A Software Architecture for Network Communication [1]

David P. Anderson

Computer Science Division
EECS Department
University of California, Berkeley

November 30, 1987

ABSTRACT

A distributed system is based on a layered set of abstractions of network communication. At a low level, current distributed systems use primitives such as datagrams, request/reply message-passing, reliable virtual circuits, or a combination of these. Future distributed systems will use high-performance large-scale communication networks, and will support a range of communication-intensive applications. What are the appropriate communication abstractions for such systems?

In answer to this question, we have developed a communication abstraction called *Real-Time Message Streams* (RMS). An RMS is a simplex (unidirectional) stream with several performance and security parameters. These parameters express 1) the needs of RMS clients (user programs and communication protocols), and 2) the capabilities of the RMS provider (network and higher layers). This information can be used in two ways. First, RMS providers can eliminate unnecessary or redundant work, and can optimally schedule resources such as network bandwidth and CPU. Second, the RMS client can use the parameters to select optimal methods for achieving whatever reliability and flow control are needed.

RMS is the communication primitive of the DASH distributed system currently being developed at UC Berkeley. This paper describes 1) the RMS abstraction itself, 2) the role of RMS in the DASH communication architecture, and 3) techniques and algorithms for providing RMS at various system levels.

[1]Sponsored by MICRO, IBM, Olivetti, MICOM-Interlan, Defense Advanced Research Projects Agency (DoD) Arpa Order No. 4871. Monitored by Naval Electronic Systems Command under Contract No. N00039-84-C-0089.

1. INTRODUCTION

Progress in low-level software mechanisms for distributed systems (virtual memory, process control, kernel structure, and communication software architecture) has not kept pace with that at higher levels (distributed data, distributed computation, transactions, and so forth). The focus of the DASH project at UC Berkeley [1] is on the development of low-level mechanisms for next-generation distributed computer systems.

The DASH distributed system is intended to run on multiple types of computer architectures and communication networks. To accommodate multiple network types, a large part of the DASH network communication system is *network independent*. The lower-level *network dependent* part has a network-independent interface (see Figure 1).

In existing distributed systems, the corresponding interface has typically provided a simple abstraction such as unreliable, insecure datagrams. Higher software layers then use this facility to provide higher-level abstractions such as reliable request/reply message-passing [5], reliable secure typed message streams [12], or reliable byte streams [9]. This approach simplifies the task of porting the system to different network types. However, it suffers from several basic problems, stemming from the overly simple nature of the basic abstraction (such as datagrams):

- Communication clients cannot express their performance, reliability and security needs to the communication provider. Some applications, for example, may not need data integrity, and therefore do not need data checksumming. Typically, however, data integrity is a mandatory part of the primitive. Conversely, a network interface might do link-level data checksumming in hardware; there is no means for software layers to learn of this and avoid doing checksumming themselves.
- It does not provide a well-defined means for the communication provider to dictate limits on client behavior, such as the amount of client data outstanding within the

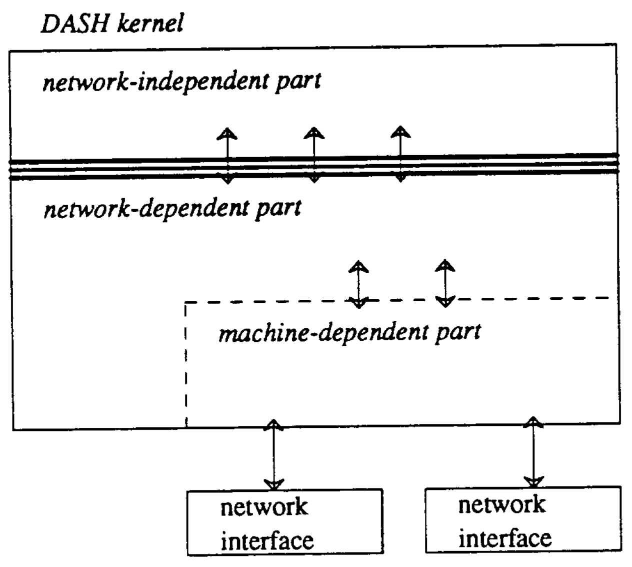

Figure 1: The Interface to the Network-Dependent Part of the DASH Kernel

network. This makes congestion control in large networks difficult.

- It makes no provision for real-time performance guarantees. Such guarantees are needed for interactive high-bandwidth traffic such as digitized audio [3] and video. This shortcoming leads to restricted, hardware-expensive solutions.

Finally, in future large-scale general-purpose systems, request/reply communication primitives will not be sufficient, because they cannot efficiently provide stream-style communication (as is needed for digitized audio and video) on high-delay long-distance networks.

Work in the area of real-time distributed systems for robot control has led to the development of communication primitives that are parameterized by real-time constraints or reliability needs [13, 14]. This work is in the same spirit as that described here, but addresses a restricted domain: request/reply communication, small networks, and no security concerns.

In an attempt to solve these problems, the DASH network communication system has been based on an abstraction called *real-time message streams* (RMS). An RMS is a simplex stream with several performance, reliability, and security parameters. The RMS abstraction appears in the interface to the network-dependent part, and at higher levels of the DASH system as well. RMS also is the basis for a request/reply communication facility, and the RMS features serve to optimize request/reply performance.

