XPORT: Extensible Profile-driven Overlay Routing Trees*

Olga Papaemmanouil, Yanif Ahmad, Uğur Çetintemel, John Jannotti, Yenel Yildirim†

Department of Computer Science
Providence, RI 02912, USA
{olga, yna, ugur, jj, yenely}@cs.brown.edu

ABSTRACT

XPORT is a profile-driven distributed data dissemination system that supports an extensible set of data types, profiles types, and optimization metrics. XPORT efficiently implements a generic tree-based overlay network, which can be customized per application using a small number of methods that encapsulate application-specific data filtering, profile aggregation, and optimization logic. The clean separation between the "plumbing" and "application" enables XPORT to uniformly and easily support disparate dissemination applications.

We demonstrate the basic XPORT system, featuring its extensible optimization framework that facilitates easy specification of a wide range of commonly used performance goals. We demonstrate iterative tree transformation protocols that allow XPORT to continuously optimize its operation to achieve these goals under changing network and application conditions. We will use two different underlying applications, an RSS feed dissemination application and a multiplayer network game, along with visual system-monitoring tools to illustrate the extensibility and the operational aspects of XPORT.

1. INTRODUCTION

XPORT (eXtensible Profile-driven Overlay Routing Trees) is a generic profile-driven distributed data dissemination system. It is designed to provide the core dissemination infrastructure for a growing set of dissemination-based applications and services, including web feed dissemination (RSS/Atom), multicast-based content distribution, massively multiplayer network games and stock ticker distribution, and large-scale distributed collaborative applications.

Dissemination-based applications often exhibit diverse application logic and performance requirements. At the same time, they all require several *common* core facilities, which include dissemination overlay construction, maintenance and optimization, (content-based) routing logic, and membership management. These applications are often developed from scratch, requiring substantial effort and investment to "get it right" for each specific case. In contrast to the existing approaches that provide point solutions to point problems, XPORT's goal is to develop an application-agnostic solution that can be easily customized and extended for a specific target application through a small number of methods that encapsulate application-specific behavior and optimizations.

Extensibility is the central design consideration for our system, which supports an extensible set of data and profiles types, and optimization metrics. Specifically, XPORT supports two types of extensibility. *Profile-related extensibility* refers to the ability to easily accommodate new data and profile types, and is key to supporting diverse applications. *Cost-related extensibility* refers to the ability to express application-specific performance goals, and allows applications to define their own criterion of an efficient and effective dissemination system. Given application-defined data and profile types, and performance metrics, XPORT automatically builds, maintains, and optimizes an overlay dissemination tree consisting of the available broker machines in the system.

This demonstration will introduce XPORT, highlighting its basic optimization framework, which uses a novel metric-independent multi-level aggregation model to define system cost. The model allows the uniform specification of many commonly used system cost metrics, as well as new ones, through combinations of different aggregation functions and local cost metrics. During run time, the system iteratively applies tree transformations in order to converge to a minimal cost configuration, optionally subject to constraints (e.g., "minimize the total bandwidth consumption in the system while ensuring that the dissemination latencies do not exceed 100ms"). XPORT provides a set of primitive transformation rules, which can be composed to obtain more powerful ones.

2. XPORT API

Based on the common functionality of data dissemination systems, we identified two types of methods an application needs to define in XPORT, *profile-related* and *cost-related* methods. For simplicity of exposition, we abstractly describe these methods, without providing their full signatures or semantics.

Profile-related methods. These methods specify how profiles are processed, stored, indexed and maintained at each broker of the system. Message matching is specified by the method **match(**m**,** p**)** that returns true if a data message m matches a profile p or false otherwise. Our API provides a **merge(**p**,** q**)** function that allows an application to define how two profiles p and q are merged to a more general profile covering p and q. It also allows applications to integrate an index structure by specifying three methods: (i) an **init()** method for declaring and initializing the index structure; (ii) an **add(**p**)** method that adds a new profile p to the index; and (iii) a

*This work is supported in part by the NSF under grants IIS-0325838 and IIS-0448284.

