WhiteBoard P2P: A Peer-to-Peer Reliable Data Dissemination Infrastructure for Collaborative Applications*

Panayiotis Neophytou, Alexandros Labrinidis, Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
{panickos, labrinid, panos}@cs.pitt.edu

Abstract

In this paper we present a new Peer-to-Peer approach for enabling data dissemination, using filtering techniques, that provides support for many types of collaborative applications. Our architecture, called WhiteBoard P2P, is designed to act as an innovative distributed data stream processing system which is vendor-independent and technology-independent. Our infrastructure can support the interconnection of existing legacy systems as well as new systems, and allows dynamic joining and leaving of collaborative participants on a need-to basis. Furthermore, our system is designed to support mobile and ad-hoc networks that are unstable, by allowing disconnected operations while enabling dynamic restructuring of the network as required. We are currently testing our system in the context of disaster management, as part of the Secure-CITI project.

1. Introduction

Reliable message delivery has become a hallmark feature of many data systems, as has the scalable delivery of data. The major goal of any data delivery system is to provide both of these major features, along with perhaps other features such as security. All of these features become even more vital when the application is for time critical applications. Messages stating where and what is happening, along with on the fly updates need to be received by the right people at the right time (i.e. in real time). Currently, the way in which these features have been addressed varies according to the system architecture in place. In general, there are two predominant architectures for current system designs that provide some of the required functionality. These architectures are the client/server architecture and the peer-to-peer (P2P) system architecture. Both of these designs have their benefits and their weaknesses.

The client/server design distinguishes between two types of nodes: servers which are responsible for maintaining the data and responding to queries, and clients which are the ones issuing requests for data. Examples of such designs/systems are CORBA [27] and Grid [12]. Such designs are more rigid with where data is served from and where requests for data are sent. However, these also provide a high level of security, as being able to centralize the authentication of messages and message delivery. The client/server architecture also tends to be reliable in the delivery of messages and data throughout the network, as the central server can record data delivery and keep track of missed or unable to be delivered messages. The design has its drawbacks, however, when it comes to flexibility and adaptability to network load and network configuration. Also, it provides a single point of failure and contains a single bottleneck, which prohibits scalability.

The P2P design, on the other hand, accomplishes many things the client/server design lacks. P2P systems [22, 26, 31, 21, 20] are strong in adaptability, scalability, and fault tolerance, since the system and data are distributed across many sites, so the failure of one does not dramatically affect the network. However, since P2P systems are distributed by nature, they tend not to be very secure, and also lack some of the rigid structure that would be needed to impose hierarchy into the system. P2P systems are also not usually concerned with the reliable delivery of data to all nodes. Many P2P systems rely on best effort in both the propagation of requests and the returning of responses. Furthermore, most of the systems involve searching for data, which again is performed in a lossy manner, where not all nodes are necessarily included. This will not be acceptable in a system that wants to include the ability to send messages to all nodes and guarantee that all the nodes receive those messages.

We are currently developing a disaster management sys-

*Funded in part by NSF ITR Medium Award (ANI 0325353).

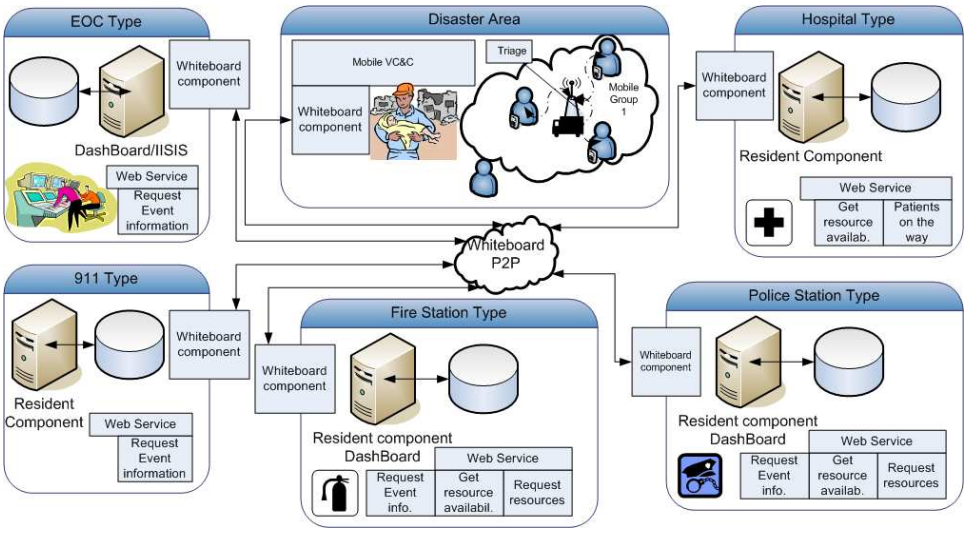

Figure 1. Design of overall Disaster Management System

tem, which none of the above approaches support. It is a collaborative system built on top of a distributed P2P infrastructure.

