LANDMARK-BASED SERVICE RECOVERY FOR DISTRIBUTED PUB/SUB SERVICES ON DISASTER MANAGEMENT

C.-S. SHIH, H.-Y. CHEN, L.-Y. CHEN & L.-J. CHEN
Graduate Institute of Networking and Multimedia, National Taiwan University, Taiwan and Research Center for Information Technology Innovation, Academia Sinica, Taiwan

ABSTRACT
A communication system that supports timely, scalable, and highly available information exchanges over a heterogeneous plug-and-play network systems is called an Open Information Gateway (OIGY). Our work focuses on design for disaster resiliency, i.e. means to provide an OIGY with capabilities to deliver critical messages to parties in a disaster management system even when the underlying physical network has sustained serious damages. This article first presents the design of such an information gateway in general. It then presents the design and implementation of a middleware layer service recovery mechanism, essential for the gateway to achieve the aforementioned objectives. The mechanism is called landmark-based service recovery mechanism. It aims at recovering real-time publishing and subscription services within a disaster affected region. The article describes its design, together with its performance, measured in terms of its transmission and recovery overheads as a function of the number of message brokers in the network.
Keywords: Message publish and subscription, middleware, service recovery.

1 INTRODUCTION

Timely and effective disaster response during and after a disaster requires collaborations among numerous parties, including service providers, decision makers, responders and rescuers, victims, and general public. Making essential information exchange services disaster resilient, capable of delivering critical messages to them on a timely, scalable, and always available basis is of paramount importance. In the last few decades, many special communication devices have been developed and communication channels have been reserved for disaster response and rescue. Examples include satellite phones [1], IP-based 911 [2], and rescue radio [3]. However, the applicability of these technologies has been limited. For example, satellite phones can be particularly effective for rescuers in mountainous areas and over the sea. Unfortunately, transmissions to the satellite can be blocked by thick clouds and heavy rain, making satellite phones ineffective during typhoons [4] and other weather-related disasters. Moreover, high deployment cost typically prevents the inclusion of a satellite phone in every emergency kit.

The work on disaster resilient information on exchange described in this article and in ref. [5] was motivated by information exchange models that exploit all available communication pathways. Experiences in response efforts during recent major disasters show that effective use of all available communication devices and services is a key to successful operations. For example, during 2010 Haiti earthquake, victims trapped in the damaged buildings sent text messages via their cell phones and thus allowed the rescuers to locate them in the left-behind area. When 2009 Morakot typhoon flooded parts of Taiwan [4], heavy damages of communication infrastructure prevented victims from contacting their family members and emergency responders. Their family members posted messages on social network services and marked their possible locations on online maps. The information was broadcast by phones and social

ISSN: 2041-9031 (paper format), ISSN: 2041-904X (online), http://www.witpress.com/journals
DOI: 10.2495/SAFE-V5-N1-13-28

network web services. Rescuers were able to locate many victims using information obtained in this way.

An information exchange framework designed to collect and distribute data and information on a timely, scalable and highly available basis even when parts of the telecommunication network systems have been damaged by disaster is called an Open Information Gateway (OIGY). An OIGY stays responsive by exploiting complementary merits of different network access technologies, approaches, and network types to make the physical connectivity as robust as possible during and after disasters. Section 2 will provide more details on this approach to maintain physical connectivity. The attention of this article is primarily on the OIGY component that is responsible for resilient logical connectivity necessary to facilitate real-time information exchanges among information sources, fusion modules, applications, and end-users. Section 2 will also describe the OIGY component responsible for local connectivity.

The contributions of this work on disaster resilient communication include the middleware-layer service recovery mechanism described here. The mechanism, called landmark-based service recovery mechanism, is designed to keep the gateway responsive and available as much as possible when parts of message delivery services are not available due to damages to the underlying communication network and computing services. Today's communication systems used to support disaster management typically provide static communication services. It can take from a few days to a few months to restore damaged communication infrastructure, and the deployment of temporary communication services typically requires manual efforts by communication experts. In contrast, landmark-based service recovery mechanism is automatic; it makes the message exchange services support by OIGY self-healing. As performance data presented later in this article will show, recovery can take place within minutes and hours.

The remainder of this article is organized as follows. Section 2 presents the structure and key components of OIGY and provides background and motivation of the landmark-based service recovery mechanism. Section 3 presents related works. Section 4 describes the design and implementation of the landmark-based recovery algorithm for publication and subscription services. Section 5 presents the performance evaluation results on the overheads and service delay achievable by the algorithm. Section 6 summarizes the article and discusses the work to be done in the near future.

2 SYSTEM ARCHITECTURE OF OIGY

An Open Information Gateway (OIGY) is designed to provide the desired features of information exchange services mentioned above. Figure 1 shows the system and software architecture of OIGY. In OIGY, information exchange services are provided by a distributed middleware over heterogeneous networks: OIGY services execute on diverse devices and computers, including computationally weak platforms, such as cell phones owned by individuals and computers in convenient stores, supermarkets, schools, etc., as well as computationally powerful platforms, such as cloud servers for weather forecast service, etc. and data repository servers in data centers. When there is no need to be specific, all of them are called points of service (POS) or POS resources. It is also assumed that by arrangements made during disaster preparedness phase, POS resources are pervasively deployed in the affected region for the sake of availability.

