July 1989 Report No. STAN-CS-89-1273

Sirpent™ : A High-Performance Internetworking Approach

by

David R. Cheriton

Department of Computer Science

Stanford University

Stanford, California 94305

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-89-1273	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION STANFORD UNIVERSITY; 6b OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION
6c ADDRESS **(City, State,** *and ZIP Code)* STANFORD, CA 94305	7b ADDRESS **(City, State, and ZIP** *Code)*
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA; 8b OFFICE SYMBOL **(If applicable)**	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00039-84-C-0211
8c. ADDRESS *(City,* **State, and ZIP** *Code)*	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE **(Include** *Security Classification)*

Sirpent^{TM}: A High-Performance Internetworking Approach

12 PERSONAL AUTHOR(S) David R. Cheriton

13a TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT 12

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

A clear target for computer communication technology is to support a high-performance global internetwork. Current internetworking approaches use either concatenated virtual circuits, as in X.75, or a "universal" internetwork datagram, as in the DoD Internet IP protocol and the ISO connectionless network protocol (CLNP). Both approaches have significant disadvantages.

This paper describer Sirpent^{TM}(Source Internetwork Routing Protocol with Extended Network Transfer)', a new approach to an internetwork architecture that makes source routing the basis for interconnection, rather than an option as in IP. Its benefits include simpic switching with low per-packet processing and delay, support for accounting and congestion control, and scalability to a global internetwork. It also supportr flexible, user-controlled routing such as required for security, policy-bared routing and real-time applications. Wc also propose a specific internetwork protocol, called VIPER^{TM2}, as a realisation of the Skpcnt approach.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE *(Include Area Code)*; 22c OFFICE SYMBOL

Sirpent™: A High-Performance Internetworking Approach

David R. Cheriton
Computer Science Department
Stanford University

Abstract

A clear target for computer communication technology is to support a high-performance global internetwork. Current internetworking approaches use either concatenated virtual circuits, as in X.75, or a **"universal"** internetwork datagram, as in the **DoD** Internet IP protocol and the **ISO** connectionless network protocol (CLNP). Both approaches have significant disadvantages.

This paper describes **Sirpent**^{TM}(**Source** Internetwork Routing Protocol with Extended Network Transfer)[1], a new approach to an internetwork architecture that makes source routing the basis for interconnection, rather than an option as in IP. Its benefits include simple switching with low per-packet processing and delay, support for accounting and congestion control, and scalability to a global **inter**network. It also supports flexible, user-controlled routing such as required for security, policy-based routing and real-time applications. We also propose a specific internetwork protocol, called VIPER^{TM}[2], as a realization of the **Sirpent** approach.

1 Introduction

A high-performance global internetwork is required to support the next generation of computer communication and distributed systems. Current internetworking approaches use either concatenated virtual circuits (CVC), as in X.75, or a 'universal' internetwork datagram, as in the Internet IP protocol and the IS0 connectionless network protocol (CLNP). The CVC approach requires a circuit setup between endpoints before communication can take place, introducing a full roundtrip delay. It also requires a significant amount of state in the gateways to maintain connection state. (However, the circuit provides a basis for access control, accounting, resource reservation and efficient addressing.)

The highly bursty traffic characteristic of most computer communication makes the CVC approach ill-suited for computer networks. Either the circuit setup cost is incurred frequently or else circuits are held and not well utilized over long periods of time. The latter incurs a connect time cost forced by the costs of switch state and bandwidth reservation associated with a circuit. This traffic is expected to become burstier as the data rates of networks move much higher than that sustainable by computer peripherals and that required by the applications. For example, an 8 Mb data stream appears as periodic bursts of packets on a gigabit channel, using less than 1 percent of the bandwidth. Moreover, increases in transactional traffic, such as credit card transactions, make the logical connections even shorter. Performance improvements in data rates, processing speeds and software leave the roundtrip time a dominant factor in response time.

Datagram-oriented internetworking requires no connection setup and little state in the gateways, avoiding the problems of the CVC approach. However, the IP approach does not provide for access control or congestion control and requires a significant amount of per-packet processing in the routers. In particular, each router must (or at least, is supposed to) determine the next hop of the route from the destination address, update the Time To Live (TTL) field, possibly fragment the packet and update the header checksum before sending on the packet. As a consequence of this processing, each packet suffers a reception, storage and processing delay at each router.

The cost and complexity of these internetworking approaches are growing concerns because of the increasing speed of networks, the increasing demands for functionality, and the growing scale of internetworking. Local and wide-area networks are progressing into 100 megabit data rates and soon gigabit data rates. Requirements for congestion control and policy-based routing [6] appear to require even more per-packet processing and mechanism in each router. Thus, these problems are expected to become far more pronounced.

This paper describes **Sirpent** (Source Internetwork Routing Protocol with Extended Network Transfer)[3], a new approach to an internetwork architecture that makes source

[1] **Sirpent is a** trademark of Stanford University.
[2] VIPER is **a** trademark of Stanford University.

[3] **The** name ***Sirpent*** *is also* **supposed** to be suggestive of the way **a** packet **'snakes'** its way through the network using **cut**-through switching. ***Cut-through*** routing or **switching** [18] refers to switching the packet out the outbound port before the entire **packet has been** received, eliminating the delay for packet **recep**tion and storage **that arises** with conventional **store-and-forward switching.**

routing the basis for interconnection, rather than an option as in IP. Its benefits include simple switching with low per-packet processing and delay, support for accounting and congestion control, and scalability to a global internetwork. It also supports flexible, user-controlled routing as required by security considerations, policy-based routing and real-time applications. We argue that source routing is the correct fundamental basis for an internetwork architecture, showing that IP-style addressing can be viewed as a restricted special case. We also argue that the Sirpent design is a result of further application of the end-to-end argument [21]. Function is moved out of the lower layers, making them simpler and providing the higher layers with more functionality. The lower layers only provide this functionality (routing in this case) when beneficial as a performance enhancement.

The next section describes the Sirpent approach, its realization in a high-performance gateway, the handling of access control, accounting and congestion, and its scalability. Section 3 discusses the extension of internetwork directories to provide a routing service in support of Sirpent. Section 4 describes some implications for transport protocols that use Sirpent. Section 5 describes a specific protocol proposal, VIPER, for realizing Sirpent. Section 6 provides some preliminary performance analysis of the Sirpent approach using the VIPER protocol specifics. Section 7 relates Sirpent and VIPER to prior work. We conclude with a brief assessment of the approach and an indication of the future directions.

2 Sirpent Design

Each Sirpent packet is structured as a sequence of *header segments* followed by user data, followed by the Sirpent trailer. Each header segment corresponds to a Sirpent router along the route and contains the following information:

port - identifier of the output port that the packet should take in the router to which the header corresponds.

type of service - priority and handling of the packet when it is blocked at the router, namely preempt, save or drop.

portToken - authorization for the outport port. The port token may also provide authoriration to return through this port.

portInfo - potentially network-specific field that specifies information required as part of transmission through the specified output port. This field includes a tag field indicating the format of the rest of the packet. In particular, it indicates the protocol for interpreting the next header, if any.

