Request Based Channel Access Protocol on Folded Bus Topology

Sushil Kumar
Computer Science Department
Colorado State University
Fort Collins, CO 80523

Anura P. Jayasumana
Electrical Engineering Department
Colorado State University
Fort Collins, CO 80523

Abstract

Multichannel optical networks promise bandwidth in the gigabits/sec domain. This current trend in optical device technology demands a simple yet scalable protocol for future high speed networks. The Request Based Channel Access (RBCA) protocol provides solution based on folded bus topology. RBCA protocol communication consists of two steps, namely, bandwidth request and data transmission. The communication is slot based and supports variable packet lengths. A dedicated monitor node in the network performs all protocol processing and error detection/correction functions, and minimizes all protocol overheads in nodes. The protocol provides fair channel access to all nodes, irrespective of their relative position from the fold of the network. In a multichannel network, the communication of each channel is independent of all other channels. Thus multichannel networks can be implemented as better optical communication devices, supporting large number of parallel channels, become available.

Key-Words: Optical Networks, Network Protocols, High Speed Networks, Folded Bus

1 Introduction

Optical networks promise bandwidths in the order of hundreds of Gigabits/sec and above. The limiting factor at this time is the available optical devices and their cost. As new optical communication devices become available new protocols will emerge, making use of the new devices. Access protocols for optical networks depend a lot on the network topology. Generalized network protocols defined for any network topology has its associated overheads and are not suitable for high speed communications [5] [6].

Folded bus topology is one of the topologies which is suitable for high speed communication using optical networks. It eliminates the need for intermediate packet buffering and retransmission, like in FDDI, and allows for a very easy configuration with reduced overheads. The protocols that have been proposed for folded bus topology include the S/S++ protocol [1] and RIB [2] [3]. The main challenge using the folded bus topology for protocol designs is ensuring fairness among all nodes in the network. Protocols need to ensure that transmission from a node does not collide with transmissions from other upstream nodes. In S++ [1] and RIB [2] [3] protocols the nodes farthest away from the fold of the network have an advantage over the nodes closer to the fold in terms of channel access. This inherent priority due to the physical location of the nodes is not desired and needs to be eliminated.

The Request based channel access protocol (RBCAP), presented here, is designed after a careful analysis of S++ protocol. Some of the S++ features which may be hard to implement are listed below to understand the problems associated with folded bus network protocol design:

- Channel Access is controlled by free channel detection which is difficult as the nodes are arranged in a serial order on the fiber. Each consecutive node in the packet direction needs to detect one more free slot to make sure that there are no more packets following the free slot.

- Clock Synchronization is difficult as there is almost half fiber length between the write and the read channel.

- Priority of the nodes depend on the physical location of the node in the network.

- Folded Bus Architecture is based on having receivers and transmitters on two separate sections of the fiber. This means that if a free slot detection is required in the WRITE bus of the fiber, there is an inherent requirement of a sensor on the WRITE bus. S++ protocol is based on detection of a free slot when it passes the WRITE bus.

- S++ Protocol detects a single bit, in real time, from each slot header and processes it to mark out some flags in it's state table and then utilizes the rest of the slot to send or not send a packet. When packets are transmitted at higher speed (greater

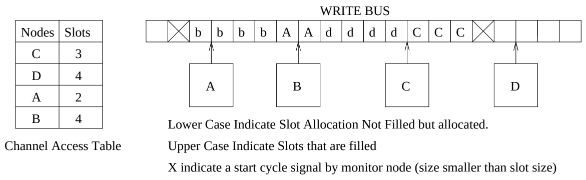

Figure 1: Network Access in Folded Bus

than 1Gbps) however, detecting and processing a single bit from a stream is very difficult.

These problems were carefully looked into while designing the RBCA protocol described below. A more practical and realistic approach was chosen in the design. For more information refer to [4]. Section 2 describes the protocol, Section 3 describes the control flow at different nodes and simulation based performance results are given in section 4.

2 Protocol Description

This protocol is based on channel requests. On a multiple WDM channel configuration, requests for all channels can be send on any channel and the network behavior will be same as a single channel protocol, only the performance and the average wait time for each packet will decrease. Let us first consider the protocol for a single channel and later discuss how this can be extended easily to multiple channel configurations.

2.1 Protocol Features

Some of the features of the protocol include:

1. *Round Robin Channel Access:* Each node gets to access channel once in the round robin cycle. If there are n nodes in the system then each node will access channel after the other $n-1$ nodes have completed their access time. A single channel access by all nodes is called a cycle and cycles repeat one after the other. See Fig. 1.

2. *Channel Access Time Request:* Each time a node accesses a channel it sends request for the next cycle. This next cycle request might not be the cycle immediately following the current one. It depends on how many cycles are currently pipelined on the network.

