Scalable High-Performance Event Filtering for Dynamic Multi-point Applications

Douglas C. Schmidt
schmidt@cs.wustl.edu
Department of Computer Science
Washington University
St. Louis, MO 63130, (314) 935-7538

This paper appeared in the 1^{st} International Workshop on High Performance Protocol Architectures, held in Sophia Antipolis, France, December 15-16, 1994.

Abstract

High-performance event filtering reduces the large volume of event traffic processed by dynamic multi-point applications in high-speed networks. We are developing an object-oriented (OO) framework for event filtering based on CORBA, which is an emerging standard for open distributed object computing. The OO framework supports the automated generation, optimization, and configuration of event filters in distributed systems. This paper outlines the key characteristics of dynamic multi-point applications, reviews related work, and describes the features provided by our event filtering framework.

1 Introduction

The demand for high-performance event filtering to support dynamic multi-point (DMP) applications is increasingly. Examples of DMP applications include satellite telemetry processing systems, fault management in large-scale network management systems, real-time market data analysis systems, on-line news services, and distributed agents in mobile personal communication systems.

The work described in this paper is motivated by the high-performance event filtering requirements in several next-generation distributed communication systems. Event filtering is used in these systems to improve the performance of DMP applications such as satellite telemetry processing and automated fault management. The issues and requirements involved in designing and implementing filtering for other DMP applications (such as real-time market data analysis systems) are very similar.

The structure of a typical DMP application is shown in Figure 1. These applications typically exhibit the following characteristics:

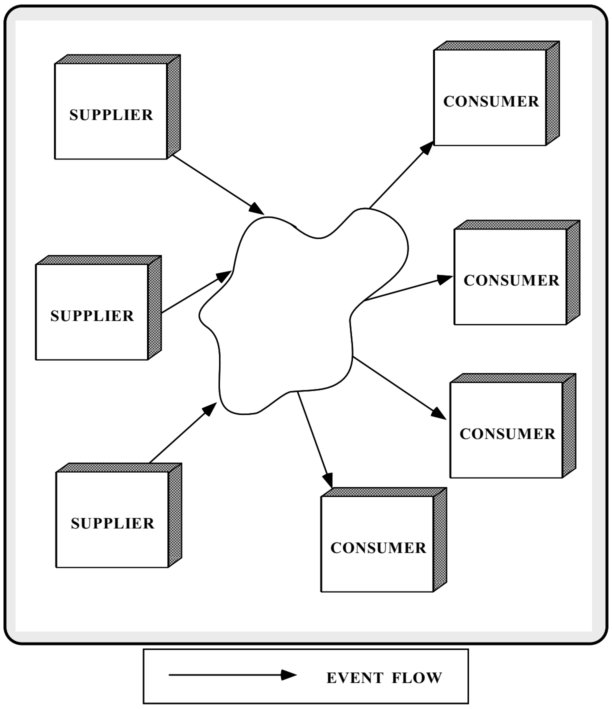

Figure 1: General Structure of a Dynamic Multi-point Application

• **A high volume of event messages are generated continuously in real-time by one or more suppliers:** In communication systems, these events originate from suppliers (such as satellites), as well as from traps generated by managed objects (such as gateways and earth terminals).

• **The message formats of events are potentially complex:** Message formats contain both header fields (*e.g.,* timestamps, source/destination addresses, routing ids, priority levels) and data payload (*e.g.,* vectors of telemetry measurand values).

• **Zero or more consumers subscribe to a subset of the total events generated by the supplier(s):** Unlike traditional static multi-point applications (such as teleconferencing), each generated event may be received by a different subset of consumers, depending on consumer subscriptions and message contents. Filtering based upon message content may involve complex demultiplexing requires.

• **Event consumers reside on hosts that are geographically distant from event suppliers:** Moreover, consumers and suppliers are often separated by very high-latency communication links ,such as low-earth-orbit (LEO) and geosynchronous-orbit satellites, that exhibit high variation in delay.

• **Events may have different priorities and consumers may need to receive events in priority order:** This characteristic implies that some form of priority scheduling may be necessary to meet stringent DMP application performance requirements (*e.g.,* the automated event correlation in fault management subsystems).

• **Consumers may add, delete, or modify their subscriptions dynamically:** Thus, low setup/removal latency may be necessary to achieve the high levels of dynamism required to monitor, detect, and recover from resource failures that occur during system operation.

