LA-CRADA--96-6

Gigabit Network Technology

Final Report

CRADA No. LA92C10052

October 1996

RECEIVED
JAN 17 1997
OSTI

edited by

Carolyn M. C. Davenport

MASTER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Table of Contents

PART I. FINAL CRADA TECHNICAL REPORT	1
A. PARTIES	1
B. PROJECT SCOPE	1
1. Deliverables.....	2
2. Specific Tasks	2
C. TECHNICAL	5
1. Background	5
a) Wavelength Division Multiplexing	6
b) All-Optical Broadcast Networking.....	7
c) HIPPI.....	8
d) Circuit Switching versus Packet Switching.....	9
2. The Rainbow II All-Optical Network	11
3. Media Access Protocol.....	12
4. Microsecond Optical Switching.....	15
a) Structure of FFP-TF.....	15
b) Characterization of high-speed FFP-TF.....	17
5. Future Work.....	19
D. PARTNER CONTRIBUTIONS	20
E. DOCUMENTS/REFERENCE LIST.....	21
F. APPENDIX: THE RAINBOW-II GIGABIT OPTICAL NETWORK*	23
G. APPENDIX: ONA OPERATION AT LOS ALAMOS NATIONAL LABORATORY.....	41
H. APPENDIX: RAINBOW II SOFTWARE OPERATION GUIDE.....	51
I. APPENDIX: MICROSECOND OPTICAL TUNING IN PIEZOELECTRIC FIBER FABRY-PEROT TUNABLE FILTERS	57
J. ACKNOWLEDGMENT	71
PART II. FINAL ABSTRACT (ATTACHMENT I).....	73
PART III. PROJECT ACCOMPLISHMENT SUMMARY (ATTACHMENT II).....	75
PART IV. FINAL QUARTERLY REPORT (ATTACHMENT III).....	77
PART V. CLOSEOUT CHECKLIST	79

* Published in the IEEE Journal on Selected Areas in Communications, Vol. 14, No. 5, June 1996.

List of Figures

FIGURE 1. BLOCK DIAGRAM OF THE RAINBOW-II ALL-OPTICAL NETWORK	7
FIGURE 2. TIME-SPACE DIAGRAM FOR CIRCUIT SWITCHED AND PACKET SWITCHED SINGLE-HOP NETWORKS.	9
FIGURE 3. DIAGRAM OF MEDIA ACCESS PROTOCOL.	12
FIGURE 4. FIBER FABRY-PEROT TUNABLE FILTER.....	16
FIGURE 5. FABRY-PEROT TUNABLE FILTER DEFINITIONS AND FORMULAS.....	17
FIGURE 6. THE SCANNING PROCESS. THE SQUARE-WAVE SCAN AND THE OPTICAL RESPONSE OF THE FILTER ARE SHOWN FOR SINGLE LASER SOURCE: A) 50 μ s, B) 10 μ s, C) 5 μ s RISE AND FALL TIMES, RESPECTIVELY.	58
FIGURE 7. THE SCANNING PROCESS. THE SAWTOOTH SCAN IN SHOWN AT THE BOTTOM AND THE OPTICAL RESPONSE OF THE FILTER IN SHOWN AT THE TOP. TWO LASER INPUTS AT 1518 NM AND 1528 NM ARE USED.	59
FIGURE 8. IMPEDANCE TRANSFER CHARACTERISTICS FOR A) TF1 AND B) TF3	60
FIGURE 9. IMPEDANCE TRANSFER CHARACTERISTICS FOR A) TF5 AND B) TF6	61
FIGURE 10. IMPEDANCE TRANSFER CHARACTERISTICS FOR A) TF7 AND B) TF8	62
FIGURE 11. (A) FASTEST RESPONSE TIME FOR TF5 EXCESS OPTICAL RINGING WITH T_{F0} = 3.9 μ s; (B) FASTEST RESPONSE TIME FOR TF5 WITH T_F = 1 μ s (NOTE THE TWO SETS OF TRACES OF DIFFERENT TIME SCALES).63	
FIGURE 12. DEPENDENCE OF TUNING SPEED ON DRIVING RATES FOR TF1, TF2, TF3, AND TF5.	64
FIGURE 13. OPTICAL RESPONSE FOR TF3: (A) T_F = 5.9 μ s FOR MOI CONTROLLER, AND (B) T_{F0} = 5.9 μ s FOR LANL CONTROLLER.	65
FIGURE 14. FASTEST RESPONSE TIME FOR E-TEK TUNABLE OPTICAL FABRY-PEROT FILTER (TOFF) WITH T_{F0} = 189.8 μ s.....	66
FIGURE 15. EXPERIMENTAL SET-UP FOR TWO NODES USING LANL'S HIGH-SPEED FFP DRIVER.....	67
FIGURE 16. THE SCANNING PROCESS. OPTICAL RESPONSE ACHIEVED FOR TWO REFERENCES AT 1545 NM AND 1680 NM USING LANL'S FFP DRIVER.	67
FIGURE 17. "OFF-ON" OPTICAL SWITCHING. OPTICAL RESPONSE ACHIEVED FOR TF5 AND 1565 NM SOURCE WITHOUT RINGING T_{F0} = 44.0 μ s.....	68
FIGURE 18. "ON-OFF" OPTICAL SWITCHING. OPTICAL RESPONSE ACHIEVED FOR TF5 AND 1565 NM SOURCE WITHOUT RINGING T_{F0} = 13.7 μ s.....	69
FIGURE 19. "ON-OFF" OPTICAL SWITCHING. OPTICAL RESPONSE ACHIEVED FOR TF5 AND 1545 NM SOURCE WITHOUT RINGING T_{F0} = 8.4 μ s.....	69

List of Tables

TABLE 1. CRADA PROJECT MILESTONES INCLUDING MILESTONES LEADING UP TO IBM/ LANL COLLABORATION.	5
TABLE 2. HIGH-SPEED FFP-TF EXPERIMENTAL RESULTS	18

Gigabit Network Technology

Final Report

CRADA No. LA92C10052

Part I. Final CRADA Technical Report

A. Parties

The project is a relationship between Los Alamos National Laboratory (LANL) AND IBM/TJWatson.

Los Alamos National Laboratory
PO Box 1663
Los Alamos, New Mexico 87544

IBM Research Division
T. J. Watson Research Center
PO Box 704
Yorktown Heights, New York 10598

B. Project Scope

Current digital networks are evolving toward distributed multimedia with a wide variety of applications with individual data rates ranging from kb/sec (e.g., file transfer, telephony) to tens and hundreds of Mb/sec (e.g., HDTV, full motion uncompressed video). Link speed requirements are pushing into the Gb/sec range (e.g., campus backbones, supercomputer interconnections, and interconnections of high-speed LAN's) and beyond the envelop of electronic networking capabilities.

There is a vast amount of untapped bandwidth (about 24×10^{12} hertz) available in the low-attenuation communication bands (i.e., in the $1.3 \mu\text{m}$ and $1.5 \mu\text{m}$ optical transmission bands) of an optical fiber. The capacity in one fiber thread is enough to carry more than two thousand times as much information as all the current radio and microwave frequencies. And while fiber optics has replaced copper wire as the transmission medium of choice, the communication capacity of conventional fiber optic networks is ultimately limited by electronic processing speeds.

All-optical networks based on wavelength division multiplexing (WDM) can overcome this electronic bottleneck and exploit the very large bandwidth available in optical fiber

to provide connectivity across large distances at high bit rates per node with considerable flexibility of topology and protocol. Data traveling at the speed of light over fiber optics can support aggregate data transfer significantly higher than gigabit per sec.

The focus of this collaboration between IBM and Los Alamos was to move toward next-generation all-optical networking (i.e., gigabit per second transmission between network nodes). A primary goal is to meet the high-speed demands of scientific, medical, academic and research facilities that work with gigabit data throughput rates. In addition, the availability of such a network would facilitate the development of powerful data-handling applications, such as those dealing with multimedia, full-motion video and supercomputer imaging. Specifically, this was accomplished by developing optical switching technology and network hardware and software to implement an all-optical network. The prototype network was successfully demonstrated at the SC'94 Supercomputing Conference and at the Optical Fiber Communications International Conference, OFC'95.

1. Deliverables

The deliverables for this project were as follows, by project phase.

Phase 1 (months 0 through 12):

- 1-1. Optical Network Nodes (ONN) developed by IBM.
- 1-2. Wave division multiplexed optical network components delivered to LANL.

Phase 2 (months 12 through 30):

- 2-1. Experimental results and reports on running various applications over optical network using LANL's massively parallel computers.
- 2-2. Experimental results and reports using other protocols (such as fiber channel).
- 2-3. Final Report

These deliverables were proposed and agreed to based upon the current state of optical networking technology in 1993 and projections regarding this and related technologies and standards. As the project progressed modifications were made, as discussed below. The final report (Deliverable 2-3) is contained herein.

2. Specific Tasks

The network that was developed under this CRADA is based upon the Rainbow-I network originated at IBM. During Phase 1, Rainbow-I network technology was modified and extended. For example, the photonics was upgraded from a 0.3 Gb/s design to a 1.0 Gb/s design. In addition, a commercial High Performance Parallel Interface (HIPPI) chipset was integrated to allow HIPPI connection to the host computers. The Rainbow-II network is implemented in the form of Optical Network Nodes (ONNs). Per Deliverable 1-1, these nodes were developed by IBM during phase 1; however, the ONNs were not delivered to LANL by the end of phase 1 as dictated by Deliverable 1-2. Delivery was delayed because design and debugging of the ONNs took longer than had been projected. In addition, once the units were built, delivery

was further delayed so that the network could be demonstrated at two technical conferences that occurred in 1994 and 1995. Three ONNs were delivered in 1995.

At Los Alamos the first year effort was concentrated on modifying the testbed to utilize the circuit switched Rainbow network along with the ONNs. This involved design interaction between Los Alamos and IBM to insure that the ONNs would integrate successfully into the testbed. Los Alamos procured the first phase of individual components for the network that were sent to IBM for assembly and check out. In addition, Los Alamos developed expertise in the single-mode fiber technology on which Rainbow is based. It was anticipated that the transport as well as media-access protocols used in such all-optical networks, whether circuit-switched or packet-switched, would require unique characteristics; therefore Los Alamos participated in the development of those protocols. The protocol development encompassed the full two year effort.

Offload protocol studies were performed for TCP implemented on the Los Alamos HIPPI crossbar interface (CBI). A joint conclusion was drawn that the desired very slim transport protocol should be based on TCP/IP rather than on IBM's RTP or some new direction because of the wide implementation and acceptance of TCP. Implementation on the RISC processor was originally planned but was not begun due to time constraints, limited resources, and a realization that a sufficiently fast optical tuning method did not exist to take advantage of such an implementation. For this reason, work was initiated to develop a Fabry-Perot tunable filter that could be tuned on the order of a few microseconds.

It had been anticipated that beginning at the end of the first year, experiments in interconnecting some Los Alamos HIPPI resources would be performed and comparisons made between the all-optical approach and the more classical electrical one (i.e., standard commercial fiber distance extenders together with crossbar switches). However, due to the delayed delivery of the ONNs, these experiments were performed during 1995. These results (Deliverable 2-1) were published and are included as an appendix (Section F).

Regarding Deliverable 2-2, Los Alamos began media-access (MAC) protocol development in cooperation with IBM, and the following progress was made.

- Los Alamos identified six appropriate MAC algorithms to be tested.
- Computer simulations of each MAC protocol program were performed to determine overall throughput, speed of transfer, and comparative algorithm complexity.

However, due to resource reallocation at IBM, MAC protocol development was not completed. The results obtained are discussed in Section 3.

The replacement of HIPPI by Fiber Channel (FC) had been anticipated since it was believed that the FC Standard would have stabilized and that reliable vendor chipsets would be on the market. This step would have necessitated a replacement of one of the two daughter cards and possibly a complete rework of the mother board. However, IBM decided not to support Fiber Channel in the ONNs. Cost and scheduling were contributing factors in this decision.

One significant challenge for this project was how to get the data between the host computer and the protocol offload hardware in the ONN. To solve this problem, the Simple Host Intersocket Protocol (SHIP) was developed. SHIP defines a simple means of passing blocks of data between a local host and an off-board protocol processor, in this case a Crossbar Interface (CBI), which would then provide the TCP/IP processing on behalf of the host. SHIP provides a more cost effective solution for providing TCP/IP processing than what many expensive supercomputers can achieve with their native implementations. This protocol is discussed in detail in the appended journal article (Section F) and in Reference 3 listed in Section E.

Los Alamos was to continue to procure components for additional ONNs for assembly and checkout at IBM. The number of ONNs was to be significantly increased to encompass most of the HIPPI nodes on the Los Alamos HIPPI Testbed. However, the unit cost limited production to the three original ONNs. It was also determined that three ONNs would suffice for demonstrating network switching capabilities in such a system. This eliminated the high cost (~\$500K) required to build ten more ONNs. These funds were used instead to support further research in optical components that will be used in gigabit optical networks.

C. Technical

In this section, technical background for some of the major components of the project is provided, as well as a brief overview of the Rainbow-II all-optical network. In Addition, the work that was begun on the media access protocol is described, and the development of a microsecond piezoelectric fiber Fabry-Perot tunable filter is discussed. Deviations from original goals/objectives are included with the rationale for changes. Finally, a discussion of future work is included.

1. Background

A summary of the project history and milestones is given in Table 1. In the following subsections brief background information will be provided on wavelength division multiplexing, optical switching, broadcast networking, and circuit versus packet switched networks. This background information will be followed by further information, discussion, and the results achieved in the areas of fast optical switching, media access protocols, and the Rainbow-II All-Optical Network.

Summary of Project History and Milestones

1987	LANL begins development of new gigabit/sec network technology: HIPPI.
1989	LANL develops first HIPPI crossbar switch.
1990	First commercial HIPPI switch. LANL begins development of HIPPI network adapter.
1991	IBM approaches LANL for collaboration to integrate HIPPI technology into all-optical networks. HIPPI becomes ANSI standard.
1992	Technology Commercialization Initiative (TCI) in DOE approves four lab program in "Gigabit Network Technology." Each lab to develop CRADA with industrial partner.
1993	CRADA between LANL and IBM signed. LANL provides HIPPI Expertise and protocol development, IBM develops hardware.
1994	Rainbow-2 All-Optical Network with HIPPI interfaces demonstrated at SC '94 and OFC '95.
1995	IBM delivers Rainbow-2 network to LANL for evaluation with supercomputers <ul style="list-style-type: none"> • Rainbow nodes placed in LDCC and ACL and are interconnected to LANL's HIPPI Testbed Network. • Results of test show high speeds attained but performance is limited by slow response of optical filters. • LANL proposes to research and develop fast packet optical network. IBM agrees but has limited resources to support new task. LANL pursues technology because of potential use to Laboratory Programs.

Table 1. CRADA project milestones including milestones leading up to IBM/ LANL collaboration.

a) Wavelength Division Multiplexing

As described above, conventional single-channel networks may not be sufficient to provide the projected level of service that will be required in the near future. In a network employing WDM, a unique transmission frequency is assigned to each data channel. The channels are then optically multiplexed such that multiple channels travel down the fiber simultaneously and with negligible interference. Therefore, WDM allows the large bandwidth of the optical fiber to be more fully utilized. Optical fiber is not used simply as a 1:1 replacement for copper wires (one channel per fiber); WDM enables an $N:1$ capability (N channels per fiber, where each channel may operate at the full electronic limit, allowing multi-gigabit per second aggregate data rates). WDM technology also offers network planners considerable flexibility in that the optical channels are both bit-rate and data-format independent.

All-optical networks that utilize WDM overcome the electronic bottleneck by removing electronics from the network. The requirement for a complex central switch is eliminated and the channel selection task is distributed among all of nodes on the network. With sufficient physical bandwidth, it is possible to simulate any type of logical switch. This "virtual" switch can function exactly the way physical switches do. It is possible to send messages anywhere in the network, include needed codes and instructions for correcting, decrypting, and reading them, and allow each terminal to tune into its own messages on its own wavelength. At this point, a fiber optic network becomes an *all-optical network*. The all-optical network is entirely photonic. From the first conversion of the signal at the source to the final re-conversion to data at the destination, it propagates through glass and does not require electro-optical conversion in between.

b) All-Optical Broadcast Networking

The Rainbow-II All-Optical Network developed is a broadcast network based upon the Rainbow-I network that originated at IBM. Rainbow-II is implemented in the form of optical network nodes (ONNs), each attached to a host computer via the High Performance Parallel interface (HIPPI) developed by Los Alamos¹. A block diagram is shown in Figure 1.

