# Sago: A Network Resource Management System for Real-Time Content Distribution

Tzi-cker Chiueh, Kartik Gopalan, Anindya Neogi, Chang Li, Srikant Sharma, Sheng-Ming Shan, Jiawu Chen, Wei Li, Nikolai Joukov, Jie Zhang, Fu-Hau Hsu, Fanglu Guo, Sheng-I Doong

Rether Networks Inc.
Centereach, New York 11720

*Abstract*— **Content replication and distribution is an effective technology to reduce the response time for web accesses and has been proven quite popular among large Internet content providers. However, existing content distribution systems assume a store-and-forward delivery model and is mostly based on static content. This paper describes the design, implementation, and initial evaluation of a network resource management system for real-time Internet content distribution called *Sago*, which provides facilities to provision and allocate network resources so that multiple bandwidth-guaranteed and fault-tolerant multicast connections can be multiplexed on a single physical network. *Sago* includes a novel network resource mapping algorithm that takes into account both physical network topology and dynamic traffic demands, a network-wide fault tolerance mechanism that supports both node-level and link-level fault tolerance, and a hierarchical network link scheduler that provides performance protection among multicast connections sharing the same physical network link. Moreover, *Sago* does not require any IP multicasting support from underlying network routers because it performs application-level multicasting. The technologies underlying *Sago* are important building blocks for real-time content distribution networks, end-to-end quality of service guarantee over global corporate intra-nets, and application-specific adaptation of wide-area network services.**

*Keywords*— **Content Distribution Networks, Application-Level Multicasting, Network Resource Mapping, Hierarchical Link Scheduling, Fast Network Restoration and Reliable Multicast**

## I. Introduction

An emerging information technology (IT) trend in today's corporate enterprises is to out-source computing and communication services so as to focus on the development of business logic functions that are the core competence and that have the highest profit margins and barriers of entry. This is why a wide varieties of service providers are sprouting out in the Internet space in the recent years, ranging from basic connectivity (ISP) and storage management (SSP) to end-to-end application deployment and hosting (ASP). The central technical problem that all *x*SPs face is how to multiplex multiple logical resource entities, each corresponding to a distinct customer, on a single physical resource in a way that conforms to individual customer's service level agreement (SLA) and that at the same time achieves the highest system utilization efficiency. This paper focuses specifically on *resource virtualization* techniques for wide-area network resources and describes the design, implement, and evaluation of an real-time content delivery network management system called *Sago*, which allocates, provisions, and manages application-level multicast connections on a single physical network such that each multicast connection could have its distinct quality of service (QoS) guarantee in terms of network bandwidth.

Despite extensive research efforts on network QoS in the past decade, the actual impacts of these efforts on commercial LAN and WAN devices appear to be relatively minor. In particular, the holy grail of end-to-end network QoS guarantee remains largely elusive. For network applications to enjoy end-to-end latency/bandwidth guarantee over wide-area packet networks, the only available alternatives are (1) reserving a dedicated virtual circuit on ATM networks (or in general MPLS-capable networks) or (2) over-provisioning the underlying network. (2) is clearly too expensive while (1) is too cumbersome and inflexible to set up and manage in practice. In the *Sago* project, we start with a baseline network, which could be a physical network or a logical network each of whose links is an ATM permanent virtual circuit (PVC), and develop a fully operational network management system that can create multicast connections with bandwidth guarantee on demand for distribution of real-time content.

*Sago* consists of two components, a global resource manager (GRM), which oversees the allocation of node and link resources on the baseline network, and a local packet processing engine (LPPE), which is placed at each baseline network node and performs actual resource usage control and management at run time. In *Sago*, users can request a real-time multicast connection and specify it as follows:

- A data sender and a set of data receivers,
- The network bandwidth requirement,
- The transport mode, which indicates whether data should be transported in the reliable mode, and
- The reliability requirement in terms of whether network redundancy is called for.

Given such a request, the GRM allocates physical link and node resources on the baseline network to meet the request's specification such that as many connection requests can be supported in the future as possible. The GRM then configures the LPPE on the baseline network nodes that are chosen to implement the given multicast connection, by setting up appropriate routing and QoS state. At run time, the LPPE dynamically recognizes packets that belong to this multicast connection, enforces their resource usage according to the corresponding specification, and invokes the associated control/data processing functions on the packets.

The key architectural design decision of *Sago* is that it performs multicast routing, reliable transport and QoS enforcement completely at the application level. As a result, it does not require any special support from network routers. In addition, *Sago* provides a novel reliable multicast transport service that is tightly integrated with node fault tolerance and link fault tolerance mechanisms.

In Section 2, we review previous works in the network resource management area. In Section 3 and 4, we present the system architecture that *Sago* assumes and its network resource mapping algorithm. Section 5 describes the implementation of traffic shaping and control. Section 6 portrays the fault tolerance scheme built into *Sago*. Section 7 presents preliminary performance results from the first *Sago* prototype. Section 8 concludes this paper with a summary of the unique features of *Sago* and an outline of on-going work in the *Sago* project.