DMP applications are *multi-point* since any event that matches a subscription is delivered to the corresponding consumer. Likewise, DMP applications are *dynamic* since (1) each event potentially may be delivered to a different subset of consumers and (2) consumers are capable of updating their subscriptions at run-time in response to changes in their environment. In contrast, conventional multi-point applications (such as teleconferencing) generally exhibit much less dynamism. For instance, once a consumer has joined a multipoint group in an ATM switch, all data sent by a supplier is received by all other consumers who are members of the group [1].

Since consumers in DMP applications do not subscribe to every event, aggregate system performance may be enhanced significantly via *event filtering*. Event filtering is a data reduction mechanism that eliminates unnecessary network traffic and unnecessary processing at consumer endsystems. In addition, filtering is used to demultiplex and classify events, which supports network monitoring and automated fault management.

Events may be filtered based on characteristics such as event type, event value, event generation time, event frequency, and event state changes. An event filter may contain relational operators, which enables arbitrarily complex filter expressions to be composed. To improve performance and flexibility, it may be necessary to install filters at various locations (such as event sources, destinations, and/or intermediate nodes) throughout a distributed system.

The remainder of this paper is organized as follows. Section 2 outlines and evaluates related research on event filtering; Section 3 describes the optimization techniques we are developing to support scalable, high-performance event filtering for DMP applications; and Section 4 presents concluding remarks.

2 Research Background

2.1 Related Work

Work on event filtering spans a number of domains, including distributed systems [2, 3], network management and network monitoring [4, 5, 6, 7, 8], user-level communication protocols [9, 10, 11, 12], and active databases [13, 14, 15]. This section outlines relevant related work on event filtering and evaluates this work in terms of its support for the characteristics of DMP applications outlined in Section 1.

2.1.1 Distributed Systems

• **Isis News:** Isis [3] supports event filtering in its *Reliable Distributed Objects* (RDO) News service [2]. In Isis RDO News, filtering is performed at the destinations since all consumers in a process group receive all events sent by suppliers. Consumer filtering is limited to matching on character strings "keywords." Depending on the available network bandwidth and endsystem processing capacity, the RDO News filtering architecture may not scale to accommodate a large numbers of consumers that possess complex filtering requirements.

2.1.2 Network Management

• **HP OpenView:** HP OpenView [4] provides an implementation of the ISO OSI event report management services [7]. HP OpenView filtering allows *Event Forwarding Discriminators* (EFDs) to be installed on remote agents in a network. An EFD contains an expression that filters events based on their type, value, generation time, and/or frequency. EFD filter expressions are described using GDMO, which is a schema for defining managed objects in a network. OpenView's implementation of OSI EFDs is inadequate for high-performance event filtering. It suffers from a highly inefficient process architecture that requires 4 context switches and 3 interprocess communication exchanges to receive, filter, and deliver each event end-to-end.

2.1.3 Packet Filters and Packet Classifiers

The majority of related experimental research has focused upon various types of *packet filters* (also known as *packet classifiers* [12]). Packet filters were developed originally to support efficient demultiplexing for user-level implementations of network protocols [9]. In addition, they have been used to monitor traffic unobtrusively on promiscuous-mode networks [6, 5, 8].

The CMU/Stanford Packet Filter (CSPF) [9] and the Berkeley Packet Filter (BPF) [10] are two influential first-generation packet filter tools. Second-generation tools (such as the Mach Packet Filter (MPF) [11] and the PathFinder packet classifier [12]) enhance the scalability of first-generation tools by enabling the composition of multiple filters.

• **CSPF and BPF:** CSPF is a stack-based packet filter that operates on binary instructions (*e.g.,* `pop` and `push`). CSPF uses boolean expressions, along with a tree model, to configure its filtering engine. BPF extends and enhances CSPF to overcome certain performance limitations with CSPF. BPF uses a register-based execution model and and acyclic control graph, instead of CSPF's stack-based language and tree graph, respectively. In addition, BPF extends CSPF by handling packet fragmentation, delivery of out-of-order fragments, and variable-length headers.

Both CSPF and BPF maintain a list of configured packet filters (the list maybe reorganized dynamically to move frequently accessed filters to the front of the list). This approach works well if there are relatively few packet filters, or if only a small number of consumers are active simultaneously. When there are hundreds of filters and/or hundreds of consumers, however, this approach does not scale up since the time required to filter packets grows linearly with the number of filters.