The use of RMS in DASH is based on anticipated needs and on projections of future network technology; RMS are not supported on current networks, and they cannot be built on top of simpler abstractions such as datagrams or virtual ciruits. However, we feel that our approach is necessary for exploiting the advances in communication technology that will occur in the near- and long-term future.

This paper is organized as follows: Section 2 defines the RMS abstraction and lists some possible uses. Section 3 describes the DASH communication architecture, which is based on RMS. Section 4 discusses implementation techniques for RMS in the areas of multiplexing, flow control, and process scheduling. Section 5 summarizes the work.

2. REAL-TIME MESSAGE STREAMS

A *real-time message stream* (RMS) is a simplex communication channel between a *sender* and a *receiver*. The sender is a process or set of processes that can invoke a *send* operation on the RMS. The receiver is typically a passive object such as a port; a message is considered *delivered* when it is enqueued on the port or given to a process waiting at the port. *Messages* are untyped byte arrays. They may in addition have *source* and *target* labels identifying the sender and receiver. The sender and receiver are *RMS clients*. The hardware and software system supporting the creation and use of RMS is the *RMS provider*. A client at one level may be a provider at a higher level.

An RMS has the following basic properties: 1) message boundaries are preserved; 2) messages are delivered in sequence; 3) client are notified of an RMS failure. In addition, an RMS has other parameters as described in the following sections.

2.1. Reliability and Security Parameters

An RMS has the following Boolean parameters:

Reliability: if true, then all messages that are sent on the RMS are delivered, unless the RMS fails.

Authentication: if true, then impersonation (delivery of a message with incorrect source label) is impossible.

Privacy: if true, then eavesdropping (access to a message by a host or process other than that specified by the target label) is impossible.

2.2. Performance Parameters

An RMS has the following performance parameters:

Capacity: an upper bound on the amount of data outstanding within the RMS at any point (i.e., sent but not yet delivered). This limit is enforced by the RMS clients, not by the provider (see section 3.4).

Maximum message size: an upper bound (enforced by the sender) on the size of individual messages. This limit cannot be greater than the RMS capacity.

Delay bound parameters: message delay is the elapsed real time between the start of the send operation and the moment of delivery. An RMS has an upper bound (guaranteed by the RMS provider) on message delay. The components of the delay may include network transmission delay, queueing and processing delays at the sender and at intermediate switches, and processing at the receiver. The bound is expressed as

$$A + B^*(message\ size),$$

where A and B are parameters of the RMS. This bound may be *deterministic*, *statistical*, or *best-effort* (see section 2.3).

Statistical workload parameters: if the delay bound is statistical, an RMS has *average load* and *burstiness* parameters (supplied by the client) and a *delay probability* parameter (guaranteed by the provider).

Average bit error rate: this parameter reflect the combination of 1) the error rate of the underlying transmission medium, 2) the effectiveness of the checksumming algorithm, and 3) the expected rate of packet loss from buffer overrun. It is guaranteed by the RMS provider.

Initially it might seem that an RMS should have a ''guaranteed bandwidth'' parameter. However, this is implied by the other parameters. If M is the maximum message size, D is the maximum delay of a message of size M, and C is the RMS capacity, then a client can send a message of size M every DM/C seconds without violating the capacity rule, since at any point at most C/M messages (of total size C) will have been sent within the previous D seconds, and all earlier messages are guaranteed to have been delivered already. This will provide a bandwidth of about C/D bytes per second. The actual maximum bandwidth may either be lower (because of errors and protocol overhead) or higher (if actual delays are smaller than the upper bound).

2.3. Delay Bound Types

The delay bound parameters of an RMS have the following types:

Deterministic: the delay bounds are ''hard''; only an RMS failure will cause them to be violated. System resources (buffer space, media bandwidth) are allocated to individual

RMS's. The RMS provider rejects an RMS request if its worst-case demands cannot be met with free resources.

Statistical: the delay bounds hold probabilistically, and may require a statistical description (average load and burstiness) of the offered workload. An RMS creation request is rejected if either its expected message delay or its expected bit error rate (which is affected by the possibility of buffer overruns) is higher than acceptable. Failure to observe the delay bounds is not necessarily reported to the clients.

Best-Effort: RMS creation requests are never rejected. Delay bound parameters are used only to schedule resources based on message delivery deadlines (see Section 4.1).

2.4. RMS Creation and Ownership

RMS creation operations (offered by RMS providers) specify the direction of the RMS; the creator of an RMS may act as either the sender or the receiver. If there is accounting, the creator *owns* the RMS in the sense of being responsible for paying for its use.

RMS parameters are established at the time of creation. A set of actual RMS parameters is said to be *compatible* with a set of request parameters if

(1) the actual reliability and security properties include those requested;

(2) the actual capacity and maximum message size parameters are no less than those requested, and

(3) the actual delay bound and error rate parameters are no greater than those requested.

An RMS creation request includes *desired* and *acceptable* parameter sets. The actual parameters of the resulting RMS (if any) are returned to the client. These parameters must be compatible with the request's *acceptable* parameters. The RMS creation request is rejected if this is not possible. The RMS provider tries to match the *desired* parameters as closely as possible.