3. *Variable Packet Lengths (In Slots):* Each time a node accesses the channel in a cycle the node does the following things: It sends the amount of data it had requested for in the past cycles, It sends request for channel for the future cycle, else it just sends a zero slot request which means that the node has zero request. If the node requested a zero slot for the current cycle in the past then it gets a free slot for its use. A free slot is a much smaller than normal slots size and has space enough to send its request for a future cycle.

4. *Request and Send Steps:* Each packet is variable in size and it will need a different number of slots to send it. After a packet arrives at a node for transmission the node sends a request for channel access and the request for the number of slots for that packet and then waits for the actual cycle to start and sends it's packet when the access time comes. So there is always a wait associated with the packet after it arrives a node. This wait can be as small as the total transmission time of the free slot around the fiber, and very large if the total load in the network exceeds the network capacity. The idea in this protocol was to ensure a good packet wait time at loads close to the maximum network capacity on the higher end and improved performance in terms of packet wait time for packets when loads remain much below the total bandwidth supported by the system.

5. *Monitor Node:* The node farthest from the fold in the network acts as a monitor node of the network. This node could be an independent monitor node just catering to process the protocol functions or it could be another node in the network taking care of the network protocol functions. The monitor node in this protocol is an important protocol processing station. It receives all the node requests and allocates the channels depending the request. After the completion of each cycle it starts a new cycle by sending a new cycle start signal on the channel. This is what all nodes look for before a waiting packet is send on the network. Some of the main function of the monitor node are:

   - **Initialization Phase:** The monitor node sends a series of *JOIN_NETWORK* signal. Each signal is followed by two slots. The first slot is used to fill in to indicate that this signal slot is filled and the next slot is used to signal the actual data in the network initialization.
   - **Cycle Management:** The monitor node controls the start of every cycle by sending the *START_CYCLE* signal. This signal is received by each node in the network for synchronization to wait and send packets when the access time for the node comes.
   - **Re-Initialization Phase:** The monitor node sends *JOIN_ACTIVE_NETWORK* sig-

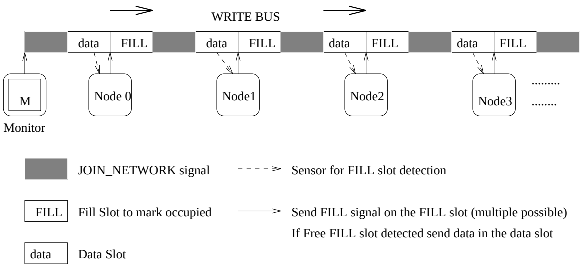

Figure 2: Dual Slot Signal Mechanism for JOIN_NETWORK Phase

nals in the network at the end of cycle (not all cycles) to enable new nodes in the network to join the network. This is done at regular intervals which depends on the number of nodes that were inactive in the network in the last cycles.

This mechanism of detecting and responding to a signal going on the write bus of the network is used in S++ protocol too. However in [1], the time required for nodes down the write bus to detect a free slot is not accounted for. This means that if a free slot needs to be detected and the detection time for a node is d' then for the nth node, down the write bus, to detect a free slot has to wait at least $d' \times n$ before it can detect a free slot as the upstream nodes might have occupied a free slot before this node. This aspect of the free slot sensing is ignored in S++ and is important when number of nodes in the network is large as it might cause delays which might be critical to the network throughput.

To overcome this problem, we use a **Dual Signal Slot Mechanism For Free Slot Detection:** Consider Fig. 3 which shows the transmitter end of any node in the network.

The transmitter at each node consists of two access points to the write bus. The *Signal Sensor Unit* senses the signals on the write bus of the network. Depending on the signal sensed on the network it sends a signal to the *Tx Unit* to send packets on the network. One extra element of control is the *Delay Unit* which delays the signal from the *Signal Sensor Unit* to the *Tx Unit*. Let us consider the two situations for which this system works.

– **Data Transmission:** When a node needs to wait for, lets say 3 slots after the *START_CYCLE* signal to send two slots

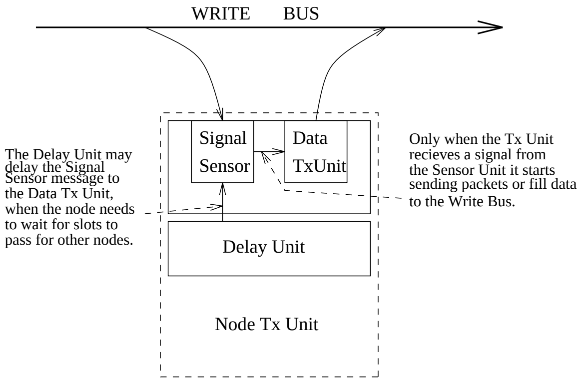

Figure 3: Node Transmitter Block Diagram

of data. The *Signal Sensor Unit* senses the *START_CYCLE* on the network and prepare to send its message to the *Tx Unit*. The delay unit knows that it needs to wait for three data slots to pass before sending the packet and it does so by delaying the signal to the *Tx Unit* by three slots. After three slots are passed, after the passing of the *START_CYCLE* signal on the write bus of the node, the node transmits the data waiting to be send.

