Chapter 1

Introduction

The birth of packet switched networks in the early 1960's has spawned the dream of an all-encompassing digital network that would span the globe and serve all our communication needs; this revolutionary network would carry voice, images, video and data, and bring together people from all corners of the world. Today that dream is becoming reality in the form of the Internet (Fig-

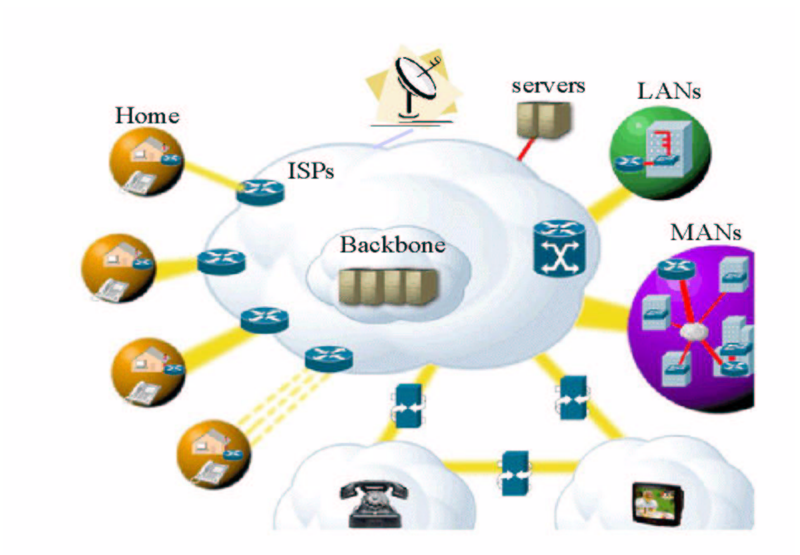

Figure 1.1: A representation of the Internet

ure 1.1).

The Internet has come a long way since its beginnings, when it was known as the ARPANET. It started as a handful of low bandwidth links used to share resources among computers, back when computers were scarce and frightfully expensive. Since its birth, the Internet has been growing exponentially; it started with the computer technology revolution, which lead to the network technology revolution, and finally to the Internet revolution. The result is that today we have a network that spans across continents, connecting several million computers and putting a staggering amount of information at our fingertips. Perhaps the biggest measure of its success is that the Internet today supports a huge number of commercial applications with revenues measured in the billions, some of which would not have been possible without the Internet.

One major component of the success of the Internet is its radical architectural shift from traditional networks like the telephone or cable television networks. Traditional networks are mostly purpose-build networks, conceived with one application in mind like voice or television. Such networks are based on a simple communication model: for example, the telephone network emulates direct wire connections by providing isolated, fixed capacity channels (called circuits). Thus, when two entities wish to communicate, they ask the network to establish a private circuit between them for their exclusive use; at the end of the conversation, the circuit is released and becomes available for use by other entities. Links are shared using *time-division multiplexing (TDM)*, which works by dividing time into fixed slots and allowing an application access to the network only during its allocated slot; thus, the bandwidth an application may use is governed by the speed of the link and duration of the slot, which is always a fraction of the total link bandwidth. Despite their limitations, existing networks like the telephone network have been highly successful, and have established a huge infrastructure around the world. They are still the main means of data transmission today, including Internet data.

Internet designers realized early on that engineering a network that emulates a wire (i.e., goes to great lengths to provide a completely reliable communication infrastructure), while well-suited for fixed bandwidth dumb terminals and telephones, is expensive and inflexible. The designers realized that by abandoning the wire-like robustness and adding intelligence at the endpoints to compensate, a network can not only be built on a much simpler and cheaper infrastructure, but can also adopt a more flexible service model that can serve a wide range of applications rather than just one. The key ingredient to such flexibility is *asynchronous packet switching*. Unlike TDM, where data is carried in slots, packet switching allows data to be carried in variable size chunks called *packets*. In TDM the destination of the data is identified by the slot; packets, however, carry a *header* which contains the *address* (or a circuit identifier) of their destination and are therefore, *self routing*. This property allows packets to be stored in the network and forwarded immediately when the link becomes available, rather than wait for their slot; this technique, called *store and forward*, allows applications in packet-switched networks to use all of the available channel bandwidth. Packet switching has proven so successful that today about 50% of the network traffic is packet switched; that figure is expected to rise to about 80% in a few years. Therefore, it seems highly likely that network engineers will concentrate on building packet switched networks in the future.