2.5. RMS Examples

To see the importance of RMS parameters, consider the case of a client (say a transport protocol serving a user program) that requires data privacy. The protocol requests an RMS from the subtransport layer (see Section 3.2). The desired and acceptable parameter sets both request privacy. Depending on the network, the following situations are possible:

(1) Privacy is provided by data encryption in the subtransport layer.

(2) The network has link-level encryption hardware; The subtransport layer learns this (it is a property of network-level RMS's) and does no data encryption;

(3) The network is considered secure, so no data encryption is done.

In any case, the optimal mechanism is used for privacy. If a client does not require privacy, no mechanism is used (which is again optimal). Without the RMS security parameters, this optimization would not be possible. A similar situation exists for data integrity: the optimal checksumming mechanism can be used based on RMS parameters.

The following examples illustrate the uses of the RMS capacity and performance parameters:

- Initial request and reply messages in a request/reply protocol should use RMS's with low delay bound. The precise delay bound and the delay bound type are determined by application needs. The RMS capacity may be large, unless it is known that request or reply messages will be small and infrequent.
- A stream protocol for bulk data transfer should use a high capacity, high delay RMS for data. RMS's for acknowledgements are discussed below.
- Reliability acknowledgements should use low capacity, high delay RMS's.
- Flow control acknowledgements should use a low delay, low capacity RMS. In DASH, the subtransport layer provides a "fast acknowledgement" service to reduce response time and RMS establishment overhead.
- Real-time communication may require deterministic or statistical delay bounds.
- Digitized voice should use a high capacity, low delay RMS, perhaps with a statistical delay bound. A high bit error rate may be acceptable.
- Communication involving a human user interface traffic (such as for network window systems [7]) can tolerate a moderate amount of delay because of human perceptual limitations. The RMS from user to application carries mouse and keyboard events, and can have low capacity. The RMS in the opposite direction carries graphic information, and generally requires higher capacity.

In all these cases the specification of client needs increases the likelihood that the provider can accommodate them. For example, if packet queueing in an internetwork gateway is done using RMS-specified deadlines, then a low-delay packet can be sent before high-delay packets that would otherwise cause it to be delivered late. A network may be capable of providing low delay or high capacity, but not both. The RMS parameters allow the client to choose.

3. THE DASH COMMUNICATION ARCHITECTURE

In this section we briefly describe the DASH communication architecture. This is done primarily to motivate the material in Section 4. Details of addressing, naming, encryption schemes, and specific operations are omitted. The structure of the DASH network communication architecture is shown in Figure 2.

3.1. The Network Layer

DASH allows multiple network types. Each network type to which a DASH host is connected is represented in its kernel by a software module with a standard RMS-based interface. These *network objects* provide host-to-host *network RMS's*. They encapsulate network-specific protocols for RMS creation, deletion, and transmission, and for non-RMS network maintenance tasks such as routing.

Networks are abstract entities, and need not be physically or logically disjoint. For example, the DARPA Internet and a local Ethernet (both with the addition of an RMS protocol) could be separate DASH networks, although they might share host interfaces and network media.

A network object has the following parameters:

- Whether all hosts on the network are trusted. If so, optimizations by the subtransport layer (see below) are possible.