A distributed means of communications is needed in order to coordinate the large number of emergency responders. Such an infrastructure must be robust, reliable, resilient to failures and adaptable to the application needs, supporting nodes joining and leaving, new roles to become part of the teams etc. We observed that a lot of communication is needed to take place and simple radio style broadcast would not work. Centralized approaches have a single point of failure and in a disaster environment you cannot rely on having such a point. A completely distributed and flat architecture would not work either because of the hierarchical nature of all the participating organizations. Such a hierarchical organization can be seen in almost every existing organizational structure nowadays. For these reasons, we developed an adaptable hybrid P2P architecture that is able to support hierarchical organization on the fly as needed.

Figure 1 shows the basic interactions that must be able to take place when a wide-spread disaster occurs. It illustrates that there are resources from several sources that will need to get together in order to manage what is happening. This diagram also shows the need, however, for an infrastructure which can handle the mass delivery of messages to a wide variety of sources while adhering to whatever constraints may be in place. This infrastructure, especially based on the sensitivity and importance of the information, must provide the functionality mentioned above of scalability, reliability and security.

In this paper we propose a new architecture for data delivery aimed at solving issues that arise when disaster management is taking place by combining the best of both approaches mentioned above into a single approach while minimizing the weaknesses each original design has. We accomplish this with a design which makes its basis a P2P system, but interjects additional security and reliability checks on data. The system is also designed to act as a continuous query processing system working on streams of data that come from the nodes in the network. This is a new paradigm for P2P systems as well, since our architecture envisions continuous queries to be registered at nodes and, if necessary, report data to the client that requested the data, but in the case of many nodes requesting that data, to facilitate multi-casting of the data throughout the network.

We follow the two tiered architecture which has become popular lately in several P2P systems [30, 25, 9], where some nodes will be super peers within the system and thus act as servers for a group of leaf nodes, but interaction between the super peers is completely done in a P2P fashion. We designed the system to act as a whiteboard program which can also act as a wrapper for legacy systems throughout the network and interact through standard APIs with any top-level component. Thus, the proposed architecture is vendor and specific technology independent.

We present our current architecture which is an innovative data stream processing system, which builds upon and extends current systems like [2, 5, 8, 23], called the Whiteboard Peer to Peer (P2P) which supports an incident-driven, adaptive structuring of control and information flow among components.

In the next section we review the current efforts for collaborative systems. Sections 3 and 4 introduce our new architecture. We present open issues in Section 5 and conclude in Section 6.

2. Related Work

Many forms of collaborative environment paradigms have been developed since the emergence of computer networks. Typically these are referred to as multi-tier architectures. Technologies and platforms that enable the development of collaborative software environments today include CORBA, Grid Middlewares, J2EE, .NET etc.

In general, CORBA [27] "wraps" code written in another language into a bundle containing additional information on the capabilities of the code inside, and how to call it. These wrapped objects can then be used by other programs (or CORBA objects) over the network simply by calling them. In this sense CORBA can be considered as a machine-readable documentation format. A very similar approach in describing the capabilities of the code is used in Web Services [29].

Grid computing [12] offers a model for solving massive computational problems using large numbers of computers arranged as clusters embedded in a distributed telecommunications infrastructure. Grid computing's focus on the ability to support computation across administrative domains sets it apart from traditional distributed computing.

Java 2 Enterprise Edition [19] and Microsoft's .NET [18] platforms both present a platform-independent software development environment, with many built-in features for the development of collaborative software. J2EE uses several technologies including JDBC and CORBA, and extends their functionality with Enterprise Java Beans, Java Servlets and XML technologies. .NET framework supports over 40 languages and technologies. Currently Sun's J2EE is fully available in many platforms but .NET is currently only available on Windows.

Generally, in Peer-to-peer network systems there existed three phases of evolution. The first generation had a centralized indexing service where all the information concerning the state of the network was kept in a single point. The most popular example of this was Napster. The second generation followed a fully distributed approach where each node was responsible of keeping it's own state. Learning about other nodes' state is done by a peer, by sending a large amount of messages to it's neighbors and those messages are propagated further to those peers' neighbors until a reasonable amount of results about others were returned. Gnutella is a popular example of this architecture. The third generation was called hybrid systems because they combine features from both of the mentioned architectures. In this generation an upper class of nodes, called super-peers, are responsible for the propagation of messages and lower class nodes, called leaf-nodes, are connected to one or more super-peers and only send and receive messages from them. The most popular implementation of this network is Gnutella2. We believe that new generations of Peer-to-peer networks will include more levels/classes of nodes in an evolving network. This will enable a more structured way of message propagation for efficiency and organization of the nodes inside administrative domains.