## II. Related Work

Resource virtualization is a technique that has been applied in various aspects of computer systems design ranging from CPU, memory system and virtual machine to more recently storage virtualization and Internet server virtualization. The *Sago* project focuses on the development of resource virtualization techniques for wide-area networks to support bandwidth-guaranteed fault-tolerant multicast connections, which can be used as the basis for Internet-scale distribution of real-time content. Content distribution networks (CDN) such as Akamai replicate Web content to minimize the user-perceived Web access latency. The next evolution of CDN is real-time digital media delivery over wide-area networks. The technologies developed in the *Sago* project are readily applicable to this problem, especially its support for bandwidth-guaranteed multicast distribution trees.

The bandwidth-guaranteed fault-tolerant multicast connections that *Sago* provisions and manages can also be thought of as small-scale overlay networks. Virtual private network (VPN) [19] is a technology that leverages public networks to extend and connect corporate intranets, with the main focus on the security and ease of deployment of island-connecting tunnels. However, VPN does not address QoS and reliability issues of these tunnels. Because of the great commercial success of VPN, the *Sago* project intentionally leaves out the security issue of overlay network set-up and operation, and plans to take advantage of the advances that VPN makes in this area.

Perhaps the most well-known overlay network is M-Bone [10], which uses tunneling to connect islands of routers that are capable of IP multicast. Similar overlay networks for IPv6 (6-Bone) [1] and active networks (A-Bone) [4] exist. The X-Bone project [20] recognized the underlying similarity among these overlay networks and developed a more general framework to facilitate the process of setting up, configuring, monitoring, and managing overlay networks with application-specific control/data planes. The X-Bone project does not address the issue of QoS mapping and fault tolerance. Cornell's VON project [17] and the the IETF's emerging VPN framework share a similar goal with X-Bone. The Genesis project [5] at Columbia advocates the notion of spawning networks, and supports a retrograde variant of recursion - deploying parent overlays, where each parent can spawn multiple child overlays.

The Darwin system at CMU [6] has a similar goal as *Sago* in that it also focuses on the system support for user-customizable value-added service on a baseline network. Darwin allows customized resource management and control at network switching points and provides hierarchical packet scheduling for network links. However, Darwin did not address the network resource mapping issue. Neither did it support node/link fault tolerance or performance isolation for overlay networks. Argonne's MORPHnet [2] is an overlay system that supports virtual networks at all layers, from virtual physical, to link, to network, on up to application. MORPHnet was designed for use in supercomputer networks, where performance requirements necessitate low- and multi-layer solutions. CRATO's Supranet [8] extends this multi-layer notion with multi-layer optimizations. Columbia's Virtual Active Networks (VANs) are part of the Netscript project [21] and deploy link-layer virtual networks

MIT's Resilient Overlay Network project (RON) [3] applied overlay network technology to the problem of Internet congestion, similar to the Detour project at University of Washington [18], and to a less degree to the distributed denial of service problem. Berkeley's Sahara project [11] also attempts to apply the overlay network technology improve the performance of Internet services, but with an emphasis on those services that are composed from multiple sites over a wide-area network.

Kodialam and Lakshman [12], [14] proposed a minimum interference routing scheme that is very similar to the network resource mapping algorithm described in Section 2.2. However, their algorithm focused only on network topology and did not take into account input workload statistics. In addition, they did not address the performance cost of their algorithm, which is an important issue for highly dynamic overlay network management systems such as *Sago* and will be one of the main focuses of this project.

Network path restoration has been studied extensively in the optical transport network and ATM network literature. Kodialam and Lakshman [13] proposed an integer programming formulation of the N:1 shared redundancy problem for IP networks. Murakami [15], [16] formulated the circuit restoration problem in ATM networks into a linear programming problem, and found that in general path restoration is more efficient than link-by-link restoration.

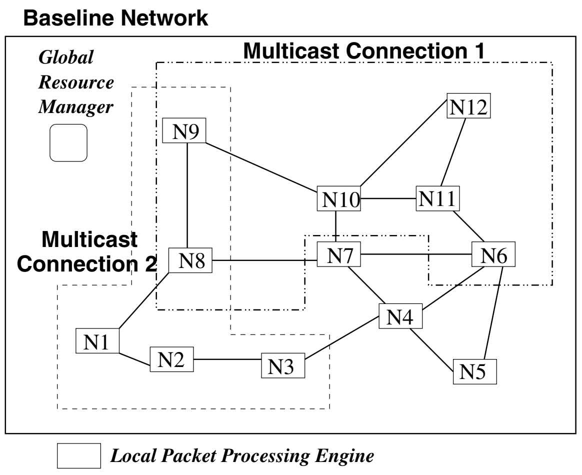


Fig. 1. A baseline network consisting of 12 nodes that supports two multicast connections. Multicast connection 1 consists of N12 (root), N11, N6, N10, N9, N8 (in depth-first order), whereas multicast connection 2 consists of N1 (root), N2, N3, N8 and N9 (in depth-first order). A single baseline network node can participate in the support of multiple multicast connections, e.g., N8 and N9. Each multicast connection is isolated from the others in terms of performance and reliability behavior. There is a *local packet processing engine* (LPPE) associated with each baseline network node for run-time resource usage monitoring and control, and there is a *global resource manager* (GRM) to oversee the resource allocation and usage reporting for the entire baseline network.

