Network Working Group
Request for Comments: 1306

A. Nicholson
J. Young
Cray Research, Inc.
March 1992

Experiences Supporting By-Request Circuit-Switched T3 Networks

Status of this Memo

This RFC provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Abstract

This memo describes the experiences of a project team at Cray Research, Inc., in implementing support for circuit-switched T3 services. While the issues discussed may not be directly relevant to the research problems of the Internet, they may be interesting to a number of researchers and implementers.

Developers at Cray Research, Inc. were presented with an opportunity to use a circuit-switched T3 network for wide area networking. They devised an architectural model for using this new resource. This involves activating the circuit-switched connection when an application program engages in a bulk data transfer, and releasing the connection when the transfer is complete.

Three software implementations for this feature have been tested, and the results documented here. A variety of issues are involved, and further research is necessary. Network users are beginning to recognize the value of this service, and are planning to make use of by-request circuit-switched networks. A standard method of access will be needed to ensure interoperability among vendors of circuit-switched network support products.

Acknowledgements

The authors thank the T3 project team and other members of the Networking Group at Cray Research, Inc., for their efforts: Wayne Roiger, Gary Klesk, Joe Golio, John Renwick, Dave Borman and Craig Alesso.

Overview

Users of wide-area networks often must make a compromise between low cost and high speed when accessing long haul connections. The high money cost of dedicated high speed connections makes them uneconomical for scientists and engineers with limited budgets. For many traditional applications this has not been a problem. Datasets can be maintained on the remote computer and results were presented in a text-only form where a low-speed connection would suffice. However, for visualization and other data transfer intensive applications, this limitation can severely impact the usability of high performance computing tools which are available only through long-haul network connections.

Supercomputers are one such high performance tool. Many users who can benefit from access to supercomputers are limited by slow network connections to a centrally located supercomputer. A solution to this problem is to use a circuit-switched network to provide high speed network connectivity at a reduced cost by allocating the network only when it is needed.

Consider how a researcher using a visualization application might efficiently use a dedicated low speed link and a circuit switched high speed link. The researcher logs in to the remote supercomputer over the low speed link. After running whatever programs are necessary to prepare the visualization, the high speed connection is activated and used to transfer the graphics data to the researcher’s workstation.

We built and demonstrated this capability in September, 1990, at the Telecommunications Association show in San Diego, using this type of visualization application. Further, it will be available in a forthcoming release of our system software.

Architectural Model

We developed our support for circuit switched services around a simple model of a switched network. At some point in the path between two hosts, there is a switched network connection. This connection is likely to connect two enterprise networks operated by the same organization. Administrative overlap between the two networks is useful for accounting and configuration purposes. We believe that with further investigation circuit switched network support could be extended to multiple switched links in an internet environment.

The switch which makes the network connection operates on a "by-request" basis (also called "on-demand"). When it receives a request

to make a network connection it will do so (if possible), and breaks the connection when requested. The switch will not activate automatically if there is an attempt to transfer data over an incomplete connection.

We also made the assumption that the circuit would be switched on a connection basis rather than a packet basis. When an application begins sending data utilizing the switched connection, it will send all the data it has, without stopping, until it is finished. At this time it will release the connection. It is assumed that the quantity of data will be large enough that the circuit setup time is negligible relative to the period of the transfer. Otherwise, it is not worth the effort to support the circuit switched network for small data transfers.

This model requires that just before the application begins a large bulk transfer of data, a request message is sent to the switch asking that the switched network connection be activated. Once the link is up, the application begins sending data, and the network routes all the data from the application through the switched network. As soon as all the data has been sent, a message is sent to the switch to turn off the switched link, and the network returns to routing data through the slower link.

The prototype system we built for the TCA show was designed around this model of circuit switched services. We connected a FDDI backbone at Cray Research in Eagan, Minnesota to the TCA show's FDDI network through 2 NSC 703 FDDI/T1/T3 routers. MCI provided a dedicated T1 line and a switched T3 line, using a DSC DS3 T3 switch located in Dallas, Texas. These networks provided connectivity between a Cray Research computer in Eagan to a Sun workstation on the show floor in San Diego.