Figure 1. Block Diagram of the Rainbow-II All-Optical Network.

Each node has a fixed optical transmitter that emits light at a wavelength different from other transmitters in the network and a tunable receiver. A fiber leads from each station to the network hub, an all-glass, totally passive star coupler with N inputs and N outputs, N being the number of stations in the network. The star coupler combines the transmissions from different nodes and sends approximately $(1/N)$ th of the optical power from each transmitter along a fiber path to each receiver. Thus, the network is broadcast in nature; all transmissions reach all nodes. At each node, a tunable optical filter selects one of the N transmission wavelengths to be received. In addition, each network node in the Rainbow-II network contains protocol processing hardware to offload the protocol processing work from the host computer onto the node such that full gigabit-per-second bandwidth may be provided to end-user supercomputer applications.

¹ Optical-fiber transmission systems can easily achieve HIPPI rates (0.8 and 1.6 gigabits/sec) and, in fact, are used today to transmit HIPPI over 10 km distances. In addition, these transmission systems have incredibly low bit error ratios (typically BERs of 10^{-9}) compared to copper-wire links (typically 10^{-5}).

c) HIPPI

In order for networks like Rainbow-II to be useful, they must be suitably attached to host computers via a host interface. The host interface of choice for Rainbow-II is the High-Performance Parallel Interface (HIPPI), an ANSI standard for connection of data processing equipment using multiple twisted-pair copper cabling². HIPPI, a point-to-point interface, supports data transfer rates of 0.8 Gb/s simplex (1.6 Gb/s duplex) using very simple signaling sequences. Furthermore, it is widely used for the interconnection of supercomputers, many of which have HIPPI interface hardware available.

Two of the major limitations of HIPPI are that it uses a cumbersome connector and cable (100 wires per connector, two connectors/cables per full-duplex connection), and that the maximum distance between HIPPI nodes is 25 m (limited by the electrical characteristics of the cable). These problems have been addressed by a version of HIPPI known as Serial HIPPI which uses HIPPI protocols and signaling methods over optical fiber. Serial HIPPI is essentially a fiber extension cord for parallel HIPPI.

Another ANSI standard interface is the serial 1 Gb/s Fiber Channel Standard (FCS)³. Unlike HIPPI, Fiber Channel is complex in order to be rich in features. It specifies four data rates, three kinds of media, four transmitter types, three distance categories, three classes of service, and three possible fabrics. The result of this complexity is that Fiber Channel products have been slow to market and have not performed to the full potential of the standard.

Work completed under this CRADA did not include Serial HIPPI or FCS. Serial HIPPI was not considered for inclusion into the WDM effort because the coding and signaling techniques used in serial HIPPI were not adaptable to the WDM protocols needed for Rainbow-II. The ONN converted HIPPI framed data to a special low-level protocol that could be used by the optical filters to tune to the right channel. Of course, a serial HIPPI link could easily have been added to the host side of the ONN, replacing the copper version, but this would have added nothing to the research effort.

Fiber Channel (FCS) interfaces were never developed mainly because very few implementations of FCS exist, and there are none at Los Alamos. As the standard progresses and becomes more widely implemented, FCS could be reconsidered for future versions of the Rainbow-II network.

² ANSI X3.183-1991, High-Performance Parallel Interface, Mechanical, Electrical, and Signaling Protocol Specification.

³ ANSI X3.230-1994/AM 1-1996, Information Technology Fibre Channel Physical and Signaling Interface (FC-PH) Amendment 1 (supplement to ANSI X3.230-1994)

d) Circuit Switching versus Packet Switching

In general, data transmission is accomplished in three phases: initiation, transfer, and disconnect. The application of these phases is dictated by the access protocol of which there are two types: circuit-switched protocols and packet-switched protocols as depicted in Figure 2.

In a circuit-switched network, an end-to-end physical path is established in the initiation phase, the user payload is sent during the transfer phase, and the circuit is disestablished during the disconnect phase. An end-to-end physical path is maintained during the full communication. These networks are particularly suitable for high-speed continuous streams of data, especially if the duration of the connection is relatively long. Examples of "connection oriented" services are video-conferencing and large file transfers.

Packet switching encompasses the same three phases, but it differs in that a real physical path (circuit) is not established. Instead, a virtual circuit consisting of a stream of associated packets that are all addressed to the same destination is created. Whereas in circuit switching the physical circuit is dedicated to the designated pair of users for the duration of the communication, in packet switching each link may be shared across many concurrent connections going to different places. A "connection" may consist of a single packet or of a long stream of packets. The requirement is that each packet must be carefully identified (by suitable fields in the header of each packet). Packet switched networks are particularly suitable for bursty data streams that

are typical of computer traffic. Increasing data rates and the burstiness of traffic sources has lead naturally to increasing interest in systems based on fast packet switching.

The Rainbow-II network demonstrated under this CRADA employs a circuit switched protocol. This is due to the slow tuning speed of the tunable optical filter. However, it is recognized that a fast packet-switched network would be a superior solution for this application, and considerable progress was made in attaining suitable tuning speeds to allow a packet-switched solution in the future. Part of this collaboration called for placing the Rainbow-II All-Optical Network at the Los Alamos National Laboratory computer testbed to determine its feasibility in a "real world" environment with super computers and other high performance systems interconnected with this network. This evaluation showed the need for a packet-switched network. These results are described in Section F below. Further discussion regarding the proposed packet-switched network is also contained in the section describing future work.

2. The Rainbow II All-Optical Network

The Rainbow-II Optical Network supports 32 nodes each at 1 Gb/s over a distance of 10-20 km. The network is implemented in the form of optical network nodes (ONNs) connected in a broadcast star architecture as shown in Figure 1 and previously described in Section 1b above. The ONNs were built by IBM. Details concerning the ONN hardware design are given in Section F (Section II of the appended paper).

The Specific goals of Rainbow-II are the following:

1. to provide connectivity to host computers using standard interfaces,
2. to deliver 1 Gb/s throughput all the way to the application layer, and
3. to study real applications requiring Gb/s capacities.

The first goal listed above was to provide connectivity to host computers using standard interfaces. In the testbed experiments, connectivity to HIPPI-based hosts was successful. Usually, the hosts were CRAYs or IBM RS6000s.

The Rainbow-II network was a featured demonstration at Supercomputing '94 and at the Optical Fiber Communication Conference '95. The demonstration used three RS6000 workstations, each connected to an ONN and are described in Section F (Section IV of the appended paper). Data throughput was limited primarily by the speed at which data could be outputted from the RS6000s and not by the ONNs, resulting in a demonstrated throughput of 60 Mb/sec at these events.

The Rainbow-II network is currently deployed at LANL where its performance has been evaluated against established gigabit testbeds, such as the LANL HIPPI Testbed, by comparing data-transfer rates between supercomputers/workstations and other high-end HIPPI-based systems. The LANL testbed network is shown, and the experiments are described in Section F (Figure 4 and Section IV, respectively of the appended paper). Additional test results as well as a report on the operation of the ONNs may be found in Section G. A description of the installation and operation of the software that runs in and supports the Rainbow-II ONNs may be found in Section H.

The study of real applications showed that for visualization (i.e., a network of HIPPI frame buffers connected to CRAYs, etc.) the "circuit-switched" nature of Rainbow-II worked very well, delivering full HIPPI bandwidth to the displays. However, for real "network" applications requiring TCP/IP, the slow circuit-switching greatly hindered performance, and a packet-switched solution should be explored in future versions of the Rainbow-II network.

3. Media Access Protocol

In computer networks the physical connection between two users is set up via a Media-Access Control (MAC) protocol. When one node has a message for another, it seizes some of the capacity of the network and addresses the message directly using a MAC protocol.

The demonstrated Rainbow-II network used a simple, light-weight, IBM-proprietary transport protocol called Optical Transport Protocol (OTP) that was chosen for expediency. OTP is a slow, circuit-switched protocol. Los Alamos, in cooperation with IBM, began development of a packet-switched MAC protocol that would operate in conjunction with a fast optical filter on the order of microseconds. The progress on this development is described below.

Los Alamos identified six appropriate MAC algorithms to be tested. In addition, computer simulations of each MAC protocol program were performed to determine overall throughput, speed of transfer, and comparative algorithm complexity. A MAC protocol consists of a series of events (scan, tune, receive) as shown in Figure 3.

A computer simulation of the MAC level protocol was written to determine how the algorithms would work in actual use, since hardware was not available. For purposes of simulation, time was divided into a scanning period, tuning period, and slot period. There is a tuning period (to tune the receiver to the desired wavelength) before and after the scanning period, as well as between all slot times. The slot time is the time allotted to transmit one burst of information from a transmitter to a receiver.

During each slot time, any number of transmitters can transmit their bursts to receivers, as long as the receiver is free (not receiving from a different transmitter). During the scan time, all receivers scan all frequencies (wavelengths) to read the header packet that each transmitter is continuously transmitting. This packet identifies the transmitter and all receivers with which this transmitter would like to communicate. Once the scan time is completed, all receivers have a table of all transmitters and all packets they need to send. The MAC algorithm is then run to allocate the slots to the packet bursts, with the goal of using as many slots as possible to successfully transmit the maximum amount of data across the network. Since each transmitter uses a unique transmission frequency, any number of transmitters can transmit a burst during a single slot time, as long as each is destined to a different receiver.

In the simulation, packets were randomly queued to each transmitter, using a distribution of destinations as well as packet sizes. The maximum and minimum packet sizes were specified on initialization of the simulation, as was the distribution of destinations. This allowed some nodes to be simulated as servers (non-server nodes had a higher probability of sending to server nodes). The number of slot times per cycle was also specified at initialization.

For each cycle, a scan packet was built for each transmitter, specifying the receivers to which data would be sent during that cycle. At the end of the scan cycle, the algorithm simulation was performed, and the data to be sent for each slot time was calculated. Then, simulating each slot time, each burst was recorded as sent, and the data remaining to be sent was updated. The number of slots used, the time it took to send each packet, and the time it took to start transmission of one packet was noted. This was done for the number of cycles specified during initialization. At the end of the simulation, The percentage of slots used, the throughput in bytes/second, and the delay for the packets (maximum, minimum, and average) was calculated.

It was found that allocating too many slot times wastes time with a lightly loaded network. When a packet arrives during a cycle, it must wait until the next cycle to be recorded and slot times allocated. On the other hand, added slot times increase throughput for a heavily loaded system. This is also true for the length of the slot times. One upgrade to the algorithms would be to calculate the load on the network and increase or decrease the slot times as necessary.

There was not that much difference in the average throughput, delay, and slots used for the different algorithms. However, the minimum and maximum times varied a bit more. Thus, choice of algorithm depended upon which of the factors was most important. It also depended on how heavily the network was loaded.

Work on the MAC layer protocol was stopped, as it was decided to build a simple demonstration system that would be simplified to have fixed transmit and receive times. This increased the probability that we could build the demonstration system on time and decreased its complexity. Since the MAC protocol was not needed for this

demonstration, further work on the MAC level protocol was put on hold. Further work would include refining the simulation, deciding on what parameters should go into the MAC protocol, and then building hardware to implement the MAC level protocol.

4. Microsecond Optical Switching

Fast switching is an essential requirement for broadcast high-speed gigabit optical networks and for packet switched networks in particular. Communication networks require a discriminating component that can detect the destination of incoming traffic and route that traffic to the proper end node. For conventional high bandwidth networks, this component usually takes the form of an electronic switch in the center of the network. At the switch optical-to-electrical conversion and reconversion of the data-stream is required, and the incoming message is electronically routed based upon encoded address information. In gigabit networks, this must occur in a few microseconds in order not to impact the overall performance of the network.

In an all-optical WDM network, the electrical-to-optical conversion and reconversion at the switch is eliminated. At the receiving-end, a wavelength discriminating optical component followed by a photodetector provides simultaneous selection and detection of the desired optical signal, creating the opportunity to more fully utilize the inherent bandwidth of the optical fiber. The performance of the optical discriminator is vitally important to the operation of the network. For this reason considerable effort was concentrated on developing fast optical switching capability and significant progress was realized. The results are summarized in this section and discussed in detail in Section I below.

One approach to achieve fast optical tuning is to use high-speed piezoelectrically-driven *Fiber Fabry-Perot Tunable Filters* (FFP-TFs). While piezoelectric (PZT) Fabry-Perot filters are typically reported to have millisecond tuning speeds, microsecond optical tuning, as well as polarization insensitivity and low loss are necessary to perform high-speed optical packet switching. Achieving microsecond tuning raises a whole new set of challenges with respect to optical ringing, thermostability and mechanical inertia control. Here we report tuning for FFP-TFs approaching one Free Spectral Range (FSR) per microsecond. To achieve these results, special high-speed FFP-TFs (Micron Optics Inc. (MOI)) were selected, and a novel stabilizing driver was developed at LANL.

a) Structure of FFP-TF

A sketch of the inner optical configuration of a FFP-TF is shown in

Figure 4. Low voltage PZT actuators are used to position two highly reflective mirrors. Tuning is accomplished by varying the air gap between the mirrors, l_c . The PZT actuators are capable of positioning accuracy of atomic dimensions. Constructing the FFP-TF with the single-mode optical fibers inside the cavity avoids the need for lenses and collimated beams, eliminates the "walk-off" tendency, and provides single-mode guidance and low-loss coupling.

The free spectral range (FSR) of an ideal FFP-TF (Figure 5) is related to the cavity length (distance between mirrors), l_c , by

$$FSR = \frac{c}{2nl_c},$$

where c is the speed of the light in a vacuum, and n is the index of refraction of the cavity material. The finesse, F , of an ideal, lossless FFP-TF is related to mirror reflectivity, R , by

$$F = \frac{\pi\sqrt{R}}{1-R}.$$

The bandwidth, BW, or wavelength resolution of the FFP-TF is simply

$$BW = \frac{FSR}{F}.$$

The FFP-TFs selected were fabricated with a cavity length of 8 μm and a 1.0 μm air-gap resulting in an FSR of 100 nm.

Figure 5. Fabry-Perot Tunable Filter definitions and formulas

b) Characterization of high-speed FFP-TF

The five critical performance parameters of high-speed FFP-TFs include tuning time per FSR, mechanical resonance, response linearity, and overall switching and settling time. Correlation between the mechanical resonance and optical response of the filter is important to obtain the optimal high-speed tuning design and for mounting hardware and control circuitry optimization.

In early experiments, the fastest tuning speed obtained for the new small FFPs without optical ringing was 50 μ s. The switching speed of the filters was limited by acceleration of the mechanical motion and electrical loading of the drive voltage. Increasing the driving slew rate induced extensive optical ringing, a combination of the driving waveform ringing and internal effects. Therefore, effort was focused on improving FFP-TF switching time and raising the mechanical resonance to beyond 100 kHz.

Reducing the total mass of the FFP-TFs was one of the keys in obtaining high-speed operation (standard (heavier) FFP-TFs had a tuning speed on the order of one msec). In experiments at LANL, the response time, T_r , of the filter was evaluated over the free spectral range for different FFP constructions. The fast rising and falling edges of the electrical driving signal (low-frequency square-wave) were used to rapidly scan the filter to obtain the response. Eight FFP-TF mechanical fixtures were characterized, differing in mounting-hardware, mass, and PZT length, as listed in Table 2.

Filter	PZT length (mm)	PZT type (V/ μ m)	Hardware mass (g)	Total mass (g)	first resonance (kHz)	T_{En} (μ sec)	T_{F} (μ sec)
TF1	16	12	2.6	9.6	40	7.00	1.76
TF2	10	18	2.6	8.3	43	9.4	1.76
TF3	10	18	1.5	6.4	60	5.9	1.66
TF4	16	12	1.5	7.8	34	7.00	1.36
TF5	10	18	0.6	4.4	68	3.90	1.10
TF6	10	18	0.4	4.0	75	3.30	1.00
TF7	10	18	0.1	3.8	75	2.68	0.94
TF8	10	18	0.1	2.0	113	2.44	0.84

Table 2. High-speed FFP-TF experimental results

A special high-speed stabilization controller was built (taking into consideration the characterization results and the FFP-TFs' high capacitance (approximately 0.5 to 1 μ F)) to obtain a 'clean' square waveform for the electrical driving signal. The controller substantially improves the shape of the driving signal and the response of the filter, it enables tuning the high-speed FFP-TF three-orders-of-magnitude faster than possible with standard commercial FFP-TFs.

