

Infopipes: The ISL/ISG Implementation Evaluation

Galen Swint¹, Calton Pu²,
Younggyun Koh, Ling Liu, Wenchang Yan
Center for Experimental Research
in Computer Systems (CERCS),
Georgia Institute of Technology
¹swintgs@acm.org, ²calton@cc.gatech.edu

Charles Consel
INRIA/LaBRI/ENSEIRB, Bordeaux, France
Koichi Moriyama
Sony Corporation, Tokyo, Japan
Jonathan Walpole
OGI School of Science and Engineering,
OHSU, Beaverton, OR

Abstract

We provide a performance comparison of generated Infopipes that have been translated from the Spi/XIP variant of Infopipe specification into executable code. Infopipes are an abstraction to support information flow applications. These tools are evaluated through a realistic application: a continuous image streaming program. We implement the application in C and compare its performance to both a hand-written application and one that uses SunRPC.

1. Introduction

One of the fundamental functions of systems software (e.g. operating systems and middleware) is to provide a higher abstraction on top of hardware to application programmers. More generally, one important aspect of system software research is to create and provide increasingly higher levels of programming abstraction on top of existing abstractions. Remote Procedure Call (RPC) [1] is a successful example of such abstraction created on top of messages, particularly for programmers of distributed client/server applications. Unfortunately, RPC, despite its success, has proven to be less than perfect for some applications, particularly the class of applications that operate on information flows, also called *information flow applications* – examples in this application domain include data streaming and filtering [2], sensor networks, e-commerce transactions, and multimedia streaming. This paper provides comparative performance of an implementation of generated communication code to support these information flow applications using the Infopipe abstraction (Section 3 and [2], [3], [4]).

The Infopipe abstraction was defined to cover some aspects of programming information flow applications unsupported by RPCs. For example, many information flow applications involve multiple transformational stages and multiple communication links between the stages. A second example is the support for quality of service (QoS). Some existing efforts in supporting the specification and maintenance of sophisticated information flow applications using RPC-style middleware [5] show the difficulties in such an approach.

The main contribution of this paper is an evaluation of current Infopipe implementations compared to traditional RPC programming. We wrote a realistic distributed multimedia application (with QoS requirements) using Infopipes and compared it to equivalent programs that are (1) hand-written and (2) built with SunRPC. On the programming side, we show that Infopipe-based code is significantly smaller than RPC programming. On the performance side, we show that Infopipe-based code achieves latency and bandwidth compared to the best hand-written code, and gaining in QoS support (comparable or smaller latency) when compared to SunRPC-generated code.

Many important Infopipe features such as QoS properties, Aspect Oriented Programming (AOP) support, or Infopipe composition (topics of active research) are beyond the scope of the paper but are partially treated in [6].

2. Background and Motivation

We believe that an appropriate programming paradigm for information-driven applications should embrace information flow as a core abstraction. Unsurprisingly, there are already concrete examples of existing information flow software. (e.g. combining UNIX's filters). This abstraction aims to offer the following advantages over RPC: first, data parallelism among flows should be naturally supported; second, the specification and preservation of QoS properties should be included; and third, the implementation should scale with the application. We emphasize this new abstraction is intended to complement RPC — not to replace it. For true client/server applications, RPC is still the natural solution.

In the Infopipe research, we followed a methodology analogous to the development of the RPC IDL, and stub generator. We developed a domain specific language for describing and building Infopipes. It abstracts the connection and data typing and frees the programmer from the necessity of uncovering architecture-specific and language-specific. Other advantages of the IDL+generator approach include enhanced program correctness and shorter development cycles since

generated code is pre-written and tested. Infopipe specifications are translated by our toolkit into code in a fashion analogous to an RPC stub generator. Similarly, the difficulties of marshalling and unmarshalling data, system initialization, etc. are hidden from the developer.

3. The Infopipes Toolkit

Like RPC, Infopipes ([2], [3], [4]) raise the level of abstraction for distributed systems programming and offer certain kinds of distribution transparency. Beyond RPC's IDL, which provides minimal data typing information, an Infopipe is specified by the syntax, semantics, and QoS properties captured in its *typespec*. To capture typespec information, we developed Spi (for Specifying Infopipes), a text-based Infopipe Specification Language, to support information flow applications. It is similar to other domain-specific languages such as Devil [7] or the RPC IDL in that it encapsulates domain knowledge and provides similar development benefits.