## III. System Architecture

*Sago* assumes that there exists a baseline network with a fixed topology and each link on the baseline network has a certain bandwidth capacity. Such a baseline network can be a physical network such as an optical transport network, or itself a logical network such as an ATM-enabled network with each link being a virtual circuit. In addition, on each baseline network node, there is a dedicated *Sago* server (LPPE) that performs resource usage control on packets that go through it. In *Sago*, packets of a multicast connection are required to traverse through baseline network nodes that are chosen to implement the multicast connection. For example, in Figure 1, N1, N2, N3, N8, and N9 are involved in implementing multicast connection 2, and therefore all packets from N1 to N3 have to go through N2. Typically baseline network nodes correspond to Internet points of presence (PoP). The LPPEs only need to be co-located within the PoPs' network operation center; they do not need to be directly on the paths that connect neighboring PoPs. Therefore, LPPEs are physically separate from Internet routers. The reason that co-location is sufficient for *Sago* is because the internal local area network within a PoP's operation center is typically provisioned with a much higher bandwidth than the aggregate of its external links. Therefore, the only network resource that *Sago* manages is the bandwidth on wide-area network links between PoPs.

Given a multicast connection specification, the GRM first chooses a set of baseline network nodes and links that are equipped with sufficient resources to meet the multicast connection's bandwidth/reliability QoS, and then makes necessary resource reservation requests to the LPPEs on the selected baseline network nodes. After a multicast connection is set up and in operation, the GRM collects resource usage information from LPPEs and either presents them through real-time display or stores them in persistent storage for subsequent on-demand querying.

As each baseline network node's LPPE may support multiple multicast connections, it has to ensure that packets from each multicast connection do not overuse the link bandwidth on the baseline network. Therefore the LPPE needs to support link bandwidth management to enforce network resource usage control and thus provide performance isolation among multicast connections that are multiplexed on the same physical nodes and links.

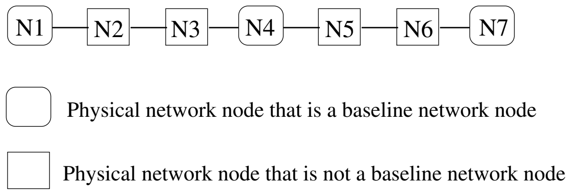

N1
N2
N3
N4
N5
N6
N7
Physical network node that is a baseline network node
Physical network node that is not a baseline network node

Fig. 2. An example physical network topology that shows how packets belonging to a *Sago* multicast connection are tunneled and routed through the underlying physical network. In this case, N1, N4 and N7 are physical network nodes that are also baseline network nodes each of which has an LPPE, whereas N2, N3, N5 and N6 are physical network nodes that are hidden from the baseline network.

Unlike IP multicast, *Sago* supports a *reliable link* abstraction between a pair of baseline network nodes, with the same level of reliability as TCP. Such an abstraction is particularly useful for multicast applications that need reliable data transport. Note that bandwidth guarantee does not imply no packet drop, because in the presence of transient packet bursts, LPPEs still need to drop those packets that overflow smoothing buffers.

A multicast connection is implemented as a distribution tree. Each link of a multicast distribution tree is either implemented as a TCP connection if it is a reliable link or a UDP connection if it is an unreliable link between the LPPEs that are on the two ends of the link. Payload packets are tunneled through the links of a multicast distribution tree. For user hosts to inject packets into or receive packets from a *Sago* multicast connection, an *Sago* gateway, similar to VPN gateways, is needed to tunnel and de-tunnel packets on the transport path. Currently *Sago* uses the combination of source port number and destination port number of the TCP or UDP connections as the unique ID of a multicast connection. In other words, within *Sago*, all TCP or UDP connections used to support a multicast distribution tree use the same pair of source port number and destination port number, which is chosen by the GRM at the time of setting up the multicast connection. LPPEs base both bandwidth management and packet routing on this connection ID.

There are two levels of routing in *Sago*. When a packet is tunneled between two LPPEs, the packet goes through the intermediate routers according to standard Internet routing protocol. However, the routing decision of going from one LPPE to the next is based on the unique ID of multicast connections and the routing table entries that the GRM sets up in the LPPEs at initial configuration time.