The portToken and portInfo fields are variable-length so an actual header contains length fields as well, which are not shown here for brevity. The portInfo field commonly contains a network-specific header that has additional network-specific information. The portToken is optional. For example, a packet routed to an Ethernet without a port token would contain as next header segment:

```
[port,tos,enetHdr]
```

where "tos" is the typeOfService field. The 'enetHdr' field is a standard Ethernet header consisting of two 48-bit addresses, for source and destination, and a 16 bit protocol type field. The protocol type field serves as a tag field specifying the format of the rest of the packet. In particular, if the packet requires at least one more hop to reach its destination, the protocol type field contains a value associated with Sirpent indicating that the following portion of the header is another Sirpent header segment. Alternatively, the type field could designate a transport protocol if the destination Ethernet address is that of its final destination.

On transmission, a Sirpent packet has an initial header segment that corresponds to the type of network on which it is being transmitted, so the initial header segment format is implicit from the network type. For example, on transmission of a Sirpent packet on an Ethernet that is to be routed through router R on to another Ethernet, the initial portion of the packet would look as follows:

```
[enetHdr1,port,tos,portToken,enetHdr2,data]
```

The 'enetHdrl' field specifies as the Ethernet source address the address of the sending host, and the destination address of the router R. Its protocol type field is set to the value reserved to designate the Sirpent protocol on the Ethernet.

On reception of a Sirpent packet at a router (and demultiplexing the packet to the Sirpent module, based on the network-specific protocol type field), the router removes the network header from the front of the packet as well as the port, typeOfService and portToken fields. It checks the authorization provided by the portToken, if present, and aborts the transfer of the packet if not authorized. Otherwise it then revises the network-specific portion, if any, so that it constitutes a correct return hop through this router and appends the return port and network header fields to the end of the packet. For example, with an Ethernet header, the destination and source addresses are swapped. The packet is then forwarded out though the port specified by the port field. If this port is connected to a point-to-point link, the next router (or destination) is the node at the other end of the link. Otherwise, the portInfo field is a network-specific set of fields that include specification of the next recipient on the link. The format of the portInfo field is determined by the type of the port designated by the port field. For example, if the port field specifies an Ethernet, the Ethernet header (in the portInfo field) specifies the next node on the Ethernet attached to this port. So, continuing the previous Ethernet example, the forwarded Sirpent packet looks as follows.

```
[enetHdr2,data,returnPort,tos,portToken,rEnetHdr1]
```

The "rEnetHdrl" matches the original Ethernet header "enetHdrl" except that the source and destination fields are reversed. A length field (not shown) indicates the size of the Ethernet header, allowing network-independent manipulation of the header/trailer segments[4]. In this example,

[4]The network-specific portion for an Ethernet need not contain the full Ethernet header. For example, by agreement between the router and sources, the network-specific portion may contain only the destination and type fields, in which case the router would be responsible for supplying the Ethernet source address to form a full Ethernet header before forwarding the packet. It would also replace the datination address with the source address when moving the origind header segment information to the trailer.

the **portToken** is actually a link Token, authorizing transmission of packets back through this port as well. Otherwise, the destination must provide different tokens for the return route.

This process is repeated at each **Sirpent** router, so the packet is source-routed by the sequence of header segments from source to destination. The packet arriving at the final destination contains only the last header segment at the start of the packet with the rest of the original header contained in the trailer, modified so that it can be easily made into a return route to the source of the packet. In particular, to generate the return route, the receiver locates the beginning of the trailer of (former) header segments and copies each segment into a separate return address area in reverse order, swapping the port fields and updating the length fields as appropriate. Because the network-specific portions of the header segments have been modified as required by the routers along the original route, the reversal process is entirely network-independent.

Multicast can be supported in **Sirpent** by three mechanisms. First, port values can be reserved to specify multiple ports, rather than just one port. Designating one of these values as a port field causes the packet to be forwarded on each of these ports. A simple case of this approach is using a value as the broadcast port, causing transmission out **all** ports.

A second approach is to allow a tree-structured specification of the multicast route, as proposed with Blazenet. Effectively, there are multiple header segments specified for a routing point, with each header segment causing a copy of the packet to be routed according to the port it **specifies.**

Finally, one can use multicast agents at various points and route packets to these agents for 'explosion'. In this case, the **portInfo** field specifies the multicast agent protocol and contains the information required to explode the packet. This approach differs from the previous scheme primarily in that the full header is delivered to each of the multicast agents, rather than only delivering its portion of the route.

A combination of these approaches can be used. For example, the tree approach might be used for a source to route a packet to several wide-area broadcast networks which then deliver the packet simultaneously to a number of multicast agents, which in turn then handle local delivery.

Sirpent does not provide for fragmentation and reassembly. When a packet arrives that is too large for the next hop, this situation is only discovered when a portion of the packet has been transmitted, assuming the router is doing cut-through. We assume that the router has enough lookahead to realize this situation has **occured** before that actual maximum has been achieved. It then appends a special segment on the **trailer** (which is not a legal **Sirpent** header segment) indicating that the packet has been truncated. Thus, the receiver can detect packet truncation even when it only affects the packet trailer. As described in Section 3, the routing service provides the maximum transmission unit (MTU) along with the route to the client so there is no need to do MTU discovery in the same sense as conventional IP.

It is useful to compare the **Sirpent** approach to IP to show that **Sirpent** design does adequately cover the issues recognized in the IP design. **Sirpent** does not provide version or packet length fields. The network needs to be able to determine the end of a packet so the length is not transmitted explicitly by **Sirpent.** For instance, an Ethernet trailer sequence indicates the end of packet on the Ethernet?. **Sirpent** does not provide fragmentation and reassembly, eliminating the need for the associated fields. **Sirpent** does not provide options but does explicitly provide source routing and the **"record** route" facilities by its basic mode of operation. The other options, for security and timestamping are not supported as being too expensive to warrant inclusion. Secure handling of the packet can be ensured by the source selecting a secure route for the packet to take*.

The **Sirpent** header does not include a time-to-live field or other explicit means to limit packet lifetime. A **Sirpent** packet cannot loop infinitely at the **Sirpent** level because the header is finite and is reduced by each router. Infinite looping within a network can be handled by **network-specific** information if necessary, which can be provided in a **Sirpent** header segment as a network-specific version of the **portInfo** field. Each transport protocol be prepared to reject packets whose packet lifetime in the internetwork exceeds that which the protocol can handle correctly. (See Section 4)

Absence of checksum means that the header can be corrupted without that fact being detected immediately. As a consequence, the packet may be misrouted rather than dropped immediately, as done with IP. However, we note that the header is normally a small percentage of the full packet so, assuming random sources of corruption, the header has a low probability of corruption. Also, the probability of a packet with a corrupted header successfully routing further in the internetwork **is** quite low. So, with the low error rates expected for current and future networks, header corruption is a low probability event and the increased load on the internetwork of routing packets further after corruption is insignificant. With **Sirpent,** the transport layer must deal with misdelivered packets, as described in Section 4.

The advantages of **Sirpent** are further illustrated by considering a high performance realization (following some of the techniques of **Blazenet [11, 13]),** techniques for resource management and scalability, as described in the following three subsections.

2.1 High-Performance Routing

A high-performance **Sirpent** router uses cut-through routing as follows. As the packet header start to arrive, the router strips the header off to a **loopback** register. Placing the port field first allows the router to make the switching decision while the **typeOfService, portToken** and **portInfo** fields are being received. During this time, the router determines which of the following three actions is applicable:

- Route onwards.
- Route to a blocked packet handler.

[5]A packet can be padded with null bytes between the end of the actual data and beginning of the Sirpent trailer without confusion.

[6]However, the selection of a secure route is purely to reduce exposure to insecure portions of the network because errors in header information can cause a packet to be misdelivered into an insecure portion of the network. Security must be (and can be) provided by packet encryption.