A typical Infopipe has two ends – a consumer (*inport*) end and a producer (*outport*) end – and implements a unidirectional information flow from a single producer to a single consumer. Between the two ends a developer provides a function, the *middle*, which can process, buffer, filter, or transform information. In operation, an information producer exports and transmits an explicitly defined and typed information flow, which goes to a consumer Infopipe's *inport*. After appropriate transportation, storage, and processing, the information flows to a second information consumer.

The C/TCP implementation of Infopipes operates in a straightforward manner, transmitting bytes in native binary between the sender and receiver. Dynamic arrays and strings are only transmitted up to their used size as set during runtime. Datatypes as specified in Spi are translated into C `struct` statements. To avoid copying, there is one structure per port into which the port can write data (for an *inport*) or from which it can read data (for an *outport*).

4. Experimental Evaluation

First, we provide a short performance evaluation of the C implementations; and second, we have implemented an image streaming application using C and Infopipes. On C, we tested the application using the socket and the ECho middleware communication layer, and in one test, a mixed communication path which used sockets for the first link and ECho for the second. We compared the latency and throughput to the performance for SunRPC.

For this paper, our measure of latency is the time it takes for a sending Infopipe or process to send an application packet and receive a reply (in the case of our

latency tests the connection between sender and receiver is symmetric). Throughput is also important, especially for applications with large data payloads, we measure throughput by timing the sending of many application packets from a source to a sink and then calculating based on the total volume of bytes sent.

Each benchmark we ran using hand-written code, Infopipes code, and RedHat 9's `rpcgen` code on three different kinds of application packets: a small packet containing a single integer (4 bytes), a mixed packet containing an array of 100 characters, a 10 element float array, and 3 integers (152 bytes total), and a packet containing an array of 3,072 integers (12288 bytes). For latency, we measure the time it takes to send the test packet and receive a 4-byte acknowledgement. For each the large and mixed packet tests, we perform 1000 measurements per trial, and then we report the average of 50 trials. For the small packet payload, we extend a trial's size to include 10,000 measurements, but we again report the average of 50 trials. For throughput, each trial measures the time it takes to transmit 10,000 application data packets for the small packet case or 1,000 packets for the large packet case, and measure on the sender side from sending the first byte to receiving a 4-byte acknowledgement when the receiver accepts the last byte. Again, we report the average of 50 trials. The small and mixed packets sizes were both small enough to be transmitted using RPC's UDP protocol binding (which has a payload limit of 8KB), so we included those results, as well. The Infopipes were written in C and used the TCP socket protocol. Two node benchmarks moved data from one node to another. Three node benchmarks moved data from node 1, to node 2, then to node 3 (one NIC per node).

Table 1. Latency in microseconds for data transfer between two compute nodes.

	Small	Mixed	Large
Infopipe	172.1	249.5	462.5
RPC-TCP	164.6	307.3	670.0
RPC-UDP	160.6	271.0	N/A
Hand	137.5	191.6	381.8

Table 2. Throughput (Mbps) for data transfer between two compute nodes.

	Small	Mixed	Large
Infopipe	27.38	265.07	398.03
RPC-TCP	0.17	3.92	140.77
RPC-UDP	0.18	4.39	N/A
Hand	30.16	301.45	450.76

Table 3. Latency (μ s) to transfer across three nodes.

	Small	Mixed	Large
Infopipe	385.1	473.3	799.9
RPC TCP	463.9	440.8	1337.4
RPC UDP	455.2	439.1	N/A
Hand	380.2	557.0	674.9

Table 4. Throughput (Mbps) on three node benchmark.

	Small	Mixed	Large
Infopipe	35.24	166.13	270.17
RPC TCP	0.07	2.47	73.61
RPC UDP	0.08	4.64	N/A
Hand	16.17	234.72	272.26

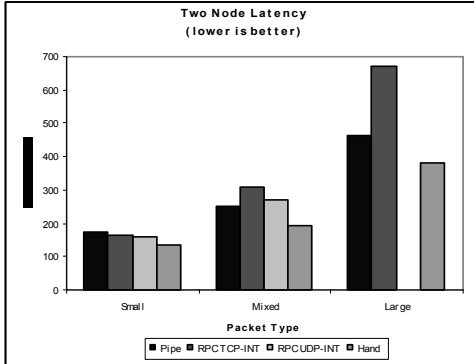


Figure 1. Latency, two nodes.