– **Protocol Processing:** In almost all protocol processing phase the node needs to detect a different type of signal on the write bus and then send or not send its reply. Let us consider a situation when a node is waiting to join an already active network. The node actually waits for a *JOIN_NETWORK* signal from the monitor node and then start sending *Fill Bytes* to fill the FILL Slot. The *Signal Sensor Unit* senses the *JOIN_NETWORK* signal on the network and immediately configures the *Tx Unit* to start sending *Fill Bytes* to fill the FILL Slot. By the time the *Tx Unit* sends *Fill Bytes* to fill all the FILL slot the *Signal Sensor Unit* detects an unoccupied or an already occupied slot. If the slot was unoccupied then the Fill Signal send by the *Tx Unit* reserved this slot for this node and the *Signal Sensor Unit* of the node sends a signal to the *Tx Unit* to send data after the Fill Slot Bytes are send. If the slot was already occupied the *Signal Sensor Unit* does not allow the *Tx Unit* to send any data in the Data Slot. See Fig. 2.

The important thing to note here is, that the length of the slot is variable without affecting the working of the protocol. The size of the FILL slot is independent of the size of the data slot. Let us assume that the time taken to detect and send a message to the *Tx Unit* is d', then the size of the FILL slot can be configured (in hardware) to make sure that the delay of the circuit is covered in the transmission time of the slot. For example if the smallest time to detect and send a signal is d' then the minimum size of the slot needs to be $d' * data\_rate$ of the channel. This is the reason why the timing considerations of the FILL slot detection for synchronization with the Data slot is not an important issue.

The other good point about the dual slot mechanism is that it is used only for protocol related action which is already minimized in this protocol so the overheads caused by this is negligible.

- **Inter-Cycle Patch Up:** Whenever a request for the next cycle of all the nodes do not reach the monitor node. The monitor node patches up the network transmission with empty time. It waits for all next cycle request to arrive before sending the start cycle signal.

6. *Reduced Protocol Processing Overheads:* The monitor node does all protocol processing related to channel allocation and network configuration and reconfiguration and sends the results to all the nodes. This reduces overheads at individual nodes.

7. *Load Sharing:* All nodes are treated equal in terms of channel access and their position from the fold does not change their access rights. The simple argument to support will come from comparing the RBCAP with other protocols that face this problem. Let us consider the S++ [1] or the RIB [2] [3] protocols which face the problem of load sharing and are both based on the folded bus topology. The main reason why the nodes farther from the fold have an advantage over the nodes low down in the network is because *START_CYCLE* signal reaches nodes away from the fold before it reaches the nodes downstream. This, with the fact that there is no way for the nodes downstream to know how much data the nodes upstream are sending makes it difficult for nodes downstream to send its packet with equal priority. In RIB the nodes downstream have to wait for all the nodes to send their packets before the nodes gets channel access. So

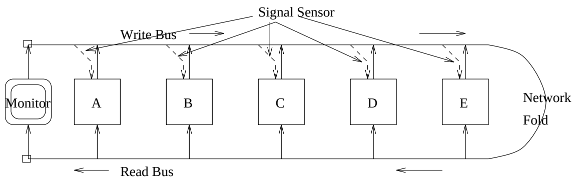

Figure 4: Folded Bus Network

the end node (closest to the fold), has to wait for all the other nodes upstream. Thus channel access is not favorable for nodes downstream.

In RBCA protocol, the problem is dealt with by making sure that all nodes know the data requirement of all the other nodes in the network. There is no channel detection required to detect free slots which makes the protocol implementation simple. As all nodes exactly know how much data all the other nodes are sending, the wait time for a node in each cycle is known even before the data is send, this allows the nodes to reconfigure and change their data requirement dynamically for next cycles. Also the start of a new cycle is now dependent on the data requirement of all the nodes in the network. Channel access is controlled by the *START_CYCLE* signal from the node and this happens after all the nodes in the network get a chance to send once. This means that every cycle each nodes makes one transmission and the transmission time of all nodes are known to all the other nodes. This is the way RBCAP achieves fairness in terms of channel access for all the nodes in the network.