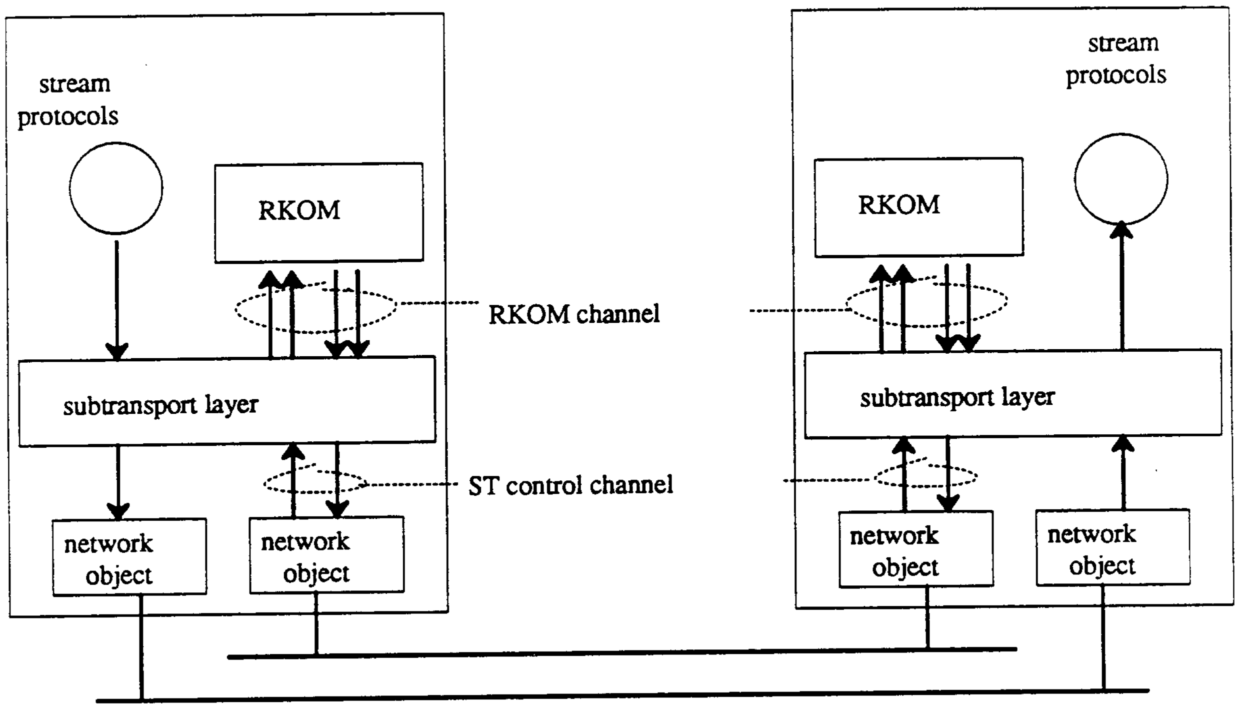

Figure 2: The DASH Communication Architecture.

- Whether the network has the "physical broadcast property": that if an eavesdropper receives an entire message, then so does its intended recipient. If so, optimizations by the subtransport layer are possible.
- For each combination of security and reliability parameters, the limits of the network's performance parameters for that combination (this may be zero if the combination cannot be directly supported).

3.2. The Subtransport Layer

The subtransport layer (ST) provides a variety of host-to-host functions. All upper-level network communication in DASH passes through the ST. The ST provides *ST RMS's* to its clients. ST RMS's are multiplexed onto network RMS's. The basic functions of the ST are to provide security [2], to do deadline-based message queueing, to multiplex ST RMS's onto network RMS's, and to arrange for "fast acknowledgement" of messages sent on ST RMS's.

For each active peer host, the ST maintains a *control channel* consisting of two low capacity, low delay network RMS's, one per direction. The ST uses a simple request/reply protocol on this channel to do authentication and ST RMS establishment. The first ST RMS creation request to a given peer triggers the creation of the ST control channel to that peer.

In addition, the ST maintains a set of *data* network RMS's to the peer. These are used to carry ST RMS traffic. Their ownership, direction and multiplicity are dynamically determined by ST client demand; see section 4.2.

3.3. Transport Protocols

All request/reply communication uses the DASH Remote Kernel Operation Mechanism (RKOM). RKOM provides kernel-level request/reply communication, and is used as a basis for user-level request/reply communication. The RKOM module maintains an *RKOM channel* to each active peer. Such a channel consists of four ST RMS's, one low-delay and one high-delay RMS in each direction. The low-delay RMS's are used for initial request and reply messages, and the high-delay RMS's are used for retransmissions and acknowledgements.

In addition to communication using RKOM, user- and kernel-level clients can establish their own communication sessions. These sessions typically consist of 1) a set of ST RMS's and 2) a set of stream protocols, each of which is a kernel-level process.

3.4. RMS Levels in DASH

In addition to the network- and ST-level RMS's described above, DASH provides the following RMS types (see Figure 3):

Subuser RMS: this spans communication protocol processes. Message sending and delivery are defined as the moments when message arrive from, or are passed to, user