Due to finite buffering, store-and-forward packet networks experience loss. Loss occurs due to burstiness, which is a result of computer communication. When a burst of packets arrives at a buffer which is already full, there is no choice but to drop some packets, because the network capacity has been exceeded. This results in a model called a *best-effort service model*, where the network promises to do its best to forward a packet, but delivery is not guaranteed. The best-effort model has proven highly successful in the Internet, because it makes very few assumptions from the underlying hardware. The embodiment of the best-effort model is the *Internet Protocol (IP)* which is capable of riding on many technologies and thus connecting together a wide variety of heterogeneous networks. IP clearly separates the network layer from the underlying physical layer; this decoupling of the layers allows network technologies to evolve independently, and yet still be part of the Internet. IP has been instrumental in making the Internet a “network of networks”, and thus allowing it to achieve the scale it enjoys today.

1.1. Loss is Part of the Internet

As a result of packet switching[1] and the resulting best-effort service model, network applications have to deal with loss. As mentioned earlier, in packet switched networks the finite link bandwidth, the finite buffers at the routers, the shared bandwidth model, and the bursty nature of many network applications, make it hard to absorb bursts. Thus, Internet routers often experience loss. While it is possible that advances in optical switching may eventually provide an abundance of

1. Packet switched networks can be made to have no loss, but that would increase their cost and complexity significantly.

bandwidth to terrestrial networks and loss will become rare, the heterogeneity of the Internet ensures that at least portions of it will always have limited bandwidth. Such portions include bandwidth limited segments such as ones served by modems, and wireless segments where loss may occur for reasons other than lack of bandwidth (e.g., weather conditions).

Congestion and the resulting loss is not just a function of limited bandwidth, but is actually a consequence of the Internet design. In a best-effort model, applications are not given resource bounds, but must discover available resources like bandwidth dynamically. This allows high utilization of the network. For example, today's leading Internet transport protocol, namely TCP[37], consumes bandwidth in a greedy fashion by deliberately and continuously driving the network to loss before backing off, in an effort to probe for available bandwidth.

The issue of loss is complicated even further by the recent emergence of Multicast. In multicast applications, which are envisioned to scale to thousands, or hundreds of thousands of receivers, data sent by a single sender often spans a substantial portion of the network; at such scale, the probability that a flow will experience loss is much greater than unicast.

Finally, even if we managed to eliminate loss by gross over-engineering of the entire network, loss can still occur due to mismatch in sender/receiver speed and subsequent buffer overrun. This is especially true of mobile receivers which typically have limited memory and CPU resources.

Given that Internet routers will continue to drop packets, applications must counteract loss by using *error control*. Error control is the part of a communication protocol responsible for detecting and recovering loss during data transmission. It typically contains three phases: (a) *error detection*, where the protocol detects that a packet was lost; (b) *error notification*, where the sender is notified of the loss; and (c) *retransmission*, where the sender retransmits the lost packet(s) to the receiver. Another option for error control is Forward Error Correction (FEC), where redundant data is sent along with the original data which allows the reconstruction of lost data at the receiver. Error control has been widely discussed in literature and good references include [1, 2]. We discuss it further in Chapter 3.

In summary, loss in the Internet is generally not a design flaw, but a feature that allows the Internet to be built on a very simple and economic infrastructure, provide a flexible service model that allows several classes of applications, and achieve high network utilization. As a consequence, however, network applications must be enhanced with error control.

1.2. The need for multiple classes of Error Control

There are many classes of applications on the Internet today, and each class has different requirements for error control. Here are a few examples:

- *Web Access* and *File Transfer* are the most popular applications in the Internet today. These applications are relatively tolerant to delay; their main requirement is that data is transferred with no packet loss. There are no strict bounds on when the transfer should complete, except perhaps user frustration. Thus, even in the presence of relatively large delays (in the order of seconds), these applications' utility remains high. Examples of such applications include web surfing, FTP and email.