Initially, the driving slew rates and amplitudes were carefully adjusted to minimize the excessive optical resonance due to electrical waveform ringing and internal effects. The fastest response times thus achieved without optical ringing for one reference, T_{FO} , are generally $<10 \mu$ s, as listed in Table 2.

Results in Table 2 show response times, T_{F} , reaching 1 to 2 μ s. There is a clear advantage gained by using the LANL's high-speed driver and lower hardware mass of MO's FFP-TFs. In additional experiments, the performance of LANL's stabilizing controller was compared with an MOI controller. LANL's driver substantially reduced the electrical oscillations of the driving signal and optical ringing in the filter thereby dramatically improving the response of the FFP-TFs.

Experiments at LANL demonstrate FFP-TF optical switching three orders of magnitude faster than that possible with standard commercial FFP-TFs. These results imply that it is possible to implement a packet-switched all-optical network. In such a network, a new packet-switched media access control protocol would be designed to minimize searching time. The filter would scans only once through the entire optical region and then tune to all of the required channels one after another in a few microseconds. The Rainbow-II Broadcast-and-Select High-Speed All-Optical network currently has a circuit-switched protocol using standard FFP-TFs. Implementation of specially selected, low mass FFP-TFs in a packet-switched network would significantly improve the usefulness of the current Rainbow-II design.

5. Future Work

The development at Los Alamos of HIPPI as the first gigabit-per-second interconnect technology to become a national standard has helped to establish Los Alamos as world leader in high-performance network research. With the end of underground nuclear testing, increased demands on DOE's high-performance computing mission have grown. The Accelerated Strategic Computing Initiative (ASCI) is an ambitious DOE program whose objective is to develop a tri-laboratory computing environment capable of exchanging terabyte data sets between massive storage units with local performance of near 100 Tflop/sec. This represents orders-of-magnitude increases over present-day performance limits. The development of ultra-high capacity local- and wide-area computer networks is also of interest to address the exponential growth of Internet traffic. The development of high speed multimedia networks puts increasing demand for higher bandwidth channels and networks. Such demands may require transmission of information at an ultra-high bit rate of terabits/sec over fiber optics.

The success of the Rainbow-II network demonstrated the feasibility of all-optical networks and was a significant step toward realizing these data rates. However, the research resulting from this collaboration pointed out limitations in the Rainbow-II design. Specifically, the relatively slow tuning performance of the optical filters would relegate the network to circuit-switched applications. Real world computing involves a broad mixture of applications that necessitate a fast packet-switched capability. Substantial improvement in the driving characteristics for the optical tunable filter has already been achieved. Further improvement would allow implementation of a novel *packet-switched media access control (MAC) protocol* to minimize the searching time for the appropriate wavelength and provide higher flexibility and optical throughput.

Specifically, future research should include the following.

- New high-speed piezoelectric fiber Fabry-Perot tunable filters should be fully characterized, including the optical response of the filter to the driving signal, the mechanical response of the piezoelectric element and the loading, and the environmental stability of the filter. This is important for further increasing tuning speed and for optimizing mounting hardware and control circuitry.
- The results of the filter characterization and optimization and consideration and optimization of the filter's capacitance should be incorporated into the design of a special controller to further improve the shape of the driving signal and the response of the filter.
- A new packet-switched MAC protocol should be designed such that searching time is minimized. The filter will scan only once through the entire optical region and then tune to all required channels one after another in a few microseconds, substantially increasing the flexibility of the optical network. This new packet-switched MAC protocol would be a central building block in a new proof-of-concept prototype network.
- A Rainbow-III prototype network should be implemented to demonstrate switching between four wavelengths within the 1.5 μm communication band.

- Following proof-of-concept, the Rainbow-III network should be evaluated in LANL's high-speed computing testbed, simulating "real world" high-speed network traffic and applications.
- All issues listed above should be revisited in the context of exploring the use of optical nonlinear soliton transmission.

In addition to high-speed lightwave systems, the concept of using optical nonlinear solitons as a transmission scheme may have great potential in next-generation data communication networks.⁴ Solitons are very-short time-division optical pulses not easily destroyed by small propagation inhomogeneities in fibers. Since pulse distortion due to dispersion does not occur, the time duration of a bit can be very short (100-200 fsec) so that it is possible to achieve a very high transmission rates. Revolutionary experiments by L. F. Mollenauer and his group at Bell Labs have demonstrated the soliton WDM transmission at 80 Gb/sec over transoceanic distances⁵. Demonstration of soliton optical transmissions in an all optical network using the fast packet switching between the links would be a significant achievement for the Rainbow-III network. Such a demonstration of near terabit/sec transmissions in a practical network setting is of extreme interest to such programs as ASCI and HPCRC where network speed and latency are critical parameters that need to be achieved in order to reach the required computational efficiency.

D. Partner Contributions

Partner contributions are described in Section F below and summarized in Part III of this report.

⁴ See, for instance, G. P. Agrawal, *Nonlinear Fiber Optics*, Academic Press, 1995 or N.L. Taranenko, L.G. Kazovsky, eds., *Three-Wave Envelope Solitons: Possibility Of Controlling The Speed Of Light In Optical Fibers*, *Journal of Lightwave Technology*, **12**, 7, 1994.

⁵ L. F. Mollenauer, P. V. Mamyshev, eds., Demonstration of soliton WDM transmission at up to 8x10 Gbit/s, error free over transoceanic distances, in *OFC'96 Technical Digest*, **2**, 1996

E. Documents/Reference List

All CRADA reports published for the project are identified below. None of the documents/references listed in this section represent CRADA-protected information. Los Alamos is not engaged in patent/copyright activity and has no pending applications associated with this project. The rights to the design of the ONN are held by IBM. There were no subject inventions disclosed by either IBM or LANL to be protected under the intellectual property protection measures (Articles XIII through XVI of the CRADA).

1. C. Barry , M. Hickey, C. Noronha, N. Taranenko, L. Kazovsky, "A multi-Gbit/s optical LAN utilizing a passive WDM star: towards an experimental prototype," *Proc. SPIE*, vol. 2024, p. 171-177, 1993.
2. C. Barry, N. Taranenko, C. Noronha, S. Agrawal, L. Kazovsky, "Network Interface Design for Multi-Gbit/s WDM Optical Networks," *J. High-speed Communication Networks*, April 1995.
3. B. Beattie, M. G. Halvorson, C. Idler, J. Kravitz, M. Sukalski, and R. Thomsen, "SHIP - simple host intersocket protocol," working document, <http://www.cic-5.lanl.gov/lanp/postscript/shipv10.pdf>, Feb. 1995.
4. E. Hall, J. Kravitz, S. Tenbrink, Demonstration of Rainbow-2 Gigabit Optical Network (IBM Research and Los Alamos National Laboratory), *Supercomputing Conference & Exhibit*, November 1994.
5. E. Hall, J. Kravitz, N. Taranenko, Demonstration of Rainbow-2 Gigabit Optical Network (IBM Research and Los Alamos National Laboratory), *IEEE Optical Fiber Communication Conference & Exhibit*, February 1995.
6. E. Hall, J. Kravitz, R. Ramaswami, M. Halvorson, S. Tenbrink, and R. Thomsen, "The Rainbow-II Gigabit Optical Network," *IEEE J. Select. Areas in Commun.*, vol. 14, No. 5, June 1996.
7. J. Kravitz, R. Ramaswami, D. Steinberg, W. E. Hall, F. Janniello, "A High Performance Wavelength Division Optical Network Adapter with Protocol Offload Features - Preliminary Specification," IBM proprietary document, January 1994.
8. N. Taranenko, S. Tenbrink, C. Miller, K. Hsu, "Fiber Fabry-Perot tunable filter for high-speed optical packet switching," *Emerging Components and Technologies for All-Optical Systems, SPIE's Photonics East'96 Symposium*.
9. N. Taranenko, C. Miller, K. Hsu, Demonstration of high-speed wavelength switching for Wavelength Division Multiplexing Gb/s optical networks (Micron Optics, Inc. and Los Alamos National Laboratory), *IEEE Optical Fiber Communication Conference & Exhibit*, February 1996.
10. N. Taranenko, S. Tenbrink, Demonstration of high-speed optical tunable filter, *Supercomputing Conference & Exhibit*, November 1995.

F. Appendix: The Rainbow-II Gigabit Optical Network*

Eric Hall, Jeff Kravitz, and Rajiv Ramaswami
IBM T. J. Watson Research Center

Marty Halvorson, Steve Tenbrink and Richard Thomsen
Los Alamos National Laboratory

* Published in the IEEE Journal on Selected Areas in Communications, Vol. 14, No. 5, June 1996.

The Rainbow-II Gigabit Optical Network*

Eric Hall, Jeff Kravitz, Rajiv Ramaswami,
Marty Halvorson, Steve Tenbrink and Richard Thomsen

Abstract

This paper describes the Rainbow-II Optical Metropolitan Area Network, which supports 32 nodes each at 1 Gb/s over a distance of 10-20 km. Rainbow-II uses optical wavelength division multiplexing (WDM), in a broadcast star architecture. Each node uses a separate fixed wavelength for transmitting data and a tunable receiver for receiving one of several data streams. The network is implemented in the form of optical network nodes, each attached to a host computer via the high-performance parallel interface (HIPPI). Each network node contains protocol processing hardware to offload the protocol processing work from the host computer onto the node. The goal is to provide full gigabit-per-second bandwidth to end-user supercomputer applications. Preliminary protocol performance measurements in a testbed network are given.

1 Introduction

Optical networking using wavelength division multiplexing (WDM) offers the potential for building large networks supporting hundreds of nodes and offering each node capacities of up to 1 Gb/s [1, 2, 3]. For an earlier WDM network project, we prototyped a 32-node, 300 Mb/s per node, WDM metropolitan-area network (MAN) called Rainbow-I [4]. This paper describes the Rainbow-II network, which is a follow-on to Rainbow-I. Rainbow-I was built primarily to gain experience with using optical tunable components and other technology in a field environment. In contrast to Rainbow-I, Rainbow-II was built to study and gain experience with higher-layer protocols and end-user supercomputer applications. The network was designed and built by IBM Research. It is currently deployed as an experimental testbed at the Los Alamos National Laboratory (LANL). LANL is making performance measurements and experimenting with gigabit applications, preliminary results of which are presented in this paper.

1.1 Rainbow-I Architecture

In Rainbow-I, each station has an optical transmitter that emits light at a wavelength different from other transmitters in the network. A fiber leads from each station to the network hub, which is an all-glass, totally passive star coupler with N inputs and N outputs, N being the number of stations in the network. The star coupler combines the transmissions from different stations, and at each output, we obtain approximately $(1/N)th$ of the optical power from each transmitter. From the star coupler, a fiber leads to each station. Thus, the network is broadcast in nature; all

*This work was supported in part by grant no. MDA 972-92-C-0075 from ARPA and CRADA no. LC9210052 from Dept. of Energy. E. Hall, J. Kravitz, and R. Ramaswami are with IBM T. J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532, rajiv@watson.ibm.com. M. Halvorson, S. Tenbrink, and R. Thomsen are with the Los Alamos National Laboratory, Los Alamos, NM 87545, sct@lanl.gov.

transmissions reach all the stations. At each station, a tunable optical receiver selects one of the N wavelengths. Each station in the Rainbow-I network consisted of a PS/2 host with an attached MicroChannel card, much like that of a token ring or Ethernet card, that provided the optical network attachment and support for implementing a simple circuit-switched media-access protocol. However, the MicroChannel was used only for control purposes; actual data entered and exited the Rainbow-I card directly using a separate serial input and output.

1.2 Goals of Rainbow-II

The specific goals of Rainbow-II are (1) to provide connectivity to host computers using standard interfaces, (2) to deliver 1 Gb/s throughput all the way to the application layer, and (3) to study real applications requiring Gb/s capacities.

In order to make networks like Rainbow more useful, we must provide suitable attachment to hosts. The host interface of choice was the high-performance parallel interface (HIPPI) [5]. HIPPI is an ANSI standard for connection of data processing equipment using multiple twisted-pair copper cabling, and supports data transfer rates of 800 megabits/second. It is widely used for the interconnection of supercomputers, many of which have HIPPI interface hardware available.

Two of the major limitations of HIPPI are that it uses a cumbersome connector and cable (100 wires per connector, two connectors/cables per full-duplex connection), and that the maximum distance between HIPPI nodes is 25 meters (limited by the electrical characteristics of the cable). These problems have been addressed by a version of HIPPI known as Serial HIPPI which uses HIPPI protocols and signaling methods over optical fiber. Another ANSI standard interface is the serial 1 Gb/s Fibre Channel Standard (FCS) [6]. The Rainbow-II network supports standard HIPPI currently. We may add a serial HIPPI or FCS interface in future when these standards are widely implemented.

Providing gigabit-per-second speeds to each node at the physical layer is only a first step. Present-day protocols are not designed and implemented to allow applications at a node to realize gigabit-per-second throughputs from gigabit-per-second-per-node networks. Classic protocols such as TCP/IP (transmission control protocol/internet protocol) were designed to minimize transmission bandwidth requirements and not CPU overheads. The maximum packet sizes supported were small because of network-imposed constraints on error rates. These protocols were not designed to be implemented in an efficient manner and are not very well suited for implementation on multiprocessors. There are several proposed approaches to solving this problem. One approach is to design lightweight transport protocols that get rid of many of the inefficiencies in TCP/IP such as unaligned header fields, complicated checksum calculation, searching to locate the transmission control block for the connection each time a packet is received, etc.[7]. It has also been claimed however, that careful implementation of TCP/IP can significantly reduce the penalty in performance [8]. However, both these approaches do not solve the major problem that many hosts, particularly large parallel computers are not suited for protocol implementation, and even in those that are, e.g., a Cray, a significant portion of the CPU time is spent on protocol processing. Thus our solution is to offload all the protocol processing work from the host onto a dedicated protocol processor (PP) [9, 10, 11, 12]. The protocols are implemented on on-board microprocessors with suitable hardware assists, rather than dedicated VLSI hardware, since that will give us the flexibility to experiment with a variety of protocols. We have implemented a lightweight transport protocol called optical transport protocol (OTP), to be described later.

Figure 1: The Rainbow-II Network

1.3 Rainbow-II Implementation

Rainbow-II uses the same broadcast-star architecture as Rainbow-I. It is implemented in the form of optical network nodes (ONNs) connected to a passive star coupler as shown in Fig. 1. The ONNs are attached to host computers via HIPPI. Sec. 2 describes the hardware design of an ONN and Sec. 2.1 focusses on the protocol offload hardware. Sec. 3 gives an overview of the software, including the optical transport protocol (OTP). Sec. 3.1 describes a slim protocol developed to offload the protocol processing work from the host computer onto the ONN. Sec. 4 gives the current status of the network and summarizes the results to date.

2 Hardware Design

The rationale above led us to design and build the Rainbow-II optical network node (ONN) shown in Fig. 2. The ONN consists of three logical sections: (a) a host-attach section which is essentially a HIPPI interface, (b) a network-attach section which contains the optical transmitter, receiver, and control hardware for the optical WDM network, and (c) a protocol offload section in the middle for implementing all the protocol layers up to the transport layer.

The ONN is realized in the form of 5 printed circuit boards, of 4 types. There are two protocol processor (PP) boards, one for each direction of data travel. In addition, there is a HIPPI I/O adaptor board, and a set of two optical network attach I/O boards, one containing the optical transmitter and the receiver, and the other containing the logic to serialize/deserialize the data stream and implement the same simple media-access protocol used in Rainbow-I [4]. These boards plug into a custom backplane containing point-to-point connections for the interfaces between the boards (i.e. no shared bus). Both protocol processor boards are identical in design, with only the backplane circuitry determining which board is connected to which I/O adapter board. The interface between the PP boards and the I/O adapter boards is such that changes need not be made to the PP boards if new I/O adapter boards are designed later. In fact, it is assumed that new I/O adapter boards will be designed in future, to support enhancements to the Rainbow network (e.g. fast tuning for packet switching) or additional host attach methods (e.g. serial HIPPI, FCS, host

Figure 2: The Optical Network Node

channel).

The boards are mounted in a small, desktop or rack-mountable cabinet and there are RS232 connections to the on board processors for bootstrapping, debugging, configuring, etc.