For example, Figure 2 shows a simple multicast distribution tree that consists of three network nodes, N1, N4, and N7, and two links, one from N1 to N4 and the other from N4 to N7. The baseline network link from N1 to N4 in turn passes through the physical network path N1-N2-N3-N4, and that from N4 to N7 through N4-N5-N6-N7. Assume that the unique network ID for this multicast connection is $< 50, 60 >$ (source port number and destination port number). So packets of this multicast connection that N1 sent out have the quadruple header $< N1, 50, N4, 60 >$. As these packets arrive at N2, they are routed to N3 based on the unique network ID $< 50, 60 >$, and similarly at N3. Eventually when these packets reach the TCP/UDP process at N4, it makes the routing decision based on the payload packet's header according to a multicast connection-specific routing policy, and decides to forward it through N7. As a result, these packets now have the header $< N4, 50, N7, 60 >$ when they are out of N4.

## IV. Network Resource Mapping

Given a multicast connection request's specification, *Sago*'s network resource mapping algorithm aims to identifying a set of baseline network nodes and links that implement this multicast connection such that as many future multicast connection requests can be admitted into the system as possible. Given that network link bandwidth is the precious resource, we will focus on the bandwidth issue in the following.

In *Sago*, a multicast connection is characterized by a sender and a set of receivers. Let's first consider a uni-directional unicast connection, which is a degenerate multicast connection. The goal of the network resource mapping algorithm is to identify a set of baseline network nodes and links that collectively implement a given unicast connection, $< s, d >$. On one hand, the end-to-end latency of the baseline network path chosen for the unicast connection, should be as small as possible. This argues for a path with minimum hop count. On the other hand, the loads on the links of the baseline network should be as balanced as possible so that the number of future unicast connection requests that are rejected because of "capacity fragmentation" is minimized. For example, it is possible that a unicast connection request is turned down because it must use a certain baseline network link, say $l$, but link $l$ does not have sufficient bandwidth. However, this lack of link bandwidth on $l$ is actually artificial because it could have been avoided had the algorithm shifted some of the load on $l$ to other links when servicing previous connection requests. This problem is similar to the fragmentation problem in segment-based memory management. Therefore, our goal is to develop a network resource mapping algorithm that constantly maintains a balance among the baseline network links' loads in order to support the maximal number of unicast connection requests that the baseline network theoretically can.

A more careful examination of this problem reveals that the notion of load balance is not as straightforward as in other computer system resources, such as CPU or disk. The key observation is that given a network topology, certain links are required to shoulder more load than others, assuming that each possible pair of nodes in the network are equally likely to be the source and destination pair of a network link request. Therefore, it may not be a good idea for the network resource mapping algorithm to attempt to equalize the residual bandwidth of each baseline network link. In other words, load balance does not necessarily mean equal residual link bandwidth in this context.

The goal is thus to identify the impact of choosing a baseline network link to satisfy a unicast connection request on the available bandwidth of each pair of baseline network nodes. The mincut of a pair of nodes $x$ and $y$, $mincut(x, y)$, represents the set of links that render $x$ and $y$ disconnected when they are removed. The maximum flow value between $x$ and $y$, $maxflow(x, y)$, is the sum of the residual bandwidth of the links in $mincut(x, y)$, and thus represent the available bandwidth between $x$ and $y$. Therefore, choosing a link $l$ to service a new unicast connection request can potentially impact the available bandwidth of all pairs of baseline network nodes whose mincut includes $l$ as a member. Once the total impact of a baseline network link on the available bandwidth between all pairs of network nodes is known, the network resource mapping problem is reduced to identifying a baseline network path for a unicast connection that have the minimal total impacts on available bandwidth.

The quantitative impact of a network link $l$ on the available bandwidth of a pair of network nodes, say $x$ and $y$, is difficult to determine exactly. However, one can approximate the impact by assuming that only links in $mincut(x, y)$ can possibly affect $maxflow(x, y)$. Moreover, the degree of impact is the same as the amount of bandwidth taken out of the residual bandwidth of link $l$ when it is chosen to service a unicast connection request. In general neither of the above assumptions is necessarily true because the mincut of $x$ and $y$ and thus their maxflow value may be completely different when some link in the baseline network is removed (or used up).

Since link $l$ has the same impact on the available bandwidth of those pairs of nodes whose mincut includes $l$, one can simply count the number of mincuts of which $l$ is a member to arrive at the total impact of $l$ on the network, i.e., $impact(l) = \sum_{l \in mincut(x,y)} 1$. But the impact on the available bandwidth of a pair of nodes should be weighted based on its current available bandwidth, so that one can meaningfully sum the impact on each pair of nodes into the total impact of $l$, i.e.

$impact(l) = \sum_{l \in mincut(x,y)} \frac{1}{maxflow(x,y)}$. However, this formula is assuming that each pair of baseline network nodes is equally likely to be used to support a unicast connection and each unicast connection request has the same bandwidth requirement. Neither assumption is true in practice. Therefore, for each pair of baseline network nodes, we compute its *past load*, which represents the total amount of bandwidth demand when the pair are involved in a unicast connection in the past. For those pairs of network nodes that are in greater demand, their impacts should be weighted more in the total impact sum. Therefore the final formula for total impact of link $l$ on the network's available bandwidth is

$$impact(l) = \sum_{l \in mincut(x,y)} \frac{pastload(x,y)}{maxflow(x,y)} \qquad (1)$$

