Chapter 3

Protocol Identification and Implementation

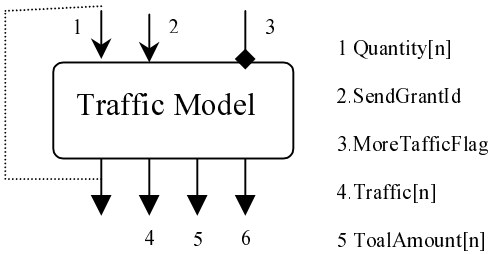
After defining the building blocks in chapter 2, we will demonstrate their feasibility in composing of MAC protocols by examples. A simulation test-bed is created using Visual C++, which allows users to choose among these building blocks to construct the protocols they desire to simulate. The necessary performance indices, such as delay, throughput and packet loss probability, can therefore obtain from this testbed.

3.1 Protocol Identification

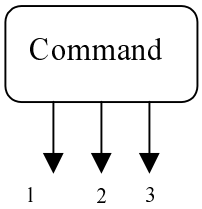
We will take CPR, UniLINK, PRMA and CSMA/CA as examples to validate our Building Blocks.

3.1.1 CPR

Traffic Model & Command : The frame structure of CPR is composed of CMS, DMS, Data. We therefore require minislot and "more-traffic request" mechanism. We set TrafficModel and Command Block as shown below. All I/O ports are valid.



1 Quantity[n]
2.SendGrantId
3.MoreTafficFlag
4.Traffic[n]
5 ToalAmount[n]
6 MoreTraffic[n]



1. ContentionFlag
2. TransmitFlag
3. MoreTrafficFlag

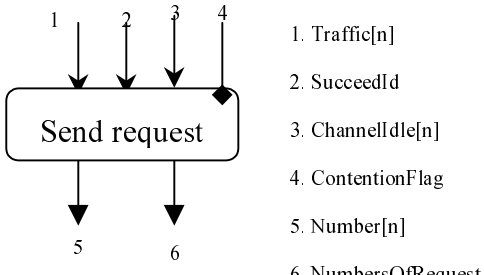
Command generates ContentionFlag, TransmitFlag and MoreTrafficFlag in turns. Here, we assume that if there are pending traffic in the channel, Traffic model will not generates any traffic for each station. However, when MoreTrafficFlag is true, Traffic Model will check SendGrantId to see if it generates more traffic or not (cf. the table below).

Frame structure	Options		No.[1]	Example / Notes
Fix frame structure	CMS	1	2	CPR MLAP
	DMS	1	2	
	Data	10	x	

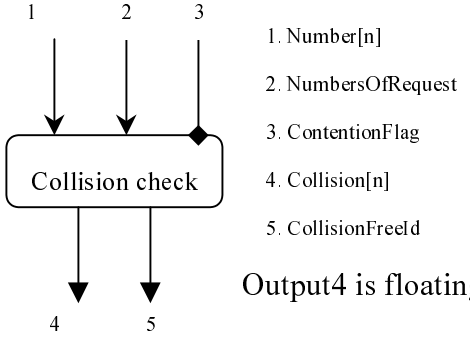
Numbers of CMS, DMS, Data and minislots in one slot can be assigned by users. In this table, "x" means Data slot doesn't have minislot mechanism.

Send Request & Collision Check & ACK & Wait ACK : CPR does not involve

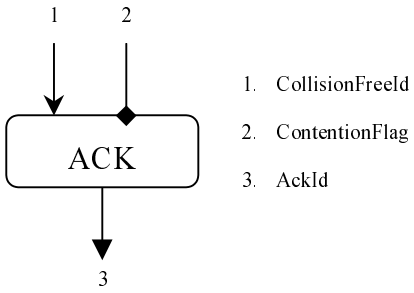
carrier sense, so input 3 of the Send Request block is unnecessary/unused. After Send Request accumulates requests, Collision check will check if there is an collision or not by means of the parameter NumbersOfRequest. In CPR, Collision check is done by the Head-End. Head-End sends an ACK back (or does nothing) to notify the stations if collision happens or not. Stations which have sent their requests will wait for ACK (performed by Wait ACK building block) in order to ensure whether or not the request needs to be re-transmitted. If stations which have sent their requests do not receive ACK, collision resolution algorithm is necessary to resolve the collision. User can choose any collision resolution algorithm provided in Random Back-off block.