- Route local.

In the first case, the portion of the packet after the packet header is routed out the appropriate port (in real-time) and the looped back portion is appended to the end of the packet after being delayed in the router until the end of the packet. That is, the packet is switched using cut-through routing to the output port. (The field-swap of the network-specific information is performed as part of the loopback process.) As part of routing onwards, the switch may abort a packet already in transmission on the given port if the new packet is of a preemptive priority and the current packet in transmission is not.

In the second case, the packet is deferred to a subsequent time, or dropped (depending on the networking technology and the type of service specified). Deferral may be accomplished by storing the packet, looping it back to a previous node (as done in Blasenet) or entering it into a local delay line to store the packet for some period of time. In any case, the original packet header can be easily retained by switching the rest of the packet to follow the header into the loopback register. The header segment can also be stripped off at this stage if appropriate. Local delivery can be handled as a special outgoing port that feeds into a local reception (memory) buffer or as a special case in the routing switch.

The type of service field determines whether a packet is retained when it is blocked and, if retained, the order of transmission of the currently blocked packets. That is, higher priority packets are retransmitted first.

If a **portToken** is present (and required), the router looks up the token in its token cache and checks authorisation. Because the token is an encrypted capability that may be difficult to fully decrypt and check in real time before the packet is forwarded, the router retains a cached version of the token such that it can check and authorise packet forwarding in real time from the cached version. A packet arriving with a token that has not been previously cached can be handled in several ways. First, it may be allowed through, deferring enforcement of full authorisation to subsequent packets, which *are* authorised by the cached token created from the first packet. This approach, what might be called optimistic *authorization* assumes that, in the worst case, one or a small number of unauthorised packets can be allowed through without significant problems[7]. Second, the initial packet can be handled as a blocked packet, the same as if the outgoing port is unavailable. The blocking action allows some time for the token to be processed, just as the blocking normally allows some time for the port to become free. Finally, the packet could be dropped. This approach might only make sense in a router in which blocked packets are dropped, thus reducing to the previous approach. In any case, the new token is decrypted, checked and cached (using the encrypted value as the key) to prepare for subsequent packets using the same token. If the token is invalid, the cached entry is flagged indicating a problem with packets carrying this token value. Subsequent packets using this token are then blocked. Cache entries are also used to maintain accounting information such as packet or byte counts to be charged to the account designated by the token.

[7]Malicious attacks of unauthorized packets with many different invalid tokens could be handled by the router switching to blocking authentication when excessive invalid tokens are received.

With this design of router, the delay through the router is reduced to the switch decision and setup time (if the output port is available and the token is cached). The switch decision and setup time can be made significantly less than a microsecond, given the simplicity of the switching decision. In any case, the "store" delay of conventional store-and-forward is eliminated so the packet delivery delay is basically the transmission time, propagation delay and sum of the queuing delays incurred at each router on the route taken by the packet. The real-time switching also preserves the gaps introduced by the sender using a rate-based transport protocol, such as VMTP [2] and Netblt [7]. (When a packet blocks, the gap is increased unless several packets going to the same source are similarly delayed, which we assume to be an unlikely event.) Finally, the type of service field allows the network to support a variety of types of traffic ranging from real-time video to file transfer while still only imposing the overhead of examining and acting on the type of service field when the packet is blocked. That is, if a packet can be routed immediately out its outgoing port with no contention from other sources, there is no need to examine its type of service field. With contention, the type of service field provides for preemption of interfering packets as well as prioritized queuing. Controlling the queuing delay and contention is the subject of the next subsection. Note that cut-through routing is only applicable when the input link and the output link are the same data rates. Because of the benefits of standardising on data rates, we expect this to be the case in significant portions of future internetworks.

2.2 Resource Management

Sirpent provides for access authorisation, accounting and congestion control using tokens and rate-based feedback control.

Authorization and Accounting Each token is an encrypted (difficult-to-forge) capability that identifies the port and type of service that it authorises, the account to which usage is to be charged, optionally a limit on resource usage authorised by this token, and whether reverse route charging is authorised (That is, the token can be used for the return route as well.). A source can only use those portions of the internetwork for which it can acquire valid tokens, and only within the type of service and resource limitations provided by the tokens. Thus, the internetwork can limit resource demands on a per-router basis by limiting the tokens issued to users. This approach is particularly appropriate for very high priority traffic for which significant queuing delay is not acceptable. (Using preemption, this traffic is not delayed by any lower priority traffic so contention only occurs between comparable priority traffic.)

Rate-Based Congestion Control For normal and low priority traffic, **Sirpent** uses rate control to limit the length of queues on a per-output port basis. In particular, using cut-through techniques and source routing, a packet is routed directly to the designated output port, avoiding input queuing. If the port is busy and the packet cannot preempt the currently transmitting packet, the packet is added to the output (priority) queue associated with the output port (assuming buffer space is available). The router monitors the output rate of the port. If the arrival rate to this port exceeds the output rate, the router signals to those "upstream" routers feeding this queue to reduce their rate of packets being transmitted to this queue. Because the upstream routers have access to the source route on each packet, they can d.etermine the packets destined for this queue. Because the congested router has access to the source route, it can easily determine the upstream routers feeding the queue. Each router rate-controlled by such a congestion point can further feed back rate control information to routers feeding its queues, indicating rate information based on packets going to specific congested router queues. (Links not feeding the queue are implicitly limited in rate and must progressively push the authoriaed rate up, similar to Jacobson's *slow* start approach [16], except applied at the network layer rather than the transport layer.)

In effect, the rate-limiting information builds up back from the point of congestion to the sources, dynamically generating *soft state* on *flows*[8]. The rate control state in the routers is similar in some ways to circuit state except: (1) as soft cached state, it can be discarded; (2) it arises per route and not per user; and (3) it is created dynamically according to need rather than explicitly when communication is initiated.

Any non-empty output queue indicates a (possibly temporary) mismatch between input rate and output rate. The buffer space allocated to the output queue provides a mechanism to absorb temporary mismatches. The rate control mechanism prevents there being a sustained mismatch. As a feedback system, this rate control approach necessarily oscillates. The degree of oscillation and its resulting effect on the utilization of the congested output link depends on the amount of output buffer space, the propagation delay to the feeding routers and the variation in traffic going to the output queue. Further determination of buffer requirements, control heuristics and performance trade-offs for this scheme are part of on-going research. In this vein, we are also exploring providing "feed forward" load information on packets transiting rate-controlled links. That is, packets include information on the number of packets queued behind them at their previous router.

We expect certain critical links in an internetwork to experience significant load, requiring rate control while many links operate with no queuing thus realising the low-delay benefits of cut-through. The critical links can be reasonably augmented with additional capacity to avoid overload. Dynamically load balancing across multiple links that provide this increased capacity is facilitated by the notion of logical hops or links, as described below.

Logical Hops and Load Balancing A network can use a port identifier to designate a group of links that are all equivalent from the standpoint of the Sirpent source. For example, a (logical) port in a San Francisco area router may designate a (logical) link to a Boston router which may in fact be implemented by many different physical links. A packet routed through this logical port can be routed over any one of the physical links by the router based on local load and availability. A port may also designate multiple hops across multiple networks to some common destination, further allowing the source to be oblivious to the intervening routing. This approach allows the network (and even portions of the internetwork) to do fine-grain load-balancing and rerouting around congestion and failures.