2.1 Protocol Offload Section

The protocol offload section of the ONN is designed to allow much of the communication protocol work to be done in it, instead of the attached host computer. Our design strategy was to try to reduce software CPU instruction path-length counts as much as possible. This is because there are very few instruction cycles available to process each packet, and so the software overhead is typically the bottleneck for high bandwidth and low latency. (For example in order to achieve 1 Gb/s throughput using 4 KByte packets, each packet must be processed in 32 μ s. With a 10 MIPS processor this means that all processing for a packet has to be done in less than 320 instructions). This strategy has led to a number of design decisions as follows:

- The operating system overhead in many protocol implementations is large, and in many cases cannot be reduced. In our system this was considered a key area, and so an operating system, per se is not used. The code is a dedicated program running directly on the hardware, with a number of support functions such as memory management, etc. being fine tuned to the hardware and the application. This does lead to increased programming effort, both to code the support infrastructure, and because an existing (e.g. Unix-based) implementation of the transport protocol cannot not be easily ported to this environment.
- Given that the hardware is being implemented specifically to assist in the protocol offload, a number of hardware features were designed into the system to improve the performance of the ONN by assisting the onboard software. The protocol offload section is comprised of two individual protocol processor boards, one for each of the two directions of data travel.

The performance enhancing features on each board include a protocol microprocessor, microprocessor program memory, data/header buffer memory, two intelligent direct-memory access (DMA) controllers, four DMA buffer queue FIFOs, inter-CPU communication FIFO, an inter-CPU DMA, and hardware calculation and checking of checksums.

2.1.1 Protocol CPU

Each protocol processor board contains a high-performance RISC CPU (Intel i80960CA) which is responsible for handling buffer management and protocol offload for data traveling in a given direction. The i80960CA processor is a 32-bit RISC CPU which has been used elsewhere in similar applications.

2.1.2 CPU Program Memory

Each protocol processor has 2 MBytes of separate RAM memory to contain the ONN software and its associated variables and data structures. The program memory is separate from the data memory to improve access speed. The ONN control software is loaded into this memory via the RS232 ports from an attached system such as a PC.

2.1.3 Triple Port Data/Header Buffer Memory

In order to minimize contention for the memories, it was decided to try using two separate memories on separate buses, one for data traveling from the network to the host computer, and one for data traveling in the other direction, in the hope that this will allow data to travel in both directions concurrently, while eliminating serious contention for the memory busses.

Triple-ported video RAM's are used for this memory. The video RAM's have two 'sequential' interfaces which are used for the DMA access to and from the memory and a random-access interface which is used for the interface to the CPU.

2.1.4 Intelligent DMA Controllers

Each protocol processor board has two DMA controllers for data traffic, one to handle data movement into the buffer memory, and the other to handle movement out of the buffer memory.

These DMA controllers are "intelligent" in the sense that they are capable of using the buffer queue FIFO's (described below) to obtain and free buffers in memory. They are therefore asynchronous (to some extent) from the operation of the protocol processors, freeing these processors from the cycle-consuming task of setting up each DMA operation and processing DMA complete functions. The "intelligent" features of these DMA controllers help eliminate some software overhead, thus improving overall ONN performance.

In addition, to eliminate the necessity of copying or moving data in memory, the protocol headers are stripped by the DMA controllers from the user data portion of the packets, and are placed in separate memory buffers, dedicated to header use. This allows different headers to be prepended to the data on the outgoing side, without the necessity of moving data in memory, again assisting performance.

To support this header/data stripping and to assist buffer management, the DMA controllers operate with two sizes of buffers, called H buffers and D buffers. The H buffers are intended to hold protocol headers and control block information, and are therefore relatively short, and the D buffers are intended to hold user data, and are relatively long. The sizes of the buffers are selectable

by software, and the H/D distinction is managed by the DMA controller hardware as well as the software.

The transmit DMA controller, in order to allow for building up of transmission units from smaller memory blocks, supports appending of DMA entries via an *append* function, and also allows such units to fall on arbitrarily aligned addresses, and have arbitrary lengths (no restrictions as to 32-bit boundaries or 32-bit multiples). This allows protocol software to build up packets from various pieces in disparate portions of memory, allowing for different protocols, protocol conversion, etc. without having to move data in memory. This again enhances performance.

The receive DMA controller has support for calculation of a TCP checksum on the received packet. When (and if) TCP/IP is implemented in the node, this will provide a very fast method of calculating the necessary checksums, which is considered a serious performance bottleneck in most systems that do this calculation in software.

2.1.5 DMA Buffer Queue FIFO's

These 32-bit wide queue FIFO's are intended as an interface between the protocol processors and the "intelligent" DMA controllers, and also are a simple mechanism to assist the processors in performing buffer management. The FIFO's queue up requests for transmission, free buffers for reception, and provide completed transmission and reception status information. These queues eliminate the necessity of immediate software response to events, since no event is lost if the software cannot respond within a small time interval. This eliminates interrupt overhead, thus increasing performance.

There are 5 logical DMA Buffer Queue FIFO's on each CPU board:

- *A receive complete queue*

This queue FIFO contains the addresses, lengths, buffer types, and error status of buffers that contain received data packets, received headers, and received "control" blocks. Each entry in this queue FIFO consists of the following "fields":

1. The address of the buffer in buffer memory.
2. The length of the received information in memory.
3. The type of block received (e.g. header, data, connection, disconnection).
4. The error status.
5. The type of buffer (i.e. H buffer vs D buffer).
6. The 16-bit 1's complement TCP/IP checksum.

- *Two free receive buffer queues*

These queue FIFOs contain the addresses of H and D buffers to be used by the "intelligent" DMA for subsequent receive operations. There are two free buffer queue Fifo's, one for H buffers (for headers) and one for D buffers (for data).

The "intelligent" DMA removes an entry from a free buffer queue FIFO at the beginning of each receive operation when it needs to determine where in buffer memory to place the data or the header.

- *A transmit data queue*

This queue FIFO contains the addresses and lengths of buffers that contain data packets and headers to be transmitted, as well as special "control" buffers to be sent to the I/O adapters for other functions (e.g. connection set up, etc.).

Each entry in this queue FIFO consists of the following "fields":

1. The address of the buffer in buffer memory.
2. The length of information to be transmitted via the DMA.
3. The function code for the buffer (e.g. header, data, connect, disconnect, etc.).
4. A tag field used by the software to keep track of the data block. This tag is returned in the transmission complete queue below.

The appropriate protocol processor builds these entries and places them in the queue FIFO when it has data or headers that need to be transmitted, or when it has special commands (e.g. connect) that it wishes to send to the I/O adapter.

- A *transmission complete* queue

This queue FIFO contains the addresses of header, data and control buffers just transmitted by the "intelligent" DMA.

Each entry consists of the following "fields":

1. The address of the transmitted buffer in memory.
2. The function code (from the Work Queue entry) of the buffer.
3. The error status (e.g. ok, parity error, connection rejected, etc.).
4. The software tag field set in the transmit data queue above.

2.1.6 Inter-Processor Communication Message Queue FIFO's

These message queue FIFO's are similar to the buffer queue FIFO's, however they are used for communication between the two protocol processors. There is one message queue FIFO for each direction of communication. Once again, these queues enhance performance by queueing up inter-processor communication information so that immediate software response is not required to prevent information loss. Fast inter-CPU communication is a requirement for most communication protocols, since the receiver and transmitter must be able to communicate for purposes of acknowledgment of received packets, for example.

Each FIFO is 32 bits wide, and the information transmitted is determined by the protocol software. This information consists of software "messages" to be passed from one processor to the other.

2.1.7 Inter-Processor DMA

The inter-processor DMA is used when it is necessary to transfer large amounts of information from the data memory (VRAM) of one protocol processor card to the other. At this point in time this feature is not used for protocol processing but is only used to download code from one processor to the other or for diagnostic/control purposes.

Each protocol processor card has a DMA controller that can be used for 3 purposes:

- Transferring data from this CPU VRAM to the other CPU VRAM (called *remote copy*)

- Transferring data from one location in the local VRAM to another location in local VRAM (called *local copy*)
- Calculating a checksum on an area of local VRAM without copying (called *checksum only*)

In addition, when the remote copy function is used, the DMA controller has a special function of copying an 8 word (32 byte) "message" from a special fixed location in local VRAM to another special fixed location in the remote VRAM, at the completion of the remote copy function. This message is used by the software to communicate command and control information associated with the block of data that was transferred via the DMA.

2.2 Optical Network Attach Section

The network attach section of the ONN consists of two cards, an optical/analog card, and a digital card.

2.2.1 Optical/Analog Card

The optical/analog card contains a distributed-feedback (DFB) laser, a fiber-Fabry-Perot tunable filter and an avalanche photodiode (APD) receiver. Each ONN in the network has a laser at a different wavelength and the laser is maintained at a constant temperature to avoid wavelength drift. The Fabry-Perot filter has a free spectral range of 32 nm and a finesse of about 100, and can tune from one wavelength to another in 25 milliseconds.

2.2.2 Digital Card

The digital card implements the *circular search* medium access control protocol used in Rainbow-I [4], and provides flow control in hardware.

Circular Search Protocol: The Rainbow-I network used a simple inband signalling protocol to set up and takedown connections between nodes in the network. Essentially in order to set up a connection a node continuously broadcasts a connection request message on its assigned wavelength. The intended destination, if idle, scans its tunable filter across all the wavelengths looking for such a request and locks to a wavelength if it sees such a request. It then sends back a connection accept message that the originator looks for while itself scanning across all the wavelengths. In Rainbow-I this protocol was implemented mostly in software. Rainbow-II implements this protocol entirely in hardware, mainly to relieve the protocol processing CPUs from performing this function.

Hardware Flow Control: Once a connection is set up between two nodes, flow control is required to prevent packets from being lost due to buffer overflows at the receiving node if it is unable to cope with the speed of the sender. A *sliding window* mechanism [13, Chap. 2] is commonly used for this purpose. At each instant the sender maintains the value of the maximum sequence number that he can transmit and the last acknowledged sequence number. Based on its available buffers the receiver sends tokens periodically to increase the value of the maximum sequence number that the sender is allowed to transmit. Usually such a scheme is implemented in software. Rainbow-II implements this entirely in hardware on the digital card, again to partially relieve the protocol processing CPUs from performing this function.

3 Software Design

The ONN software supports two modes of operation:

Figure 3: The Protocol Stack

(A) *Transparent* mode. In this mode, once a connection is established (using the HIPPI LE (link encapsulation) header to determine the destination), packets are passed through the ONN without modification. The protocol offload function is disabled in this mode. This mode allows existing TCP/IP or other protocols on the host to operate across the network end-to-end without any changes.

(B) *Protocol offload* mode. This mode allows the complexity of a full reliable transport protocol to be offloaded from the host computer to the ONN. The protocol stack used for this purpose is shown in Fig. 3. The host computer communicates with the ONN via a simple, lightweight protocol called SHIP (simple host intersocket protocol)[14] that implements the semantics of the Unix BSD sockets interface. A socket like library is provided to host applications. Thus host applications only have to be relinked (not rewritten or recompiled) to the new library in order to benefit from the performance of protocol offload. The ONN implements the reliable transport layer protocol internally to talk to the peer ONN. Currently, the transport protocol implemented between Rainbow II nodes is a simple, light-weight, easy-to-implement private protocol called OTP (optical transport protocol) which was used initially to save effort. Later, this may be replaced with a full TCP/IP implementation for two purposes: to connect with hosts and other networks running TCP/IP, and to compare the performance of a TCP implementation with the OTP implementation.

The main goal of the software was to eliminate as much software path-length as possible. This was done via the following design decisions:

- (1) No operating system or kernel is used inside an ONN. The ONN software runs completely stand-alone, specifically tailored for the ONN hardware.
- (2) No multi-tasking is used. Although this would have eased the construction of the software, it would have used precious CPU cycles. So the software runs in a "polling loop" implementation, where each function runs sequentially and cannot wait for an event or resource.
- (3) There are no interrupts under normal operation. This also reduces CPU cycle utilization and was made possible by the widespread use of FIFO's in the hardware design, thus eliminating the need for immediate response to events by the queueing function of the FIFO's.
- (4) In order to support different transport protocol software (currently OTP, future TCP/IP), the interface between SHIP and the transport protocol was kept as generic as possible. Subroutine

calls for opening and closing connections, transmitting packets, and handling errors are the only interfaces between the two protocols. There is no sharing of control blocks or other memory-based structures.

3.1 Simple Host Intersocket Protocol (SHIP)

The protocol offload hardware support in the ONN surfaces another problem, namely how to get the data between the computer running the application program (referred to below as the host) and the protocol offload hardware in the ONN. To solve this problem we developed the SHIP protocol [14] jointly with Intel. SHIP has been implemented on a variety of hosts (Cray YMP, Intel Delta and IBM RS6000) and two protocol processors (PPs) – the ONN and the crossbar interface (CBI) at LANL [15].

This immediately brings up the question “Why is SHIP less work for the host than the protocol being offloaded?” There are a number of answers:

- The connection between the host and the PP is a point-to-point dedicated connection, or a circuit-switched connection, not a packet-switched network. It may, in some future implementation, even be a memory-to-memory or internal bus connection to a PP that is part of the same computer as the host. Thus, there is no need in SHIP for routing, congestion control, packet fragmentation and reassembly and other protocol functions used in full network protocols, such as TCP/IP, or other proposed “lightweight” protocols that have attempted to solve a similar, but not identical problem.
- The connection between the host and the PP is assumed to be much more reliable, in terms of corrupted bits, than a “normal” network connection, allowing error recovery to be simpler, since the number of retransmissions is assumed to be very low and therefore not an area needing optimization. This, in conjunction with the point-to-point nature of the connection, allows for very large packet sizes, reducing the per-packet protocol overhead considerably.
- Since SHIP, by definition, will only be seen between the host and the PP, not on any external network, there is no need to support backward compatibility with a large zoo of existing protocols. This allows a number of simplifying design decisions to be made.

3.1.1 SHIP Design Assumptions

The SHIP protocol was designed with two important objectives: (1) It must allow existing application programs to run with as few changes as possible, preferably none. (2) It must be much less work for the host to process than the protocols being offloaded (TCP/IP).

The first objective led to the design decision that the SHIP protocol would basically be a protocol implementation of the sockets interface, which is widely used as the application interface to TCP. Ideally, applications that use sockets to access TCP would need only be relinked, or in some cases only need to specify different run-time parameters, and would then be using SHIP instead of TCP to talk to a PP that would then do the protocol processing. SHIP is therefore a protocol that essentially imitates the sockets interface by an actual protocol transmitted over a point-to-point connection.

The second objective led to a number of important design decisions:

- All SHIP headers are essentially of one common fixed length and fixed format, except for a few header types containing variable length extensions at the end.

- Most fields in the SHIP headers are 32 bit aligned fields.
- Data packets can be very large, potentially gigabytes in size. The actual size used is negotiated between the host and the PP on a per-transfer basis (i.e. not fixed per connection). Currently, due to buffer memory limitations in the ONN packets of 1 megabyte are considered typical.
- The protocol is a master-slave protocol, not peer-to-peer. The host is always the master, and the PP is always the slave. This has the slight disadvantage that SHIP cannot not be used as a lightweight transport replacement for TCP between hosts, but requires an intermediate PP. The reduction in overhead in both the host and PP more than offsets this disadvantage.
- A number of simplifying restrictions have been made, such as:
 - A host cannot send or request data out of order, except for repeating a transmission not yet acknowledged.
 - Data, once received by the PP and acknowledged to the host has not necessarily been received by the remote peer. It is the PP's responsibility to ensure reliable delivery.
 - Certain features of TCP, e.g. "Urgent Data" are not supported, for performance reasons.

3.1.2 SHIP Overview

The standard socket calls that are supported by SHIP are: *socket*, *bind*, *connect*, *listen*, *accept*, *shutdown*, *select*, *getsockopt*, *setsockopt*, *read*, *write*, *send/sendto*, *recv/recvfrom*, and *close*. The blocking and non-blocking versions of the appropriate commands are also provided.

The SHIP packet format itself consists of a fixed format header, consisting of 10 32-bit words, optionally followed in some cases by a variable-length extension, which must be an even integral number of 32 bit words (in order to support some link implementations that are 64 bits in width, such as 1600Mbit HIPPI).

The fixed fields in the SHIP header are:

Function Code This is a 16 bit field (in the first 32-bit word) that indicates the specific function (e.g. Create, Read, etc.) that the packet is calling for.

Host Id This is an 8 bit field (also in the first 32-bit word) that is used when a PP is attached to multiple hosts.

Status This is an 8 bit field (also in the first 32-bit word) that is used by the PP to return any protocol errors to the host.

Request Id This is a 32 bit word that is a unique number specified by the host for each Requested function, and returned by the PP with each reply, allowing the host and the PP to detect retransmissions.

