18th ACM Symposium on Operating System Principles (SOSP '01)

Mesh-Based Content Routing using XML

Alex C. Snoeren, Kenneth Conley, and David K. Gifford
MIT Laboratory for Computer Science
Cambridge, MA 02139

{snoeren, conley, gifford}@lcs.mit.edu

Abstract

We have developed a new approach for reliably multicasting time-critical data to heterogeneous clients over mesh-based overlay networks. To facilitate intelligent content pruning, data streams are comprised of a sequence of XML packets and forwarded by application-level XML routers. XML routers perform content-based routing of individual XML packets to other routers or clients based upon queries that describe the information needs of down-stream nodes. Our PC-based XML router prototype can route an 18 Mbit per second XML stream.

Our routers use a novel Diversity Control Protocol (DCP) for router-to-router and router-to-client communication. DCP reassembles a received stream of packets from one or more senders using the first copy of a packet to arrive from any sender. When each node is connected to n parents, the resulting network is resilient to $(n-1)$ router or independent link failures without repair. Associated mesh algorithms permit the system to recover to $(n-1)$ resilience after node and/or link failure. We have deployed a distributed network of XML routers that streams real-time air traffic control data. Experimental results show multiple senders improve reliability and latency when compared to tree-based networks.

1 Introduction

Our research is motivated by an interest in highly reliable data distribution technologies that can deliver information to end clients with low latency in the presence of both node and link failures. Low latency can be crucial for certain data that are extremely time critical. For example, real-time trading systems rely upon the timely arrival of current security prices, air-traffic control systems require up-to-the-second data on aircraft position and status, and gaps or delay in live network video and audio feeds can be distracting. In such environments, even a sub-second pause in a data feed while a delivery network retransmits or reconfigures may be unacceptable. Recent studies have shown the Internet recovers from failures on a much slower scale, often on the order of minutes [2, 20].

We observe that the achievable latency of a reliable data stream is bounded by the packet loss-recovery mechanism. Packet losses can be handled by retransmission or redundant coding. Retransmission methods limit recovery time to the round-trip delay between communicating nodes. In order to avoid retransmission in the face of loss redundant data must be sent.

This work is based upon the assumption that, in certain cases, the value of reliable and timely data delivery may justify increased transport costs if the cost increase allows us to meet a desired reliability goal. Systems often try to avoid the delay penalty by using loss-resistant coding schemes which encode redundant information into the data stream. We extend this redundancy to network delivery paths and senders. Recent work in overlay networks has shown that multiple, distinct paths often exist between hosts on the Internet [2]. We attempt to leverage these redundant network links. While some may consider this additional bandwidth wasteful, we believe the system described herein presents an interesting and elegant method of utilizing additional network resources to achieve levels of reliability and latency previously difficult to obtain.

Our basic approach is to construct a content distribution *mesh*, where every node is connected to n parents, receiving duplicate packet streams from each of its parents. The value of n is a configuration parameter that is used to select the desired trade-off between latency, reliability, and transport costs. By maintaining an acyclic mesh, this approach guarantees that the minimum cut of the mesh is n nodes or independent links. Thus, a mesh is resilient to $(n-1)$ node or $(n-1)$ independent link failures (we say $(n-1)$ resilient) without repair. If a mesh failure occurs, recovery algorithms restore the mesh to $(n-1)$ resilience in a few seconds.

Our architecture is based upon an overlay network that transports XML streams. An *XML packet* is a single independent XML document [7]. An *XML stream* is a sequence of XML packets, and each XML packet in a stream can have a different document type definition (DTD). When clients join an overlay network they specify an XML query that describes the XML packets they would like to receive. It is the job of the overlay network to configure itself to deliver the desired XML stream to a client at reasonable cost given reliability goals. Queries are expressed in a general language such as XQuery [11].

Our overlay network is implemented by XML routers. An *XML router* is a node that receives XML packets on one or more input

This research was supported in part by DARPA (Grant No. F30602-97-1-0283). Alex C. Snoeren was supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

links and forwards a subset of the XML packets it receives to each output link. Each output link has a query that describes the portion of the router's XML stream that should be sent to the host on that link. XML routers are components in a distributed publish-subscribe network and implement the selective forwarding of XML packets according to subscriptions described by queries.

XML has a number of advantages over a byte stream for multicast delivery. First, XML permits the network to interpret client data needs in terms of well-defined XML queries. Second, XML packets suggest what logical units of data will be processed together by a client and thus can aid network scheduling. Third, many tools and standards exist for XML making it easy for both the data originator and receiver to build robust applications. Finally, our approach allows applications and databases to push part of their processing into the network fabric. We expect that query languages such as XQuery will become standardized, allowing a single language to be used to describe data requirements. This standardization will permit applications to program our network fabric to deliver the data they need in a simple, consistent fashion.

The primary disadvantage of XML is often thought to be the increased number of bytes required to represent the same information in XML when compared to an application specific encoding. However, our experimental results suggest that conventional data compression eliminates this disadvantage. While an XML stream must be decompressed and recompressed at any router that wishes to do query matching, a router that passes all packets to every client can bypass the XML switch component entirely, and no decompression or compression need be performed. Thus, routers can include a fast-path for clients that subscribe to the unfiltered XML stream.

This paper makes three distinct, novel contributions:

- *XML Routing.* To the best of our knowledge, we describe the first packet-based network XML router to support arbitrary content routing. We believe that systems for XML routing will be useful in a wide variety of contexts and will be efficient because XML wrapper overhead can be removed by appropriate use of data compression technology.
- *Mesh-based overlay networks.* We describe the first overlay network to use multiple, redundant paths for simultaneous transport of multicast streams. Our mesh approach offers better latency performance than tree-based approaches.
- *Diversity Control Protocol.* We describe a novel protocol that uses source-independent sequence numbers to reliably reconstruct a sequenced packet stream from multiple sources. DCP reduces latency and improves reliability when compared with conventional single-sender approaches.

The remainder of this paper describes our current XML routing infrastructure in the following sections:

- Previous work (Section 2)
- Architecture of our XML routing system (Section 3)
- Mesh algorithms and distribution protocol (Section 4)
- Experimental results and our air traffic control application (Section 5)
- Issues involved in routing XML over a mesh (Section 6)
- Conclusions (Section 7)

2 Previous work

Our work on XML routers and DCP builds on a large body of past work in reliable multicast and overlay networks. We consider related work in four areas: reliable multicast, overlay networks, redundant coding and transmission schemes, and publish-subscribe networks.

2.1 Reliable multicast

Reliable multicast systems send a stream of packets to a set of receivers. Reliable multicast systems are often built on IP Multicast [3]. IP Multicast packets are duplicated by the network layer as late as possible to minimize the network resources consumed to deliver a single packet to multiple receivers. Acknowledgments are required to make IP Multicast reliable. If a packet is damaged in transmit or is lost, either a receiver will send a negative acknowledgment to the sender [14, 22, 27, 41, 43], or the lack of a positive acknowledgment from a receiver will cause the sender to retransmit [17, 22, 43]. Express [15] is a single-source multicast system that simplifies IP Multicast in the face of multiple data sources but is still integrated with the network fabric.

Of particular note is RMX [12], which shares similar goals with our work. RMX provides real-time reliable multicast to heterogeneous clients through the use of application-specific transcoding gateways. For example, it supports re-encoding images using lossy compression to service under-provisioned clients. By using self-describing XML tags, our architecture allows similar functionality to be provided in a general fashion by having clients with different resource constraints subscribe to different (likely non-disjoint) portions of the data stream.

2.2 Overlay networks

An *overlay network* is a virtual network fabric that is implemented by application level routers that communicate with each other and end clients using normal IP network facilities. Overlay networks typically use reliable point-to-point byte streams, such as TCP, to implement reliable multicast. The goal of an overlay network is typically to provide increased robustness [2, 35] or additional, sophisticated network services, such as wide-area stream broadcast [16, 30, 37], without underlying network assistance. In fact, network operators may be unaware that such services are running on their network.

One advantage of building our network as an overlay is that it is easy to modify and deploy without the cooperation of network providers. We have adopted the use of overlay networks as an effective way to build a robust mesh that can effectively route XML packets. End-system-multicast [13] is an overlay-based multicast system that constructs meshes during spanning tree discovery but does not use redundant mesh links for information delivery.

2.3 Redundant encoding and transmission

Loss-tolerant encoding schemes (often termed erasure, tornado, or forward error correcting (FEC) codes) use redundant information to support the reconstruction of a data stream in the face of a certain amount of packet loss [25]. For example, in Digital Fountain's Meta-Content protocol [9] packets are encoded to allow a receiver to recover a data stream even if a certain fraction of Meta-Content packets are never received.