• **MPF:** The Mach Packet Filter (MPF) is designed to support user-level implementations [16] of layered protocol stacks (such as TCP/IP). In this context, composing multiple packet filters helps to reduce demultiplexing overhead. Often, packets destined for separate connections on an endsystem contain a common prefix, followed by variation at a single point in the packet. For example, active TCP connections on an endsystem will have many IP header and TCP header fields in common (such as source and destination IP addresses). In this case, the only demultiplexing field that varies among arriving packets is the TCP destination port number. Thus, MPF associates a hash table with this field to demultiplex packets to different endpoints efficiently.

The MPF composition technique assumes the existence of a common prefix across protocol headers. This assumption enables low latency setup and removal of composite packet filters. However, this particular optimization strategy does not generalize to compose more complex filters.

• **PathFinder:** The PathFinder tool described in [12] presents a more general technique for coalescing filters with common prefixes. PathFinder is a packet classifier that combines software and hardware techniques to optimize the composition of complex filtering patterns. The software portion of PathFinder builds a directed-acyclic graph (DAG) interpreter based upon patterns specified by a user-defined declarative syntax. The PathFinder interpreter matches fields of incoming packets using information stored in the DAG.

PathFinder uses several novel features to support high-performance packet classification. For example, it adaptively reorders the DAG when pattern matches a composite pattern. This reordering process synthesizes and caches a new customized representation that reduces the number of comparisons for subsequent packets matching the pattern. In addition, PathFinder also offloads a portion of the packet classification logic into hardware, which may run on a network adapter board.

One limitation with PathFinder is the relatively high latency required to setup/remove patterns from the DAG (since the DAG must be searched from the root to the leaves when inserting/deleting a new pattern). In addition, the software implementation of the DAG uses an interpreter, rather than a compiler, which precludes several performance optimizations described in Section 3.2.1.

2.2 Evaluation of Related Work for DMP Applications

Most of the widely available systems described above focus on packet filtering techniques for communication protocol stacks. This section describes several key differences between the environment and requirements of emerging dynamic multi-point (DMP) applications and the protocol stacks supported by conventional packet filters.

2.2.1 Message Formats and Filter Programming Notations

Packet filters were originally designed to operate on protocol headers ranging from the transport layer to the data-link layer (*e.g.,* TCP/IP over Ethernet or ATM). These headers contain simple data types that have remained fairly stable over time. Therefore, packet filter techniques have been optimized to work well for relatively simple message formats that consist primarily of fixed-length fields containing integral values. In addition, filters are often programmed using "assembly-language" syntax that supports low-level features such as bitwise masking (which is necessary to extract bit-fields used by protocol designers to minimize network bandwidth at the expense of addition parsing overhead).

In contrast, the data formats of DMP event messages are often more complex and more diverse than the protocol headers supported by traditional packet filters. For example, filtering telemetry event messages involves comparing floating point numbers, variable-length arrays, and complex nested structures. Moreover, these message formats tend to evolve and change much more quickly than traditional protocol headers. Therefore, DMP applications typically require more flexible, higher-level notations to define message formats and to program filter expression logic.

2.2.2 Event Filtering Criteria

Packet filters typically support "stateless" filtering criteria based on simple equality and relational comparisons (such as matching the value of a field in a packet against a particular value). In general, support for "state-based" comparisons in traditional packet filters has been *ad hoc*, focusing primarily on filtering IP packet fragments [11, 12].

In contrast, DMP applications often require state-based filtering. For example, most satellite telemetry filtering is based upon changes of state (such as a telemetry measurand exceeding an aperture constraint by a specified threshold). Other state-based filtering criteria is based on frequency and time (such as detecting a particular error condition occurring x times over a certain period).

2.2.3 Scalability

Conventional packet filter tools [2, 9, 10, 11, 12] do not provide direct support for filtering other than at the destinations. This architecture is not scalable across large-scale DMP applications, where a large number of suppliers and consumers may exist. Real-time market data analysis systems used by Wall Street brokerage firms are an example of DMP application that require a high degree of scalability. These applications may be required to filter events for hundreds or thousands of geographically remote traders managing diverse portfolios in real-time.

2.2.4 Dynamism

Many DMP applications require the ability to dynamically add, delete, or update consumer subscriptions rapidly in response to changes in external conditions. For instance, the automated network fault management correlation systems install new filters in response to alarms triggered by managed objects in a high-speed network [15]. In general, conventional packet filtering tools do not provide comprehensive support for this degree of dynamism. PathFinder [12] does support the configuration of a secondary filter when a related primary filter matches (this is used to handle IP fragments). However, the functionality of the secondary filter is determined when the pattern is originally configured.