Host Socket ID This is a 64 bit number used by the host to match the replies from the PP to its own internal socket identifiers. It is sent by the host in each packet, and is returned by the PP in each reply. The PP has no other use for the field, and never interprets or modifies it.

PP Socket ID This is a 64 bit number, used by the PP to match the requests from the host to the PP's own internal socket identifiers. It is initially returned to the host when a socket is

created (via the Create function) and is then sent by the host in each request packet for that socket. The host neither interprets nor modifies this field.

Note: The use of the above two fields to identify sockets allows the host and the PP implementations to use their own optimized identifiers for sockets, allowing improved performance in the implementations.

Extension Length This is a 32 bit word that contains the length of the optional variable length extension to the SHIP header. In most SHIP packets it is zero. Only a small number of SHIP packets require more information than can fit in the fixed portion of the header, and these use this field.

Data Length This is a 32 bit word that contains the length, in bytes, of the Data portion of the packet, which immediately follows the header (for data transfer packets only, such as Read or Write).

Command Status This is a 32 bit word that is used by the PP to return any errors occurring in the execution of the specific socket command being issued (e.g. Read, Accept) as opposed to SHIP protocol errors which are returned in the Status field.

Parameter Word This is a 32 bit word that is used for various parameters that differ for different SHIP functions. Since a number of functions needed at most one parameter, it was decided to have this word in all headers, rather than to force many headers to use the variable-length extensions just for a single word. An example of the field's use is for the READ command SHIP header, in which this field contains the size of data being requested.

As can be seen, the SHIP headers were designed to make the header processing as simple as possible for the host and the PP, without too much concern for the possible extra transmission overhead that might be associated with 32-bit fields and fixed-length headers. Since the whole purpose of SHIP was to allow operation on very-high speed networks, this small transmission overhead is insignificant.

3.2 Optical Transport Protocol (OTP)

The OTP (Optical Transport Protocol) is a very simple protocol used as a transport layer protocol between Rainbow II nodes. This protocol was used because it was very easy to implement and has a potential for high performance. Standard transport layer protocols such as TCP were not initially needed, since Rainbow II nodes can only talk to other Rainbow II nodes, so the transport Layer protocol used on the Rainbow II network is never surfaced to any other entity.

The OTP protocol is a go-back-N protocol using fixed length packet headers and is capable of using large packet sizes.

The go-back-N error recovery strategy is not the most efficient, but it is easy to implement, and works well because the optical links have a very low error rate and packets are not dropped due to buffer overflows because of the hardware flow control feature.

A fixed length packet header with fixed length 32 bit fields is used. This allows for fast simple access to packet header information. In fact, to simplify the implementation, the OTP packet header is identical in size, and layout to the SHIP packet header, with a few fields used for different purposes.

OTP allows packets to be very large (megabytes), allowing for larger packet processing times. This is a potential disadvantage if the error rate is high, since retransmissions use up a lot of bandwidth, and the probability of a packet being received in error is higher. However as the

Figure 4: The LANL testbed.

Rainbow II error rate is low, this was judged not to be a problem. The implementation of OTP uses this packet size for another advantage, namely the incoming SHIP packets are converted directly to OTP packets by merely changing a few fields in the header, with no packet disassembly or reassembly required.

4 Status and Experimental Results

The Rainbow-II network was demonstrated at Supercomputing'94 and at the Optical Fiber Communication Conference'95. The demonstration had three RS6000 workstations, each connected to an ONN. All the ONNs operated in transparent mode. Two nodes served as video servers, sending out high resolution images. The third node served as a receiver and could establish connections to each of the other nodes across the network. Each image was 5 MBytes and about 12 images were transmitted per second across the network on a connection as 1 MBytes packets. The resulting 60 MBytes/sec throughput was limited primarily by the speed at which we could get data out of the RS6000s and not by the ONNs.

Rainbow II is currently deployed at LANL. LANL is evaluating the performance of the Rainbow-II network against established gigabit testbeds, such as the LANL HIPPI Testbed, by comparing data-transfer rates between supercomputers/workstations and other high-end HIPPI-based systems. The LANL testbed network in its current form is shown in Fig. 4. It consists of two ONNs interconnected by a passive star coupler (with a third yet to be connected). The ONNs are not directly connected to host machines. They are instead connected through the existing HIPPI network to either IBM RS6000 workstations or to Cray YMP machines.

Fig. 5 shows the test results for sending raw data through the ONNs in transparent mode. The data was sent from an RS6000 through the ONNs and received back by the same RS6000. The data is taken from five runs with each run consisting on sending 100 packets between the two machines. The *maximum* indicates the data rate obtained for the "best" run, the *minimum* is the data rate obtained for the "worst" run and the *average* is the data rate obtained by averaging over all the

Figure 5: Raw data transfer rates between two RS6000 workstations across the Rainbow-II network. The ONNs are operated in transparent mode.

runs. The *read* rate is the rate at which a host receives data from an ONN and the *write* rate is the rate at which a host can send data to an ONN. As expected, the data rates increase with the packet size since there are fewer packets to be processed per unit time, and for large packets we are able to obtain about 30-35 MBytes/s data rates on average. This is consistent with the 60 MBytes/s rates we were able to get earlier between two RS6000s, and again is limited by the RS6000s and not by the ONNs.

Fig. 6 shows the test results for data transfer between two Cray YMPs. In addition to the data read and write rates we also plot two additional parameters of interest: (a) *CPU utilization*, which is defined as the ratio of the CPU time taken to transfer the data to the total time taken, and (b) the *data transfer efficiency*, which is the the data rate divided by the CPU utilization, or in other words the total amount of data transferred divided by the CPU time taken. In each plot, one curve is from an experiment wherein the Crays are connected through the ONNs with the ONNs operated in protocol-offload mode. The Crays and the ONNs run SHIP in this case. The other curve corresponds to an experiment wherein the two Crays are connected to each other through the HIPPI network without going through the ONNs. The Crays run TCP in this case. In each case the data is averaged from several runs.

Observe that the SHIP write rate is higher than the read rate. This is probably because the two Crays were operating under different loads at the time of the tests. We are verifying this further currently. The maximum obtained SHIP write rate during a run was 78 MBytes/s which is close to the 100 MBytes/s maximum achievable rate across a HIPPI connection, given that there is some added overhead for framing and interpacket delays.

Observe that the average TCP transfer rate of around 55 MBytes/s is almost independent of the packet size. This is because the maximum internal packet size of TCP is 64 KBytes and larger packets are segmented and actually sent in 64 KBytes blocks. Note that for large packet sizes, SHIP achieves higher transfer rates and lower CPU utilizations, or equivalently, higher transfer efficiencies, than TCP. This illustrates the benefit of protocol offload. There are three reasons for this: (a) SHIP uses larger-sized packets (up to 1 MByte in the current implementation) without segmenting them into smaller packets, (b) SHIP has no window-size limitation, while the TCP window size on the Crays is 500 KBytes, and (c) SHIP is inherently designed to be a more efficient protocol than TCP. Given (c) it is surprising that the SHIP transfer rate is lower than the TCP transfer rate for small packet sizes. One reason for this may be that TCP is implemented very efficiently in the kernel of the Cray, while SHIP is currently an application program that has not

Figure 6: Data transfer rates, CPU utilization and data transfer efficiency between two Cray YMP machines. The curve labelled "SHIP" corresponds to the machines connected through two ONNs in protocol-offload mode, running SHIP. The curve labelled "TCP" corresponds to the machines connected to each other directly through the HIPPI network, running TCP.

yet been tuned to obtain the best performance. We are exploring this further.

5 Future Work

LANL plans to run actual applications across the ONNs using both transparent and protocol-offload modes. The transparent mode is more useful for some applications that do not use the Crays (for instance Global Climate Modeling) and that transfer non-IP (internet protocol) data between the Thinking Machines CM-5 and a High Performance Data System (HPDS) which is based on a large IBM Disk Array. Another one is a virtual reality (VR) application running between the Crays and an SGI workstation. This application will transfer data from the Cray at 800 Mb/s and use the SGI to render the images for subsequent display. The high speed link is used to speed up the overall VR processing. LANL is currently interested in exploring technologies for 10 Gb/s MANs to support some evolving applications and networks like Rainbow-II offer the potential to be scaled up to achieve these rates.

The ONN has been designed to support a variety of host interfaces as well as to support optical packet switching in future as rapidly tunable filters become available. Among the 5 cards in an ONN, only the network attach analog and digital cards need to be changed. The ONN software and the rest of the hardware would be unchanged. We are currently developing components and protocols that will enable us to support optical packet switching.

References

- [1] N. K. Cheung, G. Nosu, and G. Winzer, eds., *IEEE JSAC: Special issue on Dense WDM Networks*, vol. 8, Aug. 1990.
- [2] M. J. Karol, C. Lin, G. Hill, and K. Nosu, eds., *IEEE/OSA J. Lightwave Tech.: Special Issue on Broadband Optical Networks*, May/June 1993.
- [3] P. E. Green, *Fiber-Optic Networks*. Prentice Hall, 1992.
- [4] F. J. Janniello, R. Ramaswami, and D. G. Steinberg, "A prototype circuit-switched multi-wavelength optical metropolitan-area network," *Proc. ICC'92*, pp. 818-823, 1992. Also to appear in *J. Lightwave Tech.*
- [5] American National Standards Institute, X3T9.3, *High-performance parallel interface*, Nov. 1989. Rev. 6.9.
- [6] American National Standards Institute, X3T9.3, *Fibre channel: Signalling protocol (FC-2)*, Nov. 1994. Rev. 4.2.
- [7] W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin, and R. Williamson, "A survey of light-weight transport protocols for high-speed networks," *IEEE Trans. on Commun.*, vol. 38, pp. 2025-2039, Nov. 1990.
- [8] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications*, vol. 27, no. 6, pp. 23-29, 1989.
- [9] H. Kanakia and D. R. Cheriton, "The VMTP network adaptor board (NAB): High-performance network communication for multiprocessors," in *Proc. ACM SIGCOMM'88*, pp. 175-187, 1988.

- [10] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, and B. D. Zill, "Protocol implementation on the Nectar communication processor," in *Proc. ACM SIGCOMM'90*, pp. 135-144, 1990.
- [11] M. Kaiserswerth, "The parallel protocol engine," *IEEE/ACM Trans. on Networking*, vol. 1, pp. 650-663, Dec. 1993.
- [12] D. C. Feldmeier, "An overview of the TP++ transport protocol project," in *High-Performance Networks - Technology and Protocols* (A. Tantawy, ed.), Kluwer Academic Publishers, 1993.
- [13] D. Bertsekas and R. G. Gallager, *Data Networks*. Prentice Hall, 1992.
- [14] B. Beattie, M. G. Halvorson, C. Idler, J. Kravitz, M. Sukalski, and R. Thomsen, *SHIP-Simple host intersocket protocol*, Feb. 1995. Working document.
- [15] C. Idler and R. Thomsen, "The LANL crossbar interface: Functions and performance," in *Proc. Networld+Interop*, March 1995.

G. Appendix: ONA Operation at Los Alamos National Laboratory

Marty Halvorson
Los Alamos National Labs
Los Alamos, NM 87545

29 November 1995

Summary

This report concerns the operation of the ONAs installed at Los Alamos National Labs. The testing done here at LANL, the results obtained from the tests, and the steps necessary to make all tests operate will be covered. These tests included raw data using an IBM RS6000, TCP/IP and SHIP using Cray Y/MPs.

1. Testing Results	1
1.1 Introduction	1
1.2 Raw Data Results	1
1.3 Protocol Testing	3
A. TCP Results	3
B. SHIP Results	4
2. Operation	6
2.1 ONA	6
A. Hardware	6
B. Controlling PC and Software Operation	7
C. File Names	8
D. I-Fields and Addresses	8
2.2 IBM RS6000	10
2.3 Cray Y/MP	10
A. TCP Operation	11
B. SHIP Operation	12
2.4 Helper Software	14
3. Software Archive	15

1. Testing Results

1.1 Introduction

There were two kinds of tests run on the ONA's. The first consisted of sending raw data through them via the installed HIPPI Test Bed here at LANL using an IBM RS6000 as both the send and receive host. These tests were primarily to verify the ONA's were capable of sending and receiving data via the optical link.

The second set consisted of sending protocol data through the ONA's via the HIPPI Test Bed using two Cray Y/MP's as the hosts. The tests used two protocols to deliver data. The first used the Cray standard implementation of TCP/IP using the maximum available window size (approximately 500 KB) to assure the maximum data rate was obtained. The second protocol used the LANL designed SHIP protocol. This protocol off-loads the network routing connection set-up and control to an outboard protocol processor (the ONA's). The Cray implementation of this protocol was not optimized to maximize throughput.

1.2 Raw Data Results

This test involved sending data in the transparent mode of the ONA's from machine *acti* in the ACL through the HIPPI test bed to an ONA. The S(ource)ONA sent the data to the requested D(estination)ONA. The DONA in turn sent the data back to machine *acti*, again using the HIPPI test bed. In the simplest case, the SONA and the DONA were also located in the ACL. In other cases, either the SONA or DONA were located in the LDCC building, room 205 and the ACL. In these cases, data was transferred between the ONA's via a pair of single mode optical fibers approximately 1 km long that contained several fiber-to-fiber connections.

Raw data rates were limited by the speed of the RS6000, rather than the speed of the ONA's. In this test, the size of the packet transmitted via the ONA's and the data set size are the same. Eight different size packets were sent.¹ Data rates were computed separately for each direction.²

Raw Data Rates³

¹ 32KB, 64KB, 128KB, 256KB, 512KB, 1024KB, 2048KB, and 4096KB

² Read rate calculation did not include any latency information (i.e. time was started when the first packet was sent or received and stopped on receipt or transmission of the last packet) and thus was not dependent on write rate.

Read Data Rates

Packet Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
32 KB	14.6 MB/s	12 MB/s	13.5 MB/s
64 KB	22.9 MB/s	18.7 MB/s	21.4 MB/s
128 KB	28.8 MB/s	28.4 MB/s	28.6 MB/s
256 KB	30.5 MB/s	28.1 MB/s	29.8 MB/s
512 KB	31.8 MB/s	29.7 MB/s	31 MB/s
1024 KB	332.2 MB/s	29.8 MB/s	30.8 MB/s
2048 KB	32.5 MB/s	28.6 MB/s	31.3 MB/s
4096 KB	32.5 MB/s	31.6 MB/s	32.3 MB/s

Write Data Rates

Packet Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
32 KB	18.6 MB/s	14.9 MB/s	16.4 MB/s
64 KB	28.6 MB/s	26.5 MB/s	27.3 MB/s
128 KB	32.9 MB/s	32.2 MB/s	32.6 MB/s
256 KB	33.7 MB/s	30.9 MB/s	32.9 MB/s
512 KB	34.8 MB/s	32.3 MB/s	33.8 MB/s
1024 KB	35.0 MB/s	31.3 MB/s	33.7 MB/s
2048 KB	35.1 MB/s	30.9 MB/s	33.9 MB/s
4096 KB	35.0 MB/s	33.9 MB/s	34.7 MB/s

1.3 Protocol Testing

There were two different protocol data rate test sets run. One set was based on the standard implementation of TCP/IP. The other was based on an application library that implemented SHIP and used raw data transfer operation to send packets. Both test sets used the same application source code to send packets. The only differences were:

- Function names of the calls involving test data were different, the SHIP calls had an "S_" prepended to the standard socket function call.⁴
- The name lookup mechanism for SHIP used a special function that provided the necessary ONA formatted network addresses. Whereas TCP used calls to the standard UNIX system call `gethostbyname(...)` to obtain the network addresses.

⁴ Note that the item measured is in packet sizes

It must be clearly understood that although the tests appear to compare rates achievable with TCP to rates achievable with SHIP this is not really the case. TCP, in the Cray, runs in kernel space, and was optimized to provide maximum throughput for maximum TCP sized packets. SHIP, on the other hand, was written to provide a diagnostic environment for development of the SHIP protocol, executes entirely in application space, and was minimally optimized for throughput. In the design of SHIP, the intent was to provide a mechanism for network challenged hosts⁵ and for large data set⁶ transfers to be done in an efficient manner. As such, it can not be compared to TCP/IP, since that protocol is designed for reliable transfer of small packets.⁷

A. TCP Results

The first result shown are the data rates obtained. Then, utilizing a dimensionless number, the efficiency of TCP transfers. This number is computed by dividing the total data sent by the amount of CPU time (application plus time spent in behalf of the application by the system).