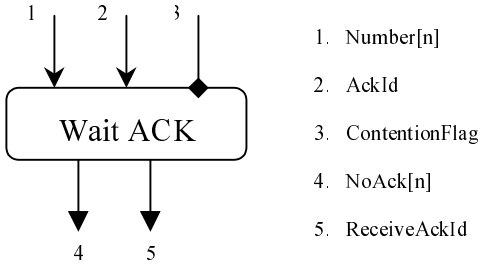


Input 3 is floating

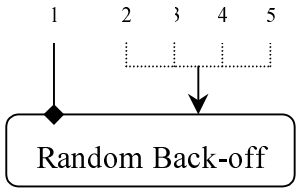


Output4 is floating.





All I/O ports in ACK & Wait ACK are valid.



1. ContentionFlag
2. NoAck[n]
3. Collision[n]
4. ChannelBusy[n]
5. ChannelIdle[n]
6. SucceedId

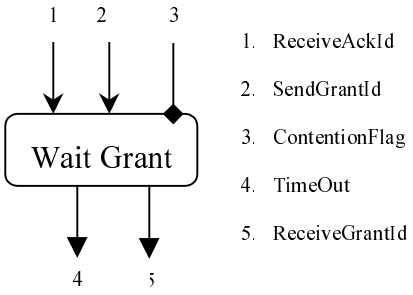
[1] Numbers of minislots in one slot

Option :

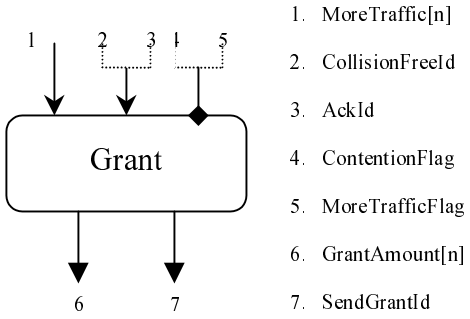
		Parameter
Case I	Dice with probability p (p-persistent algorithm)	p
Case II	Start-n (splitting algorithm)	n
Case III	Assign back-off value	b[0],b[1],b[2],……b[n]
Case IV	Binary exponential	

Although CPR uses p-persistent to solve collisions, users can choose and try other collision resolution algorithm provided in the building block.

Grant & Wait Grant : After sending requests successfully, stations should wait for the head-end's Grant message in order to know when to send their packets. If a request has been sent successfully but no grant is received in time, a TimeOut event will be launched by Wait Grant process. In such case, the packet is counted as loss. Performance block, the packet loss probability will be calcuated.

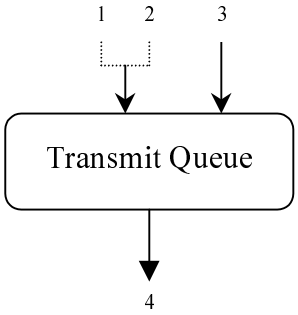


All ports are valid

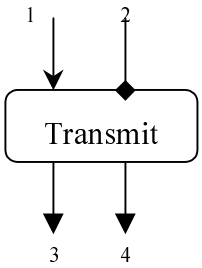


Input 2 is floating

Transmit Queue & Transmit & Performance : These three blocks are in charge of transmitting packets and calculating throughput and, delay and packet loss probability. Input 5 of Performance block will be used when minislot mechanism is applied. It helps to re-arrange time scale to adapt to minislot mechanism.

