An Internet-wide Distributed System for Data-stream Processing

Gabriel Parmer, Richard West, Xin Qi, Gerald Fry, and Yuting Zhang

Computer Science Department
Boston University
Boston, MA 02215
{gabep1, richwest, xqi, gfry, danazh}@cs.bu.edu

Abstract

The ubiquity of the Internet has stimulated the development of data- rather than processor-intensive applications. Such data-intensive applications include streaming media, interactive distance learning, and live web-casts. While plenty of research has focused on the real-time delivery of media streams to various subscribers, no solutions have been proposed that provide per-subscriber QoS guarantees to hundreds of thousands of subscribers distributed on an Internet scale. Moreover, there has been no successful approach that integrates heterogeneous end-hosts into a distributed system, connected via an overlay network, for the purpose of QoS-constrained data processing and delivery.

This paper proposes a pipelined distributed system that is the foundation for the delivery of QoS constrained data-streams over a scalable backbone consisting of heterogeneous commodity-off-the-shelf (COTS) systems. To provide a scalable routing substrate that can adapt to QoS demands and constraints, a peer-to-peer (P2P) overlay topology is constructed consisting of the end-hosts of the system. Further, end-hosts leverage a technique called user-level sandboxing, that enables application-specific stream processing agents (SPAs) to be deployed on various hosts in the system. These SPAs can be used to process and route data according to application-level service requirements without the need for explicit scheduling or data-copying overheads. Collectively, these research endeavors provide the mechanisms by which a distributed data-stream processing system on the scale of the Internet can be realized.

1. Introduction

The growth of the Internet has stimulated the development of applications that are largely data rather than CPU intensive. Examples include streaming media delivery, interactive distance learning, web-casting, group simulations, live auctions, and stock brokerage applications. Additionally, peer-to-peer (P2P) systems such as Gnutella [10, 15], Freenet [7], Kazaa [12] and more recent variations (e.g., Chord [19], CAN [14], Pastry [16] and Tapestry [23]) have become popular due to their ability to efficiently locate and retrieve data of particular interest.

While the Internet continues to grow, there is an enormous untapped potential to use distributed computing resources for large scale applications. Some projects such as *SETI@home* [17] have already realized this potential, by utilizing spare compute cycles on off-the-shelf computers to process application-specific data. Similarly, grid computing technologies (e.g., Globus [9, 5]) are attempting to coordinate and share computing resources on the scale of the Internet, to support emerging applications in areas such as scientific computing. However, there has been only limited research on the construction of scalable distributed systems to support the *delivery* and *processing* of high bandwidth, and potentially real-time, data streams transported over the Internet [4]. An Internet-wide distributed system for data stream processing would be desirable for applications such as interactive distance learning, tele-medicine, and live video broadcasts, bringing together potentially many thousands of people located in different regions of the world. Further, such a system would pave the way for new applications, such as a distributed and adaptive traffic management system, and, generally, any application that requires the dissemination of sensor data.

Essentially, applications are emerging that have requirements that extend beyond what a classical client/server paradigm can provide. In this model, a client issues a remote procedure call (RPC) to a server, sending the request and receiving the reply via IP. Some applications exist that would be better served by a pipelined, publisher/subscriber model. These models of end-host communication can be seen in Figure 1. The delivery of QoS constrained media streams to an extremely large population of interested end-hosts is a task ideally suited for the publisher/subscriber model.

This paper presents such a system based on pipelined

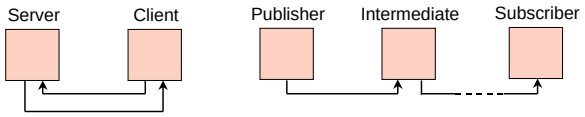

Figure 1. Client-server and publisher/subscriber models.

processing of data streams. Generally, this system can be utilized by any application domain that requires the scalable, QoS and resource aware delivery and processing of data-streams. Essentially, data streams can be transferred over a overlay topology of end-hosts on the Internet and at each hop processing can be performed on the stream by Stream Processing Agents (SPAs). These SPAs can take the form of a QoS and resource aware router, a filter to extract from a stream relevant information, an entity to perform transformations on the data, an agent to build multicast trees, a splitting agent that could separate the stream over two links to distribute bandwidth usage, or an agent to merge these streams.