†Author performed the work while he was a summer intern at Brown University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.
Copyright 2006 ACM 1-59593-256-9/06/0006 ...$5.00.

System Cost Function	Node Cost Function	Example Metrics
Additive	Additive	total (average) path latency
	Bottleneck	total path bandwidth bottleneck
Bottleneck	Additive	maximum path latency
	Bottleneck	min path bandwidth bottleneck
Holistic	Additive	variance of path latency
	Bottleneck	variance of bandwidth bottleneck

Table 1: Combinations of aggregation functions

remove(p**)** method that removes a profile p from the index. In this case method **match(**m**,** ind**)** is used that returns the set of profiles in the index ind matching the data message m.

Cost-related methods. XPORT allows applications to specify their own performance criteria for the dissemination network created. Our system uses a *two-level aggregation model* to specify the system cost. The first level computes the cost of each node as an aggregation of an application-defined cost metric. The second level computes the cost of the whole system by aggregating the node costs. Similarly, applications can also specify constraints for each node. A more detailed description can be found in [5]. We now describe the cost-related methods.

An application defines the system performance metrics, which we refer to as the **system cost**. This is an aggregation of the node cost over all nodes:

aggregate (system cost function, node cost, system cost set)

We define three categories for the system cost function: (i) *additive functions* (`SUM`, `AVERAGE`), (ii) *bottleneck functions* (`MIN`, `MAX`) and (iii) *holistic functions* (`VARIANCE`, `STANDARD DEVIATION`, `PRODUCT`). The *system cost set* allows us to aggregate the cost of all nodes or clients. Our definition permits applications to define a large variety of system cost measures, like minimum bandwidth capacity, total bandwidth consumption, average path latency, etc.

The **node cost** is defined as: (i) an application-defined local value, (ii) an aggregation of the local values of some neighboring nodes or (iii) a combinations of metrics defined as the node cost. This combination could be any function, while the aggregation function is defined as:

aggregate (node cost function, local value, node cost set)

The *node cost function* can be either a additive function or a bottleneck function. *Node cost set* defines which neighbors' costs we will aggregate. The local value can be a metric referring either to the node itself (*e.g.*, CPU usage) or to its links with its neighbors (*e.g.*, latency to the parent). Example node cost metrics are the path latency and outgoing bandwidth consumption.

Constraints can be defined as:

constraint (metric, operator, threshold)

Constraints are defined in the same way as the system cost (*i.e.*, following the two-level aggregation model), and providing additionally a threshold for the constrained metric. For example, an application might want to impose an upper bound on the path latency of every node. XPORT customizes its functionality and optimization framework to respect these constraints. Example metrics are shown in Table 1.

2.1 XPORT Optimization Framework

XPORT strives to create overlay trees that benefit application-specific cost functions. Periodically, it modifies the tree structure using local transformations to adapt to time-varying network or workload conditions. We informally define a *local transformation* as one that requires interactions among only the "nearby" brokers on the overlay tree. We refer to these brokers collectively as an *optimization unit*. Instead of performing an exhaustive search of all configurations, XPORT limits its search to a smaller set of "promising" transformations. This set contains a number of built-in *primitive transformations* as well as other composite transformations defined by an application.