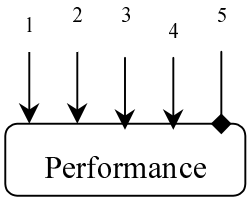


1. ReceiveGrantId
2. SendGrantId
3. GrantAmount[n]
4. Queue



1. Queue
2. TransmitQueue
3. TotalSendAmount
4. EachSendAmount[n]

Input 2 is floating.



1. TimeOut
2. TotalAmount
3. TotalSendAmount
4. EachSendAmount
5. ContentionFlag

All flow chart are presented in Fig 3.1.

3.1.2 UniLINK

Traffic Model & Command : UniLink does not incorporate minislot and "more-traffic request" mechanisms. Command block, again, generates ContentionFlag and TransmitFlag in turns.

Carrier Sense, Send Request, Collision check and **Random Back-off :** UniLINK uses CSMA/CD to resolve channel collision. Carrier Sense is used to sense medium. Collision Check judges if there is collision. When collision occurs, Random-back-off is in the position to resolve the problem. Of course, if there is only one station sending its request, this station can transmit its packet according to the arrangement of Grant Block.

Grant, Transmit Queue, Transmit and **Performance**: Grant Block arranges which station can send its packets, as well as how many packets it can transmit at a time. Transmit Queue, Transmit and Performance do the similar thing in CPR.

The status of the I/O ports in these blocks are illustrated in Figure 3.2.

Fig 3.1

3.1.3 PRMA

Traffic Mode & Command : PRMA does not have minislot and "more-traffic request" mechanisms. Command generates ContentionFlag and TransmitFlag randomly.

Send Request, Collision Check and **Random Back-off:** PRMA uses slotted-aloha to resolve the channel collision. Paramters regarding to collision resolution could be chosen flexibly to meet users' requirements.

Grant, Transmit Queue and **Transmit & Performance :** No difference from the UniLink.

The connection status of the I/O ports in these blocks are illustrated in Figure 3.3.

Fig 3.2

3.1.4 CSMA/CA

Traffic Model & Command: Users can choose any case provided by these two building blocks.

Carrier Sense, Send Request, Collision Check and **Random Back-off :** CSMA/CA has two Carrier Sense blocks and four Random Back-off blocks. Each Random Back-off can choose different collision resolution algorithm. The last part is similar to CSMA/CD.

Grant & Transmit Queue & Transmit & Performance: No difference with the previously mentioned ones.

The connection status of the I/O ports in these blocks are illustrted in Figure 3.4.

Fig . 3.3

3.2 Implementation

We implement some of our proposed building blocks in a simulation test-bed using Visual C++ to confirm its feasibility. In our simulation testbed, users select building blocks and connect I/O ports to form their designed protocol, and then use this simulation test-bed to obtain its throughput and delay. In feasibility test, we pick PRMA as an example (cf. fig. 3.5).

In fig.3.5 Command–Option-Dynamic means that ContentionFlag and TransmitFlag are generated randomly.

Next we introduce each property page.

General :

In PRMA, we can choose any option. So far, we only implement p-persistent and fix-n.

Send Request

Fig 3.5

Random Back-off

Tra heck

Fig. 3.4

Grant

Chapter 4

Conclusion and Future Work

We summarize the thesis and present some potential future work.

4.1 Conclusions

In this thesis, we survey a variety of MAC protocols. Through realizing these protocols, we find that they actually compose of common Building Blocks, and can be formed by simply reorganization of these building blocks. We therefore define a set of Building Blocks, including their I/O ports and main functions and processes. We then test its feasibility by some examples. Finally, a light simulation test-bed is constructed.

4.2 Future Work

Due the limit in time, we did not implement all the Building Blocks. In addition, no friendly user-interface is provided. Hence, one apparent future work is to complete the implementation of all building blocks, and provide a friendly user-interface. Some expansion of the building blocks to meet the requirement of more complex MAC protocols is also an interesting future task.