For example, a number of sensors, perhaps cameras, could publish the raw video they capture onto the distributed system. It would be routed through the overlay via a sequence of end-host intermediary nodes. At each of these intermediaries, a number of SPAs can be applied to the data-stream to, for instance, compress and filter the data so that a mobile device can easily receive and display the sensor output. A certain population will subscribe to these data-streams. These subscribers are the destination end-hosts and will display the video. They will also act as intermediaries, possibly further routing streams to which they are currently subscribed to other interested end-hosts, or perhaps applying SPAs to other data-streams. The distributed system is responsible for providing certain QoS levels to each of the subscribers and each stream will be processed and routed according to these constraints. This example system can be seen in Figure 2.

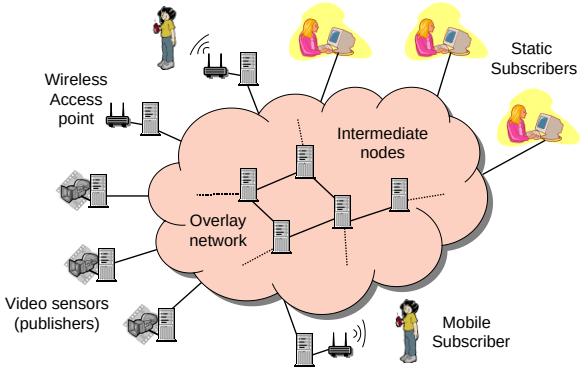

Figure 2. An example distributed system for sensor-produced data-stream dissemination.

One of the fundamental aspects of this work is to leverage off-the-shelf computers on the Internet to build a scalable distributed system for the routing and processing of data streams. Of particular interest is the use of *overlay networks* to deliver real-time media streams from one or more publishing nodes to potentially many thousands of subscribers, each with their own quality-of-service (QoS) requirements. While many existing P2P systems form overlay networks on top of the underlying physical network, they do not yet support the timely delivery of data streams between publishers and subscribers. For example, Pastry, Chord, and CAN all build logical topologies for the purpose of forwarding messages to *locate* an object in a decentralized system, rather than *transporting* data streams. Moreover, these systems make no attempt to route data in accordance with latency and bandwidth requirements.

There are systems such as Narada [4] that attempt to use end-hosts for multicast routing of data with QoS constraints. While end-system multicasting incurs delay penalties relative to IP-layer multicasting, it does have some advantages. First, IP-layer multicasting is not widely used. Second, using end-hosts for routing enables paths to be constructed and adapted based on application-specific requirements. Third, routing decisions can be made dependent on resource usage distributions throughout the system. Finally, end-hosts can incorporate additional functionality to perform application-specific data processing before data is forwarded to the next hop.

Narada advocates end-system multicast routing, but cannot scale to many hosts. A system is desirable that has the scalability of a P2P system, but that can provide the versatility of Narada's routing scheme. Several studies have been carried out in building toward this system. Section 2 describes preliminary studies relating to the construction of scalable overlay topologies which will provide the backbone over which to route and process data. This is followed by Section 3, that describes work in the area of end-system design and extensibility. In the latter case, each end-host in the distributed system will be equipped with service policies and mechanisms to ensure application-specific data processing agents will execute in a safe, predictable and efficient manner. We argue that an extensible software architecture is needed on at least a subset of all end-hosts in the distributed system to support application-specific SPAs and customizable resource monitoring functions in a manner that can guarantee the QoS. This will enable applications to deploy stream processing agents and/or monitoring extensions on remote hosts, thereby influencing the behavior of the system for their specific needs.

2 Scalable Overlay Topologies for QoS-Constrained Routing

Research in the area of P2P systems has led to the use of distributed hashing techniques for scalable overlay rout-

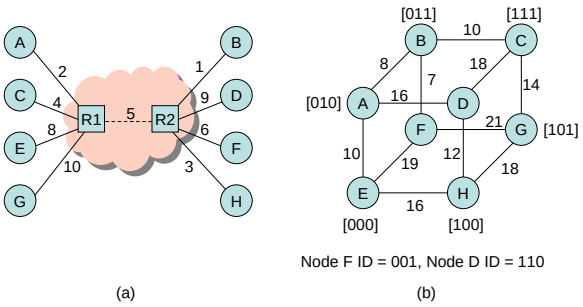

Figure 3. A sample overlay network

ing, but such approaches have not attempted to provide QoS guarantees on data delivery. In effect, these systems build *k-ary n-cube* logical overlays (or, more general n-dimensional tori) on top of the physical network.