Given a unicast connection request, characterized by source $S$, destination $D$ and bandwidth demand $B$, the network resource mapping algorithm proceeds as follows:
1. Compute the mincuts of all possible pairs of baseline network nodes based on their residual link bandwidth.
2. Compute the total impact of each baseline network link according to Equation (1), and use the result as the link's weight.
3. Eliminate all links in the baseline network whose residual bandwidth is smaller than $B$, and compute a minimum weighted shortest path on the reduced network between $S$ and $D$.
4. Decrease the residual bandwidth of all links on the weighted shortest bandwidth by $B$.
5. Increase pastload(S,D) by $B$.
The above algorithm needs to compute all-pair mincuts for each new unicast connection request, a very expensive operation in terms of performance overhead. However, there are several possible optimizations that can reduce this computation cost at the expense of load balancing accuracy. One possibility is to perform step (1) and (2) in the above algorithm once per $M$ (¿ 1) unicast connection requests, rather than once per unicast connection request.

It is interesting to note that the total impact of link $l$, a shown in Equation (1), does not have anything to do with its residual link bandwidth. That is, the notion of load balancing in network resource mapping is actually independent of the load or residual capacity of network links! However, there are second-order effects that residual link bandwidth may play a role. For example, between two links that have the same total impact, it is preferable to choose the link with lower residual bandwidth, because this potentially leads to less fragmentation.

To generalize the above algorithm to single-sender multiple-receiver multicast connections, compute a weighted minimum steiner tree rather than a weighted shortest path in Step (3) for the given multicast connection. Then in Step (4), decrease residual bandwidth of all links on the weighted minimum steiner tree by $B$. Finally, Increase the past load of each sender-receiver pair in the multicast group by $B$ in Step (5).

## V. Resource Usage Control and Management

### A. Link Bandwidth Management

Because multiple multicast connections may share a baseline network at the same time, the baseline network link bandwidth must be carefully managed and controlled to provide performance isolation among co-exiting multicast connections. In the *Sago* architecture, there is one LPPE associated with each baseline network node, which could have one or multiple baseline network links to neighboring baseline network nodes. At the same time, each baseline network node and thus its associated LPPE may participate in the support of multiple multicast connections. In general, multiple multicast connections may time-multiplex the same baseline network link.

For each out-going baseline network link, the LPPE allocates a queue for each multicast connection supported by the baseline network link. When a packet arrives through a network interface, if it is destined to a local process on the LPPE, it is sent up through the protocol stack; otherwise the LPPE looks at the packet's source and destination port number to determine the appropriate queue to enqueue it. Payload packets that go up through the protocol stack will eventually come down, and the LPPE will process them in the same way as if they arrive through the network interface. Control packets such as routing protocol or network management packets may go up but not necessarily come down.

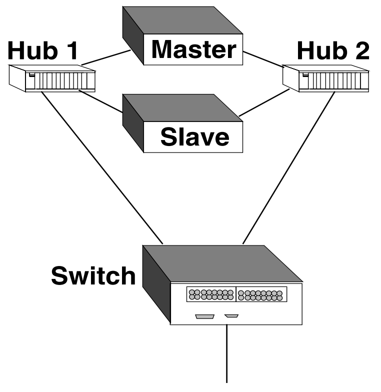
*Sago* uses a work-conserving weighted round robin algorithm to schedule the packets of competing multicast connections that share the same baseline network link. Since an LPPE may have multiple baseline network links, it needs to perform hierarchical weighted round robin packet scheduling. That is, it needs to cycle through the queues associated with each baseline network link at a different frequency, depending on the bandwidth capacity specification of each baseline network link. Moreover, spare bandwidth on one baseline network link cannot be re-allocated to queues associated with another baseline network link.

*Sago* performs weighted round-robin scheduling by visiting the per-connection queues periodically, and at each visit to a multicast connection's queue, it adds a credit to the queue's balance and continues to transmit packets from the queue until it's balance becomes negative or until the queue becomes empty. The credit is equal to the multiplication of the multicast connection's bandwidth reservation and a pre-determined cycle time, which is the visiting period. A queue can carry unspent credit over to the next cycle. However, to avoid long bursts, *Sago* sets a limit on the amount of credit that is allowed to accumulate. In this scheduling algorithm, as long as the packet scheduler can cycle through the queues within the cycle time, each queue (and therefore each multicast connection) is guaranteed its reserved bandwidth capacity. Our prototype implementation experiences show that the weighted-round-robin packet scheduler is able to keep up with fine-grained cycle time, e.g., 50 msec, for up to 10,000 connections on standard PC hardware. In all cases, packet forwarding becomes the performance bottleneck long before packet scheduling does.

### B. Packet Forwarding

On an LPPE, packets of a non-reliable multicast connection arrive at an network interface, get DMAed into a buffer memory, and eventually dispatched by the packet scheduler to the corresponding network interface. The entire process takes place inside the kernel. In addition, to avoid network interrupt overhead, network interrupt is completely disabled, and packet receiving, scheduling and dispatching are organized as a tight polling loop inside the kernel. Periodically the control is transferred to a user-level process, which performs connection set-up/tear-down and network management chore. On a Pentium-III 450 MHz machine, the packet forwarding rate of the current LPPE implementation is more than 200,000 packets/sec.