⁴e.g. S_socket (...) for socket creation

⁵e.g. CM-200

⁶nMB, where n is at least 1

⁷Maximum of 64KB

TCP Rate and Efficiency Tables⁸

Read Data Rates

Data Set Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
8 KB	49.4 MB/sec	0.1 MB/sec	38.1 MB/sec
32 KB	64.1 MB/sec	3.5 MB/sec	57 MB/sec
64 KB	63.2 MB/sec	8.3 MB/sec	58.8 MB/sec
256 KB	61.9 MB/sec	8.2 MB/sec	56.9 MB/sec
1024 KB	61.1 MB/sec	7.8 MB/sec	28 MB/sec

Write Data Rate

Data Set Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
8 KB	55.7 MB/sec	0.1 MB/sec	20.9 MB/sec
32 KB	59.6 MB/sec	3.5 MB/sec	49.7 MB/sec
64 KB	61.7 MB/sec	8.3 MB/sec	56.7 MB/sec
256 KB	61.7 MB/sec	8.2 MB/sec	57 MB/sec
1024 KB	61.3 MB/sec	7.8 MB/sec	28 MB/sec

TCP Efficiency

Data Set Size	Write	Read
8 KB	47.5	41.2
32 KB	98.4	83.9
64 KB	104.3	104.9
256 KB	122.4	105.8
1024 KB	129.7	104.1

B. SHIP Results

The results shown for SHIP contain the same information as shown for TCP. i.e. Data rate in mega-bytes per second, and a dimensionless number indicating efficiency.

SHIP Rate and Efficiency Tables⁹

⁸ Note that the item transferred is measured in data set size, which in the case of TCP is not necessarily the size of the packet transmitted.

⁹ Note that the item transferred is measured in data set size, which in the case of SHIP is also the packet size.

Read Data Rates

Data Set Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
8 KB	2 MB/sec	.2 MB/sec	1.6 MB/sec
32 KB	7 MB/sec	.8 MB/sec	6 MB/sec
64 KB	13.3 MB/sec	2.3 MB/sec	11.3 MB/sec
256 KB	35.1 MB/sec	28.5 MB/sec	30.9 MB/sec
1024 KB	48.2 MB/sec	33.7 MB/sec	41 MB/sec

Write Data Rates

Data Set Size	Maximum Data Rate	Minimum Data Rate	Average Data Rate
8 KB	4.2 MB/sec	3.4 MB/sec	3.8 MB/sec
32 KB	14.6 MB/sec	11.2 MB/sec	13.2 MB/sec
64 KB	24.7 MB/sec	17.8 MB/sec	22.3 MB/sec
256 KB	54.7 MB/sec	38.8 MB/sec	47.5 MB/sec
1024 KB	77.6 MB/sec	51.9 MB/sec	64 MB/sec

SHIP Efficiency

Data Set Size	Write	Read
8 KB	11.5	4.1
32 KB	46.6	16
64 KB	94.3	32.7
256 KB	380	119
1024 KB	1,482.6	388.5

2. Operation

There are a number of aspects of the operation of the ONA that need to be considered. First is the fact that ONA's do not use any other existing addressing scheme. Rather they depend on either: 1) The HIPPI-LE destination IEEE address field to contain an index to a table internal to the ONA¹⁰, and, currently, no implementation of HIPPI-LE makes use of the destination IEEE address field. 2) The SHIP address structure containing the same index to the internal table. As originally intended, the SHIP address structure would contain the same destination IP address as used by TCP.

2.1 ONA

A. Hardware

The ONA hardware consists of, left to right when facing unit, a power supply internal to the cabinet. A HIPPI interface board, two identical CPU boards, an I/O board, and an optical board.

On the front panel of the cabinet is the only mechanical control for the ONA, a 'reset' button.¹¹

The HIPPI interface board has two connectors, one for each direction of a HIPPI connection, marked accordingly.

Each of the CPU's has two connectors. One, marked 'P', is a PC parallel port interface.¹² This connector allows the attached control console, a PC, to load firmware, I-field tables, and script files; and well as, to receive dump files of various types. The other connector, marked 'S', is a standard RS-232 serial interface. This connector allows the control console to act as a terminal for the already loaded firmware. Either CPU board is capable of controlling the entire ONA.¹³

The I/O board has four connectors marked 'TX', 'TY', 'RX', 'RY'. Each of these connectors is cabled to the corresponding labeled connector on the optical board.

The Optical board has eight electrical connectors and two optical connectors. The connectors marked 'TX', 'TY', 'RX', 'RY' go to the respective I/O board connectors. The connectors marked 'IX', 'IY', 'OX', 'OY' are not used. The optical connectors provide for the two directions of an optical connection.¹⁴

¹⁰ The internal table used by the ONA's are: 1) different for each ONA CPU board, and 2) must be loaded each time the ONA is reset.

¹¹ Frequent pushing of this button is usually necessary because the firmware was not completely debugged.

¹² The parallel port on the PC must be capable of bi-directional operation.

¹³ There is one exception to this. When taking dumps, the parallel port must be connected to the CPU sending the dump data.

¹⁴ The optics were designed to have -16 dB optical power as compared to the transmitter. Therefore, either the optical outputs need to be attenuated, or, as is the current case, optical output needs to be sent to the optical multiplexer.

Each ONA has an identifying number. These numbers are critical to know. They are found by reading the switch settings on the CPU boards.¹⁵ This number is used by all ONA's to route packets to the other ONA's. The number read from the switch plus one is the ONA number. The ONA number multiplied by eight identifies a destination ONA. We have ONA's 1, 2, and 4. Thus the destination ONA numbers are: 8, 16, and 32.

B. Controlling PC and Software Operation

Although not necessary for normal operation of the ONA, the control console is required to:

1. Load the firmware
2. Act as the operators terminal
3. Load the internal I-field tables
4. Receive dump files

A standard PC acts as this control console. It is preferable that the PC be running a version of Windows that supports multi-tasking, as this allows one to see the results of loading the firmware or taking dumps while they're occurring.¹⁶

The operators terminal user manual follows in the next section of this book.¹⁷

Following the operators user manual is a hardware specification manual.¹⁸

Two tasks are required. The first is a DOS prompt task, the second is a terminal task.

The DOS prompt executes the `onaserv` command which is the ONA server program. In a multi-tasking Windows system, the ONA server is left running.¹⁹ The ONA server provides (see operators manual), for complete control over loading and executing the firmware.²⁰

The terminal task provides an interface to the executing ONA firmware using the PC serial port²¹. There are a fairly complete set of commands understood by the firmware for activating trace operations, initiating dumps, viewing statistics, and etc. These are explained in the ONA User Manual.²²

¹⁵ These numbers must be the same on both CPU boards in an ONA.

¹⁶ The PC attached to the ONA in the ACL is running Windows '95. The password can be obtained by asking the author.

¹⁷ Be warned that the level of detail in the operators manual is less than optimal. Some commands are noted as being present but are not documented further. For example, the debug command is shown, but the format of the dump data is not specified, nor is the specification of how dump file names are generated.

¹⁸ This manual doesn't seem to really indicate how the hardware works however. It is useful when looking at trace and error dumps to determine the meanings indicated by values in hardware registers included in the dumps.

¹⁹ No `-x` flag on command line

²⁰ Except when reloading only one CPU, the flag `(-c)` to load both CPU's should be set.

²¹ Which must be connected to the 'S' port on the ONA CPU with a 'null' modem cable.

C. File Names

In the console PC there are a number of files required by an ONA.

onaserv.exe is the file containing the server code. This is the code executing in the DOS prompt window.

onamain is the firmware file loaded by the ONA server.

For each ONA there are two files sent to the newly loaded firmware as command files. The names of these files all take the form of:

`ona[a | b][01 | 02 | 04].cmd`

Where a | b refers to the CPU ID,²³ and 01 | 02 | 04 refer to the ONA number.

When dumps are taken²⁴ the file names for the dumps take the form of:

`[aotb | trace | ssc][a | b][01 | 02 | 04].dmp`

Where aotb | trace | scb refer to the type of dump, a | b refer to the CPU ID, and 01 | 02 | 04 refer to the ONA number.²⁵

On your favorite workstation there are some helper files for interpreting the dumps:²⁶

- ona-analyzer.data contains the conversion of dump data ID's to their meanings.
- ona-aotb* are the source and executable for decoding the aotb dump files.
- ona-ssc* are the source and executable for decoding the ssc dump files.
- ona-trace* are the source and executable for decoding the trace dump files.

D. I-Fields and Addresses

The ONA has two modes, SHIP mode (AKA protocol off-load mode), and pass through. Both modes require that the HIPPI input port deliver a packet containing a HIPPI-FP header. The ULP_ID field of the header is used to differentiate between the two modes.²⁷

²² But not really completely. Some experimentation is in order to find out what everything really does.

²³ CPU are identified by the letters 'a' and 'b' for left and right CPU board, when viewed from the front, respectively.

²⁴ dump command

²⁵ Since I wasn't debugging the firmware, the only dumps I found useful were trace dumps. And only then after I discovered the hardware definitions in the hardware specification.

²⁶ For trace dumps the method is to: 1) find the trace event of interest by executing the ona-trace program, this will provide an ID containing a type field and a hexadecimal value associated with that type, this can be looked up in ona-analyzer.data to determine the ID's meaning and source code module where event is generated, 2) find the reference to the event in the code to determine the contents of the traced event, then 3) look up the meaning of the contents in the hardware specification.

²⁷ 0x02, 0x03, 0x82, and 0x83 specify SHIP mode, all others assume pass through

In pass through mode, an assumption is made that the position occupied by the destination IEEE address in a HIPPI-LE header contains the destination for the packet.²⁸ In HIPPI network byte order (big endian) the most significant byte of the 48 bit IEEE address field (i.e. the least significant byte of the second 32 bit word of the field which is the sixth word in the packet) contains the index to the I-field that will be used by the ONA sending the packet on via HIPPI (i.e. the destination ONA), and the also the destination ONA.^{29, 30}

In protocol off load mode, the host attached to the ONA has as it's host ID the destination ID number of the ONA to which it's attached.³¹ The SHIP address structure has the destination host encoded in the low order byte of the 32 bit word of the SHIP address structure address word.

²⁸ Irrespective of whether it's an -LE packet or not.

²⁹ This byte is divided into a destination ONA field (bits 3-7), and an I-field index (bits 0-2).

³⁰ As noted previously, the I-fields are loaded either manually, or at firmware load time. The destination ONA number was also explained previously.

³¹ The I-field number part of the destination number must be zero, as the ONA uses the I-field at index zero to send packets to it's host.

2.2 IBM RS6000

RS6000 operation is primarily one of verifying that the ONA's can communicate and pass data correctly over the optical connections between each ONA. As such the programs are quite simple and straightforward. There are three programs:

- onatest provides both sides of the read/write operation to the ONA³²
- fpwr provides a write only program³³
- fprd provides a read only program

Do a make to build the executables.

The test program has several parameters:

- l <n> Sets the size of the D2 data in Kbytes. Default is 1024 (i.e. 1MB)
- n <n> Sets the number of iterations. Default is 5.
- s<n> Selects the source ONA. Default is 1.³⁴
- d<n> Selects the destination ONA. Default is 2.
- r Tells program to not run the read section. Default is run read.³⁵
- w Tells program to not run the write section. Default is run write.
- v Run in verbose mode. Default is quiet mode.

Other than making sure the ONA's are properly connected to the HIPPI Test Bed, there are no special operations that need to be performed to run the test. Output is returned to two files named: read-<date>.ona contains the read rate data for each iteration of the test program, whereas write-<date>.ona contains the write rate data.

2.3 Cray Y/MP

The programs executed for the two types of testing done on the Cray are created from the same source files. Each type has it's own version of a make file.

The parameters are the same for TCP, SHIP, and ONA-SHIP versions.³⁶

³² It is suggested that this be the only program used as it is the only version that produces output compatible with the data extraction program.

³³ Both the read and write programs are derived from the onatest program.

³⁴ This and the destination ONA parameter also select the I-field to be used to connect to the ONA. These I-fields are defined in file onatest.h. Destination ONA values are also defined in this file.

³⁵ This parameter and the -w parameter are redundant because the separate programs are available.

³⁶ With some variations in the values allowed.

Common parameters:

- i Display information about created sockets. Default = False.
- r <str> Name of result data file. Default = [read | write]-rate-results
- D Display debug information. (SHIP only: display more than you ever want to know about what's going on in the Cray) Default = False.
- E SHIP only: Display Cray specific details on detected errors. Default = False.
- R Display results on standard output in addition to writing to result file. Default = False.
- S <n> Scale factor (AKA TCP window size). Default = maximum obtainable.
- T <str> Transfer type. Default = type of transfer (TCP, ONA-SHIP, ONA, SHIP)

Write only parameters:

- h <str> Name of host to which data is to be transferred. Required.³⁷
- n <n> Number of data sets to transmit. Default = 1.³⁸
- p <str> Name of packet descriptor file. Default = packet-descriptors.³⁹
- s <n> Packet size in K bytes. No default.⁴⁰

Output is in the form of a number of lines containing the information necessary to compute overall rates using the helper software.

```
[Read|Write] <transfer type> started <date and time>
Receiver: <name>, transmitter: <name or address if known>
Packet: size = <size>, count = <count>
Ratio of Application+System time to wall clock time: <value>
Type           Time           Rate
Application: <time> <rate> MB/s
System: <time> <rate> MB/s
Wall clock: <time> <rate> MB/s
```

A. TCP Operation

1. Making executables

Issue the normal UNIX make command. This will produce two executables.⁴¹

- write-packets transmits data sets to the destination defined by the -h parameter.
- read-rate reads the data sets sent from the source.

³⁷ For ONA operation the host names are in file HostConfigFile located in the same directory as the executable. This file maps host names to destination ONA numbers. This is analogous to the TCP requirement that the sender know the IP address of the recipient.

³⁸ Setting data set size or count overrides any packet descriptor file input.

³⁹ This file describes the data sets to be sent during the operation of this execution. It contains the number and size of data sets to send.

⁴⁰ Data set size is obtained from either the -s parameter or from the packet-descriptor file.

⁴¹ There are no known 'gotchas' in making the TCP versions of the test suite, other than being

2. Operation

Issue the read-rate or write-packets command appropriate for the operation on a given host.⁴² The test will send/receive the designated data sets, compute the resultant rates, elapsed times, and the dimensionless efficiency number for each data set group. This information will be written to the appropriate (write-rate-results or read-rate-results) file.^{43,44}

B. SHIP Operation

1. Making Executables

Making executables for SHIP is quite a bit more complicated than making TCP executables. There are three stages involved.

1. Building the Cray specific library.
2. Building the SHIP library.
3. Building the executable.

In each case, there is a make file to accomplish this task. In addition to the three separate kinds of make, there are also two separate kinds of SHIP. There is ONA SHIP and standard (e.g. CBI) SHIP. Thus there are a total of 6 make files involved.^{45,46} There are also a total of three (four) directories involved. One for the executable, one for the SHIP and test program source, one for the Cray source, and one for the Cray specific host type library (plus one for utility sources).

a. Making standard SHIP executables

aware that the source directories are hard coded in Makefile.

⁴² It's a good habit to get into issuing the read first. This is because the Cray in SHIP mode of operation requires an outstanding read before it will not throw away any packet arriving that does not have a known ULP_ID in the -FP header. SHIP mode in the Cray is actually raw data mode as far as Unicos is concerned.

⁴³ It's usually a good idea to rename the result file to something more descriptive of the operation. I have been using tcp2tcp, tcp2ship, and ship2tcp as suffixes to denote the type of transfer done by the test writing the data.

⁴⁴ The file can then be sent to where-ever rate calculations are being done for processing results from multiple test runs.

⁴⁵ And one shell script, ship.bldenv, that needs to be executed to provide the correct definition of what kind of Cray (Y/MP or X/MP) is the target host.

⁴⁶ Because there only one library, cray-ypm/libship.a, it is necessary to force compilation of all SHIP and all Cray specific code when building a new version of the executables. This can be done by touching *.h in both the Cray directory and in the SHIP source directory.