We use undirected *k-ary n-cube* graphs to model logical overlays in a similar manner to these P2P systems. These graphs are specified using n as the *dimensionality* parameter and k as the *radix* (or *base*) in each dimension. Figure 3 shows an example of an overlay network structured as a *2-ary 3-cube* graph and a corresponding underlying physical network. A cost is associated with each edge in the physical network, and each edge in the logical overlay maps to the shortest path between the respective end-point nodes in the physical topology. The costs associated with logical edges are derived as the sum of the costs along the corresponding path taken in the physical network. Note that the physical topology may contain end-hosts that do not participate explicitly within the context of the overlay network (i.e., R1 and R2).

2.1 Properties of *k-ary n-cube* Topologies

K-ary n-cubes have many important characteristics which can be leveraged for QoS-constrained routing [8]:

- A k-ary n-cube is an n dimensional graph with k vertices (or nodes) in each dimension. One can think of k as the *radix* or *base* of a given dimension.
- $M = k^n$, where M is the number of nodes in the graph. Therefore, $n = \log_k M$.
- Each node is of the same degree, with n neighbors if $k = 2$, or $2n$ neighbors if $k > 2$.
- The distance between any pair of nodes in the graph is no more than $W(k, n) = n\lfloor \frac{k}{2} \rfloor$ hops. This assumes there are no cycles in the path between a pair of nodes.
- The average routing path length between nodes in the graph is $A(k, n) = n\lfloor \frac{k^2}{4} \rfloor \frac{1}{k}$ hops.
- The optimal dimensionality of the graph is $n = \ln M$, to minimize the average and worst-case path lengths between a pair of nodes. However, since n must be an integer, the actual dimensionality of a k-ary n-cube should approximate to this value.
- Each node in the graph can be associated with a logical identifier consisting of n digits, where the ith digit (given $1 \leq i \leq n$) is a base-k integer representing the offset in dimension i.
- Two nodes are connected by an edge *iff* their identifiers have $n - 1$ identical digits, except for the ith digit in both identifiers, which differ by exactly 1 modulo k.

The regularity of *k-ary n-cube* graphs provides for a logical topology that is scalable in the sense that routing complexity increases less than linearly with the number of logical nodes in the system. Intuitively, the structure is regular and compact, with different values of k and n resulting in differing topology sizes and corresponding values of $A(k, n)$.

Hypercubes (k-ary n-cubes where $k = 2$) are common in NUMA architectures and much research has been done concerning these structures in that context. Building a distributed and optimal k-ary n-cube is a much more complex problem than those addressed in the past. The scale of the endeavor is orders larger and the environment in which it will be deployed is of a dynamic nature. Because nodes can join and depart from the system at any time, the optimal values for k and n must change, and the system must adapt. The problem of choosing values for k and n reduces to imposing a linear ordering on (k, n) pairs such that corresponding values of $A(k, n)$ are monotonically increasing. Further details can be found in a corresponding technical report [8]. For each (k, n) pair we define an *M-region*, or range of values for the number of *physical* hosts, for which the associated k-ary n-cube is optimal with respect to $A(k, n)$. We are not aware of any P2P systems that dynamically adapt their overlay topology to minimize the average hop distance between nodes, as in our approach.

2.2 Proximity-based Greedy Routing

While our overlay topology is bootstrapped for a particular k-ary n-cube, it can be changed dynamically as the number of physical hosts change. However, this merely affects the average and worst-case hop counts to route data between a pair of end-hosts. For QoS-constrained routing, we need to know the physical link costs of communication between pairs of end-hosts. In preliminary studies we tested three alternative approaches to routing over k-ary n-cubes:

- *Ordered Dimensional Routing*: The data-stream is always forwarded to a specified connected node determined by what hop we are on.
- *Random Ordering of Dimensions*: This is similar to the previous scheme except data streams are forwarded along randomly selected dimensions, but always toward the destination.
- *Greedy Routing*: Greedy routing is performed using some measure of physical proximity. Data-streams are

always forwarded along logical edges with the lowest cost, and, again, always toward the destination.