Our approach to redundancy is based on sender and channel diversity while loss-tolerant encoding schemes typically use only packet diversity [9, 33]. We use channel diversity because experimental data suggests that Internet packet errors are highly path dependent [2, 29, 35]. We use sender diversity because in single-sender systems a sender failure is likely to cause a stream gap during recovery [30]. Based on these assumptions, we believe that, with appropriately configured levels of mesh redundancy, sender and channel diversity can provide lower loss rates and latency than packet diversity, albeit at a higher cost.

Several previous systems have leveraged channel diversity, sender diversity, or both in an end-to-end fashion. Dispersity routing [24] and IDA [31] split the transfer of information over multiple network paths to provide enhanced reliability and performance. Simulation results and analytic studies have shown the benefits of this approach [5, 6]. In addition, tornado codes have been suggested to combine parallel downloads to improve reliability and performance [8]. Application-level dispersity routing, IDA, and parallel downloads use multiple network paths but do not provide for any loss recovery along a single path within the network fabric. Our use of application-level routers allows us to perform loss recovery inside of the network fabric and, thus, improve loss resilience. Further, the block encoding scheme used by Digital Fountain may add additional latency during decoding. We discuss our loss resilience results in Section 5.

2.4 Publish-subscribe systems

Publish-subscribe networks, such as Tibco's TIB™Rendezvous [28], Elvin4 [36], Siena [10], Gryphon [4], and XMLBlaster [1] permit receivers to specify the portion of a data stream that they would like to receive. Receivers typically subscribe to messages using a query that summarizes their interests. Streams may be encoded such that the same content, but in varying levels of fidelity, may be requested by each client [26, 42]. Siena and Gryphon both provide distributed implementations of singly connected graphs for information distribution, but neither provides XML-based routing.

XMLBlaster [1] is a publish-subscribe system based on XML packet streams, but it only permits queries over a specific header field. Our semantics permit queries over any field in an XML packet. We believe that the overhead of making each XML packet a fully formatted document is a small price to pay for the resulting flexibility and rational query semantics. This is especially the case when data compression causes the markup overhead in each XML packet to become negligible. To our knowledge, no existing stream-based publish-subscribe network uses redundant meshes for reliability or performance enhancement.

3 Resilient mesh networks

As shown in Figure 1, a typical overlay network for routing an XML stream contains one or more root routers (R1-R2), a variable number of internal routers (I1-I3), and a variable number of edge clients (C1-C3). Root routers are the origin of data and are assumed to have independent means of generating their XML stream. Internal routers receive the XML stream from their parent routers and forward elements of the stream to their children as required. Clients connect to routers and provide a query that describes the portions of the XML stream they would like to receive.

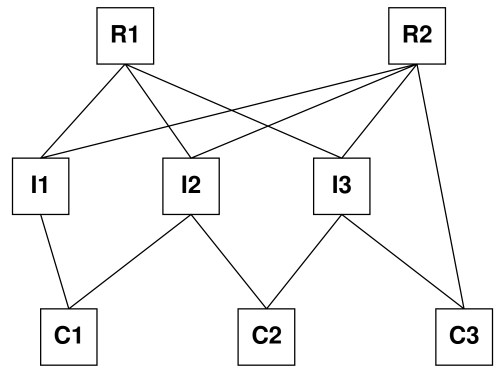

Figure 1: A mesh network comprising root routers (R1-R2), inner XML routers (I1-I3) and clients (C1-C3).

The content carried by routers in a mesh can be statically or dynamically configured. Typically, with static configuration the internal routers carry all of the XML packets available from the root routers. Thus, with a static approach to content configuration clients have a wide choice of routers that can service their request without reconfiguration delay. Unfortunately, such a mesh requires a fixed bandwidth capacity throughout. We can leverage the expressive power of XML to better control bandwidth usage.

Dynamic content configuration allows a router to carry only the packet stream necessary to service its children. In this case, a router disjoins all of the queries it receives from its children and forwards the resulting query to its parent routers. Note that since each router combines the queries of each of its children when subscribing to its parent routers, each router need only store queries for its immediate children. This results in significant bandwidth savings when clients are uninterested in the full contents of the data stream. The disadvantage of this scheme is that the mesh may not have a sufficient number of routers that currently carry the traffic needed by a node searching for an additional parent during mesh construction or repair. If a client requests information that is not available in that portion of the mesh, there will be a delay while the mesh readjusts to supply the required information although this additional startup delay is tolerable in most situations. During reconstruction, the data should be available from the current parent set. During initialization, it simply adds a slight additional startup latency.

Clients wishing to join an $(n-1)$-resilient mesh perform four distinct operations: (1) composing an XML query that describes the data desired, (2) contacting n existing routers that can service the query, (3) sending these n routers the XML query it has composed, and (4) receiving the XML stream described by the query. One particular algorithm for discovering routers is described in Section 4.

Each router includes a query table that describes the portion of the XML stream each of its children wishes to receive. Thus, each router functions as a splitter that takes a single XML stream and refines it for each child. Often a child is only interested in a subset of a stream (such as all air traffic landing in Seattle). Expressing this desire to routers saves last-mile bandwidth and end-host processing.

Our architecture also admits XML *combining routers*. A combining router merges XML feeds from different sources into a single XML feed. This can be accomplished by simply forwarding unmodified packets from both sources, or it can involve application-specific processing. For example, in our air traffic control application we are investigating merging our XML stream of air traffic data with an XML stream of runway conditions.

We will call a node k*-resilient* when any combination of k other nodes and independent links in the mesh can fail and the node will still receive its XML stream. We say a mesh is k-resilient when all of its nodes are k-resilient. The level of resilience in a network can vary according to the needs of end clients. Although we heretofore have described a uniform mesh architecture with a fixed router fan-in of n, it is entirely possible to build meshes with non-uniform fan-in. The only constraint is that in order to assure a desired level of resilience all the way to the root, the resilience of a child's parents must be equal to, or greater than the child's desired resilience. For example, one could build a core network that is 2-resilient, and certain clients could choose to be 1-resilient. The failure of a core router will most likely reduce the resilience level of many peripheral routers and clients until the mesh can reconfigure, but the mesh will continue to provide service to all clients except those clients directly connected only to the failed node. Thus, in certain circumstances, it may make sense to improve the resilience only of key portions of a network that provide service to many clients. We are investigating issues surrounding optimal mesh configuration.

4 Algorithms and protocols

An XML router implements three key algorithms and protocols:

- *XML router core*. The XML core is the engine that receives and forwards packets according to queries. Its job is to efficiently evaluate each received XML packet against all output link queries.
- *Diversity Control Protocol (DCP)*. DCP implements resilient mesh communication by allowing a receiver to reassemble a packet stream from diverse sources.
- *Mesh initialization and maintenance*. A set of algorithms automatically organizes routers and clients into a mesh and repairs the mesh when faults occur.

4.1 XML router core

Figure 2 shows the internal structure of an XML router. An XML router consists of three major components:

- An *input component* that acquires XML streams for presentation to the XML switch. The input component is responsible for maintaining DCP connections to the parents of the router and implementing the mesh initialization and reconfiguration algorithms. In addition, the input component implements data decompression. Our input component can also connect to TCP XML streams for compatibility.

  Although, in many instances, the input component will acquire a single XML stream for routing, an input component could connect to distinct meshes and merge multiple XML streams for routing. The input component is also responsible for forwarding the disjunction of its link queries to its parents.

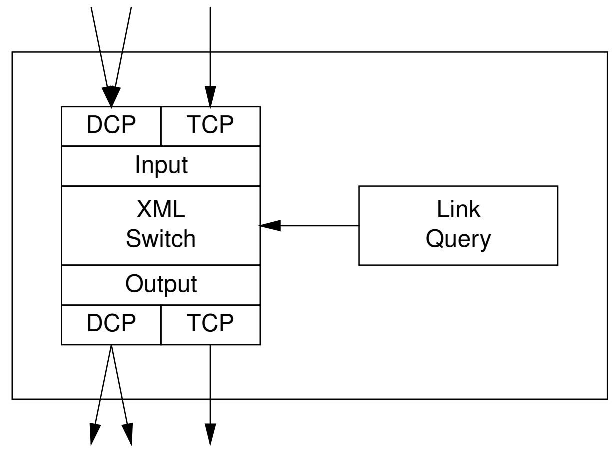

Figure 2: The internal architecture of an XML router comprises the input component, XML switch, and output component. Output link queries control XML packet forwarding.