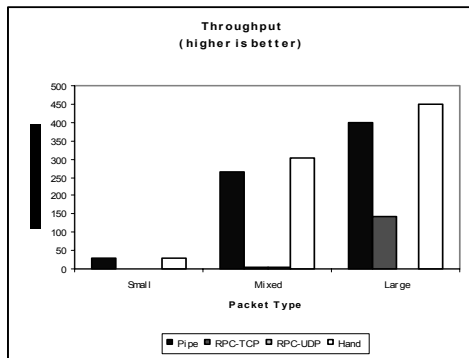


Figure 2. Throughput, two nodes.

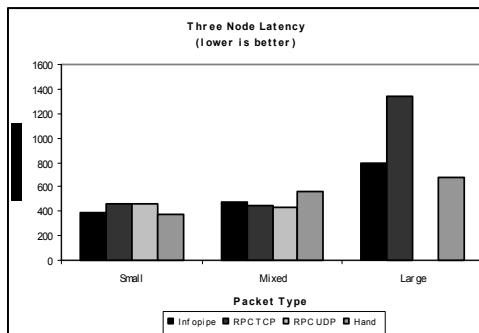


Figure 3. Latency, three nodes.

As we can see, the Infopipes implementation compared favorably in these benchmarks to both hand-written code and to SunRPC code. The high throughput of our implementation stems from two advantages. First, the Infopipes TCP version relies on native representation to send data. Since the SunRPC always converts to the XDR byte ordering, it incurs extra overhead that negatively affects its throughput, latency, and jitter. The second advantage of Infopipes is they do not demand response to a data send event as RPC does. Therefore, the sending Infopipe need spend no time

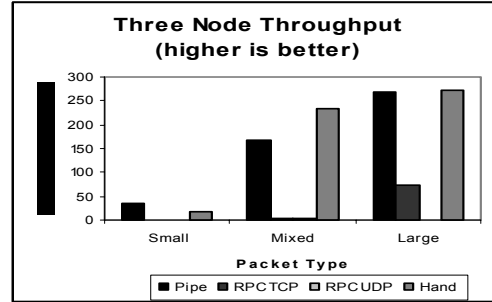


Figure 4. Throughput, three nodes.

processing return values as the RPC client must, and more importantly, the Infopipe, even with small payload sizes, can fill the outgoing TCP buffer and enjoy efficiency gains that arise from sending larger size packets. (It should be noted, however, that the same number of calls were made in each case to the communication layer.)

Our second experiment is an evaluated Infopipes for an image streaming application. This application is representative of many distributed information flow applications with bulk, continuous data streaming requirements. The application stems from DARPA's medium-sized demonstration scenario developed for the PCES project in which the application gathers video and image inputs from several sources, processes them, and redistributes the flows to several destinations including video displays and automatic target recognition programs. We use two variations of the program. In the first version, "Plain," the application simply moves an image, which is about 400KB in size, from the sender to the receiver. In the second version, "Grayscale," the sender sends the image first to a remote RPC server or Infopipe that grayscales the image, and then forwards the new image downstream to the receiver.

Table 5. Latency in ms for "Plain" and "Grayscale".

	Plain	Grayscale
Infopipe	7.05	17.23
RPC	7.47	17.33

Table 6. Throughput (Mbps) for "Plain" and "Grayscale".

	Plain	Grayscale
Pipe	644.88	826.08
RPC	471.81	206.03

Again, the Infopipes version delivers comparable latency but superior throughput as compared to the SunRPC version in the "Grayscale" variant. This time the Infopipe implementation benefits from its natural asynchronous semantics, which allows sending Infopipe to transmit data without waiting for the Grayscale Infopipe to finish its transformation.

5. Related Work

Remote Procedure Call (RPC) [1] is the basic abstraction for client/server software. By raising the level of abstraction, RPC facilitated the programming

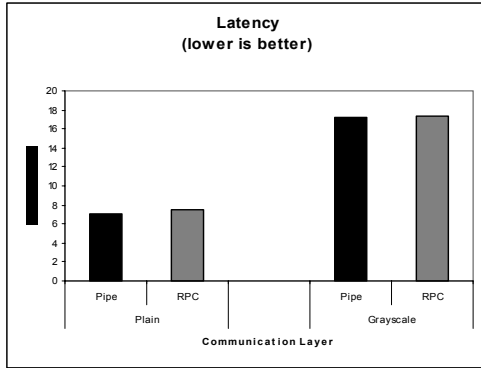


Figure 5. Latency for transmitting an image as “Plain” between two nodes and as “Grayscale” which involves three nodes.