- *Continuous Media (CM)* applications*,* which include audio and video, require that data be delivered within certain time bounds, otherwise it is considered lost. Typically, these applications have a certain degree of redundancy and thus sacrificing some packets to maintain timely delivery for the remaining packets is usually an acceptable trade-off. Notice how this is exactly opposite of the previous class. Examples include teleconferencing, video-on-demand, and visualization.

- *Transaction Oriented (TO)* applications have different requirements than either file transfer or CM applications. *TO* applications require fast and reliable delivery of typically short messages. They may also have other requirements like in-order delivery or at-most-once semantics, which error control must take into account. Examples include request-response applications like database queries.

- *Multicast Applications* are a class of applications that emerged recently with the creation of the MBONE. These applications may have large numbers (hundreds or even thousands) of participants distributed worldwide. These applications' primary concern is scalability; thus they require error control schemes that are capable of coordinating recovery among a large number of receivers. Examples include: whiteboard applications, distance learning, distributed interactive simulation and software updates.

The above list of classes of applications requiring different error control mechanisms is by no means complete; however, it serves as an indication of the many varieties of error control mechanisms that will be required in the Internet.

Defining error control mechanisms for every application is an enormous task and well beyond the scope of this work. Thus, in this thesis we address **Error Control for Interactive Continuous Media and Large Scale Multicast Applications**. We chose these classes of applications because they are very important to Internet users and pose a set of unique challenges to error control; these challenges are very different from the traditional and well-understood issues in file transfer, and have no adequate solutions yet. We describe these classes of applications next.

1.3. Error Control for Interactive, Continuous Media Applications

Continuous media applications are a class of applications employing relatively long-lived, continuous data streams, typically lasting minutes, hours, or more. Examples of CM applications include video, audio, image animation, and visualization among others. The salient characteristics of CM streams are their periodicity and strict timing requirements. For example, to produce a smooth video, *MPEG (*which is a high-quality video compression format used by most digital television programs today) requires that a frame is displayed precisely every 33 ms.

We can divide CM applications into two general classes: *interactive* and *stored media*. As the name implies, interactive applications typically involve interaction between people, and are, therefore, bidirectional; examples include teleconferencing and telephone conversations. In addition to intra-stream timing requirements, interactive applications require that data is transmitted almost instantly after it is generated, to avoid pauses in the conversation.

Like interactive applications, stored media applications demand strict timing within the stream; however, they are more flexible about when playback is initiated. The reason is that such applications are typically unidirectional and preserving interaction is not critical, except perhaps to ensure that video control operations like pause, fast-forward, and rewind remain on par with existing video players (e.g., 0.5 - 1 sec). Examples include Internet applications like RealPlayer, Video-on-Demand, and viewing of video clips from a news server.

Error control for stored media applications is relatively easier than interactive applications because the former's unidirectional nature allows a relatively large *playout buffer* at the receiver. A playout buffer is used to store new data for a short time before playback; this is advantageous because it helps absorb jitter which helps maintain the correct playback timing at the receiver, and potentially allows additional time for error recovery. Playout buffers of several seconds are common in the Internet today; the RealPlayer application for example, typically uses a playout buffer of 20 seconds.

Most existing CM applications avoid the use of error control because it makes meeting application latency constraints difficult. For example, it is often argued that retransmission cannot be used for coast-to-coast error recovery for interactive applications because there is simply not enough time to recover. Indeed, with MPEG streams generating a new frame every 33ms and with a coast-to-coast *round-trip-time (RTT)* of 40-60ms, recovering loss with retransmission appears impossible. What makes things worse is that compressed streams like MPEG are far less tolerant to loss than uncompressed streams. Thus, due to bandwidth and loss limitations, most current Internet CM applications have resorted to custom video encodings [3, 4], which use lower bandwidth and frame rate than MPEG, and are more tolerant to loss. However, these applications typically offer much worse video quality, often resulting in jerky, postage stamp size frames. Fortunately, as bandwidth availability is increasing, MPEG-style applications will become feasible. However, this is only half of the solution; due to compression (which amplifies loss) and the presence of loss, such applications still require error control.