- An *XML switch* that compares received packets against link queries, and forwards matching packets to the requesting links. An efficient XML switch attempts to combine distinct link queries into a single state machine that matches all of the link queries in a single pass over an incoming packet.
- An *output component* that forwards packets on output links using DCP. In addition, the output component is responsible for handling join requests from prospective children and implements link-based data compression. Our output component additionally can produce TCP XML streams for potential compatibility with non-DCP children.

4.2 Diversity control protocol

The *Diversity control protocol* (DCP) is so named because of the inherent sender diversity that it implements. The essential idea behind DCP is that a receiver can reassemble a packet stream from diverse senders. In DCP, the same stream of packets is sent to a receiver by multiple sources where the desired level of redundancy may vary between nodes in a mesh. As shown in Figure 3, a DCP receiver reassembles the packet stream using the first error-free packet received from any source.

4.2.1 Sequencing

Proper in-order packet stream reassembly requires that all DCP packets be assigned identifiers that admit a total ordering and that the total ordering must be known to the participants. DCP further requires identifiers obey the following invariants:

- For a given content stream, packet identifiers must be associated only with packet content and not be sender specific. This allows receivers to properly reassemble a stream based upon identifiers alone.
- Since packets may travel through a variable number of intermediate router hops, the identifiers with a particular stream must be selected at root routers and remain identifiable

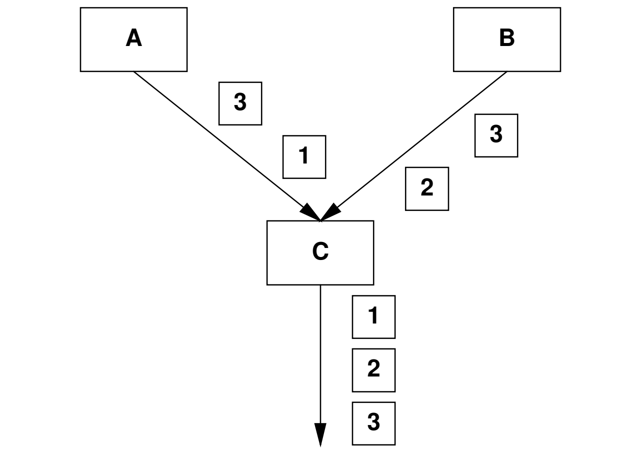

Figure 3: The Diversity Control Protocol (DCP) reassembles a packet stream from diverse senders.

throughout the mesh. Thus, the set of root routers for a particular stream must originate the same packet stream and assign the same identifiers to the same packets. This must be true even if the root routers do not generate the stream at precisely the same time or rate.

- Receiver identifier processing must admit gaps. Since intermediate routers may not forward packets containing content that was not requested by a particular receiver, the identifiers of these packets will not be received.

Our approach to satisfying these three invariants is to assign a monotonically increasing 32-bit application serial number (AN) to every DCP packet when the packet is created at a root router. Every router that forwards DCP packets maintains the last packet AN sent on each output link. The last AN sent on a link is included in the next packet along with the next packet's current AN number. Including a client-specific previous AN in each packet permits a receiver to reassemble the stream of packets from a sender in the presence of missing ANs. In our application, missing ANs are caused by filtered XML packets.

While routers may remove packets from the datagram stream, DCP itself is a reliable transport protocol. Hence, any missing datagram (as indicated by a hole in the AN sequence chain) will be retransmitted. In order to maintain redundancy invariants throughout the mesh, retransmissions are requested at each hop rather than end-to-end. Similarly, packets are buffered and transmitted in-order at each hop. This ensures that every node can consider each parent an independent source of ordered datagrams. We return to consider the implications of out-of-order forwarding in section 6.4.

DCP currently uses UDP as a transport mechanism to facilitate deployment at the application layer. Distinct DCP streams are currently transmitted on separate UDP ports. In our application, one DCP packet is used to transport one XML packet. This is possible because our XML packets are relatively small. If XML packets do not fit into a single IP packet envelope, an AN could describe both the XML packet number being transmitted and the IP packet within the XML packet. The important invariant is that an AN be based upon the content of a packet and not on when or by whom it was generated.

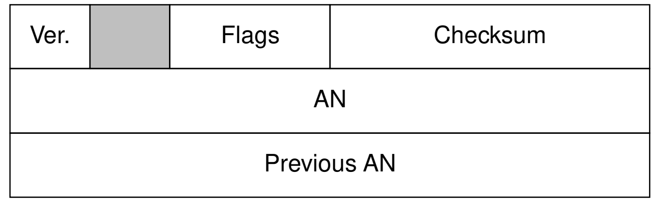

Figure 4: DCP Packet Header

Figure 4 shows the layout of a DCP packet. In addition to the ANs we have already mentioned, a DCP packet includes a 4-bit version number to allow DCP to evolve and a set of 8 bit flags. The flags permit a sender to request an acknowledgment, a receiver to send an acknowledgment or request a retransmission, and for the exchange of keep-alive and connection-establishment and tear-down information. The entire packet is covered by a 16-bit checksum which may be optionally disabled if encapsulated in UDP or when carrying streams insensitive to corruption.

While our use of DCP is as a datagram protocol, DCP is equally well-suited for the transmission of byte streams. A bit in the flags field is used to indicate that DCP is operating in stream mode. When used as a stream protocol, the AN simply refers to the sequence number of the first byte of the datagram, as in TCP. Similarly, the previous AN refers to the last byte of the previous packet in stream mode. Note that this construction allows for fragmentation or reframing of DCP packets if desired, albeit at the expense of additional complexity and buffering at the receivers. Additionally, if multiple root servers are in use each server must take care to sequence the data identically. Datagram and streaming mode cannot be used during the same DCP connection.

4.2.2 Retransmission

When a receiver joins multiple DCP senders, it waits for the first packet to arrive from any one of the hosts and uses the AN of this packet as its current AN. Packets that are subsequently received with a lower AN than the current AN are discarded and packets that are received with an AN in the future are buffered. A packet with the current AN in the previous AN field is considered the next packet in the reassembled stream and the current AN is updated. If a receiver does not receive an appropriate packet after a fixed interval, it sends a negative acknowledgment (NACK) to all senders with its current AN. This retransmission is sent only to the receiver requesting it. In a fashion similar to TCP's fast retransmit, a NACK is generated after a much shorter timeout if a packet with a subsequent AN is received, indicating either a lost or reordered packet. This NACK serves as a request for all senders to retransmit all packets after the receiver's current AN.

Assuming a regular mesh construction (equal numbers of parents and children), the negative acknowledgment process does not suffer from ACK implosion even with high degree. An individual receiver only generates a NACK if an AN is not received from any of its parents. Due to the (assumed) pairwise independence of packet loss between distinct senders and receivers, this probability drops exponentially with degree as discussed in section 5.1.2. Hence, the probability that a sender receives any NACKs at all decreases with increasing degree, avoiding the NACK implosion problem.

Senders transmit packets in order to a receiver and request an acknowledgment from a receiver from time to time. Our current implementation requests positive acknowledgment after a fixed number of packets has been sent. A receiver responds to a request for acknowledgment with an acknowledgment that contains the last AN (or last byte in streaming mode) it has processed. This serves to limit the amount of buffering required at each node and allows for rapid resynchronization of senders and receivers. If the current sender has not yet sent that AN (byte), it squelches its transmissions until after that AN (byte). A receiver can also send an unsolicited acknowledgment to squelch a sender that is behind. In contrast, if a receiver continually fails to respond to acknowledgment requests, or persistently lags behind the sequence space (indicating insufficient bandwidth between sender and receiver), the connection is terminated. The receiver must then reconnect to a new point in the mesh (presumably with a higher-bandwidth link).

4.3 Mesh formation and maintenance

A mesh begins life as a set of root routers that are all capable of supplying an XML stream of interest. We assume that failures of root routers are independent and, thus, each has an independent means of deriving the XML stream. As noted above, however, roots must be uniform in their DCP packetization and sequence number selection. Additional roots may be added to a mesh at any time provided they have a mechanism to synchronize their content stream with the existing root nodes.

Mesh discovery is outside the scope of this document, but one method of distributing the set of root nodes for a particular content stream is through DNS. All of the IP addresses for the root routers for a service could be stored in a DNS address record. For example, `stream.asdi.faa.gov` might be a DNS name that maps to a set of root routers that supply an XML stream of air traffic control data for North America.

4.3.1 *Adding routers and clients*

When a new internal router is added to a mesh, it can either be statically configured with a set of parents or the new router can select its own parents based upon performance experiments. A wide variety of automatic configuration algorithms can be used to form the mesh depending on the particular desires of the node. These may vary widely depending upon whether the mesh is controlled by a single administrative entity concerned with overall characteristics of the mesh such as its resilience or depth, or the new node has a more specific purpose. Clients join the mesh in the same fashion.