Assuming that the directory tree structure remains as it currently exists. Starting from the directory that will contain the executables:

```
cd host                # Change to SHIP source directory
touch *.h              # force compilation of all SHIP source
cd cray                # change to Cray source directory
source ship.bldenv     # get definition of Cray machine type
touch *.h              # force compilation of all Cray source
make -f ship.mk         # build Cray specific library in the
                        # directory for the Cray machine type
make -f libship.mk     # add SHIP to library
cd ../..               # go to directory containing executable
make -f ship.mk        # make executables
```

This will create two executables, read-rate, and write-packets. These two programs operate in exactly the same fashion as the TCP programs of the same name.⁴⁷

a. Making ONA SHIP executables

Assuming that the directory tree structure remains as it currently exists. Starting from the directory that will contain the executables:

```
cd host                # Change to SHIP source directory
touch *.h              # force compilation of all SHIP source
cd cray                # change to Cray source directory
source ship.bldenv     # get definition of Cray machine type
touch *.h              # force compilation of all Cray source
make -f onaship.mk     # build Cray specific library in the
                        # directory for the Cray machine type
make -f libship-ona.mk # add SHIP to library
cd ../..               # go to directory containing executable
make -f onaship.mk     # make executables
```

This will create two executables, ona-read, and ona-write. These two programs operate in exactly the same fashion as the TCP programs read-rate and write-packets respectively.

2. Operation

Issue the read-rate or write-packets command appropriate for the operation on a given host.⁴⁸ The test will send/receive the designated data sets, compute the resultant rates, elapsed times, and the dimensionless efficiency number for each data set group. This information will be written to the appropriate (write-rate-results or read-rate-results) file.^{49, 50}

⁴⁷ Except that the read-program can only be given one group of data sets because the CBI implementation always sends all the data it has available rather than what the host asked for. This means that every read in SHIP must have a buffer large enough to hold the maximum size packet the CBI is capable of sending.

⁴⁸ It's a good habit to get into issuing the read first. This is because the Cray in SHIP mode of operation requires an outstanding read before it will not throw away any packet arriving that does not have a known ULP_ID in the -FP header. SHIP mode in the Cray is actually raw data mode as far as Unicos is concerned.

⁴⁹ It's usually a good idea to rename the result file to something more descriptive of the

2.4 Helper Software

find-rate reads the raw result files produced by the test suite and creates another file containing the summary information.

This program has two parameters. One is required.

```
find-rate <input file name> -d
```

Where input file name is the name of the file containing the raw data results.⁵¹ And, -d turns on debug mode which causes something to be displayed on standard out for nearly every line of input.⁵²

The output file contains an information record for each data set size found in the raw data differentiated by whether it is a read or a write.⁵³

```
[Read|Write] <transfer type> for packet size <size>, <n> total  
packets in <m> sets
```

```
Data was generally taken in the [morning|afternoon] (average hour  
= <n>)
```

```
Utilization: Average = <util>, Max = <util>, Min = <util>
```

```
Efficiency: <effic>
```

```
Wall clock rates and times
```

```
Rate: Avg = <rate>MB/s, Max = <rate>MB/s, Min = <rate>MB/s
```

```
Time: Avg = <time>sec, Max = <time>sec, Min = <time>sec
```

```
System resources spent on application rates and times
```

```
Rate: Avg = <rate>MB/s, Max = <rate>MB/s, Min = <rate>MB/s
```

```
Time: Avg = <time>sec, Max = <time>sec, Min = <time>sec
```

```
Application rates and times
```

```
Rate: Avg = <rate>MB/s, Max = <rate>MB/s, Min = <rate>MB/s
```

```
Time: Avg = <time>sec, Max = <time>sec, Min = <time>sec
```

operation. I have been using tcp2tcp, tcp2ship, and ship2tcp as suffixes to denote the type of transfer done by the test writing the data.

⁵⁰ The file can then be sent to where-ever rate calculations are being done for processing results from multiple test runs.

⁵¹ This file can contain either read results, write results, or both, in any order. Although the usual case is the file resulting from concatenating the read-result file and the corresponding write result file.

⁵² i.e. it's pretty useless unless you happen to be debugging the find-rate program.

⁵³ The program does *not* differentiate on the basis on transfer type (e.g. the protocol used)

3. Software Archive

All software is stored in Richard's CFS file space in tar format.

RS6000 in rs6000-ona.tar.

ONA utilities in ona-util.tar.

ONA source^{54, 55} in rainsrc.tar.⁵⁶

Utilities used to process rate data produced by test programs in rate-util.tar.⁵⁷

Cray TCP make file and other assorted files are in craytcp.tar.⁵⁸

Cray source files for all of SHIP and the test suite of programs are in crayship.tar.⁵⁹

⁵⁴ As supplied by Jeff Kravitz

⁵⁵ Note that this source code is proprietary to IBM and must be protected as "**Sensitive Unclassified**" data.

⁵⁶ Note that to compile this source for the ONA, a Gnu C cross compiler, and appropriate Gnu assemblers (workstation and i960) are required.

⁵⁷ This file does not contain the executable for find-rate. The file fr.c will have to be compiled to produce it.

⁵⁸ The make file references source file directories hard coded in the makefile. These will need to be changed.

⁵⁹ There are a number of extraneous files in this tar file. There are mostly left over from previous versions of the different test programs and implementations of SHIP for the Cray.

H. Appendix: Rainbow II Software Operation Guide

IBM Research

14 April 1995

0.1 Abstract

This document describes the installation and operation of the software that runs in and supports the Rainbow II node.

0.2 Required Hardware and Software

The operational software for the Rainbow II node must be downloaded via a Parallel port from some IBM PC or PC compatible system. This system must be running DOS, probably version 3.0 or later.

It is recommended, although not required, that all Rainbow support software be placed into a subdirectory on the DOS disk of the support PC, e.g. `\rainbow`.

The server PC *must* support a bi-directional parallel port, and must have this bi-directional operation enabled (usually via some BIOS ROM option or jumper).

The parallel port of the server PC must be connected to the parallel connector of one of the CPU cards of the Rainbow II node. This cable must be as short as possible:

0.3 Onaserv - Rainbow II support server program

The *onaserv* server program runs under DOS and responds to requests from the Rainbow II node for bootstrapping, file download, and file upload operations.

The command line for *onaserv* is as follows:

```
onaserv [-c] [-x] [-e] [-l] [-r] [-d] <filename>
```

The `<filename>` argument is the DOS filename and extension of the Rainbow II executable code to be loaded whenever the Rainbow II node requests a bootstrap operation.

The various flags are defined as follows:

-c	This flag indicates that the code that is bootstrapped into the attached CPU should be copied to <i>both</i> CPU's of the node. If omitted, the code is loaded only into the CPU card that is directly attached to the <i>onaserv</i> server PC.
-x	This flag indicates that immediately after the bootstrap operation, the onserve program is to exit back to DOS, usually to allow an Asynch Terminal Emulator to be run from the same server PC for debugging purposes.
-e	This flag indicates that the Rainbow CPU is to load the code but not to execute it. The Rainbow CPU Bootstrap will load the code and then execute the Bootstrap Debugger.
-l	This flag indicates that <i>onaserv</i> will log all operations to a local disk file. This file is always named <code>onaserv.log</code> .
-r	This flag indicates that <i>onaserv</i> will operate in <i>read-only</i> mode, i.e. it will not allow the Rainbow II node to send files to the PC server (e.g. dumps, traces, etc.)

-d

This flag will turn on debugging messages from onaserv.

0.4 Rainbow II Operational Code

The operational software for the Rainbow II node is bootstrapped into the two CPU cards and runs unattended. It requires that the support server PC be attached after bootstrapping only to load an initial configuration/command file, and then may be detached from the support PC if desired (although it is recommended that the support PC be left attached permanently).

0.4.1 Initial Command File

Each Rainbow II node CPU is configured and initialized by reading in an initial command file, which it requests from the support PC after it has been bootstrapped. The name of this file is determined by a combination of the CPU ID (which is either the letter A or B), and the Rainbow Node number, which is determined by the switch settings on the CPU board. This node number is the value of the low order 5 bits of the switches on the CPU, with one added, e.g. if the switches are set to the binary value for 3, then the node number is 4. This is to prevent use of the node number 0. Warning: The switches for the two CPU's should always be set to the same value.

The initial command file for each CPU and node is named `onaXY.cmd` where X is either A or B (for CPU ID) and YY is the decimal node number.

0.4.2 Rainbow II normal Operation

The Rainbow II node will pass data from the attached HIPPI port to the WDM Optical Network, and vice versa. During this data transfer the data must adhere to a few rules.

- All HIPPI data must use HIPPI FP and HIPPI LE packet headers.
- HIPPI Data packets should not be longer than approximately 1 megabyte.
- The I-field used to connect to a Rainbow II node is ignored.
- The I-field used by the destination Rainbow II node to connect to the destination HIPPI device is determined by a mapping from the ONA Node Address in the HIPPI LE field. This mapping is determined by the *ifield* commands given to the node, usually in its initialization command file. See below.
- The Rainbow II Node operates in two modes: SHIP mode and Passthrough mode. SHIP mode is defined by the SHIP protocol. In Passthrough mode, the packets are sent, unmodified, to the destination Rainbow II node, and then to the destination HIPPI device. The mode (SHIP vs Passthrough) is determined by the ULP field in the HIPPI FP header. If the ULP field contains the values 0x02, 0x03, 0x82, or 0x83, then the packet is assumed to be a SHIP Packet, otherwise it is considered a Passthrough packet.

0.4.3 Node and I-field Mapping

Rainbow II uses the HIPPI LE destination field to determine the destination Rainbow II node, and the ultimate destination HIPPI I-field to be used. The mapping is done as follows:

1. The lowest-order byte of the HIPPI LE Destination field is all that is used to perform the mapping. All other bytes are ignored.
2. All Rainbow II Node *Addresses* are calculated by shifting the Rainbow II Node *Number* left 3 bits, e.g. The Rainbow II Node whose node Number is 1, is considered to be at Node Address 8. This allows 5 bits for Rainbow II Node Number, supporting up to 32 Rainbow II nodes.
3. The low order 3 bits of the HIPPI LE destination field are used as an index into a table of I-Field selectors at the destination node. Thus, each Rainbow II node can support up to 8 attached HIPPI destination devices.

An example:

If the HIPPI LE destination address in a given packet contains the hex value 0000001C (Decimal 28). This would correspond to Rainbow II node number 3 (i.e. the node whose internal CPU switches had 2 in the low order 5 bits), and to the 4th ifield in the internal table in this node.

Note. It can be helpful to use only the Node Address values when naming and referring to nodes. Thus the above address corresponds to node address 24, subaddress 4. Thus the LE address is just the Node address + the subaddress, simplifying the computation. If this method of referring to Rainbow II nodes is used, then each Rainbow II node has an Address which is a multiple of 8, starting at 8, and any LE destination address will contain a value whose remainder modulo 8 is the subaddress used to index into the I-field table.

0.4.4 Commands

The Rainbow II operational software accepts a number of commands, either from the initial command file, or from an attached serial terminal. These command are used to configure the system, and to aid in debugging.

Configuration Commands

```
ifield <n> <ifield> [<format>]
node    [on|off <node>...]
```

The ifield command The ifield command is used to create an entry in the I-field mapping table in a destination node. There are up to three arguments to the command.

- The first argument is the ifield table entry number, which is an index from 0 through 7, corresponding to the 8 ifield table entries, and also corresponding to subaddress calculated from the HIPPI LE address field. (see the section on Node and I-field Mapping, above).
- The second argument is the actual HIPPI I-field to be used when issuing a HIPPI Connect. This should be specified as an 8-digit hexadecimal number, e.g. 402000FE.
- The third argument, which is optional, is the HIPPI Format code. This argument is a single hexadecimal digit, used as follows...

High Order 2 Bits	Low Order 2 Bits
-----	-----
Framing Protocol Mode	Fill Mode
0 = (reserved)	0 = Fill Data if short Header
1 = FP Mode. Check FP Header	1 = Fill Data

2 = FP Mode. D2 packed with D1	2 = Fill Header
3 = FP mode. D2 starts 2nd burst	3 = Fill Both

The node command The node command is used to build the table of legal Rainbow II nodes, which is used to prevent attempts to connect to non-existent nodes. It should be initialized to only contain the Rainbow II node Addresses of nodes that exist in the network. Attempts to connect to other nodes will be disallowed.

The command contains a keyword, `on`, or `off`, followed either by a list of up to 32 node addresses (Note: the Addresses are the HIPPI LE Rainbow II node Address values i.e. multiples of 8), or the keyword `all`.

The table is usually initialized to all nodes off, via the command...

```
node off all
```

and then is set up to only enable the existing nodes, for example...

```
node on 8 16 24 32
```

Debugging Commands

```
debug    [low|med|hi|off]
trace    [errs|buff|mesg|xmit|recv|aotp|ship|host|rain] on|off
stats    [reset]
dump
info
?
```

debug

This command is used to set the debugging message level. During normal operation, this should be set to `off`. The system will display high-level debugging messages, via the attached serial port, if the value is set to `high`, and will display more and more low-level messages if the value is set to `med` or `low`. Setting this value to other than `off` will slow down the system appreciably, and can cause loss of data, and even a system crash if too much data is displayed.

trace

This command turns on or off the tracing of specific events in an internal trace table. The tracing of specific events should only be enabled for debugging purposes. Normally the system is initialized so that errors are traced.

stats

This command will display operational statistics. It can be safely issued if the system is not heavily loaded.

dump

This command requests that the current trace table, and control blocks be dumped via the parallel port to the attached server PC. The actual dump will not occur until the server PC is attached via cable to the particular CPU card issuing the dump. Each CPU card must be dumped independently.

✓ info

This command will display some configuration and other information, via the attached serial port.

?

This command displays a help screen.

The following commands should only be used by troubleshooting personnel. They are dangerous, and may cause malfunction of the system.

```

chkbuf  {a}
disp    aotb <n> | scb <n> | q <qname>
rr      x|r <addr>
wr      x|r <addr> <value>
dw      <addr> [<len>]
d       <addr> [<len>]
m       <addr> <value> [<value>...]
r
l

```

✓ chkbuf

This command is used to verify that the internal buffer allocation FIFO's contain all of the allocated buffers, and that none have been lost. It must *never* be issued during operation of the node.

disp

This command is used to display an internal control block.

rr

This command is used to read a setup/debug register.

wr

This command is used to write to a setup/debug register.

dw

This command is used to display memory in word format.

d

This command is used to display memory in byte format.

m

This command is used to enter data into memory directly.

r

This command is used, either as a command, or as a prefix to other commands, to direct commands and responses to the other CPU in a Node.

l

This command is used to return control to the local CPU after the remote *r* command was issued.

I. Appendix: Microsecond optical tuning in piezoelectric fiber Fabry-Perot tunable filters

The small total mass of MOI's FFP-TF's was one of the keys in obtaining high-speed operation (standard (heavier) FFP-TFs had a tuning speed on the order of one msec). Eight FFP-TF mechanical fixtures were characterized, differing in mounting-hardware, mass, and PZT length, as listed in Table 2.

In our experiments we evaluated the response time, T_r , of the filter over the free spectral range for different FFP constructions. The fast rising and falling edges of the electrical driving signal (low-frequency square-wave) were used to rapidly scan the filter to obtain the response.

As shown in Figure 6, in earlier experiments, the fastest tuning speed for the new small FFPs without optical ringing was 50 μ s. A 50 MHz pulse generator (HP 8112A) was used to drive the filter with a 1 V swing from approximately 1.8 V to 2.8 V. A single laser source with wavelength corresponding approximately 2.8 V is used to monitor the optical throughput of the filter. The rise and fall times of the drive voltage are controllable. Figure 6(a, b, c) shows the drive voltage overlaid on the optical power transmitted for 50, 10 and 5 μ s rise and fall times, respectively. In Figure 6b, the rise time applied starts oscillations in both the driving voltage across the piezo and the optical throughput. In Figure 6c, the ringing becomes very strong. This may be due to exciting a PZT resonance.

Figure 7 shows the filter slewing past two different laser inputs 10 nm apart (1518 nm and 1528 nm). In this case, the filter is driven with a triangle wave. The voltage for each channel is slightly different on the rise and the fall, indicating an internal delay. A triangle wave is the easiest regime for the filter. The slew rate is about 75 μ s for 10 nm (or approximately 2.5 V) which is faster than the case shown in Figure 6a (approximately 50 μ s for 1 V).

As one can see, the switching speed of the new filter used in the fast mode is limited by acceleration of the mechanical motion and electrical loading of the drive voltage. A further increase in the driving slew rate induced extensive optical ringing. There are also oscillations on the top of the square waveform that increases with the driving slew rate. The optical ringing is a combination of the driving waveform ringing and internal effects. It was suggested that it may be possible to improve the shape of a driving pulse and the response of the filter by proper design of the stabilizing electrical driver, but the resonance of the piezo element must be taken into account.