Another class of P2P networks, called structured networks, was developed to cope with the increased redundancy that unstructured systems imposed. A very popular structured P2P system is Chord [26]. Chord divides the peers into a circle and assigns ids on that circle using a hash function. It also hashes data items on that circle and the preceding node of that item's id is the node who either should store the item itself or indexing information on the item. Although Chord achieves handling of queries in $O(\log n)$ hops it reduces the autonomy of the peers. The system outperforms the unstructured systems on keyword searches. A lot of variations on handling failures, load balancing and other issues were developed and the system is also used as the base for other kinds of applications too. Other structured systems were also implemented like Tapestry [31], Pastry [22], CAN [21] etc. which are based on the same idea of dividing the space into sections and assigning data or indexing information to the peers.

As described in [10], the metaphor behind a whiteboard (aka blackboard) architecture is a group of specialists gathered around a physical whiteboard, cooperatively working to solve a problem. The whiteboard is used as a workspace where all initial data and current contributions are written.

The whiteboard architecture is intended to facilitate coordination of cooperating software modules and human participants [11]. First conceived in the 1970s for the speech recognition system Hearsay-II [17], it is well suited to many other problem solving tasks and to system or agent control. There have been many applications since Hearsay-II in areas such as command and control, process control, data fusion, case-based reasoning, speech recognition, signal and image understanding, planning and scheduling, structure identification, and machine translation [10, 6].

Many architectures use predetermined connections between modules, with the ordering of operations based on data-flow. Problems arise when the specific modules involved are not known in advance or are subject to change, and when no ordering can be determined until runtime. Whiteboards provide an attractive alternative through indirect connections via the whiteboard, which acts as an intermediary. Ordering and module participation becomes dynamically determined at runtime, and fewer communication interfaces need be supported. A disadvantage of standard whiteboards is the need for a control component or arbiter that determines which module can take the next action.

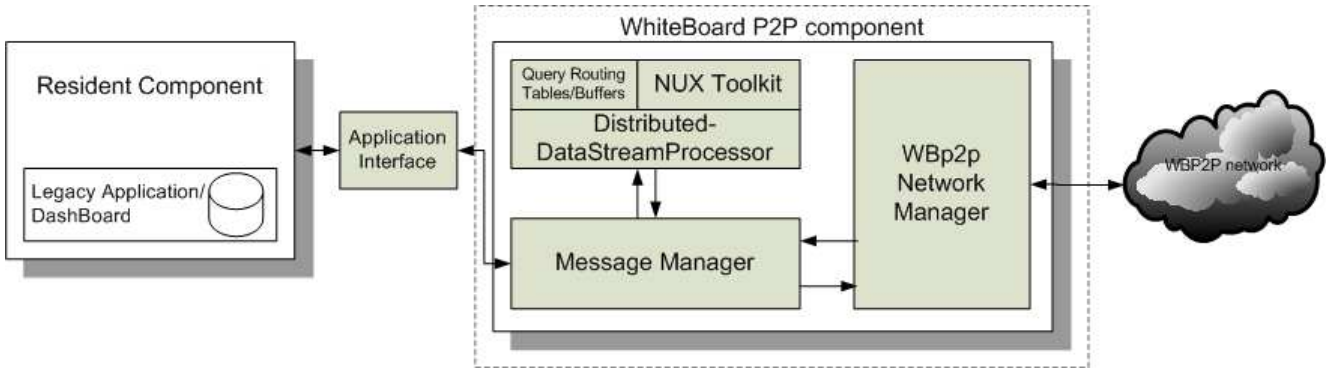

Figure 2. Nodes' core WhiteBoard P2P architecture.

3. Whiteboard P2P Design and Architecture

In this section we will describe the node's architecture that each participating peer must implement and describe the topology of our network. The topology will be forced using protocols and we have designed them in such way that will help answering queries efficiently and accurately. The Data Management part of the architecture will be described in Section 4.

3.1. Node Design

Interconnected nodes is the main ingredient of any P2P system. In order for a node to be able to participate in our system and collaborate with other nodes it must provide certain functionality and follow our protocols. In this section we describe the components that compose a node participating in our network. The overall node architecture is depicted in Figure 2.

3.1.1 Application Interface

The application interface is the means of communication between the User Interface, otherwise known as resident component or legacy application, which could have its own database and functionality. The Resident component, in order to communicate with the rest of the network, must provide two main interfaces. One to submit queries and data to the WBp2p network and one to receive data responses from other peers in the network.

The way data is sent into the network is by applying model management over the available metadata [3] in order to create a representation of the local data to a universally accepted XML format. The WBp2p software provides an internal port to receive these kind of XML messages and forward them through the Message Manager to the network. The queries, using a similar transformation technique which is implemented into the interface, are translated into XQueries [7] again in the form of XML messages.