Alternative Solution Strategies

The first aspect of using the switched services involved the circuit switch. The DS3 switch available to us was accessed via a dial up modem, and it communicated using a subset of the CCITT Q.295 protocol. Activating the switch required a 4 message exchange and deactivation required a 3 message exchange. We felt the protocol was awkward and might be different for other switch hardware. Furthermore, we believed that the dial up aspect of communicating with the switch suffered from the same drawbacks. A good solution would require a cleaner method of controlling the switch from the source host requesting the switched line.

The next aspect of using switched services involves the source host software which requests and releases the switched network. Ideally,

the switched network is activated just before data transfer takes place and it is released as soon as all data has been sent. We considered using special utility programs which a user could execute to control the link, special system libraries which application programs could call, or building the capability into the kernel. We also considered the possibility that these methods could send messages to a daemon running on the source host which would then communicate with another entity actually controlling the switch.

The last aspect of using switched services we considered is selection of the switch controlled network. This involves both policy issues and routing issues. Policy issues include which users running which applications will be able to use higher cost switched links. And packets must be routed amongst multiple connections offering varying levels of service after they leave their source.

Implementations

We have developed a model for switch control through the internetwork which we believe to be reasonable. However, we have experimented with three different source host implementations. These different implementations are detailed here.

Switch control

Our simplest design decisions involved the switch itself. We decided that the complex protocol and dial up line must be hidden from the source host requesting the switched link. We decided that the source host would use a simple request/release protocol with messages sent through the regular network (as opposed to dial up lines or other connections). Some host accessible through the local network would run a program translating the simple request and release messages into the more complicated switch protocol and also have the modem to handle the dial up connection.

This has a variety of advantages. First, it isolates differences in switch hardware. Second, multiple hosts may access the switch without requiring multiple modems for the dial up line. And it provides a central point of control for switch access. We did not consider any alternatives to this model of switch control.

Our initial implementation used a simple translator daemon running on a Sun workstation. Listening on a raw IP port, this program would wait for switch control messages. Upon receipt of such a message, it would dial up the switch and attempt to handle the request. It would then send back a success or failure response. This host, in conjunction with the translator daemon software, is referred to as the switch controller. The switch controller we used was local to

our enterprise network; however, it could reside anywhere in the Internet.

Later we designed a simple protocol for switch control, which was implemented in the translator daemon. This protocol is documented in RFC 1307, "Dynamically Switched Link Control Protocol".

Source Control of the Switched Link

This problem involves a decision regarding what entity on the source host will issue the switch request and release messages to the switch controller, and when those messages will be issued. Because we do not have very much field experience with this service, we do not feel that it is appropriate to recommend one method over the others. They all have advantages and disadvantages.

What we did do is make 3 different implementations of the request software and can report our experiences with each. These are one set of special utility programs which communicate with the switch controller, and 2 kernel implementations. We did not experiment with special libraries, nor did we implement a daemon for switch control messages on the source host.

Switch control user programs

This implementation of source host control of the switch is the simplest. Two programs were written which would communicate requests to the switch controller; one for activating the connection, and another to deactivate the connection. The applications using this feature were then put into shell scripts with the switch control programs for simple execution.

This approach has the significant advantage of not requiring any kernel modifications to any machine. Furthermore, application programs do not need to be modified to access this feature. And access to the circuit-switched links can be controlled using the access permissions for the switch-control programs.

However, there are disadvantages as well. First, there is significant potential for the switch to be active (and billing the user) for the dead time while the application program is doing tasks other than transferring bulk data. The granularity of turning the switch on and off is limited to a per-application basis.

Another disadvantage is that most applications use only the destination host’s address for transfer, and this is the only information available to the transport and network layers for routing data packets. Some other method must be used to distinguish between

traffic which should use the circuit-switched connection and lower-priority traffic. This problem can be addressed using route aliases, described below.