To implement a reliable link between a pair of baseline network nodes, a separate TCP connection is set up between the associated LPPEs for each multicast connection. Therefore packets of a reliable multicast connection need to travel through a user-level TCP server when passing through an LPPE. *Sago* chooses to implement reliable links through TCP because it attempts to reuse the standard TCP implementation in the Linux kernel. The disadvantage of this approach is that reliable multicast packets incur a higher latency because it involves two kernel-user protection boundary crossings. The advantage is that *Sago* can leverage the buffering, retransmission, and flow control of TCP to support reliable multicast for free. Because LPPE completely eliminates network interrupts, running user-level TCP server processes may cause packets to be dropped at the input interfaces because the LPPE core does not poll the interfaces sufficiently frequently. To address this issue, the TCP receive and send system calls are augmented with code to poll network interfaces so that the desired interface polling frequency is maintained even when the TCP server processes take con-



**Rest of the baseline network**

Fig. 3. The fully redundant hardware architecture of the LPPE provides node fault tolerance against single device, network interface, and hub failure.

trol of the CPU most of the time.

## VI. Fault Tolerance

*Sago* is designed to tolerate both single node failure and link failure at the baseline network level. It includes a separate mechanism for node and link fault tolerance.

### A. Node Fault Tolerance

Full redundancy is built into each LPPE in that each LPPE consists of a master and a slave device. As shown in Figure 3, the master and slave of an LPPE share two hubs to connect to the rest of the baseline network. *Sago* uses two hubs to tolerate single hub failure. Typically hub 1 is active and hub 2 is backup. Therefore, incoming packets are automatically replicated to the master and slave simultaneously. Because it is possible that transient network interface errors may cause one device to receive a packet that does not reach the other device, *Sago* cannot assume 100% reliable packet replication.

Periodic heartbeat messages are exchanged through both hubs between the master and slave to keep an eye of each other's health. If the master detects the slave is dead, it sends an alarm message to the system administrator. If the slave detects the master is dead, it sends an alarm message to the system administrator and takes over as the master. When the slave does not receive $N_{hb}$ (initially set to 10 in the current design) consecutive heartbeat messages in a row from the master through both hubs, there are several possibilities:
1. The master device is dead.
2. The master device's both network interfaces are dead.
3. The slave device's both network interfaces are dead.
4. One interface on the master and one interface on the slave are dead.

Note that a hub failure has the same effect as failure of both interfaces connected to the hub. There are two ways to determine whether a network interface is still alive: (a) by using *ping* to probe some well-known hosts through the tested network interface, and (b) by checking the status register on the network interface card. (b) is not always supported on all network interface hardware. The slave can use either of the two methods to probe the aliveness of its two interfaces. If both are dead, it does nothing. If both are alive, it takes over as the master. If one is dead and the other is alive, it broadcasts the heartbeat message through the healthy interface in the hope that the master can receive it through the other interface. The heartbeat message should include the result of network interface aliveness information. Cases (1) to (3) can be determined by the master and slave locally. Case (4) however, requires broadcast of heartbeat messages between the master and slave to make sure that this failure scenario is indeed the reason why heartbeat messages cannot get through both interfaces. For Case (4), eventually the master (slave) should remain as the master (slave) and both switch to the healthy interface if necessary; also, the heartbeat messages need to traverse through both hubs from this point on.

It is conceivable that loss of a sequence of heartbeat message is due to congestion, rather than any hardware failure. In that case, both devices may turn into a master. Eventually when congestion fades away, heartbeat messages get through again. A dual-master resolution mechanism based on the MAC address is used to turn the device with larger MAC back to a slave. To prevent the "dual masters" scenario from occurring frequently due to congestion-related heartbeat message loss, *Sago* dynamically adapts the value of $N_{hb}$, the threshold to start suspecting that there is a failure, according to instantaneous traffic load measured at run time.

*Sago* takes a *process pair* approach for fault recovery. That is, the master and slave are initialized with the same starting state and execute the same software at both the kernel and user level. They are not working in lock steps. But given the same packet stream, the critical internal states of the master and slave are tightly synchronized. The only difference between the master and slave is that packet transmission is suppressed at the slave except the heartbeat messages. Therefore, if the master dies, a slave can take over as the master without any explicit state transfer, requiring only a change to an internal state variable to indicate that it is now the master, which in turn enables the packet transmission capability.