3.1.2 Message Manager

The message manager is responsible for any actions that have to do with opening and handling of messages. It handles messages from two sources: (a) the rest of the network and (b) the local Resident Component, through the front-end interface.

When a new message comes from the network, it is forwarded to the Data stream processor for evaluation and an acknowledgment message is generated. If the message is intended for this node it will be returned back to the message manager to handle it by passing it to the application interface. If the message is a query request then the query is registered into the Data Stream Processor and a trigger is generated on the Resident Component to generate the data needed by the requestor when they become available. All these assume that the requestor has the necessary access level and trust for the requested data.

Messages generated from the Resident Component are forwarded to the message manager. The message manager adds the credentials needed and then forwards the message to the Data Stream Processor to decide to which connections the message should be forwarded.

3.1.3 Data Stream Processor

The Data Stream Processor (DSP) keeps Continuous Queries registered on the node. Each query is associated with a number of connections that currently are active between this node and other peers. Every message coming from the message manager is evaluated over the queries. Then the DSP forwards the message along with the union of the connections of the satisfied queries to the Network Manager.

3.1.4 Network Manager

The network interface provides the necessary connections needed to route data within the network. It also promotes

the structure of the network. Each node in the system will become part of an overall P2P environment in which it can take any or all of three types of roles (broker, data provider, and data requestor). Additionally, the network interface, in conjunction with the main decision module, helps to make the network adaptable to situations where incidents become highly widespread, thus requiring a more structured chain of command for decision making, or if the network becomes too big and scalability is crucial.

There are three roles that a node can take in our P2P network. The first role is that of a data provider. These nodes usually provide vital information to the decision making process. Data is pulled from the nodes with the use of continuous queries every time new updates are available. These nodes generate a new message for each update and forward it to the network through the registered queries. The second role is that of a data requestor. These nodes send queries to the network to find information that they need.

The nodes that comprehend the backbone of our network are called brokers (third role). These nodes are responsible for the propagation of messages to and from data providers, data requestors, as well as other brokers that are connected to them. Broker nodes also play a large role in the reliable delivery of data. As stated earlier, disconnected operation is available through the whiteboard application. The way this is performed is as follows. When a broker node attempts to send data to either another broker or data requestor, it expects to get back an acknowledgment message from that node. If no acknowledgment message is received, a retransmission is attempted. Two failed attempts cause that node to be marked as disconnected, and the broker begins to store the messages for that peer as it receives them. When the peer comes back online, the super peer will send to it all missed messages. In terms of reliable delivery of data, brokers are only responsible for nodes connected to them. The peer then relies on other brokers to further guarantee the reliable transmission of data. To enhance the chances of reliable message delivery, routing is done using two routes towards the requesting peer. Having these three roles defined, any node can take at most all of these role depending on its needs and capabilities.

3.2. Network Configuration - Topology

The network topology is semistructured based on layers. The layers are used to cluster nodes with common interests and help in efficiently routing messages and queries. The layered topology is closely coupled with an application specific taxonomy of the user roles. To better understand the idea of layers and taxonomies we created a sample topology in Figure 3, following a disaster management scenario. The taxonomy is a hierarchical representation of part of Region 13 in Pennsylvania, which includes the 13 counties around

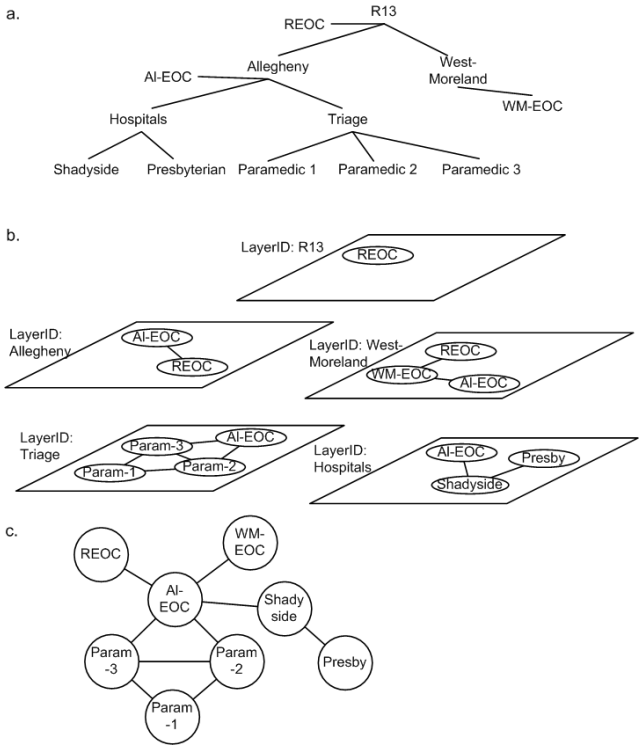

Figure 3. Taxonomy/Topology example in the White-Board P2P network. a. The application's taxonomy. b. How the taxonomy can be mapped into a possible layered topology. c. How the layered topology translates into some global unlayered view.