2.2 Protocol Steps

This protocol has two major phases. The *Initialization Phase* and the *Steady State Phase* of the network. The *Network Reset and Re-Initialization Phase* is used in case an error occurs and a node needs to be dropped from the network or a new node wants to join an already active network.

1. **Initialization Phase** The *Initialization phase* sets up the network and fills the network with cycles so that all nodes can send their first few requests. This is done when the network starts. Let us consider the network in Fig. 4 for the Initialization step. Here we consider a monitor node as the node furthest from the fold and assume that it is dedicated for monitoring purpose.

   - **START_NETWORK:** Monitor Node sends a *START_NETWORK* packet to inform all nodes that the network is being initialized. It waits for this packet to reach the end of the network. By this time all nodes waiting to join the network will know that the initialization phase has started.

- **JOIN_NETWORK:** After the *START_NETWORK* packet travels the whole network length and the monitor node receives it. It has to find out how many nodes are waiting to join the network. To do this the monitor node sends *JOIN_NETWORK* header packets one after another with an empty slot between them. When a *JOIN_NETWORK* packet reaches a node it checks if the following slot is still free or was occupied by an upstream node. If the free slot is filled it waits for the next *JOIN_NETWORK* header. The monitor node meanwhile continuously sends *JOIN_NETWORK* packets with free slots in between till it encounters a free slot that was not occupied by any node.

  This phase serves two purpose, it informs all the other nodes about the number of nodes in the network and their respective positions.

- **JOIN_PHASE_END:**
  After the monitor node receives a free slot in the *JOIN_NETWORK* phase. It stops sending any more *JOIN_NETWORK* headers and waits for all the headers to arrive. After it receives all the *JOIN_NETWORK* headers without any errors it broadcasts a *JOIN_PHASE_END* packet. This packet contains a list of nodes and their addresses in the order from the monitor to the fold.

- **INITIALIZATION_CHECK:** The monitor node then sends an *INITIALIZATION_CHECK* packet header and follows it with N free slots for all the nodes to fill. All nodes fill their entry slots and after this phase is completed without any errors the next phase starts. In case of any errors all steps are done again.

- **FILL_CYCLE:** In this phase the monitor node sends a *START_CYCLE_REQUEST* header and follows it with N empty *START_CYCLE_REQUEST_SLOT*. These slots are used by the nodes to send its first n requests, where n is the number of cycles in pipeline. After all the nodes finish their first n requests the monitor node knows how much each node require, in terms of number of slots for the first few cycles. It then starts the next phase where actual Data Transfer takes place.

- **START_CYCLE** header completes the Initialization phase of the network.

2. **Bandwidth Request and Data Transfer Phase** In this phase all nodes send requests and packets once the appropriate request is granted. All nodes know exactly how much time it had to wait for before sending its packet after a *START_CYCLE* header is detected on the write bus. This detection is done at each node at the write bus to enable the nodes to know when a cycle was started. Let us consider an example here:

   Node A has requested for 3 slots, Node B has requested for 4 slots and Node C has requested for 2 slots for some cycle. When this cycle is started by the monitor node by sending a *START_CYCLE* header. The following happens at each node:

   - Monitor node sends the *START_CYCLE* header.
   - Node A sends its 3 slot length packet immediately after the *START_CYCLE* header.
   - Node B waits for 3 time slots after the *START_CYCLE* header and then sends its 4 slot packet.
   - Node C waits for $3 + 4 = 7$ slots after the *START_CYCLE* header and then sends its 2 slot packet.

3. **Network Re-Configuration Phase** During the *Bandwidth Request and Data Transfer Phase* of the network all cycles start when the *START_CYCLE* packet is send by the monitor node. The monitor node keeps track of all the active nodes in the network. If a node stops responding in the network the monitor node can detect it and it starts a phase to reconfigure the network. The monitor node also start a phase which allows new nodes to join the network. Let us consider some conditions when network re-initialization becomes inevitable and how it is handled in this protocol.

   - **RESTART_NETWORK:** This phase is used when the monitor node detects a node from the active node list which has stopped responding. In this case a *CHECK_ACTIVE* header is send after the end of a data transfer cycle. This header is directed towards only one node who follows the header with its status (if it is active). If the monitor receives a active signal following the *CHECK_ACTIVE* header the network continues without any change.

In case the node does not respond the monitor nodes verifies the node's inactivity in the next data transmission cycle and then does the following to eliminate the inactive nodes from the active list.

- Wait for the current cycle to end and repeat the *CHECK_ACTIVE* cycle.
- Send a copy of the *ACTIVE_NODE_TABLE* without the inactive node to all the nodes in the network. This is a broadcast message to all the nodes. This message also contains a *START_CYCLE_ID* which is not the same as the old *START_CYCLE_ID*. Only nodes in the new *ACTIVE_NODE_TABLE* continue to take part in communication after this is send. This ID is also embedded in the *START_CYCLE* signal which the node reads and verifies to see if it is still included in the active node list.
- The eliminated node does not take part in the communication when it sees a *START_CYCLE* signal with a different ID than before.

