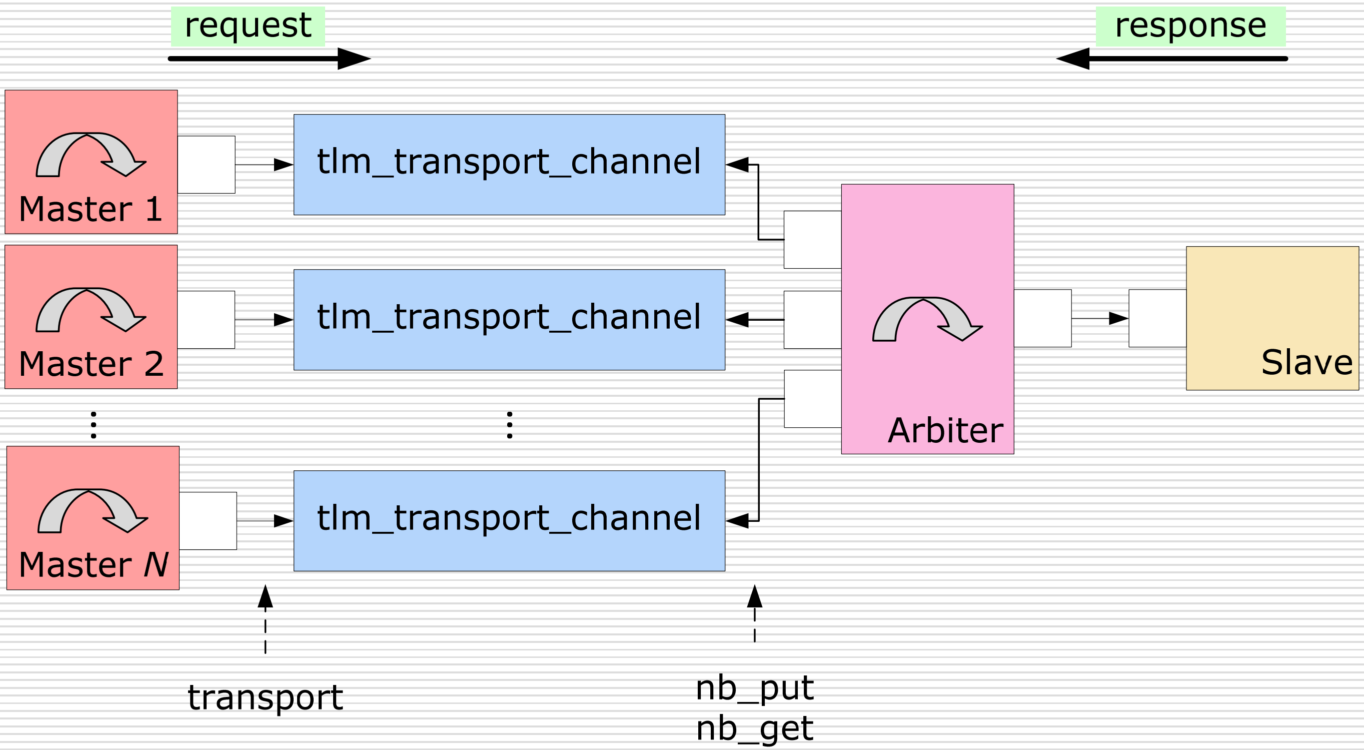
RB_lottery Algorithm Flow



1. check whether a new window starts
2. grant the most urgent master
3. stochastically grant an un-blocked master
4. record the transaction cycles
5. check the allocated bandwidth
6. reset all the blocked signals and proceed to the next window