the Pittsburgh area. Pennsylvania Region 13 covers a population of 3.1 million people over a 9,550 square mile area (http://www.pa-region13.org). Only the leaf nodes are instantiated as nodes and most internal nodes can be layers. Using this taxonomy helps us identify nodes using specific application contexts. A possible layered interconnection between the nodes is shown in Figure 3b. To have a better view of the whole interconnection network we show Figure 3c where we have a global view of the topology without the layers.

One of the major features of the whiteboard network design is its ability to adapt to changing network topology in the face of an increasing range of environmental characteristics, application needs, and dynamics of an incident. This means that as it may start with only one layer, it may soon become known that the original command centers are no longer in charge, that a new level of command centers would now take over. In this case, the nodes that are at this new level of command centers designate themselves as representatives of their current level to the level above or below and establish connections to the nodes that exist at the new

hierarchy level.

Besides the hierarchical overlays that help to better handle message routing according to the application's morphology of data and hierarchical modeling, the brokers of each level are also clustered depending on the data they can serve. For instance certain brokers are known to serve a specific type of leaf-nodes (i.e. hospital nodes), in a certain level of the hierarchy. This knowledge is stored into caches just like [14]. These caches are considered neither consistent nor reliable, but help nodes get a starting point into the network and find their place inside it after querying the network nodes for information. Caches store contact information like IP address and port numbers as well as the layers each node is part of.

4. Data Management

It has become recently widely accepted that the most successful standard for describing and handling data between heterogeneous data sources is XML [28]. A lot of work has been done in the management of XML data in P2P systems. A recent review of these technologies is in [15]. The main innovation of our system is that the queries trigger the creation of a flow of information from the sources. This is extremely useful for monitoring applications that rely on real time data. The flows of data can then be shared by various other peers that are interested in this information thus reducing the number of flows of the same data.

4.1. Data and Query Format

All the data exchanged throughout our infrastructure is expressed as structured XML messages. The messages contain metadata to describe the data inside the message in order to be able to make the necessary updates to a receiving component's local database without having to have the same schema as the sender of the message. We call the language of messages of the Whiteboard P2P as wbXML.

All communication and data transfer in the system is done with data objects that wrap different types of information. These will use a self-contained XML representation that will adhere to newly defined standards that address the application's need or standards already in use and those in development in the emergency management context, for example the NIMS Resource Typing System. The objects will be separated into meta-data and actual content data. The data section will include actual data fields of the object, such as textual information, database records, or pictures. Meta-data includes the category of the object, origin, timestamp, priority, security and credential information, desired delivery guarantees, usage guidelines, and more. It may indicate one or more specific destination peers, or a category of peers (e.g., fire stations). Collectively, this information will facilitate and influence routing decisions at the super peers and act as conditions for continuous query triggers. A more detailed description of the meta-data follows.

- *Category*: This is the basic type of the data object, identifying it as a request, a reply, a status report, etc...
- *Origin*: The peer that submitted this object to the system. This may also indicate the super peer responsible for that peer.
- *Timestamp*: The time the object was created.
- *Priority*: A rating of the urgency and importance of the data object. This also influences delivery of the message through the network. The most critical messages would need high reliability and as much speed as possible, while others may only need high reliability, but could sacrifice some speed, for example.
- *Security and Credential Information*: Includes encryption keys, trust information or tokens required to view the data, possibly allowing more limited, aggregate or summary access in some cases. This is explained in more detail in the following section.
- *Usage Guidelines*: Provides advice on how the data should be used or interpreted, or what it is relevant to. For example, this could indicate the subject of a picture object.

An example of the XML data our messages contain is displayed in Figure 4. This example shows a resource report. The meta-data specifies all needed delivery information, as well as aiding in data access and interpretation. The data in this example is a resource description for a search and rescue team. It is an XML document formatted according to the standard NIMS Resource Typing System, mentioned previously.

For multimedia data, there are two categories to account for: (1) static data like pictures, maps, and audio/video clips and (2) dynamic streaming data such as live audio and video feeds. The Whiteboard P2P handles the static multimedia as any other form of data using an XML message wrapper. Thus the deliveries go through the network of super peers and leaf nodes as usual. Streaming data requires a dedicated direct connection between the source and the receiver. This is established by exchanging XML messages through the whiteboard that indicate how and to whom the receiver should connect.