• **JOIN_ACTIVE_NETWORK:** This network phase allows new nodes to join the network. The monitor node might decide to send a *JOIN_ACTIVE_NETWORK* header after the completion of a cycle and follow it by a dual signal slot described above. When a node waiting to join the network senses this header it sends its request immediately after this header. The monitor node waits for the *JOIN_ACTIVE_NETWORK* and the following free slot to come back on the read bus and then decides to include any node which responded to the *JOIN_ACTIVE_NETWORK* signal.

It is the monitor node which decides how many *JOIN_ACTIVE_NETWORK* signal and dual slots it sends during that time. If there are a lot of nodes inactive in the network the monitor node might send more *JOIN_ACTIVE_NETWORK* signals and dual signal slots for more nodes to join together. Once this phase is done the monitor node waits for all the *JOIN_ACTIVE_NETWORK* signal and dual signal channel data to arrive and then prepare the new channel access table with the new set of nodes and post it to all the nodes with a single broadcast signal.

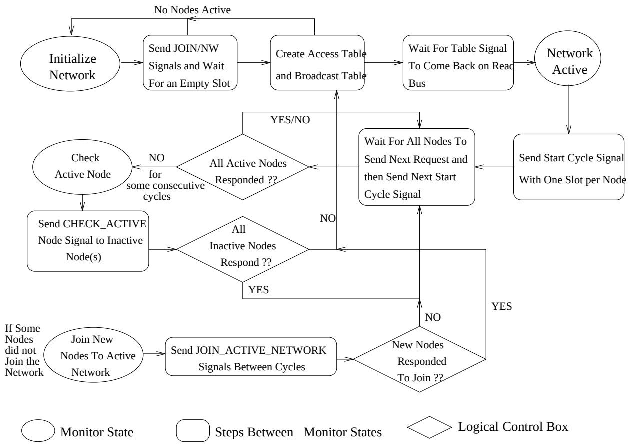

Figure 5: Control Flow in a Monitor Node

It is important to realize that for a high speed network protocol, reset or re-initialization of network should be a very small overhead to the entire system. As most of the protocol processing is done by the monitor node and copies of the result send to all the nodes the individual nodes do not get overloaded by unnecessary protocol processing. Also, the maximum delay due to this network re-initialization phase is a little more than the network delay time, which is very good for high speed networks. One controlling factor is the monitor node, which can decide when a particular re-initialization phase needs to be started to keep the overheads to the minimum.

3 Control Flow in RBCA Protocol

In this section, the flow chart for each entity in the RBCA protocol is described in detail. The main idea of this section is to introduce state diagrams with control flow that are used in the simulator design and implementation.

3.1 Monitor Node

The monitor node is the most important node in the network. The introduction of a dedicated node to monitor the network functionalities has led to a reduction of protocol overheads in the network. The monitor node is designed in such a way that its functionalities can be taken over by the first node in the network. Fig. 5 below shows the control flow diagram of the monitor node.

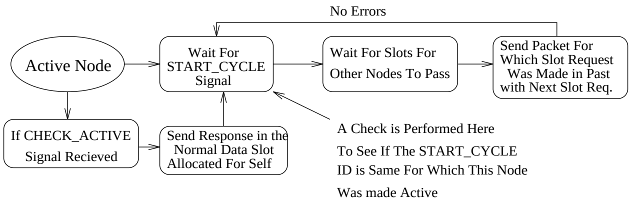

Figure 6: Control Flow of An Active Node

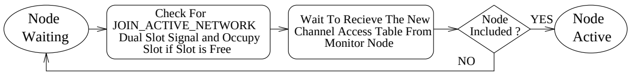

Figure 7: Control Flow of a Waiting Node

3.2 Active Node

A node is considered active in the system when the node is performing data transmission normally and it takes part in each cycle of the data transmission. The flow chart for an active node is given in Fig. 6 below.

3.3 Waiting Node

When the network is active, the set of active nodes might not include all the nodes that are physically connected to the network. A waiting node is a node that is waiting to join the active network. The node waits for the monitor node to send a *JOIN_ACTIVE_NETWORK* signal on the network before the joining process can be completed. The diagram in Fig. 7 shows the flow chart for a waiting node in the network.

4 Simulation Results

The RBCA Protocol was simulated by an event driven simulator. Simulation was done to compare the protocol performance with other similar protocols namely S++ and DQDB. The results of S++ and DQDB protocols are from [1].