Experimental Environment

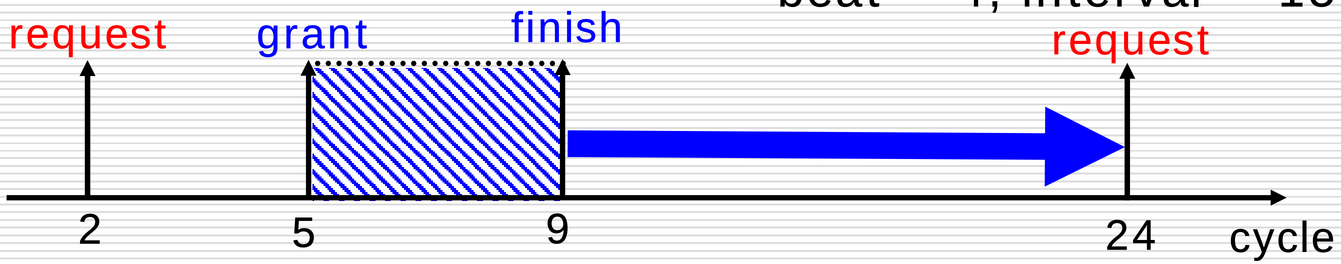
- Transaction level model in SystemC
  - different arbitration algorithms are used in the experiments, such as fixed priority, lottery, TDMA + Lottery, RT_lottery, RB_lottery



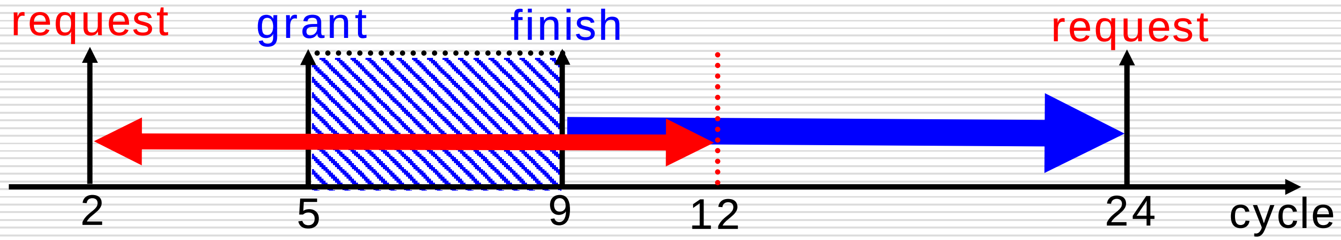
3 Types of Masters

- D type (D for Dependency)

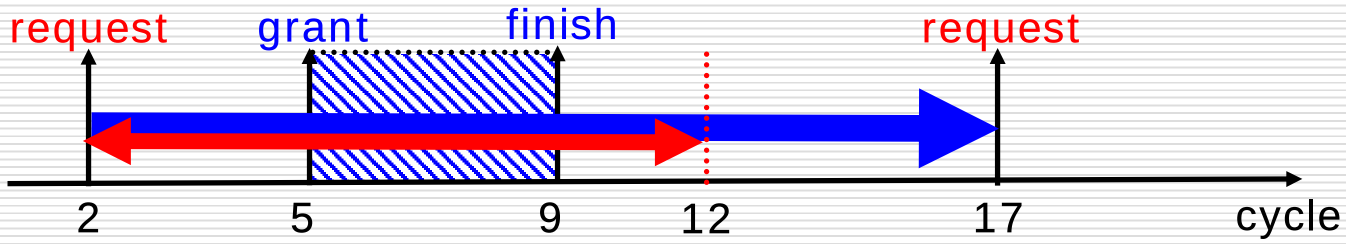
Example :
beat = 4, interval = 15, R_{cycles} = 10



- D_R type (D for Dependency, R for Real-time)



- ND_R type (ND for No Dependency, R for Real-time)



Experiment Setup

- Behavior of masters

	type	beat/prob.		interval/prob.				
Master 1	D	8/50	16/50	6/10	7/20	8/40	9/20	10/10
Master 2	D	1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 3	D	8/50	16/50	6/10	7/20	8/40	9/20	10/10
Master 4	D	1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 5	D_R	8/50	16/50	10/10	11/20	12/40	13/20	14/10
Master 6	D_R	1/50	4/50	10/10	11/20	12/40	13/20	14/10
Master 7	ND_R	8/50	16/50	65/10	66/20	67/40	68/20	69/10
Master 8	ND_R	1/50	4/50	85/10	86/20	87/40	88/20	89/10