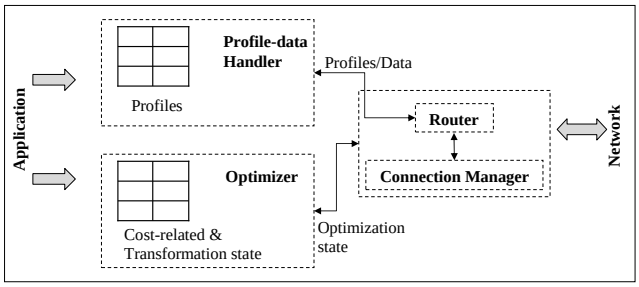

Figure 1: XPORT node architecture

The two primitive transformations of XPORT are *child demotion* and *child promotion*. The first one migrates a node along with its subtree under one of its siblings and the second one moves a node and its subtree under its grandparent. The goal of these transformations is to improve the overall system cost. XPORT calculates, through the use of general formulas that hold regardless of the transformation type or system metric, the impact of a transformation on the system's performance metric, and applies the best network reconfiguration. Details on our framework can be found in [5].

2.2 System model and node architecture

Our model is based on the publish/subscribe paradigm [3, 4], where sources publish their data and clients express their data interests through their profiles. A client connects to an XPORT node and forwards its profile to it. The new profile is merged with the existing ones and is propagated up the tree towards the root, creating a reverse routing path. Using this routing tree, a broker can now forward data only to its interested children.

Periodically, brokers reevaluate their position in the tree. They consider a set of local transformations of their optimization unit and apply the ones that improve the system's performance.

Node architecture. The high-level architecture of an XPORT node is shown in Figure 1. Applications customize two main system components. The first component is the *data/profile handler*, which is responsible for storing, indexing and maintaining profiles as well as matching them against incoming data. The *optimizer* identifies and applies network transformations given application-specific performance criteria and constraints. The tree transformations to be performed are given to the *connection manager*, which establishes and manages the node's connections with its parent and children. Both the optimizer and the profile/data handler communicate with the node's *router*, which handles all the data and metadata communication.

3. XPORT DEMONSTRATION

The demonstration showcases some of the key operational aspects of XPORT, using two popular dissemination-based applications, an RSS-feed dissemination application and a multiplayer network game.

Specifically, our demonstration shows: (i) XPORT's extensibility in defining new data types and profiles as well as various optimization metrics and cost functions, and (ii) its ability to adaptively optimize system performance based on user-defined metrics and constraints under dynamically changing conditions.

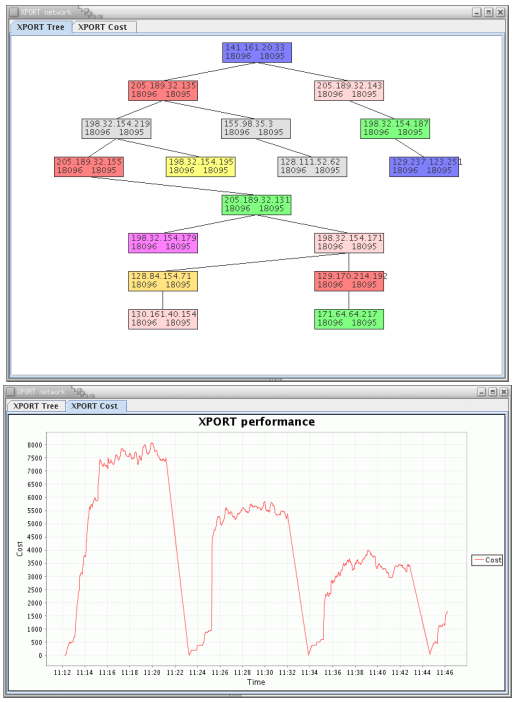

Figure 2: Network and performance visualizers

3.1 System Tools

We use two visualization tools (Figure 2) to demonstrate XPORT's features in addition to the user interfaces of the applications.

The Network Visualizer displays the current overlay network configuration. Furthermore, this tool allows us to inspect the profiles and the local statistics of each broker.

The Performance Visualizer collects statistics from the system nodes and displays the overall system performance. Changes to the system cost are reflected live here, allowing users to observe the effect of optimizations on the system cost.