A single channel folded bus topology was assumed with nodes sitting at equal distance from each other. The length of the network comprised of a read and write bus. A pseudo monitor node was assumed to be present in the system. Packet arrival at each node had a Poisson distribution similar to that used in S++ and DQDB [1]. System load was changed by changing the mean value of the packet inter arrival time. If we assume that the mean inter arrival time between packets at each node is given by I and other network parameters are d for data rate on the single channel, N the number of nodes in the system, and n as the average size of packets in bytes, then for a 90 percent load in the network we will have I defined by the following formula:

$$0.90 * d = \frac{N * 8 * n}{I}$$

and hence I was given by the formula:

$$I = \frac{N * 8 * n}{l * d}$$

where l indicated the percentage load to be used in the system.

All results are normalized to a slot time because the aim is to investigate the relative merits of the protocols being compared and not any implementation details of a specific protocol. Like in [1] for S++ protocol, a 20 node network with equal spacing between nodes, on a network of length 20 slot length was used for most of the simulation, unless otherwise stated. A channel bandwidth of 1 gigabit per second was assumed for the folded bus. The speed of light in the fiber was around $2.6 * 10^8$ m/sec for our simulation purpose.

4.1 Simulation Parameters

Some of the Simulation Parameters that were used are described below to aid the understanding of the simulation results.

- *Network Length:* Length of the complete fiber including both the *Read Bus* and the *Write Bus*.
- *Slot Size:* Size of one data slot. This was given in number of bytes.
- *Packet Wait Time:* The time difference between the arrival of the packet at a node and the time at which the packet transmission started.
- *Network Load:* Load was expressed in percentage. If the network could support 1 Gbps, and all nodes were configured to generate a total of 800 Mbps then the load was considered 80 percent.
- *Multiple Channels:* If the network fiber supported multiple channel transmission, then for each independent channel an independent transmitter and receivers was assumed. All channel communication was independent of each other. This assumption was made for multi-channel RBCAP simulation.

For all simulation run some default parameters were assumed while some parameters were varied. Some of the default values of the simulator parameters are described in the table if Fig. 8.

4.2 Avg. Wait Time vs Network Load

Fig. 9 shows the result of the simulation. *Wait Time* for a packet is the time difference between the arrival of the packet at a node and the final time when the channel was available and the packet transmission began at the node. Average for all transmitted packet gave the *Average Wait Time* of packet at the node. This value was

Parameters	Parameter Values
Number of Nodes	20
Data Rate	1 gbps / channel
Slot Size	1024 bytes (~ 2130 meters)
Light Speed	2.6e+8 meter/second
Packet Size	1024 - 4096 bytes (1-4 slots)
Network Length	20 slot lengths
Number of Channels	1 (single) OR 4 (multiple)

Figure 8: Simulator Constants for RBCA Protocol

normalized to the slot transmission time. Slot transmission time is the time necessary to start and complete transmission of a single slot of data. If a slot length was b bytes long, then the slot time t_s was given by: $t_s = (b * 8)/d$.

Let us now consider the graph in Fig. 9.

A series of simulations were done to compare with the results in [1] for S++ protocol. We see that the RBCA protocol has a lower delay than S++ for all network load. Also the S++ protocol saturates much faster than RBCA protocol. Only when the network load exceeded 90 percent the average wait time increased exponentially. For S++ the limit if somewhere close to 80 percentage.

4.3 Avg. Wait Time and No. of Nodes

The RBCA protocol sends packets in two steps. The first step is to send a request for channel access and then once the request is granted the node sends the packet. The request phase is what makes the protocol very sensitive to the delay caused due to traffic in the network. As the number of nodes increase in the network so does the average waiting time for packets and as the graph in Fig. 10 shows the relation between the two is almost linear as was expected in the protocol.

If we look at the graph in Fig. 10, for all loads in the network the behavior of RBCA protocol is much better than S++ for small number of nodes. Actually any network with 50 to 70 nodes with a 80 percent load will behave better than S++. Also as the protocol overhead is much less than S++ the performance will benefit. RBCA protocol thus can be used to connect a small number of nodes which require very high bandwidth support.