Heavy-Traffic Light-Traffic

- 4 D type masters, 2 D_R type masters and 2 ND_R type masters in the simulation system
- Half of masters are heavy-traffic

Performance Comparisons (1/2)

- Fail cases of different arbitration algorithms
  - 100 random required-bandwidth combinations for each workload
  - 102400 simulation cycles for each combination

Workload (%)	Fixed Priority	Lottery	TDMA + Lottery	RT _lottery	RB _lottery
60	100	100	95	0	0
65	100	100	98	0	0
70	100	100	100	0	0
75	100	100	100	10	0
80	100	100	100	18	0
85	100	100	100	37	1
90	100	100	100	55	12
95	100	100	100	74	44

Performance Comparisons (2/2)

- Hardware comparisons

	Fixed Priority	Lottery	TDMA + Lottery	RT _lottery	RB _lottery
Gate counts	215	4296	4917	5134	5814

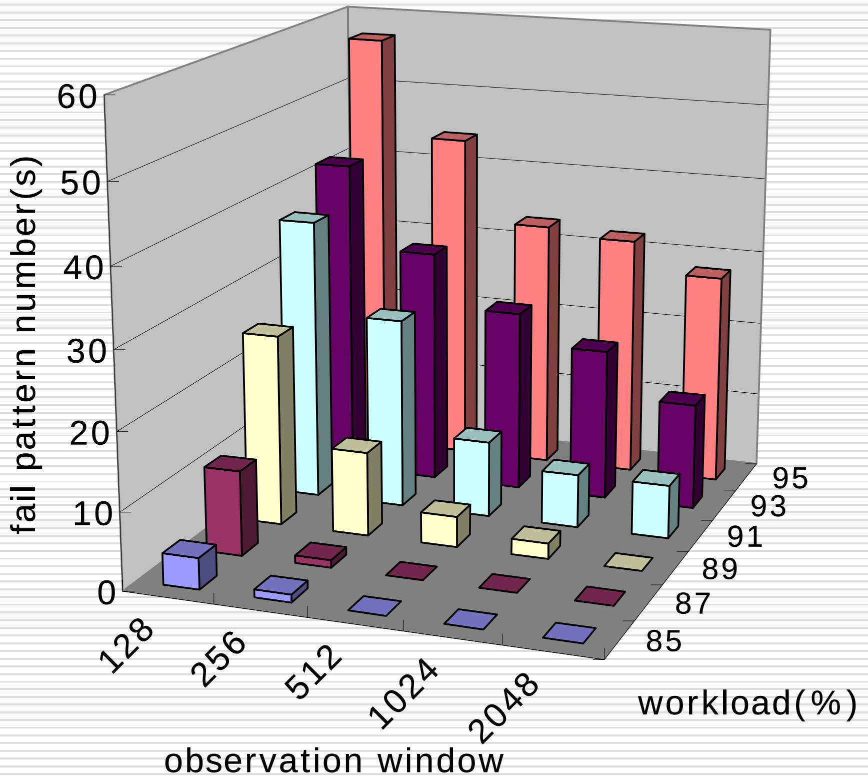
- Summary

	real-time capability	bandwidth capability
fixed priority	no consideration	poor
Lottery	no consideration	good but weight tuning is required
TDMA+Lottery	no guarantee	good only in low loaded bus (workload < 60%)
RT_lottery	always hold	good but still fails in high loaded bus (workload < 75%)
RB_lottery	always hold	good even in extremely high loaded bus

Observation Window Comparisons

- Fail cases in different size of observation window of RB_lottery
  - 100 random required-bandwidth combinations for each workload
  - 102400 simulation cycles for each combination
  - the size of observation window ranges from 128 to 2048

Workload (%)	The size of observation window				
	128	256	512	1024	2048
85	4	1	0	0	0
87	11	1	0	0	0
89	25	11	4	2	0
91	37	25	10	7	7
93	42	31	24	20	14
95	57	44	33	32	28



Conclusions

- RB_lottery is proposed to provide
  - hard real-time guarantee
  - fine-grained bandwidth control

- The observation window in the bandwidth regulator
  - the larger size of observation window, the better controllability over bandwidth requirements

Thank you!