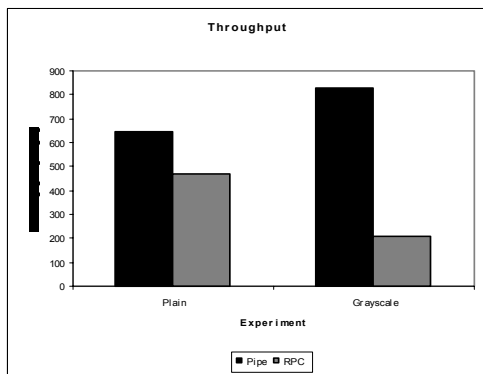


Figure 6. Throughput for transmitting an image as “Plain” between two nodes and as “Grayscale” which involves three nodes.

of distributed client/server applications. Despite its usefulness, RPC provides limited support for information flow applications such as data streaming, digital libraries, and electronic commerce. To remedy these problems, extensions of RPC such as Remote Pipes [8] and AVstreams for CORBA were proposed to support bulk data transfers and sending of incremental results. Still, RPC is usually a poor fit for distributed systems with information flows.

Infopipes have commonality with the dataflow model. Projects which touch on this computing model and code generation include Ptolemy, as mentioned, the SACRES project, and Spidle [9], [10], [11].

6. Conclusion

In this paper, we outlined the motivation for a high level abstraction called Infopipes [2], [3], [4] for distributed information flow applications such as multimedia streaming and digital libraries. Infopipes are defined by, and programmed with, a family of domain specific languages called Infopipe Specification Languages (ISL) which is compiled into executable code. The main technical contributions are results from an experimental evaluation of ISG-generated code that demonstrates the advantages of a information flow

abstraction such as Infopipe. We have written a realistic distributed application, where streaming sources uses a bandwidth-limited network. Measurements of microbenchmarks as well as application-level code show that ISG generates high quality code, with minimal additional performance overhead compared to programs hand-written in a general purpose language (C). When compared to SunRPC, results indicate that ISG-generated Infopipe code has comparable latency and significantly better throughput.

7. References

- [1] Birrell, A. and B. Nelson: “Implementing Remote Procedure Calls”, in *ACM Transactions on Computer Systems*, Vol. 2, No. 1, February 1984, Pages 39-59. Also appeared in *Proceedings of SOSP’83*.
- [2] L. Liu, C. Pu, K. Schwan and J. Walpole: “InfoFilter: Supporting Quality of Service for Fresh Information Delivery”, *New Generation Computing Journal* (Ohmsha, Ltd. and Springer-Verlag), Special issue on *Advanced Multimedia Content Processing*, Vol. 18, No. 4, August 2000.
- [3] R. Koster, A. Black, J. Huang, J. Walpole and C. Pu. “Infopipes for Composing Distributed Information Flows”. In the *Proceedings of the ACM Multimedia Workshop on Multimedia Middleware*, Ottawa, Canada, October 2001.
- [4] Pu, C., K. Schwan, and J. Walpole, “Infosphere Project: System Support for Information Flow Applications”, in *ACM SIGMOD Record*, Volume 30, Number 1, pp 25-34, (March 2001).
- [5] P. Pal, J. Loyall, R. Schantz, J. Zinky, R. Shapiro, J. Megquier, “Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration.” *ISORC*, March 2000.
- [6] Swint, G., Pu. C. “Code Generation for WSLAs using AXpect. *International Conference on Web Services*.” July 2004.
- [7] Merillon, L. Reveillere, C. Consel, R. Marlet, G. Muller, “Devil: An IDL for Hardware Programming”, in *Proceedings of the 2000 Conference on Operating System Design and Implementation (OSDI)*, pp 17-30, San Diego, October 2000.
- [8] Gifford and N. Glasser, “Remote Pipes and Procedures for Efficient Distributed Communication”, in *ACM Transactions on Computer Systems*, Vol. 6, No. 3, August 1988, Pages 258-283.
- [9] C. Consel, H. Hamdi, L. Réveillère, L. Singaravelu, H. Yu, C. Pu. “Spidle: A DSL Approach to Specifying Streaming Applications”. *LABRI Research Report 1282-02*
- [10] Benveniste, A., B. Caillaud and P. Le Guernic. “From synchrony to asynchrony.” In J.C.M. Baeten and S. Mauw, editors, *CONCUR’99, Concurrency Theory*, 10th International Conference, vol. 1664 of *Lecture Notes in Computer Science*, 162-177. Springer V., 1999.
- [11] Amagbégnon, P., L. Besnard, P. Le Guernic. *Implementation of the data-flow synchronous language SIGNAL*. PLDI 1995.