The hierarchical structure of recent internetworking infrastructure makes this type of approach attractive in countering the problem of slow route update using source routing. That is, a wide-area transit network could provide single logical hops for the various destinations it serves and the clients simply use those logical hops. Internally, the network can be handling the routing to balance the load. For example, a router connected to a Blaeenet might replace the logical hop destination by a Blazenet source route as the packet enters the Blaaenet network and remove it on exit, at the cost of the packet delay of adding this routing information (which need not cost more than the siae in bits of the route divided by the data rate). Also, a very high speed physical link, such as a 10 gigabit line, might be statically divided into 10 1 gigabit channels with all 10 links being treated as one logical link. A packet arriving for this logical link would be routed to whichever of the channels was free. This approach offers a means of exploiting high capacity physical links without forcing the higher speeds on the rest of the internetwork.

In general, one can regard logical ports and logical links as an optimization for the cases in which it is more efficient for some portion of the routing to be performed by the internetwork. The benefits are primarily: (1) later binding of routes to avoid congestion and failures and (2) shorter packet headers. The IP approach represents an extreme of this "optimiration" in which all routing between the source and destination hosts is performed by the internetwork. By the "end-to-end argument", this optimization, which increases the cost of the (inter)network layer, is only justified if it offers a net performance benefit. We conjecture that this optimisation is only justified for limited portions of some routes, such as replicated trunk links, and otherwise incurs cost and performance degradation in the internetwork service. That is, the IP approach can be viewed as an extreme in false optimisation of the Sirpent approach'.

The intra-host addressing of UDP [19] and TCP can be regarded as a degenerate form of source addressing. That is, the UDP port number is interpreted relative to the IP host address so a UDP packet is source-routed to the host and then to the socket within the host. With Sirpent, intra-host addressing is provided by the same mechanism as used for inter-host addressing. That is, a Sirpent header segment can be used to designate the port within a host to which to address the packet. In this light, it is interesting to note that IP/UDP addressing can have some of the same potential problems as source routing, namely having a fixed binding to a route that has failed or is congested. In the IP/UDP case, the "route" is bound to a particular host interface and port (which is typically bound to a particular instantiation of an application). Thus, the host interface can fail and cause the communication to fail even though the host may still be reachable through a separate host interface. The remedy to this problem involves either revising IP to represent intra-host transport endpoints (a direction

[8] We attribute the terms and concepts of *soft state* and *flows* to Dave Clark of MIT. Soft state refers to state that is easily recoverable after router crash, thereby not conflicting with the "stateless" philosophy for IP routers. The term *flow* refers to a sequence of related packets, such as those constituting a file transfer, which the router detects based purely on their dynamic behavior.

[9] Without the source routing option, IP would also fail to provide an important functionality offered by source routing, namely the ability for the source to pick the route.

being taken by the OSI standards) or to provide route rebinding mechanisms similar to that required by **Sirpent.** We argue that the latter is the simpler and more efficient approach. Given that mechanism, **Sirpent** unifies inter-host and intra-host addressing rather than treating them differently, as done with IP, and unifies the rebinding mechanism as well.

In summary, logical hops and ports allow **Sirpent** to mix both source routing and conventional **(inter)network-**controlled routing as appropriate. This combination is further illustrated in the next section which shows how IP can be used as part of **Sirpent.**

2.3 Scalability

Sirpent has a number of attractive properties for scalability. First, with variable-length source routes, there is no limit to the number of nodes than can be addressed. Even limiting the size of source routes to reasonable values allows a very large number of nodes. For example, using VIPER (of Section 5 and a maximum of 48 header segments (expected to be under 500 bytes long), one can address up to 2^{56} endpoints, far exceedingly the total required for the future global internetwork. Moreover, there is no need to coordinate the assignment of addresses; the addresses are purely a result of the internetwork topology and port assignments within each switch, which can be arbitrary.

Second, the size of state required by each **Sirpent** router is proportional to the properties of its direct connections and not the entire internetwork, unlike standard IP routing algorithms such as link state routing which store the entire internetwork topology. In particular, a **Sirpent** router needs memory for buffering, accounting and congestion control that is related to the delay-bandwidth of its links. Thus, the cost of a **Sirpent** router need not increase as the internetwork scales to a larger **size.**

Finally, **Sirpent** accommodates existing **internetwork-**ing approaches both for compatibility and to exploit these techniques as optimizations as appropriate. In particular, the **Sirpent** approach can be viewed and implemented as an extended form of IP as follows. An IP protocol number is assigned to the **Sirpent** protocol. A **Sirpent** packet can view the Internet as providing one logical hop across its internetwork, as described above. That is, the packet is source routed to an IP host or gateway so that the header is now an IP header. The host/gateway uses standard IP to route the packet to the specified destination host. At this point, the packet is **demultiplexed** to the **Sirpent** protocol module which interprets the remainder of the **packet** header as a source route on from that point. The **Sirpent** protocol module can also be invoked directly from the raw network layer. An analogous approach cab be used to exploit existing **X.25/X.75 (inter)networks,** except for the additional problem of managing the virtual circuits. In this sense, all existing networks (and internetworks) can be incorporated into the **Sirpent** approach by adding- a **Sirpent** module to each routing node and allocating a type identifier for **Sir-**pent for each network technology.

These existing internetworking protocols and techniques can also be exploited to **minimise** the **Sirpent** header size, using the approach of logical ports and links, as described previously. For example, a complex multi-hop transit route can be replaced with a single logical port designation as an optimization of header size and load balancing. In this way, the two major issues for scaling with **Sirpent,** size of source route and provision for dynamic **(re)routing,** can be addressed using conventional network routing techniques.

A key aspect of **Sirpent** is placing function in the transport layer and higher which has been traditionally placed in the (inter)network layer. The next two sections address the role of internetwork directory services and the transport layer in handling these functions.

3 Internetwork Directory Support for Source Routing

The global internetwork directory service is extended in **Sirpent** to provide routes to a host or service, given its character-string name. In this vein, the routes to a service can be regarded as just one of many attributes of the service that the directory can maintain. Thus, as internetwork directory services move from highly **specialized** and restricted name servers, such as the Internet Domain Name Service, to general distributed data base management systems, a query about a service can return routes to the service as well as other attributes of the service.

The routing information is relative to the requesting client both in the actual route as well as in the authorizing tokens. One plausible scheme for acquiring and maintaining the routing information (easily extended for authorization and accounting) is described by Singh [23]. The scheme assumes that the internetwork is structured as a hierarchy of ***regions*** with a routing directory server for each region, analogous to the Internet Domain Name service. (In fact, similar considerations govern the division of the internetwork into administrative regions for both routing and naming purposes. For example, **stanf**ord.edu represents both a naming and routing domain from an administrative standpoint. Subdomains, such as **cs.stanf**ord.edu can have similar properties as a subnetwork of the Stanford network.) Each server in responsible for maintaining the routing information for immediately higher layer servers and lower level servers within the same region. With **Sir-**pent, the hierarchical character-string names serve as the unique hierarchical identifiers for hosts, gateways and networks, required by Singh's scheme. Routing information is updated by reports from routers, hosts and networking monitors. The directory servers, as users of the **internet-**work themselves, can also observe load and failures as part of their normal operation.