2.2.5 Priority Scheduling

DMP consumers may subscribe for events that have different levels of priority. For instance, consumers may require certain events (such as critical faults and recovery procedures), to be reported within specific time constraints. This time limit may be negotiated during subscription time [17]. In conventional packet filters, incoming packets are buffered in FIFO order at the end of device driver input queues, regardless of packet priority. However, to support real-time event filtering for DMP applications, queueing and filtering must be scheduled based upon the priority of an event.

2.2.6 Protection Domain

Existing packet filter implementations are based on interpreters. This is due largely to the kernel-resident environment of traditional packet filters. To reduce security and reliability risks, most operating systems do not provide convenient ways of compiling and dynamically linking filter code into an OS kernel. In contrast, DMP applications generally run in user-space rather than in an OS kernel. Therefore, optimization techniques (such as compilation, explicit dynamic linking, and parallelism) are often more feasible.

3 Optimization Techniques for High-performance Event Filtering

We are developing an object-oriented event filtering framework that is designed to satisfy the requirements of DMP applications discussed in Section 2.2. The framework is based on OMG CORBA [18] and OMG common object services [19], which are emerging standards for open distributed object computing.

This section outlines work in progress on the framework, which currently consists of research prototypes. When complete, the framework will support the generation, optimization, and flexible configuration of high-performance, scalable, and extensible event filtering mechanisms using the techniques described below.

3.1 Event Filter Generation

To simplify and automate the description, processing, and optimization of event filters, the following high-level filter schema notation and declarative filter expression language are being developed.

3.1.1 Event Schema Notation

A schema is used to define the fields in an event. In our system, event schemas are described using a superset of the OMG CORBA interface definition language (IDL) [18]. CORBA IDL provides a simplified form of the C++ type system, but is not a full-fledged programming language.

CORBA IDL is similar in functionality to standard data definition languages such as ASN.1 and XDR. It provides support for basic types (such as long, short, char, and double), as well as composite types (such as fixed-length arrays, bounded and unbounded sequences, and discriminated unions). We are extending CORBA IDL to support packed bit-fields to support existing packet formats (such as TCP and IP headers).

The following interface illustrates the CORBA IDL syntax used to define the schema for events in a distributed logging service [20]:

```
// IDL schema definition
interface Logger
{
```

```
  // Types of logging messages.
  enum Log_Priority {
      LOG_DEBUG,   // Debugging messages
      LOG_WARNING, // Warning messages
      LOG_ERROR,   // Errors
      LOG_EMERG    // A panic condition
  };

  // Format of the logging record.
  struct Log_Record {
    Log_Priority type;
    long         length;
    long         time;
    long         app_id;
    sequence     msg_data<char>;
  };
};
```

The framework IDL compiler automatically translates this interface into an intermediate representation that may be interpreted directed or translated and optimized into a more efficient execution format (discussed in Section 3.2.1).

3.1.2 Filter Expression Language

This language enables the declaration of *filter expressions*. Filter expressions may contain relational operators, which enables the composition of arbitrarily complex filter expressions.

A filter generation compiler is used to transform a filter expression into executable code. By using a declarative language, the compiler will automatically detect inconsistencies between event schemas and filter expressions. This relieves a DMP application developer from focusing on low-level details of filter formats. In addition, the filter generation compiler may automate various filtering optimizations (such as the various techniques for composing filters described in Section 3.2.1).

The filter expression language enables filtering based upon stateless criteria (such as event type, event value, event generation time), as well as criteria that must retain state (such as event frequency and event state changes). The expression language is a superset of C++ expressions. The keyword `this` refers to the currently active message. Comparisons involving message history state are indicated by using the "`{ }`" operator. This operator may contain a range of values that refer to the previous messages (*e.g.,* `this->field{0..9}` refers to the value of a field in the current and through the previous 9 messages).