Since all of the data within our system is XML formatted, the most suitable standard for querying the data is the XQuery language [7].

```
<meta-data>
    <category> Report </category>
    <origin> R13:Allegheny:Triage:Station1738 </origin>
    <destination> R13:Allegheny:Al-EOC </destination>
    <timestamp> 200412050937 </timestamp>
    <priority> normal </priority>
    <security>
        <token> <id> 34f5a9 </id> <access> full </access> </token>
    </security>
    <priority> normal </priority>
    <usage> NIMS Resource </usage>
</meta-data>
<data>
    <resource name="Search & Rescue Task Force" cat="S&R" kind="Team">
        <component name="Personnel">
            <metric name="Number in Team">
                <type1> 70 </type1> <type2> 28 </type2>
            </metric>
            <metric name="Training">
                <type1> NFPA 1670 </type1> <type2> NFPA 1670 </type2>
            </metric>
            <metric name="Areas of Specialization">
                <type1> high angle rope rescue, confined space rescue </type1>
                <type2> basic rope rescue </type2>
            </metric>
        </component>
        <component name="Equipment">
            <metric name="Rescue Equipment">
                <type1> Pneumatic Tools, Elec. Tools, Hand Tools, Safety </type1>
                <type2> Pneumatic Tools, Elec. Tools, Hand Tools, Safety </type2>
            </metric>
        </component>
    </resource>
</data>
```

Figure 4. Example XML Message with NIMS-compliant Resource Description

4.2. Data Streams, Query Handling and Information Discovery

All message processing and routing in WBP2P is done using XQueries. There are two channels for message propagation. The PUSH and the PULL channel. Both channels are implemented using XQueries. The first thing a node wants to establish as soon as it becomes part of the network is the PUSH channel. This is the only way other nodes will be able to reach it directly. The PUSH channel is directly associated with the application taxonomy. Based on our motivating application of disaster management (Figure 1 and 3) let us see how `R13:Allegheny:Hospitals:Presbyterian` would establish it's PUSH channel. The node would have to generate the following query:

```
for $dest in
     /wbMessage/meta-data/taxonomyDest
return
 if ($dest/text() == "R13:Allegheny:
             Hospitals:Presbyterian")
   then true();
 else if (starts-with(\$dest/text(),
           "R13:Allegheny:Hospitals:")
   then true();
 else if ($dest/text() ==
                   "R13:Allegheny:*")
   then true();
 else if ($dest/text() == "R13:*")
   then true();
```

This query is to be flooded to all the peers that have the same primary layer as this node and go just one hop beyond to any nodes that are connected to this layer, but it is not their primary layer. This flooding leaves a trail that goes back to the originating node. That is every hop points to the previous hop. This way if a message hits on this query, it propagates back to the originating node (e.g., the EOC nodes for our example). Notice that the second `if` clause covers all the messages intended to go to hospitals. The third one is for messages intended to go to any Allegheny county node and the last one for those messages intended to go to every node in Region 13. Now let's see the PUSH channel query for the `R13:Allegheny:EOC` node:

```
for $dest in
     /wbMessage/meta-data/taxonomyDest
return
 if ($dest/text() == "R13:Allegheny:EOC")
   then true();

 else if (starts-with($dest/text(),
                      "R13:Allegheny:")
   then true();
 else if ($dest/text() == "R13:*")
   then true();
```

This is also propagated to all the nodes that have the same primary layer as the EOC and one hop beyond that. You can easily see that this way all the messages exchanged between layers would be routed through the EOC's layer which is `Allegheny-county`.

As you can see the PUSH channel is a set of helpful queries that propagate the messages to the correct nodes. The PULL channel corresponds to application specific queries that help gather information from various information providers. The PULL queries are propagated through the PUSH channel and are registered along the way, creating a trail from the information provider to the requester. These queries could be simple XQueries, aggregate queries, or even join queries that join information from two or more information providers.

Simple queries can be addressed to multiple information providers (i.e. getting status information from all of the hospitals, where `taxonomyDest=R13:Allegheny:Hospitals:*`). Dealing with these queries is simple. Aggregate and join queries need a defined window to function properly. For instance, if you want the number of patients triaged in the past hour and update this every five minutes you need a buffer that contains an hour's worth of data and evaluate it every 5 minutes against the query. The way we implement this is depicted in Figure 5. The queries have an extra admission query which filters out messages that have nothing to do with the data concerning the actual query. This makes evaluating the aggregate or join query

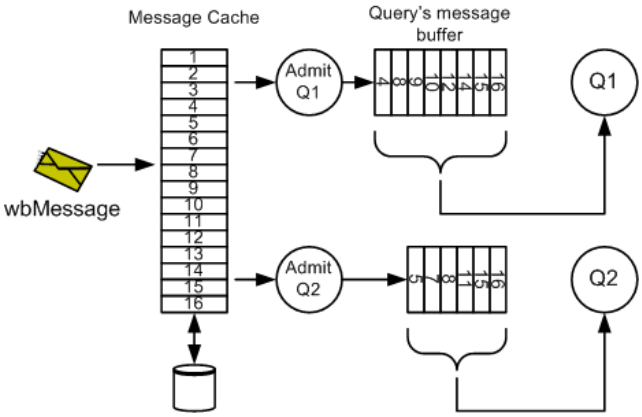

Figure 5. Processing of aggregate/join queries using buffers.