This approach has several advantages. First, the clients of the directory/routing service can exercise more control over the routes taken by their packets than with conventional routing. A client can request and receive multiple routes to a service. It can also request a route with particular properties, such as low delay, high bandwidth, low cost and security. For example, transactional application would prefer a low delay route over one with higher bandwidth and higher delay. A client can also use an independent routing service or formulate its own routes to meet special application requirements and considerations. In particular, **policy-**based routing can be handled within this framework along the same lines as proposed by Clark [6]. These extensions in function and flexibility are feasible because routing decisions in **Sirpent** are done at a higher layer than conventional networks so the extensions do not impact the performance- and reliability-critical switching nodes. The use of caching, on-use detection of stale data and hierarchical structure for the routing information, as has been proposed for directory systems [3], reduces the expected response time for routing queries and the expected load on directory servers".

As a second advantage, the client can have more information about the route it is using for packets. For example, the directory service can return information on the bandwidth, propagation delay, maximum transmission unit, etc. for each portion of the route it returns. With this information, a client can determine (up to variations in queuing delay) the roundtrip time and MTU for packets on this route, rather than discovering these parameters over time. This property is particularly important for transactional communication where the single request-response behavior does not provide enough data to discover this information before the communication activity has completed. The conventional approaches of discovering this information over multiple roundtrips takes more time, uses the communication resources less efficiently during this time and can be invalidated at **any** time by the internetwork deciding to reroute packets.

Finally, using the directory services for routing eliminates the considerable duplication between directory services and routing services- (incorporated in all routers in conventional schemes). That is, there is no need for **IP-like** addresses and thus no need for the mechanism and protocols for mapping these addresses to routes and maintaining all this information in each router. Merging the routing and directory services facilitates supporting **authorisation** and accounting as part of routing, which is required for effective resource management. The directory systems must deal with authorization and access control in any case. The authorization and accounting information represents a data base. Thus, there is considerable potential for sharing of mechanism and protocols between authentication, naming and routing.

4 Transport Layer Implications

A transport layer protocol using **Sirpent** must **implement** some **functions** normally provided by the network and internetwork layers, including **recognising** misdelivery, enforcing maximum packet lifetimes and handling very large packets. We argue that these functions are more efficiently implemented at the transport layer and, by the end-t-end argument, must necessarily be implemented by the transport layer in any case.

4.1 Handling Packet Misdelivery

A transport protocol using **Sirpent** must provide a unique transport layer address independent of the network layer because a packet may be **misdelivered** if the **Sirpent** packet header is corrupted. (Because **Sirpent** does not use a checksum, it can detect this corruption [11].) As an example of such a protocol, VMTP [2] provides a **64-bit** transport layer identifier which is unique independent of the **(in**ter)network layer addressing.

[10]**Acquiring** a route req**uires** a full round trip to the region server for the destination. **Thus,** without caching, the time to acquire the route **incurs** a **similar** round trip delay to that incurred by circuit **setup** in a circuit-switched network.

[11] One could provide a checksum of the original **Sirpent** header and then reconstruct **this** checksum from the trailer but a host would not in general have the knowledge to reconstruct the original header from the trailer.

This type of transport layer addressing has several additional advantages. For instance, the network-independent addressing in VMTP is used to support process migration, multi-homed hosts and mobile hosts. It also facilitates the use of different network layers with the same transport module and makes the transport layer more independent of the network addressing and functionality. The major cost, the larger **sise** of transport identifiers **(64-bits** in VMTP versus 16 bits in TCP), is not significant with the higher network data rates. Thus, we conclude that placing greater requirements on the transport level to handle misdelivery independent of the (inter)network layer is justified.

In contrast, conventional transport protocols rely on (inter)network layer information to detect misdelivery. For example, TCP requires that the IP header be correct because it relies on the IP address as well as the TCP port number to form a complete unique transport address. (In reality, it includes a 'pseudo-header' of IP-layer information in its checksum to ensure the **IP** information can be trusted by the transport layer.) This approach precludes moving TCP connections between hosts or even different host interfaces on the same host unless the IP address is also reassigned.

4.2 Enforcing Maximum Packet Lifetime

Transport protocols require bounds on maximum packet lifetime (MPL) b**ecause** they use fixed-size fields for packet identification". The conventional approach to limit MPL is to include a time-to-live (TTL), as employed in IP, or a **hopcount** in the (inter)network header which is **decremented** by each router. The packet is discarded when the value reaches sero. However, correct implementation of this facility requires that the TTL is updated by every router on the packet's route. This observation exposes a significant flaw in this approach: The transport layer is dependent on the correctness of the network layer for its correct operation, violating the basic objective of the transport layer (at least with TCP and **ISO** TP Class 4) of providing reliable process-to-process communication in spite of unreliability at the network layer.

Appealing to the end-to-end argument [21], we require that the transport layer include a creation timestamp in every transport protocol packet and require that the sender and receiver have roughly **synchronized** clocks. Undetected failure of docks is viewed as equivalent to other host failures that could cause incorrect protocol behavior. The receiver then discards packets that are older than an acceptable period based on its recent history of communication. For example, a host with a low reception rate that has not crashed recently can accept relatively old packets without risk whereas a recently booted machine might discard packets older than its boot time.

The inclusion of a timestamp at the transport layer is illustrated by (recently revised) VMTP, which includes a 32-bit timestamp in the trailer of the packet (along with the checksum). The 32-bit timestamp represents the time in **milliseconds** since January 1, 1970, **modulo** 2^{32}. The

[12]**That is,** after **some** number of packet **transmissions,** a field value **must** be reused. If a copy of a previous packet **using** the **same identification is still in existence** at the time of this reuse, it **may** reappear at the receiver, causing confusion with the new packet **assigned** that identification, and result in incorrect behavior if the receiver (unknowingly) accepts the old **data.**

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-t+-tttt+-t+-t+-tt+
|Port InfoLength|PortTokenLength |
+-+-+-t+-tttt+-t+-t+-tt+
|    Port        I Flags |Priorit I
+-+-+-t+-tttt+-t-+-+-t+-+
>            PortToken            <
+-+-+-t+-tttt+-t-+-+-t+-+
>            PortInfo             <
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: VIPER Header Segment

The flags include:

VNT VIPER Next Type - the **portInfo** field is void and another VIPER header segment immediately follows this one. The **portInfoLength** field may still be non-zero if the PortInfo field is used for padding.

DIB Drop If Blocked - drop the packet if it is blocked at this router.

RPF Reverse Path Forwarding - the packet is being returned using the route and tokens supplied in a packet received by the currently sending host.

The Priority field indicates the priority of switching and forwarding. Normally, a router only considers this field when a packet is blocked. However, a sophisticated router is free to use it to schedule a packet in competition with other communication activities as part of every packet routing decision if the processing provides lower delay to higher priority packets. Normal priority is 0 with 7 highest priority. Priorities 6 and 7 preempt the transmission of lower priority packets in mid-transmission if necessary. Values with the high-order bit set represent lower priorities, **0xF** being the lowest priority. Control and charging over the use of priorities is exercised by the token mechanism if required. For example, use of high priorities may be limited by simply charging more for higher priority packets and limiting the number of tokens for high priority traffic through a router at any given time.

The PortTokenLength field specifies the length of the **PortToken** field in octets. A value of 255 is reserved to indicate that the actual length is larger than 254 octets. In this case, the length is contained in the 32-bits starting in the standard **PortToken** field. A PortTokenLength field of 0 indicates that the **PortToken** is not present.

The **PortToken** field contains a token value that indicates authorization and accounting information, if present. The token values are provided by the routing directory servers at the time that the source determines the route. These tokens are opaque capabilities to all but the router and the administration domain that manages the router. In general, these capabilities are structured, obtained and managed in the same fashion as proposed by Clark [6] for policy-based routing.