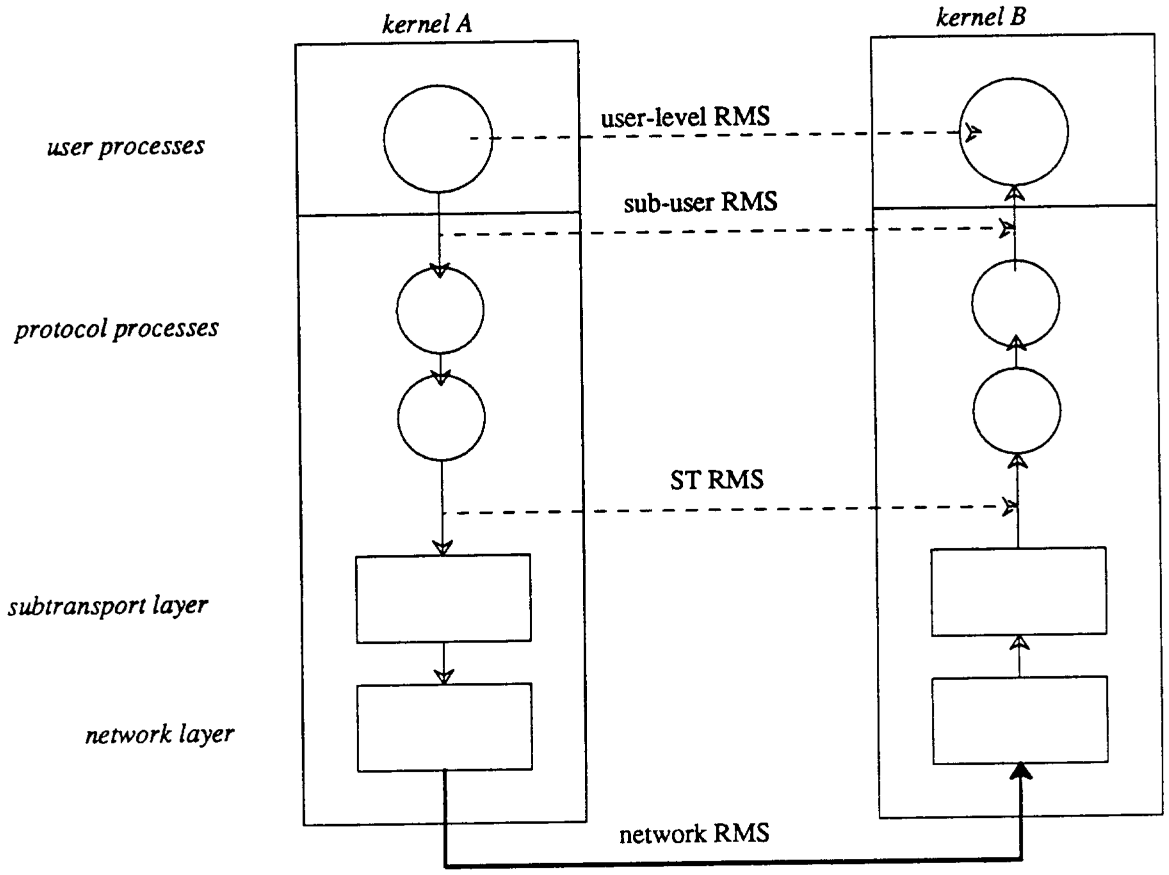

Figure 3: RMS Levels in DASH

processes. The delay bounds include protocol processing time, and their enforcement includes deadline-based process scheduling (see Section 4.1).

User-level RMS: this spans user processes. The moments of message sending and delivery are defined by the user process, and end-process CPU time is included in the RMS delay. Scheduling of these user processes must be deadline-based.

4. IMPLEMENTATION TECHNIQUES FOR RMS

This section describes a set of techniques, algorithms, and issues that arise in providing low-level RMS and in building high-level RMS out of low-level RMS. These techniques are presented in the context of the DASH architecture described in the previous section.

4.1. Process and Interface Scheduling

When an upper-level RMS is created, its total delay is divided among its various stages (send protocol processing, ST RMS delay, network RMS delay, and receive protocol processing). When a message is sent on an RMS, there is a *deadline* by which it must be handled (i.e., processed by a protocol, sent on a lower-level RMS, or transmitted on a network medium). This deadline is the current real time plus the delay allocated to the next stage of the RMS.

For subuser and user-level RMS, these deadlines are used by the short-term scheduler to determine the execution order of protocol or user processes. These scheduling deadlines are "hard" or "soft" depending on the delay bound type of the RMS. For network RMS, the deadlines are used to determine the order in which packets are queued for transmission on a network interface.

4.2. RMS Caching and Multiplexing

The ST *caches* network RMS's; i.e., it may retain a network RMS even while it is not being used by any ST RMS. This caching is motivated by two assumptions: 1) during a given time period a host will tend to communicate repeatedy with a small set of remote hosts; 2) it is slow and costly to create network RMS's.

The ST also does *upwards multiplexing* of multiple ST RMS's onto a single network RMS (see Figure 4). The motivations for this multiplexing (as opposed to simply creating a network RMS for each ST RMS) are that 1) it can eliminate the need to create a new network RMS, and 2) messages from multiple ST RMS's can potentially be *piggybacked*, i.e., combined and sent as a single network message, with a possible reduction in overhead. Multiplexing of ST RMS's increases the potential frequency of piggybacking.

Among the rules that govern RMS multiplexing are:

- A deterministic ST RMS can only be multiplexed onto a deterministic network RMS, and a statistical ST can be multiplexed only onto a deterministic or statistical network RMS.
- The delay bound parameters of the ST RMS's must be at least those of the network RMS; the difference is a potential *queueing delay* during which the ST can attempt to piggyback additional messages with the outgoing message.