Several short examples illustrating the use of the filter expression language are shown below. The fields accessed in the filter expressions correspond to the `Logger` IDL interface defined in Section 3.1.1.

```
// This filter matches if 10 consecutive
// error messages are sent from an
// application with an id of 2001.
"this->app_id == 2001
 && this->type{0..9} == Logger::LOG_ERROR"

// This filter matches if the absolute value
// of the length between two consecutive
// messages from application 2010 differ
// by more than 100 bytes.
"this->app_id == 2010
 && abs (this->length{0} -
        this->length{1}) > 100"

// This filter matches if the time stamp
// of the message is between noon and 1
// pm.
"this->time >= 12:00:00
 && this->time <= 13:00:00"
```

3.2 Event Filter Optimizations

The OO framework supports two general types of optimizations for DMP applications: *filtering optimizations* and *transmission optimizations*. Filtering optimizations are designed to reduce the time required to determine the subset of consumers that have subscribed to receive an event message. Transmission optimizations are responsible for delivering a message with minimal overhead once the subset of consumers has been identified. Several filtering and transmission optimizations being investigated for the framework are described below.

3.2.1 Filtering Optimizations

We are currently investigating a number of optimization techniques to enable scalable filtering of DMP application events. Three promising techniques discussed below include parallel processing of composite filters, trie-based filter composition, and context-free grammar-based filter composition using "skip-ahead parsing." Existing work has investigated several of these optimization techniques in the context of communication protocols [21] and packet filtering [12, 22]. However, there appears to be no comprehensive comparative study investigating all three techniques in the context of DMP applications.

• **Parallel processing of composite filters:** We are using shared memory multiprocessors (*e.g.,* a 20-CPU SPARCcenter 2000) to improve event filtering performance. In this approach, one or more filter expressions registered by a consumer are associated with each processing element (PE). When a message arrives it is stored in shared memory accessible to the PEs. Each PE then compares its collection of filter expressions against the relevant fields in the message. Messages that match filter expressions are subsequently delivered to the appropriate consumers.

An advantage of using parallelism is the relative ease of configuring and reconfiguring event filters in response to dynamically changing consumer subscriptions. A more challenging issue involves selecting the process architecture to use as the basis for parallelization. Task-based process architectures (which bind the PEs together with different filtering tasks or service layers) perform poorly on shared memory multiprocessors. Message-based process architectures (which bind the PEs with the individual messages or connections to consumers) generally perform much better [21]. However, empirical tests are necessary to determine the circumstances under which types of message-based process architectures (*e.g.,* Connectional Parallelism or Message Parallelism) provide the highest performance.

One limitation with the current generation of parallel processing platforms is the cost of delivering each message to all PEs attached to the shared memory. A fairly high cost is generally incurred when accessing shared memory on standard I/O buses [23].

• **Trie-based Filter Composition:** We are also exploring optimization techniques that compose multiple filter expressions together using dynamic trie-based data structures. This trie-based approach is a generalization of the DAG-based techniques presented in [12]. Every node in the trie implements a particular type of branching mechanism. The branching mechanism selected at a node employs a lookup function (such as a hashing function, a binary search, or a series of inlined relational expression comparisons). During trie construction, the appropriate type of lookup function is selected based upon the type and the range of data values in a node.

For tries that contain a large amount of common prefixes, this technique reduces the number of operations required to process M messages of size L through N event filters from $O(M \times L \times N)$ to $O(M \times L)$. This represents a substantial performance improvement when N becomes large (which is often the case for DMP applications containing hundreds or thousands of consumers).

• **Context-free Grammar-based Filter Composition:** A related optimization involves the use of automated compiler parsing and code generation techniques. In this approach, a finite automaton is created by combining a set of context free grammars that describe the composite filter expressions. A novel technique known as "skip-ahead parsing" [22] is used to eliminate unnecessary access to data fields within a message. An advantage of this approach is that the generated code may be optimized using compiler optimization techniques.

One drawback to using compilation is the relatively high latency required to compile and install, modify, or delete an event filter. OS support for explicit dynamic linking [24] helps to reduce this overhead to some extent.

Preliminary experiments on prototypes of our OO framework indicate that the appropriate choice of filtering optimization is affected significantly by the degree of dynamism required by a DMP application. Applications that tolerate high installation latency (such as SV telemetry processing, whose filtering constraints are generally fixed at initialization-time) benefit from the performance improvements from compiler-based optimizations. Conversely, applications that require frequent interactive updates (such as fault management applications initiated by network administrators) benefit from the parallel-based or trie-based schemes due to their lower (re)configuration latency.