3.2 Applications

RSS Feed Dissemination. RSS feeds are commonly used by news websites and weblogs to notify subscribers of changes in their content. These feeds are presented and organized by an RSS aggregator that periodically polls the websites to check for news. As a result of such polling, hosting RSS feeds requires unnecessarily high bandwidth requirements. Instead, XPORT could be used to create an RSS feed dissemination tree where clients specify the RSS feeds they wish to receive to an XPORT broker. XPORT brokers self-organize in a profile-based dissemination tree, where the root is responsible for polling the RSS sources, and forwarding the returned feeds through the tree.

Multiplayer Game "Area Of Interest" Dissemination. A massively multiplayer network game involves players pursuing a series of quests and battling other players in a shared-game world. Players continually generate events as they move and interact in the world that must be disseminated to other players. Typically, these games are implemented in a client/server model, and reduce the system's bandwidth requirement by statically partitioning the game world into zones. Zones are used to filter updates between players, allowing communication only if the two players are in the same zone. Each zone represents a player's "area of interest", and denotes the player's profile against which XPORT should match incoming events.

3.3 Applications in Context

We built an RSS feed dissemination application using our API and deployed our prototype on the PlanetLab testbed. We demonstrate XPORT by creating a dissemination tree on PlanetLab sites and fetching any requested feeds from their actual RSS sources in real time. In addition to our predefined set of client profiles, we allow users to specify subscriptions to, and view RSS feeds through an RSS aggregator. The RSS aggregator is connected to an XPORT broker through a proxy. The proxy generates a profile from any subscription request, and forwards the profile through the dissemination tree via the XPORT broker. Upon receipt of an RSS feed, a broker decides whether to forward it to a downstream broker (or a client) using the application's matching function.

We also extended the Sauerbraten game engine [2] to support profiles and event publication. For this application, our demonstration uses a local area network of laptops to run our XPORT brokers. Each player in the game interacts with XPORT in two ways. First, players continuously submit profiles representing an area of interest to XPORT as they move around in the game world. We use a zone based partitioning of the game world to control the assignment of profiles to XPORT brokers. Second, our players continuously send position data indicating motion to the root of the XPORT dissemination tree. Thus, our matching function is spatial containment of the motion events and the area of interests.

In addition to XPORT's visualizers, for the RSS feed dissemination, users will use the RSSOwl RSS Reader [1] to specify the RSS feeds they would like to receive. For our game application, our demonstration uses a game front-end, driven by the Sauerbraten game engine [2]. This front-end presents a visual form of the player profiles by explicitly rendering the area of interests for selected players.

3.4 Setup Details

For both applications, the demonstration highlights the optimization of end-to-end latency and total bandwidth usage via local transformations. We start off with a randomly chosen dissemination tree deployment over PlanetLab sites (or laptops). In our first scenario we artificially constrain the bandwidth available to specific brokers and grow the number of clients, and consequently the bandwidth requirements between system nodes. XPORT automatically detects these changes in network conditions and applies transformations to minimize the total bandwidth usage of the dissemination tree while respecting the links' capacities.

In our second scenario, we look at the latency of disseminated messages, while respecting a constraint on the total outgoing bandwidth consumed by the broker at the root of the dissemination tree. Here we simulate an increased latency on various links in our topology and subsequently transform the tree to gracefully handle the performance degradation. We witness the transformations reducing the depth of the tree, yielding lower latency overlay paths, but stop short of a star topology due to our bandwidth constraint.

4. REFERENCES

[1] RSSOwl RSS Reader, http://www.rssowl.org/.
[2] The Sauerbraten Engine, http://sauerbraten.org.
[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM TOCS*, 19(3), 2001.
[4] Y. Diao, S. Rizvi, and M. J. Franklin. Towards an Internet-Scale XML Dissemination Service. In *VLDB*, 2004.
[5] O. Papaemmanouil, Y. Ahmad, U. Çetintemel, J. Jannotti, and Y. Yildirim. Extensible Optimization in Overlay Dissemination Trees. In *SIGMOD*, 2006.