In this thesis, we have chosen to study error control for interactive, CM applications; we consider this to be an important class of multimedia applications because they facilitate human interaction. Our target environment is MPEG video at 30 frames per second, and target propagation latency is the US coast-to-coast round-trip-time (RTT), which is about 40-60 ms. An example application is coast-to-coast high-quality video-conferencing. This is a challenging application for the following reasons: (a) its interactive nature makes error control more difficult than stored media; (b) MPEG streams are very demanding compared to other video streams, due to their burstiness and high frame rate; and (c) the US coast-to-coast propagation latency is large compared to the MPEG inter-frame interval.

1.3.1. Our Solution

We believe that the issues that must be addressed in designing error control for interactive CM applications include the following:

- How can we maximize time available for recovery without hurting interaction?
- Given finite recovery time, how can we maximize the probability of timely delivery of data?
- In the event that a loss is unrecoverable, how can we help the application deal with the loss?

To achieve the first objective, we observe that studies of human interaction have shown that people can tolerate a certain amount of latency in their conversations. In other words, we perceive a response from another person to be almost instant if it occurs within about 200ms [10]. We gain leverage from this observation thereby increasing the interval available for recovery with our first technique: adding a limited amount of playout buffering (about 100ms) at each side to gain some additional time for retransmission.

Our second technique, which aims to maximize the probability of recovery, is to avoid initiating retransmissions if they would never make it in time to the receiver. Such retransmissions, if allowed to go through, may unnecessarily delay transmission of original data, thus wasting precious time. To suppress these retransmissions, the receiver maintains an estimate of the RTT to the sender and uses gap-detection to detect loss and send requests. Upon loss detection, and based on the current RTT estimate and the playback buffer occupancy, the receiver determines whether a retransmission has a good chance of arriving in time before sending the request, and thus can avoid late retransmissions.

Our final technique helps the application conceal unrecoverable loss by supplying control information about the frame in addition to the frame itself. Thus, whenever a frame is delivered to the application, the error control mechanism passes a bitmap indicating which portions of the frame are missing. The application may use this information in any way it pleases; for example, it could fill the missing part with parts from the previous frame.

In order to demonstrate the feasibility and viability of our solution, we implemented a transport protocol which employs the three techniques described above. Given our constraints, in the current implementation our protocol makes one retransmission attempt whenever loss is detected; however, we have shown how to extend the protocol to multiple retransmissions for applications with more relaxed constraints. The protocol's performance was tested both subjectively with animation streams, and quantitatively with random drop and bursty MPEG traces. We have found that even with a single retransmission, the protocol reduces the observed loss rate, sometimes by several orders of magnitude, without compromising the interactive nature of the stream. We have thus shown that the bias against using retransmission-based error control in CM streams is unfounded.

1.3.2. Contributions

Our work was done at a time when it was commonly believed that retransmission-based error control was unsuitable for interactive CM applications. Our main contribution is that with our research we have shown, against popular belief, that retransmission is not only possible, but can offer significant gains in lowering the observed loss rate in a CM stream while preserving interactivity. We have shown how to design such a protocol, and what error control mechanisms are appropriate for CM applications. Other researchers have subsequently reached similar conclusions [5, 6].

We have also contributed the implementation of a self-contained transport protocol with retransmission-based error control for interactive, continuous media applications. We have shown that such a protocol is not only feasible, but helps significantly in improving the perceived quality of continuous media and thus improves the ability of the Internet to carry interactive CM traffic. Our error control mechanism, while it works in a challenging environment like coast-to-coast MPEG video-conferencing, is tunable and can provide improved performance for other, less challenging classes of applications. Our transport protocol is integrated in the NetBSD Unix kernel and is available as an alternative to existing Internet protocols like UDP that lack error control.