For more detailed information regarding these routing policies, see [8].

Experiments were conducted using generated random transit-stub physical topologies [22] with 65536 hosts. They demonstrate the behavior of our P2P system with a significantly sized population. One host was chosen at random to be a publisher, and all other hosts were assumed to be subscribers. A message was then routed from the publisher host to each subscriber host and end-to-end latencies were recorded, as well as the unicast latency of a message routed directly between the publisher and each subscriber. The delay penalty of routing over the overlay relative to the unicast delay was calculated as the logical end-to-end latency divided by the unicast delay for each subscriber host. Figure 4 shows the delay penalties for each of the three routing algorithms on a 2-ary 16-cube, and on a 16-ary 4-cube. We can see that, as expected, different configurations for k and n yield different efficiencies for certain population sizes [8]. Moreover, we can see that greedy routing can provide better efficiency given a dynamic network with varying link states. Greedy routing allows our distributed system to adapt to network characteristics and route with per-data-stream QoS constraints in mind.

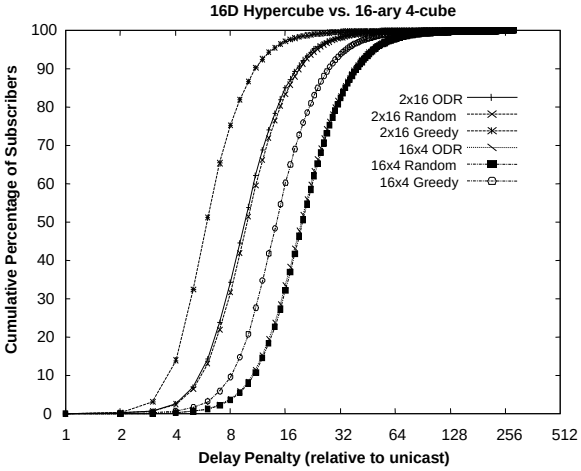

Figure 4. Comparison of routing algorithms.

While the relative delay penalty is significant, with respect to unicast costs, end-system multicast routing seems plausible for large-scale applications. We still need to investigate relative delay penalties compared to network-layer IP multicast, but Narada provides convincing evidence that host-based routing is a viable approach.

```
Subscribe(Subscriber S, Publisher P) {
  Find the neighbor i of P such that:
    (i.cost(P) < S.constraint) or
    (i.cost(P) is minimum for all neighbors);
  If host i is not a subscriber then
    swap logical positions of i and S;
  If host i is a subscriber then
    Subscribe(S, i); }
```

Figure 5. Adaptive node re-assignment algorithm

2.3 Adaptive Node ID Assignment

Bootstrapping an overlay topology and randomly assigning node IDs can result in poor proximity between neighboring nodes. In the absence of information about how publisher and subscriber hosts are associated with QoS-constrained data streams, random placement of physical hosts in the logical network is appropriate. However, as sets of hosts begin to specify interest in receiving particular data streams with corresponding service constraints, it becomes possible to re-assign such subscriber hosts to more appropriate locations in logical space. Re-assignment of a host to a new location in the overlay (based on the requested service) is accomplished by *swapping* the logical node identifier, as well as routing table information, with some other host in the system. In typical P2P systems, there is no correlation between logical ID and physical locality. The adaptive re-assignment of node ID establishes this correlation in hopes that QoS guarantees can be strengthened by decreasing communication latency. The algorithm for changing positions in the logical overlay is described in Figure 5.

Experiments have been conducted that show using adaptive node re-assignment can give significant benefit. See our previous paper [8] for details.

3 End-System Design and Extensibility

The importance of being able to provide QoS guarantees for data-streams at the end-host granularity cannot be overstated. Because of this, we intend to configure end-hosts with support for "user-level sandboxing" [21]. A sandbox will provide a safe, efficient, and predictable environment for the implementation of overlays, stream processing agents, and application-specific services, including those that monitor and react to observed resource usage and availability. Using this approach, data stream processing can take place on an end-user's machine without consuming all computing resources. Constraining the usage of resources and making them have as little overhead as possible is important because an end-user may wish to use his/her computer for more than just data stream processing; there may be other applications co-existing on a single end-host.