Rather than specify a particular algorithm or policy, we admit a host of possibilities by providing a set of mesh primitives that new nodes can use to discover the topology of the mesh and locate themselves within it. Each router supports the following primitives:

- **Join (Q):** A new node is added as a child of the router with query **Q** provided the current node is willing to admit a child with such a query.
- **Children (Q):** The router responds with its children that subscribe to a subset of **Q**. A full child list may be elicited by specifying a query that matches the entire stream.
- **Parents:** The router responds with its parent set.

1. Initialize the set S to be the root routers.
2. For each node in S, send a join request and remove the node from S.
3. If a node accepts the join, add it to the parent set P. If n nodes are in P, quit.
4. If a node declines the join, ask it for a list of its children, and add them to S.
5. If S is not empty, go to Step **2**.

Figure 5: Parent selection algorithm. Each node runs this algorithm to construct an $(n-1)$-resilient mesh.

Using these three operations, it is possible for a new node to completely walk a mesh to determine its optimal location. We note that the optimum location may vary depending on the particular desires of the joining node. We have currently implemented a very simple algorithm for automatic parent selection for a client seeking $(n-1)$ resilience shown in Figure 5.

This simple algorithm seeks to find a set of routers that are closest to the root routers and uses the timing of responses to select among candidates. The algorithm also assumes that potential parents are configured to reject join requests when they are at maximum desired capacity or they do not wish to service a requested query. We contemplate additional research on improved algorithms that are based upon both depth in the mesh and observed packet latency to select optimal parents for a new child.

Routers may refuse to serve as parents for policy reasons, if they are not receiving the portion of the XML feed necessary to service a new child's query, or if they are over-subscribed. If a prospective parent is not receiving part of the feed necessary for a new child, the prospective parent may be configured to push an expanded query up to its parent, thus propagating the information request up the mesh.

4.3.2 *Mesh repair*

Our mesh repair algorithm recovers from parent failures. If one of the parents of a node fails, the node actively attempts to join a new parent. The method used to obtain a new parent is currently identical to that used to obtain initial parents with one caveat. To guarantee that a mesh is acyclic, each router maintains a level number that is one greater than the maximum level of all of its parents. A router's level number is established when a router first joins the network. During mesh recovery, a router will only join parents that have a level number that is less than its own level number. If this is not possible during recovery, then a router must disconnect from all of its children and do a cold re-initialization to return to its desired level of resilience.

Our repair algorithm recovers $(n-1)$ resilience of the mesh if a non-root router fails. As discussed earlier, $(n-1)$ resilience is a fundamental property of any acyclic mesh where each child has n parents. This can be seen by forming an acyclic graph that is a dual of a mesh. In this dual graph each child is represented as a vertex that has directed edges to all of the child's parents. The min-cut of this graph is n vertices or edges if each vertex has out degree n. Thus $(n-1)$ nodes or $(n-1)$ distinct paths can fail and a node will still be connected to a root.

Due to occasional internal node failures, a mesh repaired using our algorithm will have a tendency to flatten out over time as nodes are forced to select parents with lower level numbers during each repair process. If the mesh structure is to serve extremely long-running streams, it may be necessary for nodes to occasionally remove themselves from the mesh and select an entirely new location in order to preserve the depth of the mesh and prevent overloading of root nodes. We have not yet explored efficient algorithms for determining when to start this process.

5 Evaluation

We have developed two separate implementations of our XML router. Our full-featured, multi-threaded Java implementation uses DCP for router-to-router and router-to-client communication. We have also implemented a prototype high-performance router based on Click [19]. The goal of our Java implementation was to adequately support our air traffic control application and it does not attempt to maximize absolute performance. In contrast, the Click router attempts to achieve production-grade performance using freely available XML parsing technology. Below, we report our experiences with both routers. We are mainly interested in understanding how routers will behave in a mesh under varying configurations. Thus, our evaluation focuses on the effects of mesh redundancy on DCP reliability and performance. We also provide performance results from our Click-based XML router.

5.1 DCP performance

The Diversity Control Protocol has several attractive features independent of the format of the data stream. In particular, DCP-based meshes can achieve substantially lower effective loss rates and latency than tree-based distribution networks. Further, the redundancy can be utilized to absorb unexpected decreases in the link capacity between nodes. In this section, we quantify these effects using our Java XML router. All results presented in this section represent the average of several experiments each consisting of 1000 to 10000 XML-encoded ASDI packets.

5.1.1 *Experimental design*

The experimental setup consisted of four 600MHz PIIIs, two running Linux 2.2.14, and two running FreeBSD 4.0. Each machine used 100Mbit Intel EtherExpress Pro100 Ethernet controllers and 128 Mbytes of memory. The roots and all intermediate XML router nodes were run on one Linux machine using Sun's JDK version 1.3. The XML client node was run on the other Linux machine, also with Sun's JDK version 1.3.

For each experiment, the root node received an XML feed containing a 1Kbyte per second substream of the live ASDI flight data described in the following section. While each node in our experimental topology requests the entire test XML stream, it does so by specifying an XML query predicate, hence each packet is parsed by the intermediate nodes as part of the forwarding process. The intermediate nodes connect to the root over the loopback interface, so there was no packet loss. The desired link loss rates between intermediate and client nodes were obtained by routing each DCP connection through one of the FreeBSD machines which passed the packets through an appropriately configured Dummynet [32] tunnel.

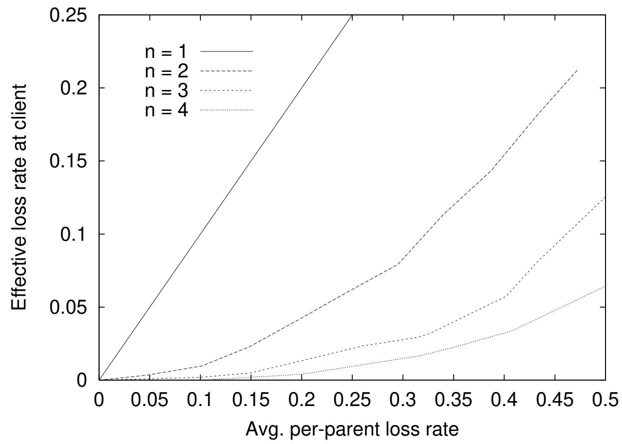

Figure 6: Loss rates experienced by a client as a function of individual parent loss rates and the number of parents.

The one-way latency between the client and parent nodes was negligible (0.1ms) with respect to the millisecond granularity of the Java-based timing mechanisms we used to measure packet latency. Variability in the observed latency of XML packets can be attributed both to the inherent non-uniformity of XML parsing times and to thread scheduling uncertainties of the JDK.

5.1.2 *Redundancy reduces loss exponentially*

We assume that packet loss is independent across parents. This assumption is false if a problematic portion of the communication path to a set of parents is shared. In our ideal model, if each parent has an identical loss rate, p, a node with n parents should expect a combined loss rate of p^n. Figure 6 verifies this experimentally, showing the DCP loss rate experienced at clients with 2, 3, and 4 parents where the loss rate at each parent is independently identically distributed (i.i.d.) with uniform probability p varying from $[0, 0.5]$.

For a traditional tree-based distribution network, the loss rate experienced at the client corresponds directly to the loss rate of its parent. The graph shows, however, that a mesh topology is able to provide acceptable delivery rates over even extremely lossy channels. A node with four parents can expect a loss rate of less than 5% even if each of the parents individually experiences a loss rate of up to 45%.

Most Internet links do not experience extremely high loss rates. In fact, typical long-term average loss rates are on the order of 2–5% with substantially higher burst rates [29]. In such cases, a mesh with $n = 2$ still limits the loss rate at the client to substantially less than 1%. Decreased loss rate is not the only gain from multiple parents, however.

5.1.3 *Latency*

Even in cases where acceptable loss rates can be provided by a tree-based network (reliability may be assured by retransmission), significant improvements in latency can be achieved by increasing redundancy. In DCP, loss is not detected until the receipt of a later packet since each individual packet is not acknowledged. However,

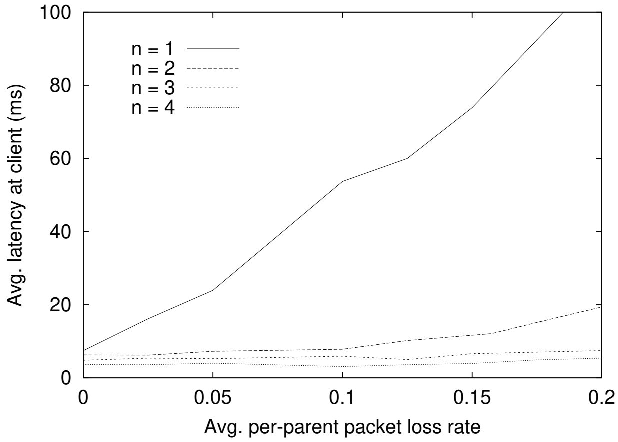

Figure 7: Average per-packet latency from root to client as a function of individual parent loss rates and the number of parents.