Kernel switch control

We have made two different implementations of switch control facilities within the operating system kernel. Both rely upon the routing lookup code in the kernel to send switch connect and tear down messages. The difference is in how the time delay between request of the switch and a response is handled.

For starters, routing table entries were expanded to include the internet address of the switch controller and state information for the switched connection. If there is a switch controller address specified, then the connection must be set up before packets may be sent on this route. We also added a separate module to handle the sending and receiving of the switch control messages.

When a routing lookup is satisfied, the routing code would check whether the routing table entry specified a switch controller. If so, then the routine requesting switch setup would be called. This would send a message on the Internet to the switch controller to setup the connection.

In our first implementation, the routing lookup call would return immediately after sending the switch connection request message. It would be the responsibility of the transport protocol to deal with the time delay while the connection is setup, and to tear down if the switched connection could not be made. This has significant ramifications. In the case of UDP and IP, packets must be buffered for later transmission or face almost certain extermination as they will probably start arriving at the switched connection before it is ready to carry traffic. Because of this problem, we decided that this feature would not be available for UDP or IP traffic.

We did make this work for TCP. Since TCP is already designed to work so that it buffers all data for possible later retransmission, this was not a problem. Our first cut was to change TCP to check that the route it was using was up if it is a switch controlled route. TCP would not send any data until the route was complete, and it would close the connection if the switch did not come up.

This did not work well at first because every time TCP tried to send data before the switch came up, the retransmit time would be reset and backed off. The rtt estimate, retransmit timeouts and the congestion control mechanism were seriously skewed before any data was ever sent. The retransmit timer would expire as many as 3 times

before data could be transmitted. We solved this problem by adding another timer for handling the delay while the route came up, and not allowing the delay to affect any of the normal rtt timers.

Our experiences with this approach were not particularly positive, and we decided to try another. We also felt that unreliable datagram protocols should be able to use the service without excessive reworking. Our alternative still sends the switch control message when a routing lookup finds a controlled route. However, we now suspend execution of the thread of control until a response comes back from the switch controller.

This proved to be easier to implement in many ways. However, there were two major areas requiring changes outside the routing code. First, we decided that if the switch refused to activate the connection, it was pointless to try again. So we changed the routing lookup interface so that it could return an error specifying a permanent error condition. The transport layer could then return an appropriate error such as a host unreachable condition.

The other, more complex issue deals with the suspension of the thread of execution. Our operating system, UNICOS, is an ATT System V derivative, and our networking subsystem is based on the BSD tahoe and reno releases. The only way to suspend execution is to sleep. This is fine, as long as there is a user context to put to sleep. However, it is not a good idea to go to sleep when processing network interrupts, as when forwarding a packet.

We solved this problem by using a global flag regarding whether it was ok for the switch control message code to sleep. If it is necessary to send a message and sleep, then the flag must be set and an error is returned if sleeping is not allowed. User system calls which might cause a switch control message to be sent set and clear the flag upon entrance and exit. We also made it impossible to forward packets on a switch controlled route. We feel that this is reasonable since the overhead of switch control should be incurred only when an application program has made an explicit request to begin transfer of data.

The one other change we made was to make sure that TCP freed the route it is using upon entering TIME_WAIT state. There is no point in holding the circuit open for two minutes in case we need to retransmit the final ack. Of course, this assumes that an alternate path exists for the the peer to retransmit its fin.

The advantage of building this facility into the kernel is that it allows a fine degree of control over when the switch will and will not need to be activated. Many applications which open a data

connection, transmit their bulk data, and then close the connection will not require modifications and will make efficient use of the resource. It also opens the possibility that applications written to use type-of-service can use the same network connection for low-bandwidth interactive traffic, change the type-of-service (thus activating the switched connection) for bulk transfers, and then release the switch upon returning to interactive traffic.

Putting this feature into the kernel also allows strong control over when and how the switched link can be used, keeping accounting information, and limiting multiple use access to the switched link.