Figure 6. The scanning process. The square-wave scan and the optical response of the filter are shown for single laser source: a) $50\ \mu\text{s}$, b) $10\ \mu\text{s}$, c) $5\ \mu\text{s}$ rise and fall times, respectively.

Figure 7. The scanning process. The sawtooth scan is shown at the bottom and the optical response of the filter is shown at the top. Two laser inputs at 1518 nm and 1528 nm are used.

A network analyzer revealed a series of resonant frequencies in the fixtures' impedance transfer functions (Figure 8 - Figure 10). For example, TF3 exhibits resonance at 60, 123, 172, 400, and 980 kHz, although the last two resonances are very weak. By reducing the total mass from 6.4 g to 4.0 g in TF6, these set of resonances moved out to 75 and 152 kHz, respectively. The second resonance returns toward the PZT's intrinsic first resonance. Furthermore, the magnitude of the first resonance decreased significantly. These PZTs have intrinsic resonances above 100 kHz; however, mechanical loading produces a first resonance below 100 kHz as shown in Table 2. Next by reducing the total mass to 2.0 g in TF8, a dramatic improvement is obtained where the first resonance moved beyond 100 kHz, reaching 113 kHz. This illustrates the improvement gained by reducing the structural mass. Correlation between the mechanical resonance and optical response of the filter is important for the optimal high-speed tuning design and for mounting hardware and control circuitry optimization.

These features together with the FFP-TF's high capacitance (approximately 0.5 to 1 μF) are being folded into building a special high-speed stabilization controller to get a 'clean' square waveform of the driving electrical signal. It has an adjustable repetition rate from tens to hundreds of Hz, adjustments on rising and falling slopes, and amplitude adjustment with a maximum of 25 V. The resultant controller substantially improves the shape of the driving signal and the response of the filter, it enables tuning the high-speed FFP-TF three-orders-of-magnitude faster than possible with standard commercial FFP-TFs.

Initially, the driving slew rates and amplitudes were carefully adjusted to minimize the excessive optical resonance due to electrical waveform ringing and internal effects. The fastest response times TF0 thus achieved without optical ringing for one reference are generally $<10 \mu\text{s}$, as listed in Table 2.

Figure 8. Impedance transfer characteristics for a) TF1 and b) TF3

(c)

(d)

Figure 9. Impedance transfer characteristics for a) TF5 and b) TF6

Figure 10. Impedance transfer characteristics for a) TF7 and b) TF8

As an example, the fastest response for TF5 is shown in Figure 11a, where $T_{F0} = 3.9 \mu\text{s}$. Further increase in driving slew-rates induces strong parasitic signals. The "ringing" is due to excessive voltage level as well as to internal effects such as excitement of PZT mechanical resonance. When the driving electrical signal is almost a step function then sharp stopping at a desired wavelength excites back and forth motion and it periodically changes the small air gap inside of the filter. It causes periodical opening and locking of the filter. The results of that are shown in Figs. 7-9, where $T_F = 1.1 \mu\text{s}$.

Figure 11. (a) Fastest response time for TF5 excess optical ringing with $T_{F0} = 3.9 \mu\text{s}$; (b) fastest response time for TF5 with $T_F = 1 \mu\text{s}$ (note the two sets of traces of different time scales).

Results in Table 2 show all response times reaching 1 to 2 μs and the clear advantage gained by using the LANL's high-speed driver and lower hardware mass of MO's FFP-TFs. Tuning speed versus driving slew rate plotted in Figure 12 reveals an ideal linear relationship for low-speed operations and a nonlinear, dramatic increase in tuning speed for high speed operations.

Figure 12. Dependence of tuning speed on driving slew rates for TF1, TF2, TF3, and TF5.

In additional experiments, the performance of LANL's stabilizing controller was compared with an MOI controller. Figure 13a indicates a very strong ringing for TF3 when we used the MOI controller. Furthermore, it was possible to achieve response times as low as 5.9 μs without any ringing with the use of LANL's controller (Figure 13b). It is important to note that all possible types of tunable piezoelectric Fabry-Perot filters were tested. All had either substantially heavier hardware mass or couple times higher than MOI FFP-TF capacitance.

Figure 13. Optical response for TF3: (a) $T_r = 5.9 \mu s$ for MOI controller, and (b) $T_{r0} = 5.9 \mu s$ for LANL controller.

These factors did not allow tuning faster than 100 to 200 μs even with the use of LANL's driver. Figure 14 shows the optical response of the fastest E-TEK Tunable Optical Fabry-Perot Filter (TOFF) that was measured using LANL's controller. In this case $T_{r0} = 189.8 \mu s$.

We also have measured the fastest response times achieved for an optical filter without optical ringing for two references at 1545 and 1680 nm using the scanning regime. Figure 15 shows the block-diagram for the experimental set up using LANL's high-speed FFP controller. Figure 16 shows the fastest optical response achieved for two references using LANL's controller. In this case, $T_{r0} = 11.48 \mu s$. It is possible to operate at a tuning speed of approximately $1 \mu s/1V$ which would be adequate for initial packet-switched systems if adequate PZT start and stop characteristics can be demonstrated. Also a switching response as fast as 10 to 15 μs will satisfy the recommendations of Society of Motion Picture and Television Engineers (SMPTE) for video switching applications.

Figure 14. Fastest response time for E-TEK Tunable Optical Fabry-Perot Filter (TOFF) with $T_{F0} = 189.8 \mu\text{s}$.

Figure 15. Experimental set-up for two nodes using LANL's high-speed FFP driver

Figure 16. The Scanning process. Optical response achieved for two references at 1545 nm and 1680 nm using LANL's FFP driver.

Also we performed 'off-on' and 'on-off' tuning experiments for TF5. A single laser source with 1565 nm wavelength corresponding approximately 4.6 V for the first maximum and 21.9 V for the second maximum of the Airy function was used to monitor the optical throughput of the filter. For Wavetek 802, 50 MHz pulse generator and LANL's driver were used to drive the filter in 'off-on' switching regime from 5.8 V to 21.9 V so the filter was operating in the second FSR. The fastest off-on switching time obtained for TF5 and 1565 nm source without the ringing was 44.0 μ s (Figure 17).

Using the same 1565 nm source in 'on-off' switching regime from 4.6 to 26.7 V (filter operated in the first FSR) reveals 13.7 μ s fastest switching (Figure 18). Note that the peak on plot corresponds to the second maximum of the Airy function when the scanning regime is applied.

Also a single laser source with shorter 1545 nm wavelength corresponding approximately 1.4V was used to monitor the optical throughput of the filter in 'on-off' regime. The same filter was driven from 1.84 V to 24.5 V. In this case the filter was able to switch in 8.4 μ s without any ringing (Figure 19).

The experiments show that for the same mechanical loading, the low voltage PZT has a better overall performance compared to the higher voltage PZT in both the speed and voltage requirements.

As the result, a new packet-switched media access control protocol is being designed to minimize the searching time. The filter scans only once through the entire optical region and then tunes to all the required channels one after another in a few microseconds. It can help update Rainbow-II Broadcast-and-Select High-Speed All-Optical network that currently has a circuit-switched protocol using standard FFP-TFs.

J. Acknowledgment

Participant's signature on the final report indicates the following:

1. The Participant has reviewed the final report and concurs with the statements made therein;
2. The Participant agrees that any modifications or changes from the initial proposal were discussed and agreed to during the term of the project;
3. The participant certifies that all reports either completed or in process are listed and all subject inventions and the associated intellectual property protection measures attributable to the project have been disclosed or are included on a list attached to this report;
4. The Participant certifies that proprietary information has been returned or destroyed by LANL.

Rajiv Ramaswami
IBM Research

[Industrial Participant] Date
11/27/96

x

[LANL Principle Investigator] Date

Attachment I --Final Abstract
Attachment II --Project Accomplishments Summary
Attachment III --Final Quarterly Report

Part II. Final Abstract (Attachment I)

The Collaboration Research and Development Agreement (CRADA) between IBM and Los Alamos National Laboratory (LANL) is part of a DOE Defense Program 1992 Technology Commercialization Initiative (TCI) that supported four different CRADAs at four DOE Laboratories. This project, entitled "Gigabit Network Technology", was initiated by Paul Green of IBM and John Morrison of LANL in 1991. This collaboration was an attempt to upgrade all-optical networks to gigabit rates..

The IBM all-optical network approach is based on a technique which uses wavelength division multiplexing (WDM) as the channel discrimination scheme. IBM had developed a system in 1990 called Rainbow-I which created a 300 megabit/sec WDM network of RS6000 workstations. WDM shows great promise as a technique to utilize more of the inherent bandwidth of fiber optics. Multiplexing several (up to 64) individual network channels on one fiber using WDM greatly increases the bandwidth carried on this single fiber. This technique, coupled with laser transmitters and optical filter receivers, makes up the core features of a WDM all-optical network.

With the development of gigabit network technology, especially that which was done at Los Alamos in 1988-89 with the introduction of the High Performance Parallel Interface (HIPPI), Paul Green and his researchers at IBM sought a collaboration that would combine WDM and HIPPI to create the first all-optical *gigabit* network to be called Rainbow-II. The initial CRADA statement of work called for four distinct efforts:

- A HIPPI to WDM network adapter to be designed and built at IBM with LANL providing technical support.
- Development of a new light weight protocol at Los Alamos to support efficient data transfers between HIPPI based computer hosts and the WDM network .
- Integration of applications on the WDM network by LANL and IBM.
- Evaluation of the resulting optical network at LANL using supercomputer and other systems at the Laboratory.

In addition, there was a fifth effort at LANL to improve the tuning response time of the optical filters used in the network. This effort was endorsed by IBM because of a desire to develop a faster switching network to support packet-switching as opposed to circuit-switching which was used in Rainbow-II.

A brief summary of the results from the CRADA are as follows:

- Three Optical Network Nodes (ONNs) were built by IBM with technical help from LANL (4/93 - 12/94).
- A new protocol (SHIP) was developed by LANL with technical help from IBM for use in the ONNs (6/93 - 7/94).
- Visualization applications were developed for Rainbow-II by LANL and IBM to demonstrate the network at several conferences in 1994 and 1995 (9/94-3/95).
- An evaluation of Rainbow II was performed at Los Alamos using Cray supercomputers in the LANL HIPPI testbed (4/95 - 12/95).

Research into techniques to improve optical filter response time has been ongoing between LANL and Micron Optics, who supplied the optical filters for Rainbow-II, and has developed to where tuning speeds have improved by a factor of 100 (1/94 - 9/96).

Part III. Project Accomplishment Summary (Attachment II)

When this CRADA was proposed, IBM had developed Rainbow-I, a prototype 0.3 Gb/s-per-node all-optical circuit-switched network composed of 32 nodes over a 25 km area that used wavelength division multiplexing. Los Alamos National Laboratory had been instrumental in the development of high performance gigabit per second computer networks and had established a testbed based on the High Performance Parallel Interface (HIPPI) standard, a standard that had seen wide acceptance throughout the computer industry. LANL's testbed contained many of the country's leading supercomputers, and current and projected projects (e.g., DOE programs for nuclear stockpile stewardship) indicated a pressing need for ever faster computer networking. The goal of this CRADA, as part of DOE's Technology Commercialization Initiative, was to show the feasibility of creating an all-optical network with individual links transferring data at nearly 1 Gb/s.

While results from this project directly supported the future technical directions of certain of DOE projects, the benefits of this collaboration may also extend to commercial efforts.. Industries benefiting from this collaboration include Telecommunications and Motion Picture industries that require Gb/s throughput rates now or in the near future. In addition, manufacturers of optical networking equipment including WDM component manufacturers have benefited from the results achieved.

In this CRADA, IBM concentrated on designing and building optical network nodes (ONNs) that provided the interface between the American National Standard High Performance Parallel Interface (HIPPI) and it's all-optical network called Rainbow. The HIPPI expertise at Los Alamos was used to provided technical support for this effort. In addition, Los Alamos concentrated on developing a new protocol to support efficient data transfers from host computers to the ONNs and also supported the capability of the ONN's to provide protocol processing of the standard TCP/IP protocol used in most networks. The ability of such adapters to process TCP/IP traffic is critical to their deployment in general computer networks. The ONN was not able to process the TCP/IP packets because of the relative slow response time of the optical filters.

IBM built three ONNs and, along with Los Alamos personnel, successfully demonstrated them as the first gigabit all-optical network at several conferences in late 1994 and 1995. Los Alamos supported these demonstrations with equipment and visualization applications.

To further show feasibility, the CRADA called for an evaluation of the optical network at Los Alamos utilizing network traffic from the Laboratory's Cray supercomputers. This effort was done during the second half of 1995. The result of this evaluation highlighted a performance constraint in the network that is mentioned above - the relatively slow tuning performance (tens of milliseconds) of the optical filters used in the network. Without the ability to tune quickly (10 to 100 microseconds), the network would be relegated to circuit switched applications such as visualization. However, in a real world computing environment there is a broad mixture of applications that

necessitate a fast packet switching capability. In particular, the TCP/IP protocol, ubiquitous in most high-end computer networks, requires fast switching of its packets.

For an all-optical network to be useful to current and future programs at Los Alamos, this fast packet capability must be present. With this in mind, Los Alamos, as part of an evolving role in the CRADA, started collaborating with an optical filter manufacturer to develop ways to improve optical filter tuning performance in 1994. By the end of the CRADA (9/96), Los Alamos researchers had evaluated and characterized new filters that can tune in less than 10 microseconds.

Conclusions:

This CRADA with IBM was successful in showing that all optical gigabit networks are feasible although the prototypes were expensive, especially the optics. The ability of off-board protocol processors such as the ONN to handle TCP/IP protocols is difficult and depends on switching capabilities in the network. This issue of quickly tuning the filters to support fast packet switching of gigabit data transfers is critical to the deployment of such networks. Efforts to build and deploy networks supporting 5 to 10 Gb/sec are currently being pursued under the HPCRC and ASCI programs. To support the anticipated increasing demand for more bandwidth for networks above the 10 Gb/sec realm may require the use of all-optical networks such as those developed under this CRADA. This is evident since the capability of electronics used in the 5 to 10 gigabit range is reaching the limits of electronic technology.

Points of Contact

For further information please contact the DOE Facility/LANL Point of Contact for Project Information:

Steve Tenbrink,
Deputy Group Leader, CIC-5
505-667-4935
P.O. Box 1663, Mail Stop B255
Los Alamos National Laboratory
Los Alamos, NM 87545

The industrial partner for this CRADA was IBM Research Division. IBM is an international corporation. Its 1994 Annual report listed world-wide revenue of \$64 billion (US revenue of 24.1 billion) and 270,000 part-time and full-time employees. The point of contact at IBM for this project is:

Rajiv Ramaswami
Manager, Optical Network Systems Group
914-784-7356 (voice)/914-784-6225 (fax)
rajiv@watson.ibm.com
IBM T.J. Watson Research Center
PO Box 704
Yorktown Heights, New York 10598

Part IV. Final Quarterly Report (Attachment III)

During the final quarter of this CRADA, experimentation continued to characterize and further improve the optical switching performance of the Fabry-Perot tunable filters and specially designed LANL driver. In addition, a plan was established and equipment ordered to demonstrate data transmission reliability through fast optical filters that are a critical component for fast packet switching optical gigabit networks. In particular, a Burst Error Rate generator was procured along with optical components that will form the core of a demonstration at the Optical Fiber Conference during March, 1997. In addition, funding from internal Laboratory sources was identified to continue this work through FY97.

The last quarter also included an increase of scientific staff working on this technology. Dr. Carolyn Davenport has been hired to work with Dr. Taranenko in the further research and development of a fast packet gigabit optical network. Also, Dr. Taranenko's position has been changed from post-doctoral to technical staff member at the Laboratory.

YES NO Any non-DOE property loaned has been returned.

YES NO Any DOE property loaned has been returned.

YES NO Final Report has been submitted to the CRADA partners and is attached.

YES NO Final Report has not been completed. The anticipated completion is _____

YES NO Final Abstract is attached.

YES NO Project Accomplishment Summary is attached.

YES NO All known subject inventions, including partners' and joint, have been reported to LC/BPL and disclosed to the partners. Please list Los Alamos Disclosure Numbers:

☒ YES ☐ NO Has any material subject to copyright (i.e., software) been developed under the CRADA?

☒ YES ☐ NO Industrial Partner has expressed interest in an exclusive license for intellectual property generated under the CRADA.

Please list any CRADA deliverables that are not complete. Indicate the anticipated completion date or recovery plan.

Date _____