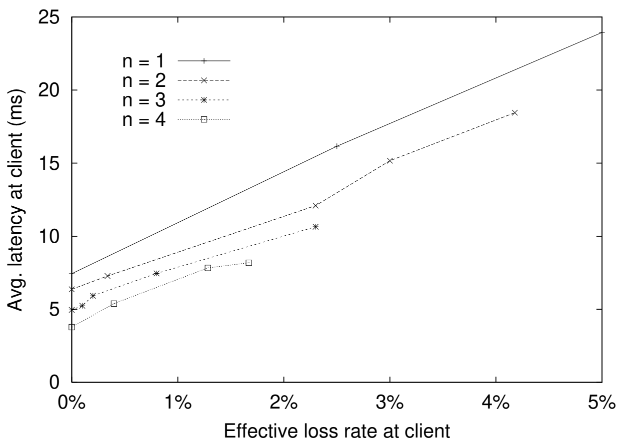

Figure 8: Average per-packet latency from root to client for a range of *effective* loss rates. Effective loss rates for each level of redundancy are taken from Figure 6.

because we expect the Internet to reorder packets [29], a packet is not assumed lost until some time after its successor arrives (currently 5ms).

In order to magnify the effects of retransmissions in a LAN environment with short round-trip times, our test XML feed was specifically constructed to have relatively long inter-packet intervals. In our experiments, a single retransmission adds approximately 300ms to the latency for the lost packet. In practice, streams are likely to have a shorter inter-arrival period but longer RTTs, resulting in a similar effect. As can be seen in Figure 7, packet loss significantly impacts the average packet latency at the client for non-redundant configurations. Meshes with higher levels of redundancy perform much better.

A redundant topology performs even better than the effective loss rate of Figure 5 suggests. This is because clients with multiple parents use the *first* copy of each XML packet they receive. In general, the expected minimum of multiple samples from any distribution

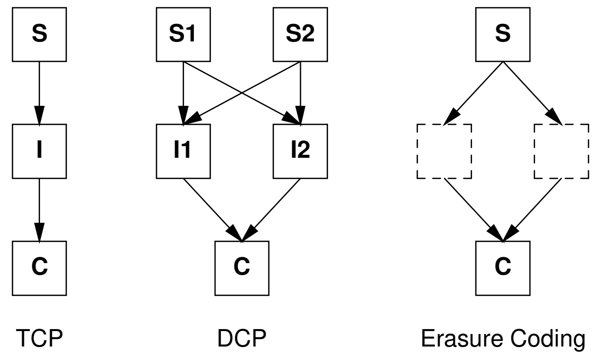

Figure 9: Experimental multi-tier topologies. A two-hop TCP path with a TCP splice in the middle, a 1-resilient DCP mesh of depth two, and an erasure code using two disjoint paths of length two with simple forwarding at the intermediate nodes. The loss rates on all links is identical.

is guaranteed to be at least as good as expected value of a single sample. Figure 8 shows the average latency for several levels of redundancy with respect to *effective* loss rates.

5.1.4 Multi-tier meshes

The improvement in latency and throughput becomes even more dramatic as the the depth of the mesh increases. We demonstrate this by measuring the throughput performance of a two-tier mesh using both DCP and TCP and analytically derive the expected performance of a carousel-based erasure code scheme. As shown in Figure 9, our experimental 1-resilient DCP mesh has five nodes: two servers delivering identical streams, two intermediate nodes, and one client. In the case of TCP, the mesh has only three nodes: a server, client, and one intermediate node that splices the two separate TCP connections. In both cases, the client, server, and intermediate nodes are connected with point-to-point Ethernet links. We analyze the performance of erasure coding over a hypothetical topology consisting of a server and client connected by two disjoint, two-hop paths. The nodes in the middle simply forward packets and, unlike TCP and DCP, do not request retransmissions of lost packets.

In our experiments we used Dummynet to limit the bandwidth of each link to 75Kbits per second and set the server to transmit data in 262-byte bursts at a rate of 19Kbits per second—significantly under link capacity. Each link in the mesh has a one-way latency of 10ms. Figure 10 shows the throughput observed at the client as the loss rate is adjusted for all links uniformly. Note that TCP's throughput drops rapidly as the loss rate increases. The redundant links are of no use to TCP as a duplicate TCP connection on a redundant path would suffer the same fate. DCP, on the other hand, is able to utilize both links at the same time to provide successful transfer at much higher loss rates.

We note that in the case of multi-hop networks with sufficient bandwidth, DCP outperforms carousel-based erasure coding techniques such as those used by Digital Fountain [9]. Such schemes do not retransmit lost packets. Instead, they encode the data stream at a fixed rate using an erasure code which enables any lost packets to be recovered by simply receiving an additional number of encoding

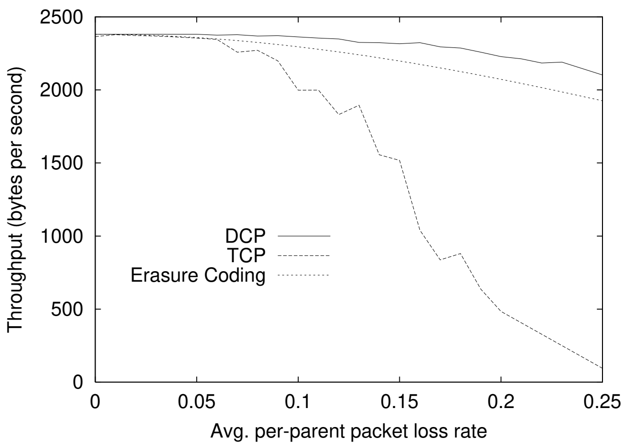

Figure 10: Observed throughput of a two-tier mesh with uniform link loss rates using both 1-resilient DCP and TCP. The stream is served in 262-byte chunks at a rate of 2381 bytes per second. DCP downloads utilize two parents at each tier while TCP can support only one at each tier. We also plot the expected performance of a simple carousel-based erasure code using two disjoint, two-hop paths.

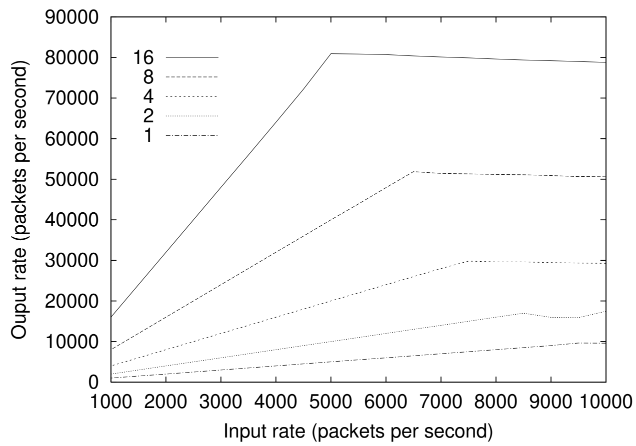

Figure 11: Forwarding rates for a Click-based XML router with a varying number of children. Each child requested the entire input XML stream specified through a trivial XPath expression.

packets. A maximum-distance-separable erasure code requires that the client simply receive as many packets as comprised the original data stream, regardless of which packets they are. In practice, many codes (including those used by Digital Fountain) are not quite maximum-distance-separable, requiring a few additional packets.

Because carousel erasure coding is typically deployed end-to-end with no retransmissions within the network, the loss rates at each hop are cumulative. Whereas, DCP reassembles the stream and retransmits a full set of redundant packets at each tier of the mesh. A naive carousel-based distribution network could support applications such as ours by ensuring each packet contains all the data necessary to decode the current input packet plus any additional redundant data required to support the erasure code.

A carousel erasure code can utilize redundant links by sending an encoded version of each data packet down all available links. Hence, we can calculate an upper bound on the performance of such an erasure code by assuming only one packet of any encoding set must be received. Given a distribution network with n separate paths, each comprised of l hops with link loss rate p, it is easy to see that each path successfully delivers the packet with probability $(1-p)^l$. In the best case, each data packet can be successfully decoded by the client if only one of the encoding packets is received, which occurs with probability $1-(1-(1-p)^l)^n$. To recover lost packets, the client must receive additional encoding packets which are lost with the same probability. Hence, the throughput of such a scheme can be computed by simply multiplying the input data rate by the effective reception rate. Using this formula, Figure 10 shows the expected performance of a sufficiently low-rate maximum-distance-separable erasure code over the two-tier topology shown in Figure 9.