Using our sandboxing mechanism, we allow applications to configure and deploy SPAs at user-level that may execute

in the context of *any* address space. Using Linux as the basis for our approach, we focus specifically on the implementation of a user-space network protocol stack, that avoids copying data via the kernel when communicating with the network interface. Our approach enables services to efficiently process and forward data using configurable SPAs via proxies, or intermediate hosts, in the communication path of high performance data streams. Unlike other user-level networking implementations, our method makes no special hardware requirements and can be widely deployed on COTS systems. Due to benefits gained from *user-level sandboxing* we achieve a substantial increase in throughput over comparable user-space methods. Additionally, the elimination of scheduling overheads greatly reduces the delay variation between service invocations, which is critical to jitter-sensitive data-streams such as real-time media streams.

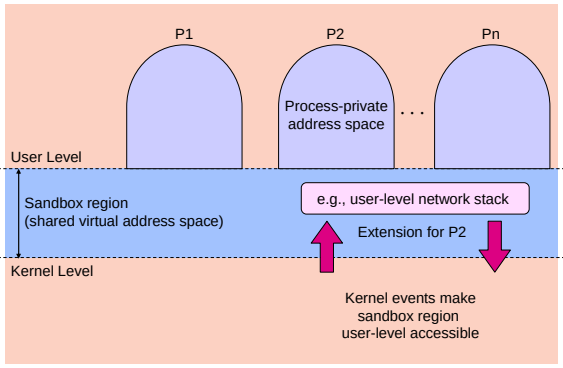

Figure 6. User-level sandboxing with an extension mapped into the shared virtual memory region.

Traditional operating systems provide logical protection domains for processes mapped into separate address spaces. With user-level sandboxing (Figure 6), each process address space is divided into two parts: a conventional process-private memory region and a shared, but normally inaccessible, virtual memory region. The shared region acts as a sandbox for mapped extensions. When extensions are executed via upcalls from the kernel, the shared virtual-memory region is made accessible at user-level. Kernel events, delivered by upcalls to sandbox code, are handled in the context of the current process, thereby eliminating scheduling costs.

Because extensions running in the sandbox execute at user-level, they can use application libraries, and typical application programming practices[1]. Also, in contrast to kernel-level extensibility mechanisms, they cannot issue privileged instructions, and therefore they cannot directly modify or influence the networking hardware to copy packets directly to and from the sandbox using DMA. This protects the system, but the ability to utilize DMA to zero-copy is extremely important to minimize the overhead of stream processing on the end-hosts. Typical implementations of zero-copy, require specialized hardware [20]. Instead our scheme allows the host kernel and sandbox to interface through a well defined set of communication channels to pass the desired memory location for packet arrival or transmission. In this way, the kernel address space is isolated from sandbox extensions. Independent of which process is running at any given moment, the networking card can DMA directly to or from the sandbox region. Figure 7 demonstrates the structure and execution of the sandbox networking stack and the SPA execution as well as the DMA interaction with the network interface card. A thorough explanation can be found in [13].

[1]With a few restrictions such as not being able to use `mmap[21]`

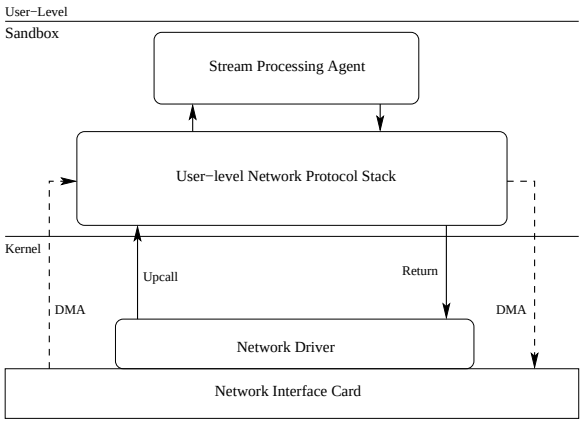

Figure 7. Packets are transferred by DMA between the NIC and the sandbox. The network driver upcalls into the sandbox protocol stack and SPAs compute on the packets.