The disadvantage is that significant kernel modifications are required, and some implementation details can be very difficult to handle.

Switch control libraries

The switch control programs we used were built on a library of simple switch control routines; however, we did not alter any standard applications to use this library. We did consider some advantages and disadvantages. On the plus side, it is possible to achieve a satisfactory degree of switch control without requiring any kernel modifications.

The primary disadvantage of this approach is that all applications must be altered and recompiled. This is particularly inconvenient when source is not available.

Link Selection

When an application wishes to send data over a circuit-switched connection, it will be necessary to select the switched link over other links. This selection process may need to take place many times, depending on the local network between the source host and the bridge to the circuit switched connection.

For example, if the kernel routing code is controlling the link, then there must be a way to choose a controlled route over another route. Further downstream, there must be a way to route packets to the switched link rather than other links.

This issue has the potential for great complexity, and we avoided as much of the complexity as possible. Policy routing and local routing across multiple connections are fertile areas for work and it is outside the scope of this work to address those issues. Instead we opted for simple answers to difficult questions.

First of all, we added no special policies to link accessibility beyond that already found in UNICOS. And we handled local routing issues to the NSC FDDI/T1/T3 routers with routing table manipulation and IP Type-of-Service.

We came up with three solutions for selecting a routing table entry. The first possibility is to use the type-of-service bits, which seemed natural to us. We changed the routing table to include type-of-service values associated with routing entries, and the routing lookups would select using the type-of-service. UNICOS already supports a facility to mark connections with a type-of-service value. A controlled route could be marked with high throughput type-of-service and an application wishing to transfer bulk data could set the socket for high throughput before making the connection. It could also be possible to change the type-of-service on an existing connection and start using the switched link if one is available.

Using the type-of-service bits have the advantage that downstream routers can also use this information. In our demonstration system, the NSC FDDI/T1/T3 routers were configured to transfer packets with high throughput type-of-service over the T3 connection and all others over the T1 connection.

Another possibility is to take advantage of the multiple addresses of a multi-homed host. Routing tables could be set up so that packets for one of the addresses get special treatment by traveling over the switched link. The routing table in the source host would have an entry for accessing the switch controller when sending to the high throughput destination address.

We also derived a method we call route aliasing. Route aliasing involves associating extra addresses to a single host. However, rather than the destination being an actual multi-homed host, the alias is known only to the source host and is used as an alternative lookup key. When an application tries to connect to the alias address the routing lookup returns an aliased route. The route alias contains the actual address of the host, but because of looking up the special address, the switch is activated. The alias could also specify a type-of-service value to send in the packets so that downstream routers could properly route the packets to the switched link. We realize that some may bemoan the waste of the limited Internet address space for aliases; however, only the source host is aware of the alias, and the primary shortage is with Internet network addresses rather than host addresses. In fact, we argue that this is a more efficient use of the already sparse allocation of host addresses available with each network address.

Future considerations

We believe that by-request services will become increasingly important to certain classes of users. Many data centers make high performance resources available over a wide area, and these will be the first users to take advantage of wide-area circuit-switched networks. Some users, such as CICNet ([2]), are already interested in deploying this capability and telecom vendors are working to satisfy this need. However, there are a lot of issues involved in providing this functionality. We are working to involve others in this process.

References

[1] Nicholson, et. al., "High Speed Networking at Cray Research", Computer Communications Review, January 1991.

[2] CICNet DS3 Working Group, "High Performance Applications on CICNet: Impact on Design and Capacity", public report, CICNet, Inc., June 1991.

[3] Young, J., and A. Nicholson, "Dynamically Switched Link Control Protocol", RFC 1307, Cray Research, Inc., March 1992.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Andy Nicholson
Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55123

Phone: (612) 452-6650
EMail: droid@cray.com

Jeff Young
Cray Research, Inc.
655F Lone Oak Drive
Eagan, MN 55123

Phone: (612) 452-6650
EMail: jsy@cray.com