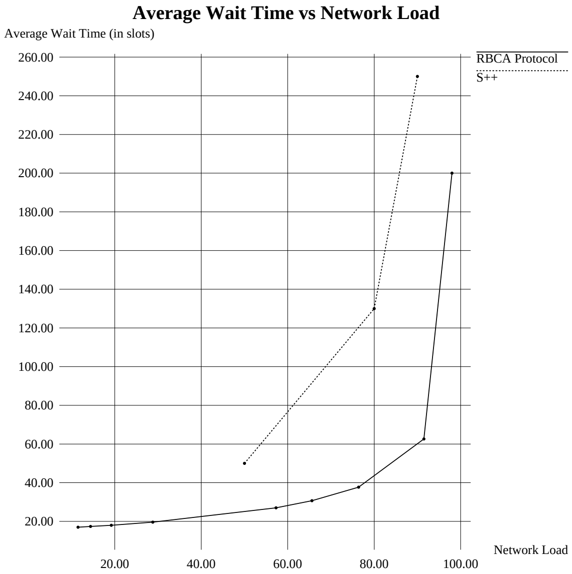

Figure 9: Average Wait Time vs Network Load

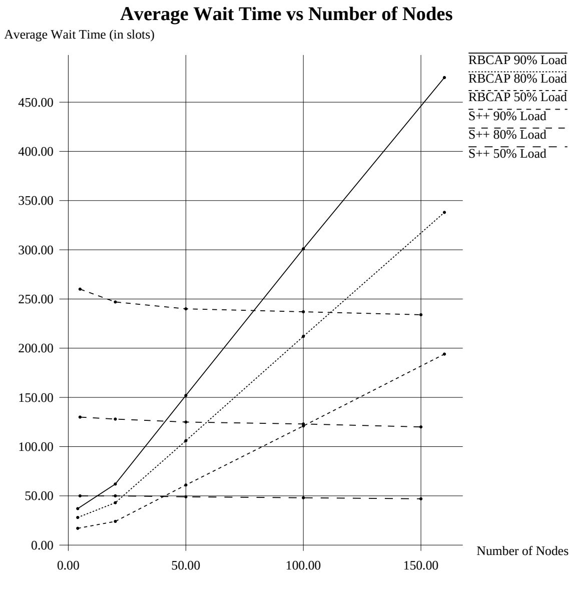

Figure 10: Average Wait Time vs Number of Nodes

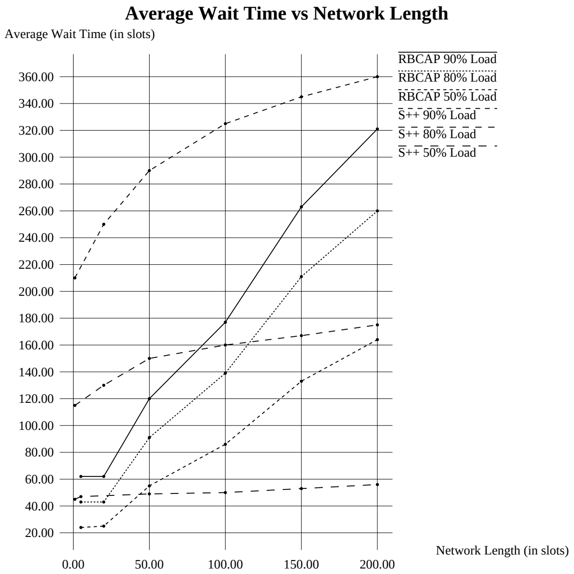

Figure 11: Average Wait Time vs Network Length

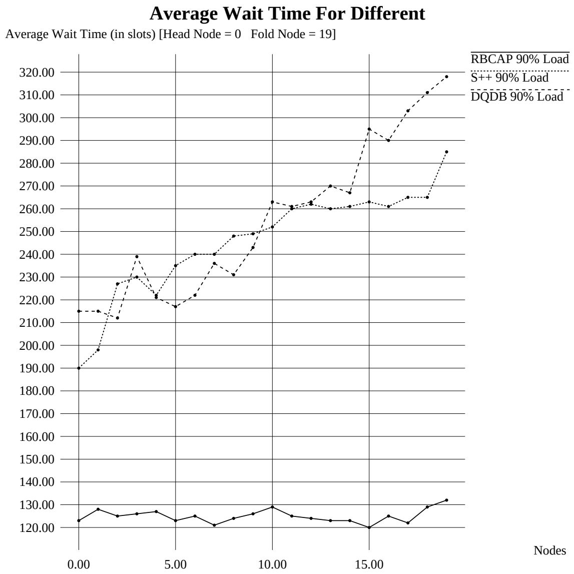

Figure 12: Average Wait Time For Each Node

4.4 Avg. Wait Time vs Network Length

The behavior of the RBCA protocol relies a lot on the network delay to send request for channel access. The longer the network fiber the larger is the delay. But the simplicity of the protocol assures a much better performance than expected. Three simulations were done with different system load and results were compared with S++ protocol. We see that for all network load the RBCA protocol behaves much better than S++ for a small network. As the network length increases the request delay increases and the protocol performance deteriorates. For networks in the range of 50 to 125 slot length long fiber the RBCA promises much better performance.

Even though RBCA protocol behavior deteriorates much faster than S++, for a given load, with network length the performance drop when load increases is not as bad as S++. In the graph, in Fig. 11, the difference between the 90 percent and 80 percent load graphs is much less than the difference for S++ protocol for the two loads. This means that in S++ protocol the average wait time deteriorates much faster than in RBCA.