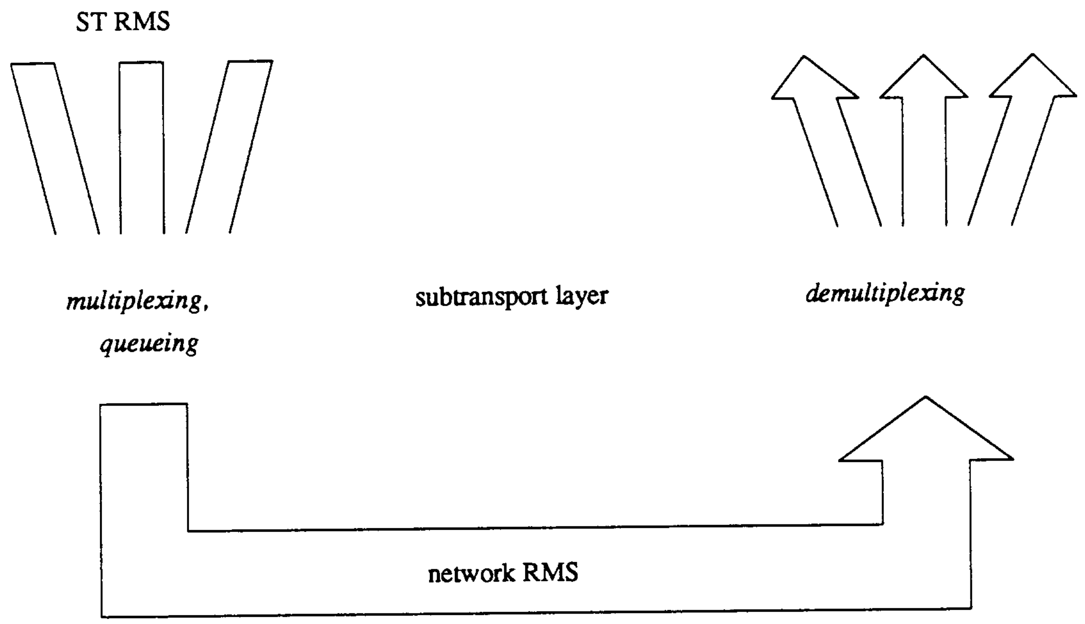

Figure 4: RMS Multiplexing

- The capacity of the network RMS must be at least the sum of the capacities of the ST RMS's.
- The maximum message size of the ST RMS's may exceed that of the network RMS. This requires fragmentation and reassembly by the ST (see below).

It would also be possible to downwards-multiplex an ST RMS across several network RMS's. If there were multiple networks paths between the hosts, this technique could be used to increase capacity beyond that available in a single network RMS. However, this has not been included in the DASH design because the expected gain may not outweigh the additional ST protocol complexity.

4.3. Increased Maximum Message Size

It is an issue whether the ST should offer a much larger maximum message size to its clients than that provided by the network layer. This is done, for example, in VMTP [6]. At the network level there will always be a message size limit (e.g., the 1.5KB Ethernet packet size limit) because of hardware restrictions, RMS capacity, nonzero bit error rate, or the need for fairness.

To offer its clients much larger message lengths, the ST would have to use a transport protocol providing both reliability and RMS capacity enforcement. In general, this places an undesirable burden on the ST, and may duplicate work being done at higher levels. On the other hand, providing a somewhat larger maximum message than that provided by the network layer may reduce protocol process context switching and other overhead.

For these reasons, the ST may provide a larger maximum message size than the network layer. A maximum message size is chosen with the object of maximizing potential throughput based on the combination of network RMS error rate and context switch time. The ST does fragmentation and reassembly to support this larger message size. It does not retransmit fragments; if a message is incomplete when a fragment of the next message arrives, the partial message is discarded.

4.3.1. Message Queueing and Ordering

When the ST sends a fragment for transmission on a network RMS, a *transmission deadline* parameter is passed to the network RMS send routine. If the network interface has a nonempty transmission queue, transmission deadlines determines the order in which messages are sent. In any case, the network layer must guarantee that if message *A* is sent after message *B*, and has a transmission deadline greater than or equal to that of *B*, then *B* is delivered first (note: this is a refinement, particular to network RMS, of the sequential delivery property of RMS)

If the ST does no piggybacking, and immediately sends each client message on the associated network RMS using transmission deadlines that are monotonically increasing for a particular ST RMS, then ST RMS messages will be delivered in the correct order. The current real time could be used as the transmission deadline. However, it would be preferable to use

(current real time) + (ST RMS delay bound) - (network RMS delay bound)

since this would better reflect the true deadlines.

It is also possible (and perhaps desirable) for the ST to internally queue messages in the hope of piggybacking. In doing this, care must be taken to preserve the ordering of ST RMS messages while still honoring deadlines. The following set of policies can be used to achieve this:

- For each outgoing network RMS, the ST maintains a *piggybacking queue* of client messages awaiting transmission. This queue never exceeds the network RMS maximum message size. ST client messages that require fragmentation are not piggybacked.
- The *maximum transmission deadline* of an ST client message is its arrival time plus the ST RMS delay bound minus the network RMS delay bound.
- The *minimum transmission deadline* of a client message is the actual transmission deadline of the previous message on the same ST RMS. This ensures that messages on the ST RMS are delivered in order.
- The minimum (maximum) transmission deadline of a piggybacking queue is the minimum (maximum) of the corresponding deadlines of its component messages.
- When a piggybacking queue is finally sent on the network RMS (either because its maximum transmission deadline is reached or because it overflows) the transmission deadline passed to the network layer is the queue's maximum transmission deadline.
- If a client submits a message whose maximum transmission deadline precedes the minimum transmission deadline of the corresponding queue, then it is sent on the network RMS immediately. Otherwise it is appended to the piggybacking queue if possible.

This algorithm maximizes the possibility of piggybacking, while ensuring correct ordering and optimal interface scheduling.

4.4. Flow Control and RMS Capacity Enforcement

Where there is a limited buffer space in a communication system, flow control can be used to avoid performance loss from buffer overrun and dropped packets. Flow control mechanisms are often necessary for even minimal performance levels. On the other hand, flow control mechanisms may not be needed in certain situations (for example, if the data rate of the sender is known to be low) and in that case may impose an unnecessary overhead.

It is useful to factor communication system buffers into three groups:

(1) Buffers between the sending process and the send protocol.

(2) Buffers in the network switches and gateways, and in the receiver's network interface and low-level driver.