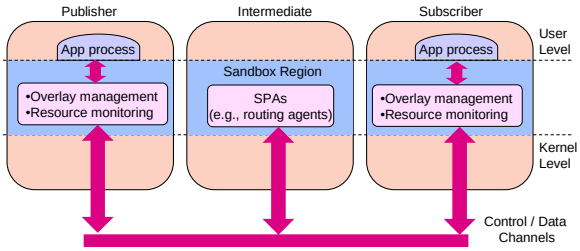

Figure 8. A publisher/subscriber model using user-level sandboxing.

Sandboxed Networking Efficiency: We use a simple *UDP forwarding agent* to test the performance characteristics of our implementation. This application forwards UDP packets for specific streams of data from one host, the publisher, through the intermediary forwarding host, to the subscriber. A generic demonstration of a publisher/subscriber model can be seen in Figure 8. Such a generic application would

be used to route data along the P2P overlay described in Section 2. The forwarding would typically be directed and arbitrated by routing protocols, such as the greedy routing algorithm explained in 2.2.

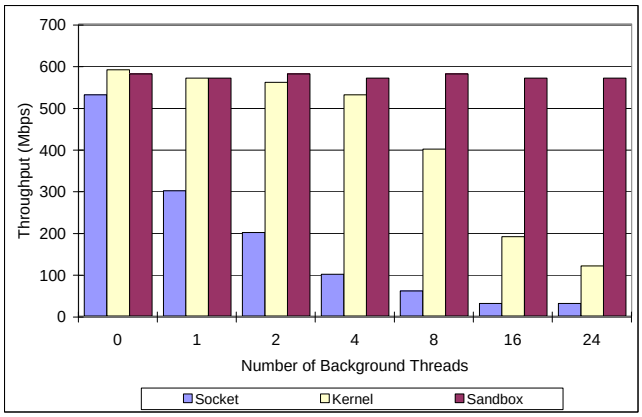

Figure 9. UDP Throughput comparison of an optimized sandbox stack versus both user-level sockets and kernel implementations.

We show the efficiency of our sandboxed networking implementation by demonstrating the raw bandwidth achievable. Figure 9 shows a comparison between the throughput routing capability of the fastest possible middleware forwarder, an application that simply reads data from one socket and writes to another, a kernel forwarder that uses a kernel-thread to forward data from one socket to another, and a forwarding agent using the sandbox networking stack. The main differences between the implementations is that the sandboxed networking stack does all forwarding at interrupt time because SPAs can be executed in the context of any running process, while the other implementations do network processing in the context of a thread, so they suffer from scheduling delays. Both the kernel and sandbox networking case use DMA to execute zero-copy communication while the middleware case cannot. Background processes are run to measure the effect of system load on performance. These background processes are simple CPU-bound tasks with minimal working-set sizes. It is important to measure the throughput of our methods with background processes because we target COTS systems on the Internet that probably will be performing other tasks concurrently.

One can see that the sandboxed networking stack's bandwidth remains nearly constant as the number of processes increases while the other approaches do not. Even with no background processes, our user-level networking does better than the minimal user-level application and only slightly worse than the kernel itself. A thorough analysis of these results can be found in [13].

Quality of Service Reliability: Because it is important to provide QoS guarantees at every level in a system, from overlay to end-hosts, we measure the amount of jitter in the arrival time of the packets experienced by the node receiving the stream routed through the intermediary node. This allows us insights into the granularity of QoS guarantees we can make on an end-host level. Experiments conducted to test the QoS characteristics of the forwarding SPA in the sandbox as compared to the one in the kernel, and a middleware program can be found in [21]. The jitter in the data-stream sent from publisher to subscriber is measured, with a certain amount of background threads. The jitter experienced by the sandbox networking scheme is independent of the number of threads currently running and is constant. Because the other two schemes suffer from scheduling delays, they experience a significant increase in jitter as the number of background threads grow. Isolation of the network computation strengthens the QoS guarantees we can make in the distributed system, because on each end-host along the routing path of a data stream, jitter can be tightly constrained. Because a data-stream may be forwarded through multiple end-hosts through the overlay, it is important that each of these nodes avoids introducing unpredictable delays. Any detriment incurred at the node level will be compounded as the data-stream is forwarded through multiple hops.