faster. Each of these queries may be evaluated upon each new message admission in the query buffer, or every x time units interval specified with the query's meta-data. In the future we plan on extending the query language to include the epoch between evaluations of the queries, for increased expressibility. Each evaluation that produces data is compiled into a new wbMessage with the results as the payload and a new time stamp etc. If necessary, the generated message may go into the message cache. The message cache is maintained by a global thread which is bound to application-specific rules (i.e. how big the cache should be, how long should the messages be kept for, etc.)

5. Open Issues regarding our design

Although some of the techniques presented thus far have been used in the past to address individual parts of the issues raised by our motivating application, we believe this is the first attempt at a holistic solution that provides security, robustness and high-availability of data, which are very much needed features for many applications.

The new research problems arising from our design are:

Data encryption and message routing using continuous queries seems impossible to handle because to evaluate a query the data must be unencrypted and available to the peer doing the evaluation. A new security model, that will support multiple levels and dimensions of decryption keys that on the one hand will help preserve sensitive data, and on the other hand allow for the evaluation of the basic information needed to correctly route the messages, must be designed.

Preserving trust while routing messages is also difficult, since the interactions that a peer has to make are not only end-to-end. There are many intermediate nodes that handle the data without certainty for who the intermediate node

will be and how it will handle the data. The trust mechanisms have to be extended for this to work.

Another interesting optimization problem is how to efficiently satisfy query routing without having to go all the way to the sources of data, but instead use intermediate, existing streams. That is, if some existing stream is already serving a query that is identical to the one being routed, or produces results that satisfy this query, then there is no need to go all the way to the sources of data. The query can be satisfied from that intermediate point. The bigger problem is to guarantee that all the necessary sources are involved in the current stream. A step further would be to also satisfy queries for which their data requirements are a subset of the data currently being routed.

The current query routing scheme can handle queries to nodes such as: `R13:Allegheny:hospitals:*`, which will serve all the hospitals in Allegheny county. Another possible requirement in query routing would be: how can you send a query in all the hospitals in Region 13? Such a query would be `R13:*:hospitals:*`.

Software Security Layer: After entering the system, encryption, and transmission to the appropriate peer, data availability will be determined at the application level. Each message belongs to a class and by using these classes we define roles for certain types of users/peers. Using these roles, connected users can only access data and messages included in their roles. Specifically, roles define who, from where, what and when someone can access certain classes of information. These roles exist to define the jurisdictional relations between the users of our system, a major requirements for any real-life disaster management system.

Trust management is also critical since the messages are routed throughout the network by several peers. Trust must be established to make sure that sensitive information remains confidential and secure. Although data is encrypted, decryption has to be allowed for some peers to be able to evaluate the queries registered to them and route the messages correctly. Since there has been a lot of work done in this area [4, 13, 24] we have decided to use existing technologies, test them and then modify whatever is needed to suit our needs. The most promising technique from what we have encountered so far is Trust-X [4]. Trust-X is a comprehensive XML-based framework for trust negotiations in peer-to-peer environments. It provides feature like trust tickets and support for different negotiation strategies.

Another requirement that is needed, is to provide guarantees that each request for data is fully evaluated upon the network and the replies received are from all possible sources of the requested data. For example, if the query misses a hospital peer then the decision support that the system provides is degraded and any decisions made will be wrongful. Exhaustive testing of the considered clustering techniques in conjunction with caching and routing mechanisms and the hierarchical topology setup, will show which is the best combination to exhaustively answer any query.

6. System Status and Future Work

In this paper we presented a new architecture for enabling reliable message passing and filtering for disaster response management. Our architecture, which is called Whiteboard P2P, attempts to aggregate the benefits that exist from two predominant architectures, the client/server and the peer-to-peer architecture. The system we are designing acts as an innovative data stream processing system which is vendor-independent and specific technology-independent. We are specifically designing this system to work in conjunction with responding to disasters, by providing quick, easy, and accurate information to a variety of actors in disaster response while enabling coordination following the necessary rules and guidelines already in place in the real world.

Our main design contains several features which are necessary in an architecture which is to be vendor-independent and innovative, such as message formats, system security, and the ability to reliably send and deliver messages while scaling up to high numbers of participating nodes and data/queries. Our message format, which we termed wbXML, adheres to the standard of the NIMS Resource Typing System which allows for easy use in disaster management and contains many features to ease information passing. Security in our system is done at both the software level, through the use of filters, and at the network level, through secure message passing hardware. Finally, our system design for message delivery is intended to not only be highly scalable, but also enable disconnected operation, while allowing for dynamic restructuring of the network as needed.