5.2 XML routing performance

Figure 11 shows the forwarding rates achieved by our Click XML router installed as a kernel module. We measured the forwarding capacity by generating a constant stream of identical UDP packets containing a 262-byte XML-encoded ASDI flight update (similar to the one in Figure 12) at a fixed rate and sending them to our XML router. The router parses each XML packet, applies the appropriate child predicates, and forwards the packet to the children with matching predicates.

The tests were conducted on an 800Mhz dual-processor Intel PIII in uni-processor mode with two Ethernet controllers: an on-board Intel EtherExpress PRO 100Mbit/s PCI controller and an Intel PRO/1000 Gigabit Ethernet PCI card. We use uni-processor mode because our XML parsing code is not known to be SMP-safe. Packets were received on the 100Mbit interface and forwarded out the Gigabit interface. Because Click does not support polling on the 100Mbit controller packet input was interrupt driven.

The maximum loss-free forwarding rate varies with the number of children. Additional children add processing overhead for additional link queries. With only one client and a simple query expression, our implementation is able to forward slightly more than 9,000 262-byte packets per second, or about 19 Mbit/second. The more complicated the packet and expression, the slower the forwarding rate.

In order to better understand the impact of query complexity, we timed the XML parser and query evaluator separately. Our Click XML router uses the Gnome XML library, *libxml*. The library provides both an XML parser and an XPath (a subset of XQuery) evaluator. We have made no attempts to optimize the performance of this library. Thus, our measured performance represents a lower bound, and we expect an efficient implementation could perform much better. Table 1 shows the time taken to apply a variety of queries to the same 262-byte sample XML flight packet. We find that complex expressions can take over twice the time to evaluate in the context of an XML packet than simple ones. In all cases, packet processing cost is dominated by XML parsing time.

XQuery	Time (μs)
Parse	64.2
true()	4.5
/flight/flightleg/altitude $>$ *300*	7.1
starts-with(string(/flight/id),'TWA')	8.9
substring-before(string(/flight/flightleg \ /coordinate/lat),'N') $>$ *2327*	14.5

Table 1: Time to evaluate various queries in an 800Mhz PIII. Parsing a standard 262-byte XML flight update requires 64.2μs. The four XQuery expressions shown here select the entire feed, flights above 30,000 feet, Trans World Airlines flights, and flights currently north of the Tropic of Cancer, respectively.

5.3 Experience with air traffic control data

Our original motivation for developing XML routers was to build an infrastructure for distributing and processing real-time air traffic control data. Our laboratory receives the Aircraft Situational Display to Industry (ASDI) [40] feed via a private IP intranet connection to the U.S. Department of Transportation (DOT). The ASDI feed provides detailed information about the state of North American airspace. ASDI messages include information on flight plans, departures, flight location, and landings. A position update is received approximately once a minute for all enroute aircraft. The ASDI feed is directly distributed to most major airlines and is used for collaborative planning between the FAA and the airlines.

The ASDI feed as distributed by the DOT is encoded in ASCII with a specific compact character encoding for each ASDI message type. Efforts were made to make native ASDI messages a compressed format by virtue of their terseness. The ASDI feed is the union of feeds from multiple Air Route Traffic Control Centers (ARTCCs) and countries (USA & Canada). Unfortunately, messages that cannot be parsed using the ASDI specification arise. Thus, at the outset of our work, we built an ASDI feed parser and carefully gathered examples of non-standard messages. We slowly tuned our ASDI feed parser to handle undocumented cases and, today, still find the occasional new message format.

Early in our work, we decided to convert each ASDI message into a corresponding XML packet to create an XML packet stream. This decision guaranteed that all of our applications would have an easy to parse and well-defined XML DTD to consume. Furthermore, it centralized our interpretation of the ASDI feed so that it could be updated as new undocumented message types were identified. We call the XML stream that is created from the ASDI feed the XML ATC stream. Figure 12 shows a sample flight in both ASDI encoding and our XML encoding.

The XML-encoded ASDI packets contain widely varying amounts of data depending on the type of event being reported: flight departures, arrivals, position updates, or other auditing information. Packet size ranges from around 250 bytes to almost 1000 bytes, for an average of about 350 bytes per packet. The stream is diurnal, peaking in the early evening with an average packet inter-arrival time of about 14ms, resulting in an XML data stream of about 25 Kbytes per second.

ASDI Format:

```
153014022245CCZVTZ UAL1021 512 290 4928N/12003W
```

XML Format:

```
<?xml version="1.0"?>
<messageid>153014022245CCZVTZ</messageid>
<flight>
  <id>UAL1021</id>
  <flightleg status="active">
    <speed type="ground">512</speed>
    <altitude type="reported" mode="plain">
      290
    </altitude>
    <coordinate>
      <lat>4928N</lat>
      <lon>12003W</lon>
    </coordinate>
  </flightleg>
</flight>
```

Figure 12: The same flight data formatted in ASDI and XML. In practice, we omit the Message ID field from the XML encoding.

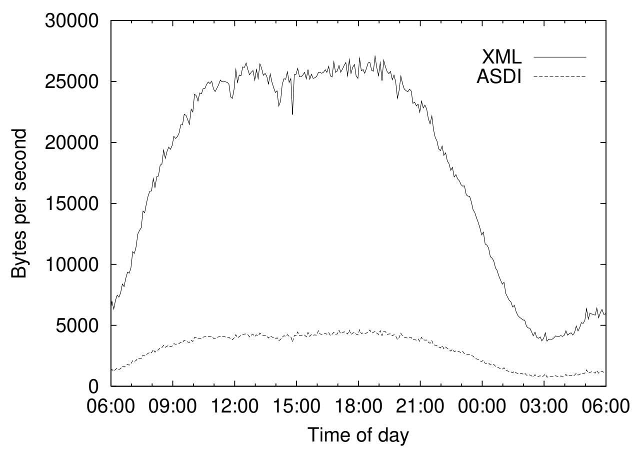

Figure 13: Average bandwidth utilization of the full XML stream and the native ASDI format vs. time of day. In both cases the Message ID field (see Figure 12) is removed from all packets at the root nodes.

Our primary concern in converting the ASDI feed to XML was potential bandwidth bloat. Figure 13 shows the bandwidth of the ASDI feed in both native and XML formats averaged over five minute intervals. Simply converting the feed to XML results in approximately a four-fold increase in bandwidth when compared to the native ASDI feed. We ran both the XML and native ASDI streams through a Lempel-Ziv [21] data compressor. Figure 14 shows the bandwidth of compressed forms of the same streams shown in Figure 13. While the ASDI feed compresses over a factor of two, the XML feed compresses over a factor of 10. The net result is a compressed XML ATC stream is only slightly larger than a compressed ASDI feed and more efficient than the raw ASDI feed.

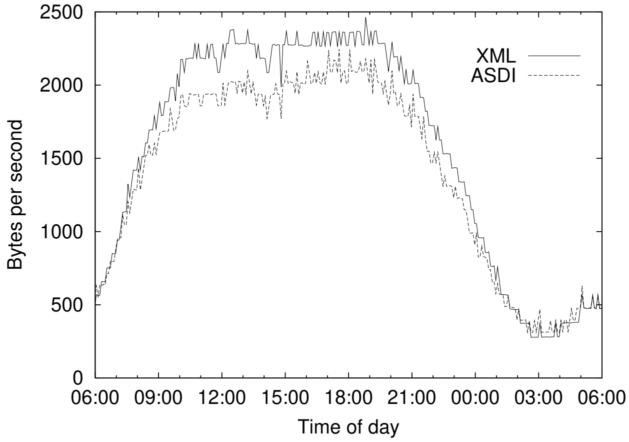

Figure 14: Average bandwidth utilization of the compressed XML stream and the compressed ASDI format vs. time of day. Again, in both cases the Message ID field is removed from all packets at the root nodes.

We have run mesh networks with two root routers and four internal routers but a single root router is more typical. This is because our DOT link is a single point of failure and terminates at our root router(s). We are adding a second communication line to the DOT to connect to their backup ASDI system. This will enable us to have two root routers with independent failure modes. Application serial numbers (ANs) in our ATC application are provided by the FAA. Hence, synchronizing multiple roots is straightforward. Each ASDI message includes a Message ID that we use as the AN of the corresponding XML packet.