To maintain a reliable link between a pair of LPPEs, each of which consists of a master and a slave, requires careful synchronization between the master and slave to preserve TCP's reliability semantics. First of all, when the master and slave are started, they are given the same *seed* variable that is used to generate subsequent TCP sequence numbers. This guarantees that the master and slave always use the same starting sequence numbers for new connections. Second, instead of simply dropping all TCP packets, the slave forwards the ACK sequence number to the master, which can ACK a sequence number if and only if it receives an ACK from the slave that acknowledges the same sequence number. Of course, this additional synchronization constraint should be turned off when the slave is determined to be dead. These two modifications guarantee that the master and slave of each end of a TCP connection maintain the same session state for that TCP connection, including both sequence numbers and buffered segments. Therefore, a slave can immediately take over as the master without losing any TCP packets, thus upholding TCP's reliability guarantees. Because modification of an LPPE's critical state, such as routing state and bandwidth reservation requests, is triggered by messages sent through TCP, preserving TCP's semantics has the nice side effect of ensuring that connection-related states of the master and slave are always kept consistent with each other.

### B. Link Fault Tolerance

A multicast connection request specification includes a reliability attribute in terms of link failure recovery time. To achieve comparable fault recovery time as Sonet, 50 msec, *Sago* sets up a backup path for each *link path* of a multicast connection that requires fast failure recovery. A link path is a sequence of adjacent links on a multicast distribution tree that starts with a fork node or the root, ends with a fork or a leaf node, and does not pass through any fork node. Given a path, there are three general approaches to path redundancy management:
- *1+1 Redundancy*: A dedicated backup path is set up for each active path and packets are sent on both the active and backup paths.
- *1:1 Redundancy*: A dedicated backup path is set up for each active path but packets are sent only on the active paths.
- *N:1 Redundancy*: Multiple active paths share a backup path and packets only sent on the active paths.

*Sago* adopts the N:1 Redundancy scheme to minimize the cost of link fault tolerance. Another issue of backup path construction is whether to use link-by-link backup or path-disjoint backup. *Sago* chooses path-disjoint backup because previous study [15] showed that path-disjoint backup is in general more bandwidth-efficient.

To detect link failures, LPPEs on neighboring baseline network nodes exchange heartbeat messages between themselves. Loss of a consecutive sequence of heartbeat messages triggers an aliveness probe, which results in either a node or a link failure indication. The LPPE that pinpoints the failure notifies the GRM, which consults the network resource map to identify all link paths that are affected, and informs the LPPEs at the two ends of each of the affected link path. When an LPPE receives such a notification from the GRM, it modifies the low-level routing table entry corresponding to each affected link path to follow its backup path. The routing state of the LPPEs on the backup paths are typically set up appropriately at the network initialization time.

Finding an optimal shared redundant path for multiple active paths has been shown to be NP hard [9]. *Sago* chooses a heuristic algorithm called the shared backup-path construction (SBC) algorithm to solve this problem. Each link path in a multicast distribution tree is serviced by an active and a backup bath. Given a new link path request, the SBC algorithm needs to reserve the request's bandwidth demand on each link of the chosen active path. However, it does not always need to reserve the same amount on the links of the backup path, because backup link bandwidth can be shared among multiple link paths. If the active path of a link path request $R1$ shares at least one baseline network link with that of another link path request $R2$, then $R1$ and $R2$ are *conflicting* and the bandwidth reservation on the baseline network links that $R1$ and $R2$ both use as part of the backup path should be the sum of the bandwidth demands of $R1$ and $R2$, i.e., no redundancy sharing. On the other hand, if the active paths of $R1$ and $R2$ are disjoint, then the bandwidth reservation on the links that $R1$ and $R2$ both use as part of the backup path is the maximum of the bandwidth demands of $R1$ and $R2$. Therefore, for a baseline network link that serves on the backup path of a set of link path requests, the total amount of backup bandwidth reservation on this link is the maximum of the bandwidth sums of all subsets of these link path requests in which at least one member is conflicting with all the rest. Give a set of active link paths to be protected, the heuristic SBC algorithm builds up the backup path for each active path one by one as follows:

1. Take out the active path being considered, say $P$, from the baseline network to form a reduced graph so that the backup path to be constructed is guaranteed to be path-disjoint with $P$.
2. Apply a weighted shortest path algorithm on the reduced graph to find a backup path for $P$, where the weight of each link on the backup path is the *increase* in the link's backup bandwidth reservation as a result of including this link as part of the backup path for $P$.
3. Update the residual bandwidth and backup bandwidth of the links on the chosen backup path.

The incremental bandwidth cost for including a baseline network link as $P$'s backup path is calculated by first identifying all the active paths that are conflicting with $P$ and that also include the link under consideration into their backup path, then summing up the bandwidth demands of these active paths and $P$'s, and finally subtracting this sum from the backup bandwidth already reserved on the link. If the subtraction result is negative, the incremental cost is zero; otherwise the subtraction result is the incremental cost. By using the bandwidth increment as the weight, the weighted shortest path algorithm is more likely to include those baseline network links that allows backup bandwidth sharing. For the active path $P$ whose backup path is being constructed, these are the links that are used as part of the backup path of those link path requests whose active path is not conflicting with $P$.