The **PortInfoLength** field specifies the length of the PortInfo field in octets. A value of 255 for the PortInfo field is reserved to indicate that the actual length is larger than 254 octets and contained in the 32-bits starting in the standard PortInfo field. The **PortInfo** field is **network/protocol**-specific and potentially contains the type format for the rest of the packet as well as possibly other information, as the earlier examples of using the Ethernet illustrated. For example, the length would be 14 for an Ethernet header, including 12 octets for the 2 48-bit Ethernet addresses in the standard Ethernet header and 2 octets for the Ethernet type field.

The size of the VIPER header segment is minimized by the use of small fields, the smallest segment size being 32 bits. It is important to minimize the header segment size because its multiplicative effect using source routing: with N hops, there are N VIPER header segments on the front of a packet. Minimizing the total header size is important to reduce the need to fragment transport data segments and to minimize header overhead for small amounts of data.

The fields are ordered to minimize the difficulty of handling the packet header segment in cut-through switching hardware. In particular, the fixed-length portion is first and provides the length information on the variable-length portion as far in advance as possible of the variable-length portion arriving, allowing for hardware setup times.

The VIPER transmission unit is 1500 bytes. The large size is justified by the de **facto** standard created by Ethernet, the larger expected size from FDDI and other new networks, and the higher data rates of future networks, making large maximum packet sizes feasible without increasing the maximum channel occupancy time per packet. The 1500 byte size allows for roughly 1 kilobyte transport packet plus up to 500 bytes of VIPER header information. This size represents **a** comparable convention to the 576 byte unit used by the **DoD** Internet because VIPER does not provide fragmentation and reassembly.

A VIPER router follows the **Sirpent** algorithm of strip ping the current header segment **from** the front of the packet, checking the port token for proper accounting and authorization (if present), and appending the return port, token and network-specific information onto the end of the packet.

6 Performance Evaluation

Key measures of **Sirpent** performance include the **per**-packet switching delay, header overhead and handling of congestion and link failures. All three factors affect the response time for clients as well as the network utilization.

6.1 Switching Delay

The switching delay with a cut-through **Sirpent** switch is the switch decision and setup time plus the queuing time. Cut-through switching eliminates the reception and storage time for the packet, which is proportional to the size of the packet. The switching decision and setup time can reasonably be significantly less than a microsecond and handled entirely by a hardware-optimized path in the common case. Thus, with links of low utilization, the switching delay is a fraction of a packet time. With reasonable load (up to about 70 percent utilization), M/D/l modeling of the queue suggests an average queue length of ap proximately one packet or less, including the packet currently being transmitted. The average queuing delay is then approximately the transmission time for half of an use of a 32-bit timestamp with 1 millisecond granularity means that wrap-around occurs in roughly one month, which should protect against all but maliciously delayed packets. When a VMTP packet is received, the packet is discarded if its timestamp indicates it is too old by the above considerations. A timestamp value of 0 is reserved to mean that the timestamp is invalid and should be ignored. This value is for use by query operations when a machine is booting before it knows the current time accurately.

The millisecond accuracy is motivated by the desire to provide a basis for recreating relative time frame information of arriving packet information in real-time traffic. For example, packets representing a video stream may experience different delays in transit; the timestamps allow the the receiver to recreate the appropriate timing as well as discard very old packets. Because packet lifetimes are generally limited to multiple seconds if not minutes, inter-host clock synchronization need not be more accurate than multiple seconds. In fact, as an optimization, the timestamp comparison could be restricted to the high-order bits so that a simple equality check of the creation time with the current time would succeed most of the time, only resorting to a more complex modulo difference check (that provides for wraparound) when the simple test fails.

The inclusion of the timestamp has several advantages. First, the correct operation of the transport protocol is made independent of the **(inter)network** layer delaying of packets for long periods of time. There is also no need for a pseudo-header at transport layer that includes the **(in**-ter)network layer information. That is, if IP provided such a creation timestamp, it would have to be included in a transport layer pseudo-checksum for strict correctness, the same as done for IP addresses with TCP.

Second, unlike the **TTL** field in the IP packets, the creation timestamp requires no update in intermediate routers, thereby eliminating the associated processing load on the routers. This approach matches with the **Sirpent** philosophy, namely using slightly more bandwidth (in the form of extra bandwidth to send around the creation timestamp) to reduce the processing load at the routers.

Third, it relieves the sender of the job of choosing a TTL, a decision which fundamentally belongs to the receiver. Restrictive choices of TTL in some software have already caused problems in the context of the rapid growth of the Internet. (Note that the scope use of TLL in **multi**cast [9] does belong to the sender and is only an **optimiza**tion for switches and network, not an issue of transport protocol correctness.)

The timestamp information can also be used by the receiving host to aid in estimating the round-trip time, to detect congestion when significant increase in delay takes place, to aid in maintaining synchronized time between hosts, and to recreate the time frame for real-time traffic, as described above.

This approach requires approximately **synchronised** clocks among the communicating hosts (unless suppression of old packets is not necessary). This requirement is feasible for several reasons. First, there are reliable clock synchronization protocols available [8, 14]. Second, clock **syn**chronisation is useful, if not required, for a variety of other reasons in communicating hosts. For example, **file** transfer should avoid creating (or apparently creating) a file before its creation time at the sender. Thirdly, external sources of time synchronization, such as the WWV radio source, are available to provide extra redundancy and thus reliability for this facility. In particular, file servers and other major machines can have access to such a service by radio receivers (if battery-backed up clocks are insufficient). The availability of accurate and standardiaed radio broadcast sources around the world allows clock synchronization among computer systems to be scaled globally. The coarseness of clock **synchronization** that is adequate for reasonable transport protocols, including VMTP, makes these sources of time sufficiently accurate for this use.

4.3 Handling Large Logical Packets

The **Sirpent** protocol provides no support for fragmentation and reassembly, unlike IP. Handling the logical packet fragmentation and reassembly at the transport layer simplifies the internetwork layer and makes the communication system more resilient to error and packet loss. The transport protocol can provide selective transmission and flow control on the logical packet fragments, avoiding the all-or-nothing behavior of IP in the reassembly of packets and the systematic failures that can arise because of overrun. For example, with VMTP, rate-based flow control is used between packets within a packet group to avoid overruns, and selective retransmission is employed when a packet is lost within a packet group. Other recent transport protocols, including XTP [5] and Rx [22], also provide selective retransmission with comparable benefits.

The **Sirpent** approach requires a minimum transfer unit guarantee that is large enough to contain the minimum transport packet plus the header segments. Any network that does not provide this minimum on its links must provide an encapsulation layer that transparently delivers this minimum **size,** as was done with PUP [1].

Traditionally, the **(inter)network** packet is the unit of host transmission, so it appears that **Sirpent** may impose significant host overhead in sending smaller packets than would be feasible with IP. However, the transport layer can provide a unit of transmission that decouples the host unit of transmission from that of the network packet sise. An example of such a unit is the packet group in VMTP. Using a network adaptor like the NAB [17], the host can initiate the transfer of a packet group and let the NAB handle the per-packet transmission, including the per-packet **Sirpent** overhead [13].

5 VIPER

We propose a specific protocol to be used as a realization of the **Sirpent** approach, called the Versatile **Internet**work Protocol for Extended Routing (VIPER).