(3) Buffers between the receive protocol and the receiver.

The RMS approach to flow control treats these buffer groups separately. The capacity parameter of an RMS prevents overrunning buffers in group (2). Since there is no RMS capacity adjustment, it is assumbed that the sizes of these buffer do not change dynamically. If they do, the RMS provider must delete the RMS, and the clients must establish a new RMS. In contrast, the flow control of TCP does not protect gateway buffers; ICMP source quench messages [8, 10] provide an *ad hoc* and often ineffective solution to this flow control problem.

RMS clients are responsible for enforcing the RMS capacity. If they fail to do so, the provider's guarantees are voided; messages may be delivered late or discarded. The RMS provider is not responsible for detecting potential capacity violations and blocking the sender. This simplifies the task of RMS providers, and it means that in cases where no flow control is necessary (perhaps because the sender is known to be sufficiently slow) there is no wasted overhead for capacity enforcement.

If a capacity-enforcement mechanism is needed, the following approaches are possible:

- *Rate-based*: using timers, the sender ensures that during any time period of duration $A + CB$, the number of bytes sent does not exceed C. This approach is pessimistic in the sense that it assumes the maximum delay for all messages.
- *Acknowledgement-based*: the sender receives *flow control acknowledgements* for messages received. This may achieve higher maximum throughput at the cost of the reverse message traffic.

If the receiver can, on the average, process incoming data faster than it arrives, then it is possible that no flow control is needed (a large receive buffer may be needed; the size is determined by several factors, including the variability of the receiver's speed). If not, then a *receiver flow control* mechanism is needed; the protocol must stop sending data when the limit of the receive buffer is reached. The need for this flow control mechanism is independent of the need for RMS capacity enforcement; if both are needed they could be integrated into a single protocol.

There are no capacity-enforcement mechanisms in the DASH ST or network layers. Where needed, transport protocols can enforce RMS capacity using either rate-based or acknowledgement-based mechanisms. In addition, they may provide a receiver flow control mechanism if needed.

If the sender can produce data faster than it can be sent on the ST RMS (because of capacity enforcement, receiver flow control, or both) then there must be a *sender flow control* mechanism. This is done in the DASH kernel using a flow controlled local IPC port for message-passing between the sender and the send protocol. A sender blocks when a port queue size limit is reached. The sending transport protocol stops reading messages from the port while it is prevented from sending because of RMS capacity enforcement or receiver flow control.

If mechanisms for sender flow control, RMS capacity enforcement, and receiver flow control are all present, then there is end-to-end flow control ([11]). However, in cases when this is not necessary, performance optimizations (simpler protocols and fewer messages) may be possible. Figure 5 shows the different options for flow control.

5. CONCLUSION

We have proposed the *real-time message stream* (RMS) abstraction as a network-independent communication primitive in future distributed systems. RMS allows the client and the provider to negotiate parameters. Client can specify their performance, reliability and security requirements to the provider, and the provider can dictate limitations on the behavior of the client. Compared to simpler abstractions, this has the following benefits:

- The solution of the congestion control problem is simplified by the availability of (and control over) network load information. This may be critical in the design of large-scale high-performance networks.

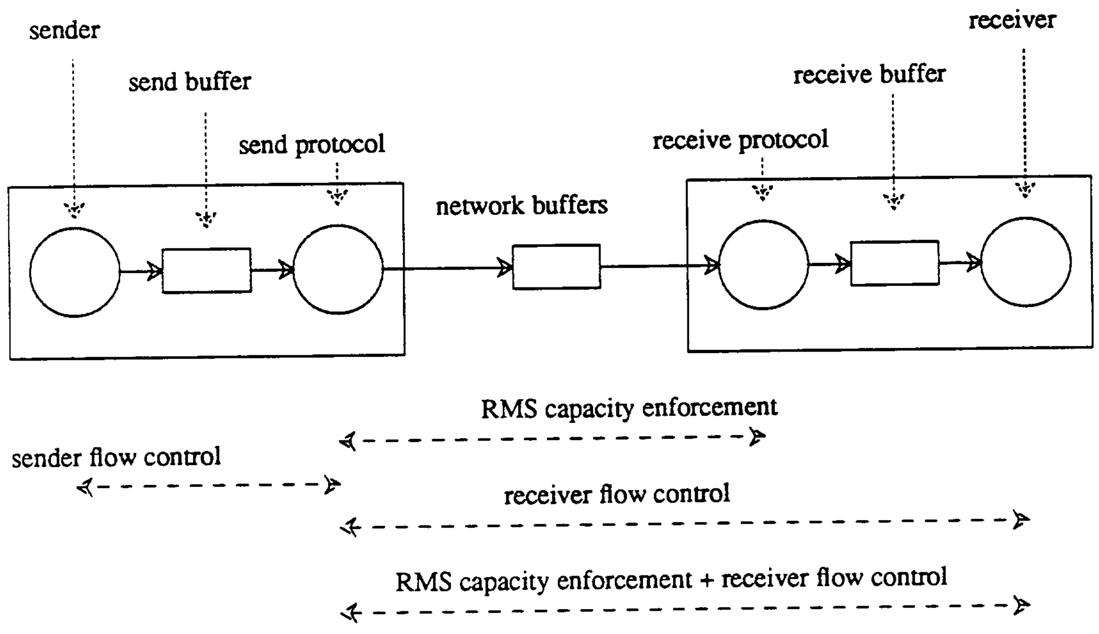

Figure 5: Flow Control Options.