4.5 Avg. Wait Time For Each Node

RBCA protocol has been designed to provide a much stable network where all nodes have equal access to the network bandwidth irrespective of the position of the node in the network. This is the main achievement of the RBCA protocol. In most of the channel access protocols for folded bus (S++, RIB etc) or dual bus (DQDB) the nodes down the line have a disadvantage over the upstream nodes. This is a feature which is not desired in network protocols. All nodes should be treated fairly and RBCA guarantees this. This feature is the basis of the RBCA protocol.

Fig. 12 shows the average wait time of packets at all the 20 nodes in the network. For RBCA, even though the average time is much smaller compared to both DQDB and S++, all nodes have almost the same wait time irrespective of their positions. For DQDB and S++ as we get closer and closer to the network fold the nodes have an increase in their average wait time.

4.6 RBCA Protocol On Multiple Channels

The main bottleneck of the RBCA protocol is the network delay inherent in every packet transmission. One way to improve this is to find a faster way to send channel requests to the monitor node. This will reduce network delays. If we assume a multiple channel network, and assume that each channel is independent of each other, then each channel can be used to send requests for a few other channels for the next cycle channel access. This was tried and results are shown in Fig. 13. There was a substantial decrease in the average packet wait time for high network load but network with low load did not benefit much from the multi-channel request. Also longer network had better increase in performance than smaller network.

5 Conclusions

The RBCA Protocol was presented. Bandwidth sharing and high throughput, and a low average wait time for small network configurations is what RBCA

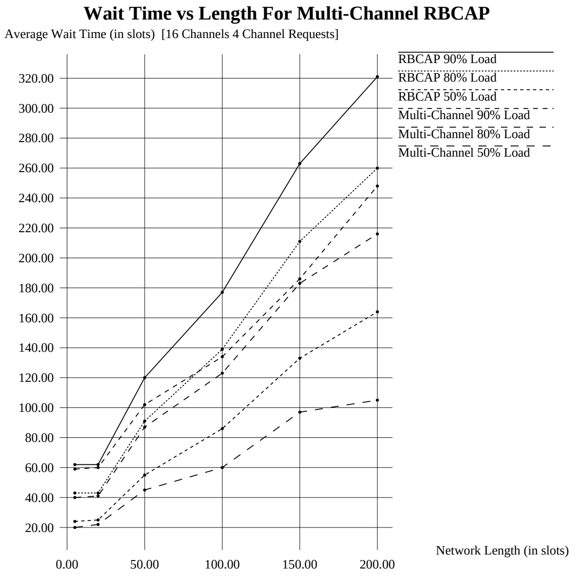

Figure 13: Multi-Channel Requests in RBCA Protocol

Protocol promises. Multiple channels can be used independently by having a single channel RBCA protocol for each channel. As long network and large nodes in the network lowers the performance of this protocol, RBCA can be used for smaller network configurations running for like 50-100 kilometers with 50-100 nodes in the network. The RBCA protocol is also suitable for networks where rapid reconfiguration is not necessary, i.e. networks handling circuit switched type applications because each data transfer is a two step process involving a bandwidth request phase and a data transmission phase.

One other feature which make RBCA protocol very interesting is the fact that on a multiple channel network all nodes need not have access to all the channels and channel communications can be kept independent of each other. Nodes with less bandwidth requirement can attach itself to a smaller number of channels, and nodes with large bandwidth requirement can have access to more channels. This does not affect the network performance at all as long as channels have similar loads.

References

[1] *S++ - A New MAC Protocol for Gb/s Local Area Network*, Greg C. Watson and Samir Tohme, IEEE Journal on Selected Areas of Communications, May 1993.

[2] *Register Insertion Bus Architecture For Fiber Optic Local Area Networks*, J.M. Ott and A. P. Jayasumana, Electronic Letters, Nov 1990.

[3] *Bandwidth Balancing of the Register Insertion Bus (RIB) Fiber Optic Local Area Network*, James M. Ott and Anura P. Jayasumana, Proc. 18th Conf. on Local Computer Networks, Sep 1993.

[4] *High Speed Communication Protocols For Optical Networks*, Sushil Kumar, Master's Thesis, Computer Science Department, Colorado State University, Nov 1994.

[5] *Channel Allocation Protocols in Frequency-Time Controlled High Speed Networks*, Imirich Chlamtac and Aura Ganz, IEEE Transactions on Communications, April 1988.

[6] *Toward Alternative High-Speed Network Concepts: The SWIFT Architecture*, Imirich Chlamtac and Aura Ganz, IEEE Transactions on Communications, April 1990.