A VIPER header is shown in Figure 1. The Port field **specifies** the output port to be used by the current switch or router. Reserving 0 as a special port value meaning 'local', the effective number of ports per switch is limited to 255. We require that larger fan-out switches be structured hierarchically as a series of switches, each with a fan-out of at most 255. The hierarchically structuring has a number of advantages in the development of a switching fabric and imposes no significant additional delay given the use of **cut**-through routing at each stage.

"Actually, a NAB can **easily** handle multiple packet groups if **minimising host** processing **overhead** is critical, but this **optimization seems** unwarranted in general.

average packet size. Thus, we expect the savings in delay for cut-through to be significant for all but significantly loaded links. Moreover, previous work on Blaaenet [12] shows that circuit-switched networks cannot run links at comparable utilization with the bursty traffic characteristic of computer communication. Thus, we argue that, when queuing delay in a Sirpent-style internetwork becomes significant, other schemes would be severely congested or deny service because of their slower switching (in the case of IP approaches) and more static allocation of resources, as in the CVC approach.

The size of the Sirpent-style header, which depends on the length of the route and other factors, also contributes to switching delay. However, we argue below that the expected size of the header is relatively small, possibly smaller than with IP.

0.2 Header Overhead

The header overhead is dependent on the average packet size, the number of hops a packet **travels**[14] and the size of each header segment.

Previous network measurements [4] suggest (as a rough approximation) that half the packets are close to minimum size (for the transport layer), one quarter are maximum size and the rest are more or less uniformly distributed between these two extremes. Using this approximation in general, the average packet **size** is roughly **3/8** of the **maximum** packet size.

The topologies of **internetworks** are evolving to **minimise** hop count and improve **managability** (although there are some counterexamples). Looking at a relatively mature communication system such as the world-wide telephone system, one sees hop counts of 5 or **6** for global **communication**. In addition, an increasing amount of computer communication is local, within a cluster of local networks or campus. Moreover, clusters with significant inter-cluster communication are likely to be supported by low **hopcount** routes. Thus, we argue that locality of communication causes the expected number of hops per packet for many applications significantly less than one.

Finally, several trends in networking suggest that network addressing may evolve away from the large addresses used in the Ethernet. First, some multi-access networks, such as the Token Bus [15], use dynamically assigned addresses, so the address **size** can be approximately that required to handle the number of possible nodes on one network, rather than the total number of this type of network interface ever manufactured. That is, the address could be 16 bits or fewer, rather than 48 bits. Second, work on very high-speed networks has been favoring point-to-point networking rather than multi-access techniques. The address is then implicit in the route taken, rather than having to be specified in the packet, further reducing the address **size.** Finally, the **Blazenet** design and its general rationale suggest that networks themselves may use source routing. In this case, we argue that again the expected length of the source route is small because of locality to communication.

Combining these observations, the expected overhead for **Sirpent** addressing as a percentage of packet size is small. As an estimate, assume that the maximum packet size is 2 kilobytes (so that average packet size is about 633 bytes). Assume that the average header size is 18 bytes per hop (which is a VIPER header plus Ethernet header) and the average number of hops is **.2** (counting 0 hops as local to the same network). Then the average VIPER header overhead is 0.5 percent. This figure is indicative of our rationale for the design and suggests that header overhead should not be a problem. Further measurements and experience are required to provide a better indication of the actual overhead.

[14] We use the convention of counting hops **as** the number of routers traversed, not the number of networks traversed.

The variable-sized header required by source routing is a potential cause of complication and cost. However, it is easy to support on transmission from the source with a simple multi-segment DMA facility, as provided by many network adaptor chips and boards. On reception, the header has been reduced to that comparable to current network headers, if not simpler. The trailer then contains the source route. Of course, "intelligent" network adaptor boards, such as the NAB [17], can be extended to support this source routing scheme. In particular, the trailer can be removed by the NAB on reception to avoid transferring the trailer to main memory and "polluting" the user data area with the trailer. A router can handle a **Sirpent** packet in software by, after fully receiving the packet, copying the first header segment to the end of the trailer (with suitable modification) and then transmitting the packet starting at the following header segment.

8.3 Response to Congestion and Link Failure

Another performance concern with source routing is the time required for a client to reroute packets in response to congestion or link failure. A related concern is the lack of network layer control over load and load distribution. **Sirpent** addresses these issues as follows.

Clients can request multiple routes (rather than a **single** route) to the desired host or service, and switch between these routes based on the performance of the different routes. Because the client knows the base round trip time for the route, measures the actual round trip time as part of reliable communication, and receives feedback **from** the rate-based congestion control mechanism in the presence of real congestion, it is able to quickly detect and react to congestion and link failures. In fact, we argue that the client can react **faster** and more reliably to optimise its **end-**to-end performance than can the hop-by-hop optimization of conventional distributed routing, except in one expected case, namely the replicated transit link. This case is handled by making the replicated transit link appear as one ***logical link*** to the source routing mechanism (as described in Section 2.2) and allowing the router to select between the physical transit links based on local load conditions.

The rate-controlled congestion control precludes excessive load on portions of the internetwork independent of the routes chosen by the sources. The back pressure exerted by the congestion control mechanism causes sources to switch to other routes, as described above.

The routing directory servers maintain reasonably **up-**to-date load information on links using report received from network monitoring stations, individual routers and sources experiencing problems with routes they are using. The **in**ternetwork topology **is** slowly changing and easily tracked by the routing servers. The clients benefit from these **rout**ing updates by periodically requesting route advisories from the routing servers. The problem of load distribution in **re-**

sponse to (the far more dynamic) changes in load is handled by the use of logical hops.

Using these techniques, we conjecture that **Sirpent** can provide better performance than competing and established internetwork architectures.

7 Related Work

Source routing is an established concept [10, 24]. Saltzer [20] advocated the use of source routing in campus networks for some of the reasons to cited here, but without considering congestion control or accounting. Although their focus was also on simplifying the routers (and although they identify a number of other key advantages we do), they do not address the requirements of cut-through or construction of the return route in the trailer. As part of this earlier effort, Singh [23] developed the specification for a source routing server. This specification identifies the need for hierarchical naming of internetwork hosts but it does not explore the idea of combining this level of identification with character-string host identification, as provided by directory servers.

Blazenet [11, 12, 13] is a network design that exploits source routing with some of the same motivation as **Sirpent**. The **Blazenet** design provides a gigabit network that that may require **Sirpent** techniques for adequate internetworking performance.

The token-based authorisation and accounting scheme builds on Clark's approach to policy-based routing [6]. Clark's proposal also uses source routing. However, **Sirpent** provides an approach that subsumes the Internet with focus on very high-performance routing, rather than focusing on policy enforcement within the current Internet. In fact, **Sirpent** reverses the design base of IP. **Sirpent** makes source routing the basis (rather than an option like in IP) and reduces (inter)network distributed routing (in the form of logical links and ports) to an optimization, rather than the basis of the design, as in IP.

8 Conclusions

Sirpent provides a high-performance approach to internetworking with significant advantages for performance, cost, resource management and scalability. We conjecture that its performance is adequate for a wide-range of traffic from real-time video to file transfer, assuming adequate bandwidth. In particular, there is no need to resort to circuit switching and resource reservation techniques to provide for this range of traffic. **Sirpent** also suggests a means to move "intelligence" out of the (inter)network, freeing the internetwork to be fast while also providing hosts and routing servers the option of providing more intelligence, user control and flexibility. In particular, policy routing issues, whether for security, reliability or accounting reasons, can be made by the source host and routing server with no complication of the internetwork routers beyond the token authentication and accounting mechanism. This migration of function up the layers seems key to higher performance, reliability, security and functionality.