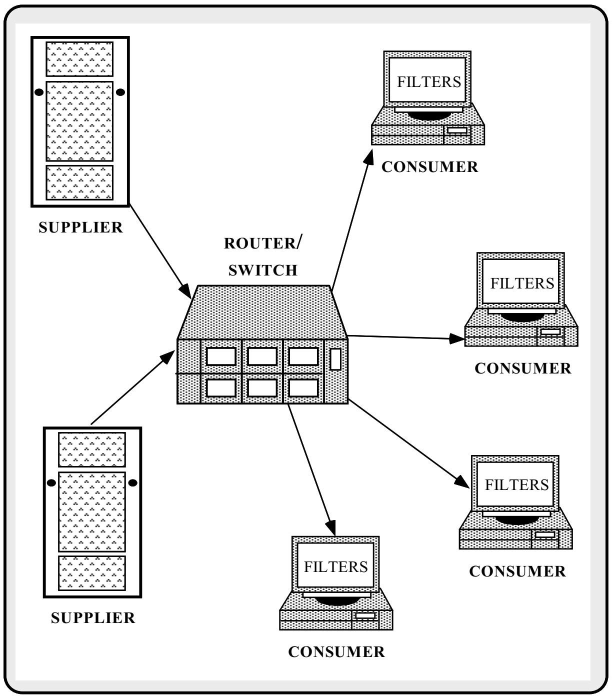

Figure 2: Decentralized Event Filtering

3.2.2 Transmission Optimizations

Applying the composite event filters to a message produces a multicast set that contains the address(es) of the consumer(s) for each message. Once this multicast address set is determined, additional transmission optimizations may be performed. The goal is to minimize the number of network packets used to deliver event messages to consumers. These optimizations take advantage of lower-level networking mechanisms (such as reliable transport multicast and ATM switch multicast) to minimize network traffic. For instance, ATM switch multicast uses port recycling and range-copying optimizations [1] to reduce the amount of copying necessary to multicast messages through an ATM switch. In turn, this helps to reduce unnecessary network traffic and avoid excessive event processing at endsystems.

3.3 Event Filter Configuration

Our OO event filtering framework is designed to allow the flexible configuration of filters at various locations (such as event sources, destinations, and/or intermediate nodes) throughout a distributed system. This leads to several types of event filtering architectures illustrated in Figures 2, 3, and 4. This section discusses the advantages and disadvantages of each architecture.

• **Decentralized Event Filtering:** In certain DMP environments, it is beneficial to decentralize event filtering by performing it on consumer hosts (shown in Figure 2). This configuration is appropriate when the following pre-conditions exist:

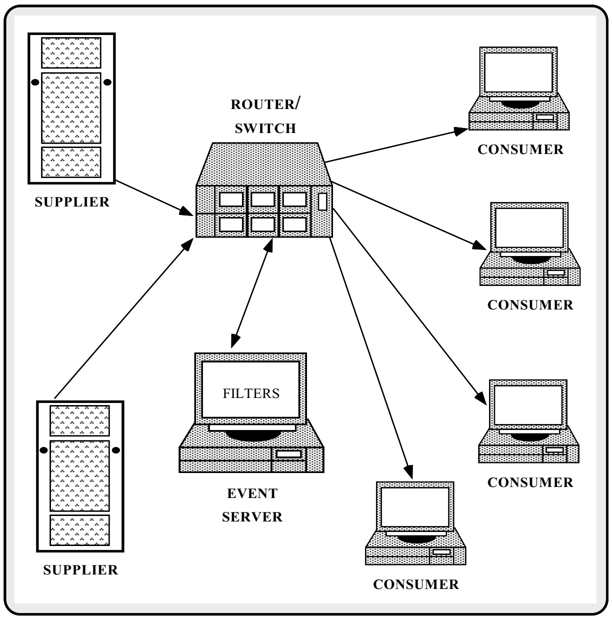

Figure 3: Centralized Event Filtering

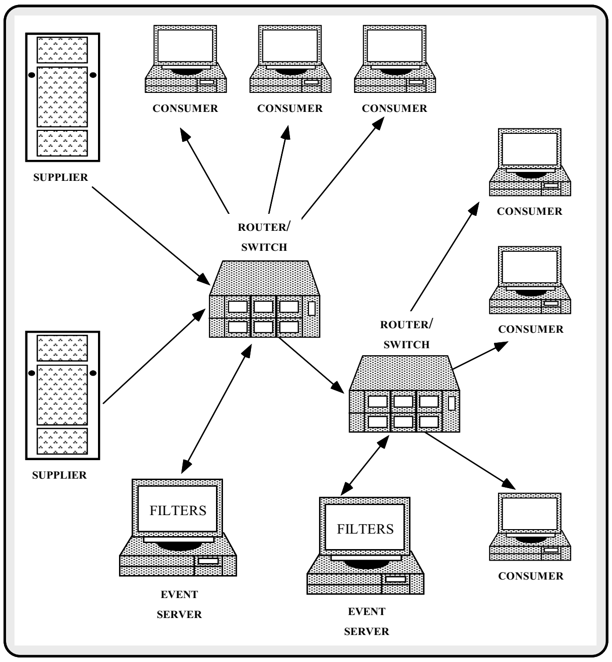

Figure 4: Distributed Event Filtering

- the consumer hosts are powerful workstation platforms;
- a high-speed network is available to connect the suppliers to the consumer hosts;
- consumers subscribe to most events;
- event filters are relatively complex.

When these conditions exist it may become more efficient to perform filtering in the consumer endsystems.

Centralized Event Filtering: In other DMP environments, it is beneficial to centralize the event filtering at a single event server (shown in Figure 3). This configuration is appropriate when the following conditions exist:

- an event server is installed on a high-performance platform (such as a multi-processor);
- the consumer hosts are run on less powerful platforms (such as inexpensive PCs);
- a relatively low-bandwidth (or highly congested) network connects the event server to the consumer hosts;
- consumers subscribe to a relatively limited subset of events;
- the complexity and number of event filters subscribed to by consumers does not produce a major processing bottleneck at the event server.

When these conditions exist, the network and the consumer hosts at the edges of the network are typically the processing bottleneck, rather than the event server. Therefore, a centralized event filtering architecture helps to off-load work from the network and the consumer hosts.

Distributed Event Filtering: More complex event filtering scenarios are also possible (shown in Figure 4). For example, network topology in complex systems may interconnect suppliers, event servers, and consumers that span multiple routers and switches, across local-area networks, as well as wide-area networks.

Conventional techniques [25, 26] for configuring flexible software systems possess performance limitations since they install distributed application services by spawning heavyweight processes and connecting these processes via heavyweight IPC mechanisms (such as pipes and sockets). In contrast, our OO framework supports flexible lightweight (re)configuration mechanisms based upon explicit dynamic linking and lightweight processes (threads) [24].

4 Concluding Remarks

This paper describes research being conducted at Washington University on optimizations for high-performance event filtering. Efficient event filtering is crucial to reduce the large volume of event traffic processed by dynamic multi-point (DMP) applications (such as satellite telemetry processing and automated detection and recovery). Unlike related work, our optimizations are targeted for the requirements of DMP applications. Existing frameworks for event filtering (such as Isis, HP OpenView, DCE, CORBA, and packet filters) do not adequately address the usability, extensibility, performance, and scalability requirements of DMP applications.

To improve usability and extensibility we are developing an object-oriented framework for event filtering based on CORBA [18, 19]. The framework supports the automated generation and optimization of filter expressions based on the CORBA interface definition language (IDL). By selecting CORBA, we plan to leverage off emerging distributed object development environments [27] and open system protocol specification techniques [28].

To improve DMP application performance and scalability, we are exploring optimization techniques to reduce event filtering overhead. These techniques employ parallel processing, dynamic trie-based search structures, and compiler technology based on context-free grammars to reduce the time required to filter messages exchanged by DMP applications in a distributed system.

Acknowledgements

Portions of the research on optimizations for event filtering are being conducted in conjunction with George Varghese and Ron Cytron at Washington University.

References

[1] J. Turner, "An optimal nonblocking multicast virtual circut switch," in *Proceedings of the Conference on Computer Communications (INFOCOM)*, pp. 298–305, June 1994.

[2] Isis Distributed Systems, Inc., Marlboro, MA, *Isis Users's Guide: Reliable Distributed Objects for C++*, April 1994.

[3] K. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*. Los Alamitos: IEEE Computer Society Press, 1994.

[4] K. S. Klemba, "Openview's Architectural Models," in *Proceedings of the* 1^{st} *International Symposium on Integrated Network Management* (B. Meandzija and J. Westcott, eds.), pp. 565–572, IFIP, 1989.

[5] R. T. Braden, "A Pseudo-machine for Packet Monitoring and Statistics," in *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, (Stanford, CA), ACM, August 1988.

[6] Sun Microsystems, Inc., Mountain View, CA, *NIT(4P); SunOS 4.1.1 Reference Manual*, Part number: 800-5480-10 ed., October 1990.