1.4. Error Control for Large-Scale Multicast Applications

The second problem we have chosen to address is error control for large scale multicast applications. Multicast applications require simultaneous transmission of data to multiple receivers whose number can vary widely from a few, to hundreds of thousands. Multicast is a powerful communication model because it not only allows a single transmitter to reach potentially everyone on the network, but it does so in a very efficient and scalable manner. Multicast applications include distance learning, Internet radio and television, distributed interactive simulation, software updates and much more.

With the current IP Multicast service model [29], it is difficult to perform reliable data transfer in a scalable and efficient manner. Traditional error control mechanisms designed for point-to-point applications (e.g., TCP) either break down completely or give poor performance. The difficulties arise because the IP Multicast model requires that all messages sent to a multicast address reach all receivers subscribing to that address. While this model is simple, elegant and works well for data transfer, it is not suitable for data recovery. The reason is that losses in a multicast environment are local, i.e., they affect only part of the multicast tree. Attempting to recover data lost locally by global means leads to several problems, which we summarize below.

Implosion: Implosion is a problem that occurs when the loss of a packet triggers simultaneous messages (requests and/or retransmissions) from a large number of receivers. In large multicast groups, these messages will swamp the sender if unicast, or even the entire group if multicast.

Exposure: Exposure is a problem that occurs when recovery-related messages reach receivers which have not experienced loss. This is mainly due to the lack of “fine-grain” multicast in the existing model.

Recovery latency: The latency experienced by a member from the instant a loss is detected until a reply is received. Latency has great implications in application utility and the amount of buffering required for retransmission.

Adaptability to dynamic membership changes: This is a measure of how the efficiency (in terms of loss of service, duplicate messages and added processing and/or latency) of error recovery is affected by changes in the group topology and membership. In large dynamic multicast groups receivers may join or leave at random intervals. Thus error control mechanisms that make assumptions about receiver population and/or location are not suitable for such groups.

Current solutions solve some, but not all of the above problems. For example, some solve implosion at the expense of latency and exposure, while others solve implosion, exposure and latency, but do not adapt well to membership changes. Nearly all existing solutions are significantly more complicated than unicast solutions, not only in terms of complexity but also in terms of state, because they require receivers to maintain topology-related information about other receivers.

1.4.1. Our Solution: Light-weight Multicast Services (LMS)

LMS [34] is motivated by the observation that the current multicast model offered by IP Multicast is not well-suited for scalable reliable multicast. The challenge, however, is to enhance the current model by adding minimal complexity, while maximizing the gain. We believe we have met this challenge. The key feature of our solution is the separation and isolation of the forwarding and error control components; the forwarding component is assigned to the routers (where it can be implemented most efficiently), while the error control component stays at the receivers. The result is that the multicast model need only be enhanced with new forwarding functionality. Freeing the receivers from the burden of topology-related operations, reduces the complexity of multicast error control to a level comparable to unicast. Like other solutions, our solution assumes some degree of collaboration among receivers, to maximize efficiency; however, it still performs well with minimal collaboration. While our solution requires modifications to the current IP multicast model, these are, however, far less complex than modifications proposed by others, including a major router vendor.

Briefly, our scheme works as follows: first, the routers create an implicit receiver hierarchy by selecting one of their outgoing interfaces as the parent for the remaining outgoing interfaces (the incoming interface is the one leading to the sender). Second, when detecting loss, all receivers immediately multicast requests; these are steered by routers to the parent interface, except for the request from the parent interface, which is forwarded upstream. This allows only one request to escape to the next level router. Third, before steering requests to the parent interface, a router marks its location by inserting its address in passing requests; we call this location the *turning point* and it identifies the root of the subtree that sent the request. Since the turning point is carried in the request, routers need not remember any state. Finally, the first parent that receives the request and has the lost data, provides the router at the turning point with a retransmission, which the router multicasts to the appropriate subtree.