Several ideas are central to the **Sirpent** approach. First, source routing is taken as fundamental, encompassing both inter-machine and intra-machine routing, with conventional network routing handled as a selective optimization, referred to as logical links. This approach is supported by the end-to-end argument given that conventional routing incurs a cost, provides benefit only in certain cases, and does not provide the endpoints adequate control of packet routes (and therefore, of the delivery service it receives). More generally, **Sirpent** seeks to move all but the essential functions out of the internetwork layer. In particular, packet lifetime limits, unique endpoint identification and fragmentation/reassembly must be handled at the transport layer with **Sirpent**. Similarly, routing is performed by the sources or routing directory services, moving this function out of the performance-critical portions of the internetwork.

Second, per-hop and per-packet accounting tokens and priority are proposed as both necessary and sufficient for internetwork resource management, without losing the benefits of packet switching for the bursty, transactional traffic that is characteristic of computer traffic.

Finally, rate-based congestion control exploiting the source routing information in packets can control queuing delay and packet loss, given adequate output queue buffer space.

The **Sirpent** approach also incorporates a number of novel detailed ideas building on the well-known concept of source routing. First, the return route or a basis for the return route is dynamically constructed as a trailer to the packet, facilitating cut-through routing and reducing switching delay. Second, logical hops are introduced to provide a trade-off between local (to router) and remote (at routing service) route binding. Third, **Sirpent** explicitly provides for compatibility with existing (inter)network approaches, including IP, by allowing (inter)network-specific fields in the packet header segments. Fourth, the routing service is provided as an extension of internetwork (name) directory services, eliminating the extra identifier space and mechanism required for current internetwork datagram approaches such as IP. Finally, the optimistic token-based authorisation using caching provides control of resource usage without performance penalty.

Several issues remain open and are topics of our ongoing research on **Sirpent**. First, there is a need to characterise the behavior of the rate-based congestion control, determining the relationship between traffic stability, link characteristics, buffer space and link utilization. Second, a demonstration implementation of VIPER together with a routing directory service is required to gain a better understanding of route selection, monitoring and reselection dynamics. Third, the details of the port token structure, management and accounting mechanism need to be developed. Finally, we are interested in experimenting with real-time traffic on **Sirpent** internetworks in which "jitter" is handled by selectively delaying data delivery to recreate the original packet transmission spacing, possibly using the VMTP timestamp for this purpose.

Overall, **Sirpent** appears to be a promising approach to building the global internetwork required for the next generation of computer communications and distributed systems. We plan to experiment with the VIPER protocol and its implementation to further evaluate this approach and the techniques we have proposed.

9 Acknowledgements

The paper has benefited considerably from the comments of Dave Clark, Steve Deering, Deborah Estrin, Ross Finlayson, Hendrik Goosen, Zygmunt Haas and Chuck

Thacker as well as the referees. The author has also benefited from discussions with members of the Internet End-to-end Task Force chaired by Bob Braden. This work was sponsored in part by the Defense Advanced Research Projects Agency under contract N00039-86-K-0431, by Digital Equipment Corporation, by the National Science Foundation Grant DCR-83-52048 and by ATT Information Systems.

References

[1] D.R. Boggs, J.F. Shoch, E.A. Taft, and R.M. Metcdfe. Pup: An internetwork architecture. *IEEE Transactions* on *Communicationr*, COM-28(4):612–624, April 1980.

[2] D.R. Cheriton. Versatile message transaction protocol (VMTP). RFC 1045, SRI Network Information Center, February 1988.

[3] D.R. Cheriton and T.P. Mann. Decentralizing *a* global naming service for efficient fault-tolerant access. *A* CM *Trans. on Computer Systems*, 7(2), May 1989. An earlier version is available as technical report STAN-CS-861098 Computer Science Department, Stanford University, April, 1986 and CSL-TR-86298.

[4] D.R. Cheriton and C. Williamson. Networkmeasurement of the VMTP request-response protocd in the V distributed system. *In SIGMETRICS '87.* ACM, 1987. Banff, Canada.

[5] G. Chesson. XTP/PE overview. *In 13th Conference on Local Computer Networka,* pages 292-296. IEEE Computer Society, October 10-12 1988. Minneapolis, Minnesota.

[6] D.D. Clark. P oli: youting in Internet protocols. Technical Report 1102, Internet RFC, May 1989.

[7] D.D. Clark, M. Lambert, and L. Zhang. Netblt: A bulk data transfer protocol. RFC 969, SRI Network Information Center, 1985.

[8] F. Crirtian. Probabilistic dock synchronization Technical Report RJ 6432, IBM Almaden Research Center, September 1988.

[9] S.E. Deering. Multicast routing in internetworks and extended LANs. In *Proc. SIGCOMM'88.* ACM, August 1988. Also available as Stanford CS tech. report STAN-CS-88-1214.

[10] D.J. Farber and J.J. Vittal. Extendability considerations in the design of distributed computer system (DCS). In *Proc. Nat. Tefecomm. Conf.*, pages 15E–1 to 15E–6, November 1973. Atlanta, Georgia.

[11] Z. Haas. *Packet-#witching* in *Future High-performance* Wde-area *Networkr.* PhD thesis, Electrical Engineering Dept. Stanford University, May 1988.

[12] Z. Haas and D. R. Cheriton. A case *for* pack&switching in high-performance wide-area networks. *In SIGCOMM '87 Workrhop,* Aug 11-13 1987. Store VT.

[13] Z. Haas and D.R. Cheriton. Blasenet: A packet-switched wide-area network with photonic data path. to appear, IEEE Transactions on Communications, An earlier version appeared as Stanford Computer Science Department technical report, Blasenet: A High Performance Photonically Implementable Wide-area Network, 1988.

[14] J. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant *clock* synchronisation. *In Proc. 3rd Annual* ACM *Symposium on Principles of Distributed* Computing, pages 89–102. ACM SIGACT/SIGOPS, 1984. Vancouver, BC Canada.

[15] IEEE. *802.4 Tok en-Parr Bur Access Method.* IEEE, New York, 1985.

[16] V. Jacobson. Congestion avoidance and control In *SIG*-COMM '88. ACM SIGCOMM, August 1988.

[17] H. Kanakia and D.R. Cheriton. The VMP network adapter board NAB: High-performance network communication for multiprocessors. *In SIGCOMM* 88. ACM, August 1988.

[18] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[19] J. Postel. User datagram protocol. RFC 768, SRI Network Information Center, September 1980.

[20] J. Saltzer. Source routing for campus-wide intemet transport. In *Proceedings* of *the* IFIP *Working Group* 6.4 *Workrhop on Local Area Networkr.* IFIP, August 1980. Also Internet IEN 144.

[21] J.H. Saltzer, D.P. Reed., and D.D. Clark. End-to-end arguments *in* system design *A* CM *Trans. on Computer Systems*, 2(4):277–288, November 1984.

[22] B. Sidebotham. Rx protocol specification. draft from ITC CMU, March 1989.

[23] V. Singh. The design of a routing service for campus-wide internet transport. Technical Report MIT/LCS/TR-270, MIT, 1981.

[24] C.A. Sunshine. Source routing in computer networks. *Computer Communications Review*, 1(7):29–33, January 1977.