[7] I. O. for Standardization, "Information Processing Systems - Open Systems Interconnection - Part 5: Event Report Management Function," Tech. Rep. ISO/IEC DIS 10164-5, ISO.

[8] J. C. Mogul, "Efficient Use of Workstations for Passive Monitoring of Local Area Networks," in *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, (Philadelphia, PA), ACM, Sept. 1990.

[9] J. C. Mogul, R. F. Rashid, and M. J. Accetta, "The Packet Filter: an Efficient Mechanism for User-level Network Code," in *Proceedings of the* 11^{th} *Symposium on Operating System Principles (SOSP)*, November 1987.

[10] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *Proceedings of the Winter USENIX Conference*, (San Diego, CA), pp. 259–270, Jan. 1993.

[11] M. Yuhara, B. Bershad, C. Maeda, and E. Moss, "Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages," in *Proceedings of the Winter Usenix Conference*, January 1994.

[12] M. L. Bailey, B. Gopal, P. Sarkar, M. A. Pagels, and L. L. Peterson, "Pathfinder: A pattern-based packet classifier," in *Proceedings of the* 1^{st} *Symposium on Operating System Design and Implementation*, USENIX Association, November 1994.

[13] N. Gehani, H. V. Jagadish, and O. Shmueli, "Compose: A System for Composite Event Specification and Detection," in *Book chapter in Advanced Database Concepts and Research Issues* (N. R. Adam and B. Bhargava, eds.), Lecture Notes in Computer Science, Springer Verlag, 1994.

[14] S. Gatziu and K. R. Dittrich, "Detecting Composite Events in Active Database Systems Using Petri Nets," in *Proceedings of the* 4^{th} *International Workshop on Research Issues in Data Engineering: Active Database Systems*, IEEE, February 1994.

[15] O. Wolfson, S. Sengupta, and Y. Yemini, "Managing Communication Networks by Monitoring Databases," *IEEE Transactions on Software Engineering*, vol. 17, pp. 944–952, Sept. 1991.

[16] C. Maeda and B. Bershad, "Protocol Service Decomposition for High-Performance Networking," in *Proceedings of the* 14^{th} *Symposium on Operating System Principles*, (Asheville, North Carolina), ACM, December 1993.

[17] S. F. Wu and G. E. Kaiser, "On Hard Real-Time Management Information," in *Proceedings of the* 1^{st} *International Workshop on System Managment*, (Los Angeles, CA), IEEE, Apr. 1993.

[18] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 1.2 ed., 1993.

[19] Object Management Group, *Common Object Services Specification, Volume 1*, 94-1-1 ed., 1994.

[20] D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching," in *Pattern Languages of Program Design* (J. O. Coplien and D. C. Schmidt, eds.), Reading, MA: Addison-Wesley, 1995.

[21] D. C. Schmidt and T. Suda, "Measuring the Performance of Parallel Message-based Process Architectures," in *Proceedings of the Conference on Computer Communications (INFOCOM)*, (Boston, MA), pp. 624–633, IEEE, April 1995.

[22] M. Jayaram, R. K. Cytron, D. C. Schmidt, and G. Varghese, "Efficient Demultiplexing of Network Packets by Automatic Parsing," in *Submitted to the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation*, ACM, 1994.

[23] P. Druschel, M. B. Abbott, M. Pagels, and L. L. Peterson, "Network subsystem design," *IEEE Network (Special Issue on End-System Support for High Speed Networks)*, vol. 7, July 1993.

[24] D. C. Schmidt and T. Suda, "An Object-Oriented Framework for Dynamically Configuring Extensible Distributed Communication Systems," *IEE/BCS Distributed Systems Engineering Journal (Special Issue on Configurable Distributed Systems)*, vol. 2, pp. 280–293, December 1994.

[25] J. Magee, N. Dulay, and J. Kramer, "A Constructive Development Environment for Parallel and Distributed Programs," in *Proceedings of the* 2^{nd} *International Workshop on Configurable Distributed Systems*, (Pittsburgh, PA), pp. 1–14, IEEE, Mar. 1994.

[26] J. M. Purtilo, "The Polylith Software Toolbus," *ACM Transactions on Programming Languages and Systems*, To appear 1994.

[27] C. Horn, "The Orbix Architecture," tech. rep., IONA Technologies, August 1993.

[28] D. Lea and J. Marlowe, "PSL: Protocols and Pragmatics for Open Systems," Tech. Rep. 94-0369, Object Management Group, September 1994.