## VII. Conclusion

As content replication and delivery moves to the center stage, distribution of real-time content over the Internet is an obvious next step. *Sago* provides the essential tool to allocate, route and manage application-level multicast connections that are reliable, bandwidth-guaranteed, and fault-tolerant. Because *Sago* greatly simplifies the process of network resource provisioning, it renders possible the business model of *bandwidth retailing*, which allows a specialized content delivery network service provider to re-sell wide-area guaranteed-QoS multicast connections to Internet content providers on an on-demand basis. The novel features of the *Sago* system include a network resource mapping algorithm that takes into account network topology and dynamic input request distribution when balancing the load on network links, a comprehensive fault-tolerance mechanism that can tolerate single link and node failure, and a highly efficient packet forwarding engine that also supports hop-by-hop reliable data transport. Measurements on a fully operational *Sago* prototype show that the average packet latency is under 0.06 msec even at input loads as high as 80 Mbits/sec.

Currently we are working on the gateway subsystem that interfaces between the data sender/receiver at the end user site and the peripheral LPPEs on the content distribution network. We are also embarking on a comprehensive evaluation of the effectiveness of the proposed network resource mapping algorithm and the shared backup-path construction algorithm using a wide variety of network topologies and input request distributions. Because *Sago* makes very little assumption on the baseline network, the *Sago* technology could also be applied to the management of optical transport networks, with the major difference being the basic unit of resource allocation for the latter is an optical channel of fixed bandwidth. Finally, we are generalizing *Sago* to a full-scale overlay network management system, which allows a physical network to support an arbitrary number of logical overlay networks each of which is specified with any combination of attributes that are traditionally associated with a physical network, e.g., topology, link bandwidth, link reliability, and even control-plane/data-plane processing protocols.

## References

[1] Testbed for deployment of IPv6. http://www.6bone.net.

[2] Aiken, R., et al., "Architecture of the Multi-Modal Organizational Research and Production Heterogeneous Network (MORPHnet)," ANL-97/1, Argonne National Lab, IL, Jan. 1997.

[3] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris "The Case for Resilient Overlay Networks," Proc. HotOS VIII, Schloss Elmau, Germany, May 2001.

[4] Braden, B., "A Plan for a Scalable ABone - A Modest Proposal," (work in progress), July 1999.

[5] Campbell, A., et al., "Spawning Networks," IEEE Network, July/Aug. 1999, pp. 16-29.

[6] Chandra, P., et al., "Darwin: Resource Management for Value-Added Customizable Network Service," Sixth IEEE Int?l Conference on Network Protocols (ICNP'98), Austin, Oct. 1998.

[7] Chiueh, T.; Vernick, M.; Venkatramani, C., "Integration of Real-Time I/O and Network Support in Stony Brook Video Server," IEEE Network Magazine, Vol.13, No.2, p 30-36, April 1999.

[8] Delgrossi, L., Ferrari, D., "A Virtual Network Service for Integrated-Services Internetworks," 7th International Workshop on Network & OS Support for Digital Audio & Video, May 1997.

[9] B. Doshi, et al., "Optical network design and restoration," Bell Labs Technical Journal, January-March, 1999.

[10] Eriksson, H., "MBone: The Multicast Backbone," Communications of the ACM, Aug. 1994, pp.54-60.

[11] Katz, R. et al., "The SAHARA Project: A Revolutionary Service Architecture for Future Telecommunications Systems," http://sahara.cs.Berkeley.edu/.

[12] M Kodialam and T. V. Lakshman, "Dynamic routing of bandwidth guaranteed tunnels with restoration," in Proceedings of the Conference on Computer Communications (IEEE INFOCOM), Mar. 2000.

[13] M. Kodialam and T. V. Lakshman. Minimum Interference Routing with Applications to MPLS Traffic Engineering. In Proc. of IEEE Infocom, 2000.

[14] P.V.N. Koppol, T.V. Lakshman, H. Sarin, and B. Suter, "RATES: A server for MPLS traffic engineering," IEEE Network Magazine, pp. 34–41, March/April 2000.

[15] K. Murakami, H. Kim, "Optical capacity and flow assignment for self-healing ATM networks based on line and end-to-end restoration," IEEE/ACM Transactions on Networking, Vol. 6 No. 2, April 1998.

[16] K. Murakami, "Comparative study on restoration schemes for ATM networks," IEEE INFOCOM 1997.

[17] Rodeh, O., Birman, K., Hayden, M., Dolev, D., "Dynamic Virtual Private Networks," TR98-1695, Dept. of Computer Science, Cornell University, Aug. 1998.

[18] Savage, S., Anderson, T., et al., "Detour: a Case for Informed Internet Routing and Transport," IEEE Micro, V19, N1, Jan. 1999, pp. 50-59.

[19] Scott, C., Wolfe, P., Erwin, M., *Virtual Private Networks*, O'Reilly & Assoc., Sebastapol, CA, 1998.

[20] Touch, J., Hotz, S., "The X-Bone," Proc. Global Internet Mini-Conference/Globecom, Nov. 1998.

[21] Yemini, Y., da Silva, S., "Towards Programmable Networks," IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, LAquila, Italy, Oct. 1996.