At the current time, we have begun building our architecture from scratch, and are making good progress towards the full-fledged implementation and deployment of the WhiteBoard P2P architecture. We have successfully been able to establish our own network overlay, which provides reliable message delivery through the use of acknowledgment messages. The system also incorporates the use of XQueries and maps them to connections to accommodate routing. The query manager used is NUX [16]. In this way, the messages are successfully multi-casted out, with leaf nodes only needing to send one message, whereas all nodes requesting such a message are receiving it. This means that we have been able to create a network of peers that is self-forming, able to do filtering at both where data is routed and where data is received, and can reliably send messages. Our future implementation plans include porting the existing components to the JXTA framework (http://www.jxta.org). This will make our implementation adhere to current stan-

dards and make it easier to expand, deploy and integrate with other systems. We did not use JXTA at this stage because the routing and network organization techniques implemented with it do not match our needs. We plan on extending JXTA to accommodate our architecture.

We also see our architecture working as a middleware for integration with new emerging technologies such as the Global Sensor Networks middleware [1] which tries to address the complications of deploying and interconnecting heterogeneous sensor networks. GSN offers virtual sensors as a powerful abstraction which enables the user to declaratively specify XML-based deployment descriptors with the possibility to integrate sensor network data over local or remote data sources. We see our architecture as a means for supporting message delivery, resource discovery and distributed query processing in such environments.

While much has been accomplished in our current implementation, there still remains a lot of work to be done. For example, one of our goals was to enable disconnected operation, which means that nodes which get disconnected will receive vital messages that appeared while they were disconnected. While we have begun implementing this, we have not incorporated it yet into the system. After the system implementation is completed, we will go through a beta-testing phase, where we will need to fine-tune different components of the system e.g., the security levels provided, according to real users' needs. Given the progress so far we expect this to happen in the near future, which will allow for the creation of a complete system that supports reliable message delivery, a fine-grained security model, and intelligent filtering to enhance disaster management.

References

[1] K. Aberer, M. Hauswirth, and A. Salehi. The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical report, 2006.

[2] S. Babu and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 30(3), 2001.

[3] P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[4] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. Knowl. Data Eng.*, 16(7):827–842, 2004.

[5] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams: A new class of data management applications. In *VLDB*, 2002.

[6] N. Carver and V. Lesser. The evolution of blackboard control architectures. *Expert Systems with Applications*, 7(1), 1994.

[7] D. Chamberlin. XQuery: An XML query language, 2002.

[8] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *SIGMOD*, 2000.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Iddues in Anonymity and Unobervability*, 2000.

[10] D. Corkill. Blackboard systems. *AI Expert*, 6(9), 1991.

[11] D. Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Int'l Lisp Conference*, 2003.

[12] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[13] H. Garcia-molina, M. T. Schlosser, and S. D. Kamvar. The eigentrust algorithm for reputation management in P2P networks, Nov. 18 2003.

[14] Gwebcache: http://www.gnucleus.com/gwebcache/, 2003.

[15] G. Koloniari and E. Pitoura. Peer-to-peer management of xml data: Issues and research challenges. In *SIGMOD Record, Vol. 34, No. 2*, 2005.

[16] B. Lab. Nux toolkit: http://dsd.lbl.gov/nux/, 2006.

[17] V. Lesser and L. Erman. A retrospective view of the hearsay-ii architecture. In *Int'l Joint Conference on A.I.*, 1977.

[18] Microsoft. Microsoft .net information. http://www.microsoft.com/net/default.asp.

[19] S. Microsystems. Java 2 platform, enterprise edition (j2ee). http://java.sun.com/j2ee/.

[20] C. G. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172, 2001.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, 2001.

[23] M. Sharaf, P. Chrysanthis, and A. Labrinidis. Preemptive rate-based operator scheduling in a data stream management system. In *ACS/IEEE International Conference on Computer Systems and Applications*, 2005.

[24] A. Singh and L. Liu. Trustme: Anonymous management of trust relationships in decentralized P2P, July 10 2003.

[25] A. Singla, C. Rohrs, and L. W. LLC. Ultrapeers: Another step towards gnutella scalability, version 1.0.26, 2002.

[26] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, 2001.

[27] S. Vinoski. Corba: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 14(2), 1997.

[28] W3C. Extensible markup language (XML): http://www.w3.org/xml/, 1996.

[29] W3C. Web-services standards, 2002. http://www.w3c.org/2002/ws/.

[30] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *ICDE*, page 49, 2003.

[31] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.