4 Related Work

Many technologies have been proposed to construct scalable content management systems on the Internet. Among these are existing P2P systems such as first [10, 15, 7, 12] and second [19, 14, 16, 23] generation P2P networks that build logical topologies for the purposes of *locating* an object in a decentralized manner. While these systems route control messages over these logical topologies, they do not use them for the real-time delivery of application streams.

Systems such as Narada [4] attempt to use end-hosts for multicast routing of data with QoS constraints. Though similar to our system, Narada cannot scale to anything beyond a few hundred hosts. In contrast, we wish to scale to the size of the Internet. While a number of other projects [1, 6, 3, 11, 2, 18] have implemented scalable multicast solutions at the application-level, most have either not addressed per-subscriber QoS requirements, or take a very different approach to ours. None have attempted to ensure QoS guarantees on the end-host level.

5 Conclusions and Future Work

We propose a framework for the processing and delivery of data-streams that is scalable to the size of the Internet and that can address per-subscriber QoS constraints throughout the entire system. Though this system is ideal for the delivery and processing of multimedia streams, it is general enough that it could be used as a distributed pipelined

super-computer. The need for such a system is derived from the inability of the client/server model to provide a resource-aware, QoS constrained data-stream delivery and processing framework. A publisher/subscriber paradigm is required for such a system, especially when the size of the population becomes large.

To provide a scalable framework over which data-streams can be processed in a manner consistent with their QoS constraints, we propose a system consisting of a P2P overlay and an end-system architecture, both engineered to provide scalability and QoS throughout. The P2P system is able to make decisions about optimal structure for different population sizes, about intelligent routing based on QoS and network characteristics, and about logical placement of nodes to further strengthen QoS constraints. The end-system sandboxing network architecture provides an efficient and QoS aware system with which to execute SPAs on data-streams. It does this while making no special hardware requirements. Experiments have confirmed the scalability and efficiency of both of these technologies.

The mechanisms for such a distributed system for data-stream processing on the scale of the Internet have been introduced in this paper. Many of the policies to build on top of these mechanisms are currently being researched. These include resource management and monitoring which will allow intelligent application of specific SPAs throughout the network, such as one to split a stream or to build a multicast tree. Additionally, new SPAs for specific types of data-streams need to be formulated. For instance, any pipelined scientific application will require specific computational stages, each of which can be mapped into a SPA. In short, this paper presents the framework for a distributed data-stream processing system and many of the policies of how to manage and use the system are ongoing work.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM Sigcomm*, August 2002.

[2] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of INFOCOM*, San Francisco, CA, April 2003.

[3] Y. Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, December 2000.

[4] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[5] I. Foster and C. Kesselman. The Globus Project: A status report. In *Proceedings of IPPS/SPDP'98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[6] P. Francis. Yoid: Extending the multicast internet architecture, 1999. On-line documentation: http://www.aciri.org/yoid/.

[7] Freenet: http://freenet.sourceforge.net/.

[8] G. Fry and R. West. Adaptive routing of QoS-constrained media streams over scalable overlay topologies. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2004.

[9] The Globus Alliance: http://www.globus.org/.

[10] Gnutella: http://www.gnutella.com.

[11] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[12] Kazaa: http://www.kazaa.com.

[13] X. Qi, G. Parmer, R. West, J. Gloudon, and L. Hernandez. Efficient end-host architecture for high performance communication using user-level sandboxing. Technical Report 2004-009, Boston University, 2004.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[15] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the International Conference on Peer-to-Peer Computing (P2P2001)*, Linkoping, Sweden, August 2001.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[17] SETI@home: http://setiathome.ssl.berkeley.edu/.

[18] S. Shi and J. Turner. Routing in overlay multicast networks. In *Proceedings of IEEE Infocom*, June 2002.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[20] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 40–53. ACM, December 1995.

[21] R. West and J. Gloudon. User-Level Sandboxing: a safe and efficient mechanism for extensibility. Technical Report 2003-14, Boston University, 2003. Now revised and submitted for publication.

[22] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom*, volume 2, pages 594–602, San Francisco, CA, March 1996. IEEE.

[23] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, Apr. 2001.