Note how our scheme addresses all problems listed earlier: implosion and exposure are reduced because the tags help routers form a virtual recovery tree, thus localizing recovery between parents and children. Maintaining the recovery tree at the routers is easy, because routers already have the necessary topology information. In addition, the tree adapts instantly to both membership and routing changes, since routers ensure that the recovery tree always tracks the multicast routing tree. Recovery latency is minimized because messages (requests, retransmissions) are sent immediately. Finally, the separation of forwarding and error control eliminates all topology state from the receivers (e.g., round-trip-time, back-off timers, parent/child assignment), along with the overhead associated with such state.

We have evaluated our solution in two ways: with simulation over topologies generated by an Internet topology generator, and with implementation in the kernel of NetBSD Unix. Our results have shown that our solution performs as well or better than existing solutions (including other solutions that change the multicast model), while being highly scalable and adaptive. We have shown that not only is receiver complexity significantly reduced, but performance has actually improved significantly: implosion is essentially eliminated, exposure is kept at very low levels (below 1% in most cases), recovery latency is better than unicast (30 - 60% of unicast latency), and adaptation to membership changes occurs instantly. We have added LMS into the kernel of NetBSD and have shown that very little new code is required. The additional code is simple and its forwarding overhead is actually less than regular multicast forwarding.

1.4.2. Contributions

It is important to understand the broader impact of our research on multicast. In our work we have taken a problem, namely multicast error control, and have shown how to decompose it into two sets of components: a set of forwarding services that the routers support, and a set of error control operations which are executed by the receivers. We have shown that this not only results in minimal changes to the multicast model, but also helps eliminate much of the complexity from the receivers.

This approach has provided us with a vantage point from where we can address other difficult multicast problems, such as congestion control and more.

The specific contributions of our work are as follows:

IP Multicast Model: We have defined enhancements to the current IP Multicast model to realize a highly scalable, efficient and light-weight error control mechanism.

IP Multicast Services: We have clearly defined a set of new forwarding services, and shown how to separate forwarding and error control functionality. We have defined the new state that must be added to forwarding entries at the routers, along with mechanisms to maintain and update the state.

Reliable multicast protocol: We have designed and implemented a scalable reliable multicast protocol that uses the above services.

Networking Code: We have written the appropriate networking code to implement the new forwarding services at the routers and demonstrated its correct operation. We have defined the new IP options required to implement these services.

Operating System Code: We have written the appropriate operating system code to allow applications access to the new router forwarding services, and integrated it with the existing IPC code in NetBSD Unix.

Comparison between RM schemes: We have done a simulation comparison between LMS, SRM and PGM. We also contributed our simulation code to the Network Simulator (*ns*) community. Our simulations for the first time evaluated all three schemes under identical conditions, and using topologies generated by an Internet Topology Generator utility, rather than arbitrary topologies. We have demonstrated that the network-assisted schemes (LMS and PGM) have much better performance than the non-assisted SRM. LMS is a much simpler scheme than PGM; however, we showed that LMS achieves comparable and often better performance than PGM, despite the difference in complexity.

1.5. Dissertation Overview

This dissertation is organized as follows. The next chapter describes our work on interactive CM applications. We present related work, followed by the description of our Retransmission-Based Continuous Media transport protocol. We present the motivation, design, implementation, testing, verification and finally the evaluation of the protocol.

Chapters 3, 4, 5 and 6, present our work on reliable multicast. In Chapter 3 we present some essential background to motivate the problem, followed by related work.

Chapter 4 presents a high-level design of Light-weight Multicast Services (LMS), and a receiver-reliable multicast protocol based on LMS. We present the specification of both LMS and the reliable multicast protocol. We discuss advantages, limitations, and extensions, and describe other potential applications of LMS besides reliable multicast.

Chapter 5 presents a simulation study of LMS, SRM and PGM. For this study we have implemented LMS and PGM in the *ns*, which contains an implementation of SRM, and used an Internet topology generator to simulate all three schemes in a variety of Internet-like topologies.

Chapter 6 presents the implementation of LMS in NetBSD Unix. We precisely describe the changes that are required in the networking code to accommodate LMS, the introduction of new IP options, and give the code that we added to implement LMS.

Finally, in Chapter 7 we conclude and discuss future work.