- Real-time deadlines can be used to schedule both network bandwidth and CPU time. Compared to systems that use only priorities (or no information at all), this optimizes usage and makes real-time communication possible. A system cannot provide real-time communication unless it is supported at the lowest level.
- RMS capacity enforcement is separated from sender and receiver flow control. Based on the values of RMS parameters it can be determined what flow control mechanisms are needed, and unnecessary mechanisms can be avoided.
- Flow control protocols can be simpler (because of the fixed window size determined by RMS capacity) and more efficient (because of the fast acknowledgement service provided by the ST) than those in traditional protocol hierarchies.
- Clients may have better control over network costs. RMS parameters correspond roughly to the network resources (buffer space and bandwidth) consumed. A network might charge a fixed RMS setup cost, plus a charge determined by the RMS parameters, the number of bytes sent, and the RMS connect time.

In addition, we have described the DASH network communication architecture. It illustrates the following points:

- It may be desirable to introduce a *subtransport layer* that does piggybacking, RMS caching, and other functions.
- A request/reply facility can be built on RMS, and can exploit its features.
- An operating system can provide higher-level RMS's that provide real-time properties similar to those of lower-level RMS's, but which take into account protocol and perhaps user processing time at each end.

Many questions related to RMS remain to be investigated, including:

- How can RMS's be supported on existing and future networks? In particular, how can it be determined whether a request for a new set of RMS's should be granted or not? Solutions may vary in complexity according to the underlying network and the types of RMS supported. An Ethernet RMS protocol supporting only the *best effort* type is being implemented in the DASH prototype.
- How can RMS's be supported in internetworks? A design for a best-effort RMS protocol for the DARPA Internet is given in [4].
- How should the workload of an RMS with a statistical delay bound be parameterized, and how can these parameters be used to determine rules for multiplexing such RMS's?
- How can deterministic, statistical and best-effort RMS's be intermixed on the same network?
- What are the optimal RMS transport protocols for providing various combinations of reliability, RMS capacity enforcement, and receiver flow control?

These and other issues are currently being investigated in the DASH research project, under the direction of Professor Domenico Ferrari and myself.

6. Acknowledgement

I would like to thank the members of the DASH project, especially Robert Wahbe, for their involvement in this work. G. Scott Graham and Domenico Ferrari gave valuable critical readings of this paper.

REFERENCES

1. D. P. Anderson, D. Ferrari, P. V. Rangan and S. Tzou, "The DASH Project: Issues in the Design of Very Large Distributed Systems", *Report No. UCB/Computer Science Dpt. 87/338, CS Division (EECS Dept.), UC Berkeley*, January 1987.
2. D. P. Anderson, D. Ferrari, P. V. Rangan and B. Sartirana, "A Protocol for Secure Communication in Large Distributed Systems", *Report No. UCB/Computer Science Dpt. 87/342, CS Division (EECS Dept.) UC Berkeley*, February 1987.
3. M. Bastian, "Voice-Data Integration: An Architecture Perspective", *IEEE Commun. Mag. 24*, 7 (July 1986), 8-12.
4. J. R. Betten, "IRMSP: A Protocol for Real-Time Communication in the DARPA Internet", *Master's Project Report, CS Division, US Berkeley*, Decemeber 1987.
5. D. R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations", *Proceedings of the 9th Symp. on Operating System Prin., Operating Systems Review 17*, 5 (October 1983), 128-140.
6. D. R. Cheriton, "VMTP: A Transport Protocol for the Next Generation of Communication Systems", *1986 SIGCOMM Symposium*, , 406-415.
7. J. Gettys, "Problems Implementing Window Systems in UNIX", *USENIX Winter Conference Proceedings*, January 1986, 89-97.
8. J. Nagle, "Congestion Control in IP/TCP Internetworks", *Internet RFC 896*, January 1984.
9. J. Postel, "Transmission Control Protocol", *DARPA Internet RFC 793*, September 1981.
10. J. Postel, "Internet Control Message Protocol", *DARPA Internet RFC 792*, September 1981.
11. J. H. Saltzer, D. P. Reed and D. D. Clark, "End-To-End Arguments in System Design", *ACM Trans. Comput. Syst. 2*, 4 (Nov. 1984), 277-288.
12. R. D. Sansom, D. P. Julin and R. F. Rashid, "Extending a Capability Based System into a Network Environment", *1986 SIGCOMM Symposium*, , 265-274.
13. K. Schwan, T. Bihari, B. W. Weide and G. Taulbee, "High-Performance Operating System Primitives for Robotics and Real-Time Control Systems", *ACM Transactions on Computer Systems 5*, 3 (August 1987), 189-231.
14. K. G. Shin and M. E. Epstein, "Communication Primitives for a Distributed Multi-Robot System", *Proceedings of the IEEE International Conference on Robotics and Automation*, March 1986, 910-917.