Figure 15 shows one interface to the XML ATC stream. This graphical client implements DCP and connects to our XML router mesh. The panel on the left of the screen can be used to control the display of aircraft information. Different colors are used to depict aircraft altitude and the client will coast the position of an aircraft between position updates. For our particular application domain of air traffic control data, XML proved to be a robust and efficient mechanism for distribution. We anticipate adding new types of clients, including an XML stream recorder, to our current system.

6 Discussion

This section considers the strengths and weaknesses of our approach to content routing using XML. While we believe that many of the techniques we developed for our ATC application are widely applicable, we would like to make our assumptions clear.

6.1 AN generation

One difficulty in providing redundant packet sources is providing a standardized sequence space for packet streams that obey the three invariants we outlined in Section 4.2.1. Often, application-specific solutions will present themselves, such as source-derived sequence numbers or time codes. However, in the absence of application-provided sequence numbers, it is necessary to use other approaches, such as cumulative byte counts, block fingerprint matching [23, 38, 39], or other derived metrics.

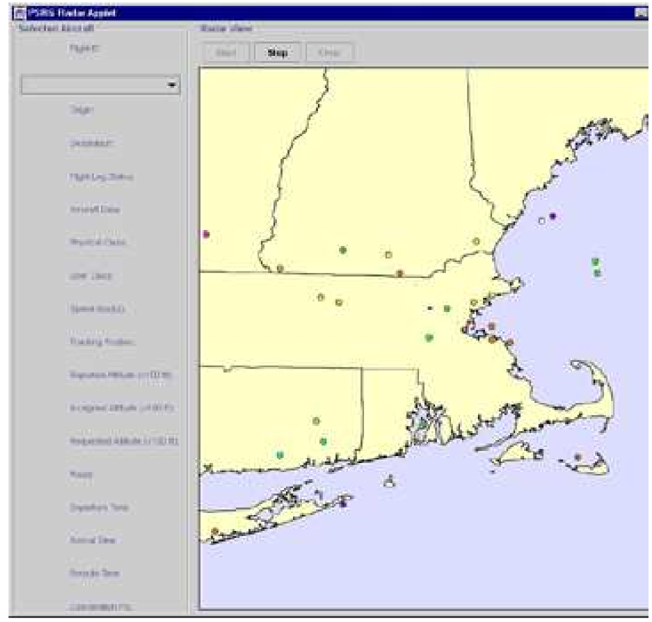

Figure 15: Java-based client for XML ATC stream showing controls and air traffic.

When a combining router merges XML streams, packets in the combined stream must have appropriate ANs. Simply using the ANs from the original uncombined packets for packets in the combined stream will typically not work as packets from different streams will in general have incomparable ANs. One solution is for the combining router to become the root of a new stream and establish its own totally ordered AN space. However, this would create a single point of failure if sequence assignments in this space are not coordinated with another combining router. Another approach is to make AN space partially ordered. For example, the AN space for a combined stream could be a pair of the AN of the source packet in its original stream along with an integer suffix that identifies the source XML stream. Packets with ANs from different source streams would not be sequenced across streams, but a client could recover the ordering of XML packets within each stream.

6.2 Flow control

Nodes are responsible for monitoring the loss rate of streams from their parent and adjusting their predicates appropriately. Limited per-child buffering is available at each node, and clients may be disconnected if they are consequently unable to consume the data stream at an acceptable rate.

The squelching mechanism of DCP allows parents to avoid wasting bandwidth sending packets to a child that the child has already received. If a child is unable to keep up with the long-term average rate of the stream, however, queues will build up and action must be taken. If the client is able to subsist with a smaller subset of the data stream, it may wish to conduct join experiments in order to determine the appropriate XML query for its bandwidth constraints [26, 42]. Otherwise, clients persistently unable to keep up with the data stream will be disconnected by their parents.

6.3 Redundancy

We expect that most mesh networks will use $n = 2$. This level of redundancy allows for single points of failure and allows mesh repair to proceed without stream flow interruption. We expect that as future networks increase in capacity a moderate amount of packet redundancy will be acceptable for high-value streams to achieve specific reliability and performance goals. Secondary storage is often replicated for similar reasons.

We have assumed in our analysis that errors from different parents are independent. This assumption can be violated in numerous ways, but the most likely reason will be shared communication path components from a child to its parents. In addition, network-wide effects, such as distributed-denial-of-service attacks, could cause independent parents to have dependent packet losses. To maximize link independence, we plan to explore using routers in distinct Internet autonomous systems (ASs) and ensuring that last-mile bandwidth is adequate to each AS. In certain applications, it may also be possible to use private intranets to better control error assumptions.

6.4 Router XML stream reassembly

Each of our routers recreates the original XML stream before it is processed by the XML switch. We do this to guarantee that every XML packet is forwarded by every router, to allow a client to ask for retransmissions from any of its parents, and to potentially allow the XML switch to keep stream-dependent state between packets that could be used by queries. The amount of buffering required is bounded by requiring positive acknowledgments as discussed in section 4.2.2.

If XML packets are forwarded out-of-order by an XML switch then a router does not necessarily need to buffer packets or recreate the original sequenced XML stream. This is, indeed, the case in our ATC application, although in our ATC application every XML router does recreate the original XML stream. If an XML router need not recreate the original XML stream, a router could process each received packet independently and would not need to process every packet in an XML stream. In this scenario, a client places increasing reliance upon the redundancy of the mesh to ensure timely delivery of packets that are not received from a particular router. In particular, since all levels may forward out of sequence, the latency induced by a retransmission request from the client may be large. Hence, we have not yet considered how to handle reliable, out-of-order delivery with bounded latency.

6.5 Packet acknowledgments

For asynchronous, variable-bandwidth data streams, packet loss can be detected either by the lack of packet acknowledgments at a sender or by a gap in packet sequence at a receiver. DCP currently relies upon the latter technique. If inter-arrival times are large, per-packet ACKs may be required to provide the appropriate level of responsiveness. Unfortunately, positive acknowledgment schemes admit a well-known implosion problem where the sender is flooded with acknowledgments from each of its children.

While our implementation currently uses unicast UDP to transport DCP packets, DCP could employ IP Multicast where available. The negative acknowledgment system we describe is capable of handling IP Multicast packet losses. If IP Multicast were employed, a DCP output component would send a single packet to an appropriate multicast group of its children based upon the children's queries.

6.6 Dynamic timer adjustment

A robust DCP implementation should be able to automatically adjust its timers to the characteristics of the link between nodes. In particular, the negative acknowledgment timer should be set only long enough to admit observed packet reordering, which clearly depends on the inter-packet arrival of the flow. Being too slow results in poor latency, being too jumpy results in wasted bandwidth. Similarly, several timers relating to mesh liveliness would benefit from automatic refinement. In particular, nodes expect to receive data from their parents every so often. If no data is available in that interval, the parent sends a keep-alive message. A similar mechanism is employed by the parent to insure the continued presence of its children. Clearly the timer should be proportional to the stream data rate, in order to avoid excessive probing. We are currently exploring applying known techniques to these problems [18].

7 Conclusions and future work

This paper presented three key ideas. First, we introduced the idea of XML routers that switch self-describing XML packets based upon any field. Second, we showed how XML routers can be organized into a resilient overlay network that can tolerate both node and link failures without reconfiguration and without interrupting real-time data transport. Finally, we introduced the Diversity Communication Protocol as a way for peers to use redundant packet transmissions to reduce latency and improve reliability.

A wide variety of extensions can be made to the work presently reported, both in protocol refinements and additional functionality. We are actively investigating methods of DCP self-tuning, both for adaptive timers and sophisticated flow control. DCP can also can be used for uninterpreted byte streams. Thus, DCP-like ideas may find application in contexts outside of XML routers. For example, contemporary work on reliable overlay networks (RONs) could use DCP as a RON communication protocol to maximize performance and reliability [2].

Just as secondary storage has become viewed as expendable in pursuit of enhanced functionality and performance [34], we believe that, for certain tightly-constrained applications, network bandwidth across multiple paths may be similarly viewed as well-spent in return for substantial gains in reliability and latency. It is unlikely that multiple disjoint paths with excess capacity will always exist on the last mile to a client. Hence, many installations may benefit from meshes that change to lower levels of redundancy at critical network points such as points-of-presence before last mile cable.

Within the scope of XML routing, our current XML routers could be extended to support.

- More sophisticated XML mesh building and maintenance algorithms.
- Combiners that integrate multiple XML streams for multicast transport as a single stream.
- Using XML routers for duplex communication.

- Other XML network components, such as stream storage and replay.
- Transcoding XML routers that produce output packets that are derivatives of input packets, based upon client queries.

Even in its current form, however, we believe our architecture demonstrates XML is a viable mechanism for content distribution, providing a natural way to encapsulate related data, and a convenient semantic framing mechanism for intelligent network transport and routing.

Acknowledgments

We would like to thank Qian Z. Wang and Micah Gutman for their work on an early version of the XML router and the graphical ASDI client shown in Figure 15. The DCP experiements were conducted at emulab.net, the Utah Network Emulation Testbed, which is primarily supported by NSF grant ANI-00-82493 and Cisco Systems. We are indebted to Benjie Chen and the members of the Click project for assistance with benchmarking our Click-based XML router. This paper greatly benfited from comments on earlier drafts by Chuck Blake, Frans Kaashoek, the anonymous reviewers, and our shepherd, Maurice Herlihy. We also remember Jochen Liedtke, Bruce Jay Nelson, and Mark Weiser as great life forces and friends.

References

[1] XMLBlaster. `http://www.xmlblaster.org/`.

[2] ANDERSEN, D. G., BALAKRISHNAN, H., KAASHOEK, M. F., AND MORRIS, R. T. Resilient overlay networks. In *Proc. ACM SOSP* (Oct. 2001).

[3] ARMSTRONG, S., ET AL. Multicast transport protocol. RFC 1301, Internet Engineering Task Force, 1992.

[4] BANAVAR, G., CHANDRA, T., MUKHERJEE, B., NAGARAJARAO, J., STROM, R., AND STURMAN, D. An efficient multicast protocol for content-based publish-subscribe systems. In *Proc. Int'l Conf. on Dist. Comp. Systems (ICDCS)* (May 1999).

[5] BANERJEA, A. Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels. In *Proc. ACM SIGCOMM* (Aug. 1996), pp. 194–205.

[6] BESTAVROS, A. An adaptive information dispersal algorithm for time-critical reliable communication. In *Network Management and Control, Volume II*, I. Frish, M. Malek, and S. Panwar, Eds. Plenum Publishing Co., New York, New York, 1994, pp. 423–438.

[7] BRAY, T., ET AL. Extensible markup language 1.0 (second edition). `http://www.w3.org/TR/REC-xml/`, W3C Recommendation, 2000.

[8] BYERS, J. W., LUBY, M., AND MITZENMACHER, M. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proc. IEEE Infocom* (Mar. 1999), pp. 275–283.

[9] BYERS, J. W., LUBY, M., MITZENMACHER, M., AND REGE, A. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM* (Sept. 1998), pp. 56–67.

[10] CARZANIGA, A., ROSENBLUM, D. S., AND WOLF, A. L. Achieving scalability and expressiveness in an Internet-scale event notification service. In *Proc. ACM PODC* (July 2000), pp. 219–227.

[11] CHAMBERLIN, D., ET AL. XQuery 1.0: An XML query language. `http://www.w3.org/TR/xquery/`, W3C Working Draft, 2001.

[12] CHAWATHE, Y., MCCANNE, S., AND BREWER, E. RMX: Reliable multicast for heterogeneous networks. In *Proc. IEEE Infocom* (Mar. 2000), pp. 795–804.

[13] CHU, Y., RAO, S. G., AND ZHANG, H. The case for end system multicast. In *Proc. ACM SIGMETRICS* (June 2000), pp. 1–12.

[14] FLOYD, S., JACOBSON, V., MCCANNE, S., LIU, C.-G., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. on Networking 5*, 6 (Dec. 1997), 784–803.

[15] HOLBROOK, H. W., AND CHERITON, D. R. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proc. ACM SIGCOMM* (Aug. 1999), pp. 65–78.

[16] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K., KAASHOEK, M. F., AND O'TOOLE, J. Overcast: Reliable multicasting with an overlay network. In *Proc. USENIX OSDI* (Oct. 2000), pp. 197–212.

[17] KADANSKY, M., CHIU, D., AND WESLEY, J. Tree-based reliable multicast [TRAM]. Technical report TR-98-66, Sun Microsystems Lab, 1998.

[18] KARN, P., AND PARTRIDGE, C. Improving round-trip time estimates in reliable transport protocols. *ACM CCR 17*, 5 (Aug. 1987), 2–7.

[19] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. *ACM Trans. on Computer Systems 18*, 3 (Aug. 2000), 263–297.

[20] LABOVITZ, C., AHUJA, A., ABOSE, A., AND JAHANIAN, F. Routing stability and convergence. In *Proc. ACM SIGCOMM* (Aug. 2000), pp. 115–126.

[21] LEMPEL, A., AND ZIV, J. A universal algorithm for sequential data compression. *IEEE Trans. on Information Theory 23*, 3 (May 1977), 337–343.

[22] LIN, J.-C., AND PAUL, S. RMTP: A reliable multicast transport protocol. In *Proc. IEEE Infocom* (Mar. 1996), pp. 1414–1424.

[23] MANBER, U. Finding similar files in a large file system. In *Proc. Winter USENIX* (Jan. 1994), pp. 1–10.

[24] MAXEMCHUK, N. F. *Dispersity Routing in Store and Forward Networks*. PhD thesis, University of Pennsylvania, May 1975.

[25] MCAULEY, A. J. Reliable broadband communication using a burst erasure correcting code. In *Proc. ACM SIGCOMM* (Sept. 1990), pp. 297–306.

[26] MCCANNE, S., AND JACOBSON, V. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM* (Aug. 1996), pp. 117–130.

[27] MOSER, L., MELLIAR-SMITH, P., AGARWAL, D., BUDHIA, R., AND LINGLEY-PAPADOPOULOS, C. Totem: A fault-tolerant multicast group communication system. *C. ACM 39*, 4 (Apr. 1996), 54–63.

[28] OKI, B., PFLUEGL, M., SIEGEL, A., AND SKEEN, D. The information bus — an architecture for extensible distributed systems. In *Proc. ACM SIGOPS* (Dec. 1993), pp. 58–68.

[29] PAXSON, V. End-to-end internet packet dynamics. *IEEE/ACM Trans. on Networking 7*, 3 (June 1999), 277–292.

[30] PENDARAKIS, D., SHI, S., VERMA, D., AND WALDVOGEL, M. ALMI: An application level multicast infrastructure. In *Proc. USENIX Symp. on Internet Technologies and Systems (USITS)* (Mar. 2001), pp. 49–60.

[31] RABIN, M. O. Efficient dispersal of information for security, load balancing and fault tolerance. *J. ACM 36*, 2 (Apr. 1989), 335–348.

[32] RIZZO, L. Dummynet: a simple approach to the evaluation of network protocols. *ACM CCR 27*, 1 (Jan. 1997).

[33] RIZZO, L., AND VICISANO, L. A reliable multicast data distribution protocol based on software FEC techniques. In *Proc. IEEE HPCS* (June 1997).

[34] SANTRY, D. J., FEELEY, M. J., HUTCHINSON, N. C., AND VEITCH, A. C. Elephant: The file system that never forgets. In *Proc. Workshop on Hot Topics in Operating Systems (HotOS-VII)* (Mar. 1999).

[35] SAVAGE, S., ANDERSON, T., AGGARWAL, A., BECKER, D., CARDWELL, N., COLLINS, A., HOFFMAN, E., SNELL, J., VAHDAT, A., VOELKER, J., AND ZAHORJAN, J. Detour: a case for informed internet routing and transport. *IEEE Micro 19*, 1 (Jan. 1999), 50–59.

[36] SEGALL, B., ARNOLD, D., BOOT, J., HENDERSON, M., AND PHELPS, T. Content based routing with Elvin4. In *Proc. AUUG2K* (June 2000).

[37] STOICA, I., NG, T. S. E., AND ZHANG, H. Reunite: A recursive unicast approach to multicast. In *Proc. IEEE Infocom* (Mar. 2000), pp. 1644–1653.

[38] TRIDGELL, A. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, Apr. 2000.

[39] TRIDGELL, A., AND MACKERRAS, P. The rsync algorithm. Tech. Rep. TR-CS-96-05, Australian National University, 1997.

[40] VOLPE NATIONAL TRANSPORTATION CENTER, AUTOMATION APPLICATIONS DIVISION. Aircraft situation display to industry functional description and interfaces. DTS-56 report, Aug. 2000.

[41] WHETTEN, B., AND TASKALE, G. An overview of reliable multicast transport protocol II. *IEEE Network 14*, 1 (Jan. 2000), 37–47.

[42] WU, L., SHARMA, R., AND SMITH, B. Thin streams: An architecture for multicasting layered video. In *ProcİEEE Int'l Workshop on Network and Operating System Support for Digital Audio and Video* (May 1997).

[43] YAVATKAR, R., GRIFFIOEN, J., AND SUDAN, M. A reliable dissemination protocol for interactive collaborative applications. In *Proc. ACM Conf. on Multimedia* (Nov. 1995), pp. 371–372.