To support the aforementioned desired features of information exchange services, this work proposes to deploy OIGY to support distributed timely information exchange over heterogeneous networks. This section presents the methodology and advantages of each component in the system and how they interact with each other.

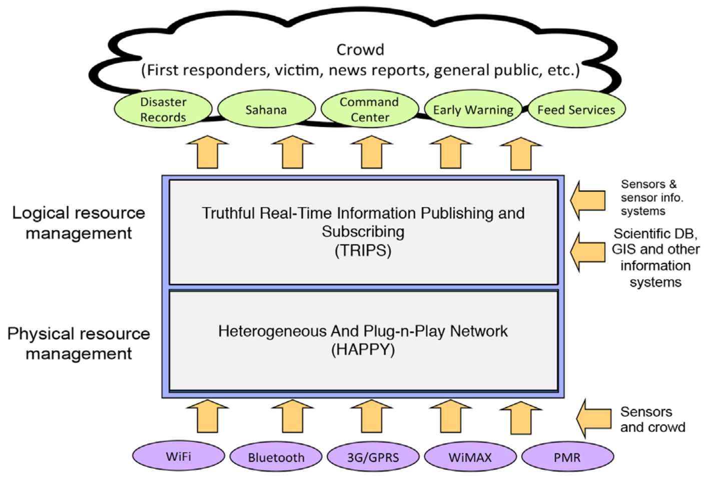
Crowd
(First responders, victim, news reports, general public, etc.)
Disaster Records
Sahana
Command Center
Early Warning
Feed Services
Logical resource management
Truthful Real-Time Information Publishing and Subscribing (TRIPS)
Physical resource management
Heterogeneous And Plug-n-Play Network (HAPPY)
Sensors & sensor info. systems
Scientific DB, GIS and other information systems
Sensors and crowd
WiFi
Bluetooth
3G/GPRS
WiMAX
PMR

Figure 1: System Architecture of Open Information Gateway (OIGY).

As discussed earlier, in response to a disaster, individual users, news agencies, government agencies, and emergency responders, etc. must collaborate with the help of information delivery services. Each of them can be a provider and/or a consumer of data and information. For instance, during emergency, a data center of a government agency may subscribe data from sensors in the disaster affected area and publishes processed and verified data to subscribers including first responders, victims, news reporters, and general public. Victims in the affected area may publish messages to the others and subscribe information from their family members, news media, and government agencies. To achieve scalability, OIGY is designed to be a middleware-layer software component in the communication system. Victims can install OIGY widgets on their cell phones; a weather forecast agency can install an OIGY message delivery service on their powerful servers. In general, one or more OIGY widgets and services for publishing and subscribing information are made available via POS resources throughout the affected region.

OIGY consists of two major components. One is the distributed Truthful Real-time Information Publishing and Subscribing (TRIPS), and the other is the Heterogeneous And Plug-n-PlaY networks (HAPPY). The objective of HAPPY [6] is to maintain physical connectivity. It does so by interconnecting network-capable devices in all possible manners: It makes use of heterogeneous network access technologies (e.g. WiFi, Bluetooth, Professional Mobile Radio (PMR), and 3G/GPRS) and diverse networking approaches (e.g. Infrastructure-based networks, wireless mesh networks, mobile ad hoc networks, and opportunistic networks).

The objective of TRIPS is to support distributed real-time publication and subscription services, hereafter referred to as P/S services. Note that a device/service in the system may be an information publisher, or a subscriber or both. Typical resource limited devices/services act as information publishers only. Sensors used to detect mud-slide, measure rainfall and water level in rivers, and monitor earthquakes are some examples. Services such as weather forecast services and GIS systems subscribe information from sensors, GIS databases, and other information sources and publish their information to the applications and end-users of the disaster management system. While requesting for P/S service, an application specifies

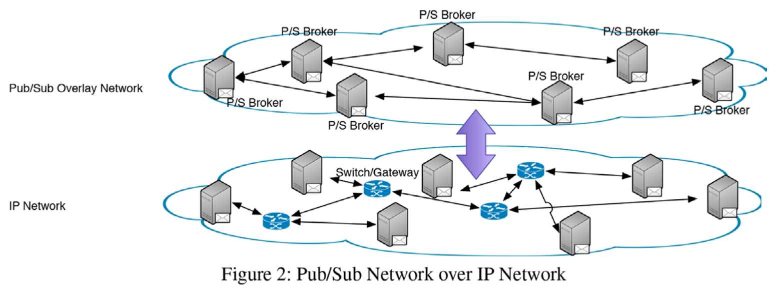

Figure 2: Pub/Sub Network over IP Network.

the Quality of Service (QoS) or Class of Service of the service using QoS parameters such as response time and resource requirements. Upon establishment of a P/S service, TRIPS will register and announce the service. Hence, one capability of TRIPS is to manage service declarations, automatically establishing connections between publishers and subscribers for a matching topics and dynamically detecting new status in the system. When a subscribed message arrives or is sent, TRIPS delivers the message to its subscribers. The component that does this work is called a P/S broker, or simply broker, and a logical network of publishers and subscribers serviced by brokers is called a PubSub network.

Figure 2 illustrates the architecture of a PubSub network over a HAPPY network. In a PubSub network, P/S brokers are responsible for storing and forwarding messages to the brokers and subscribers. Broker nodes (i.e. the POS resources on which brokers run) are connected by HAPPY networks, and the messages are routed over the switches and gateways in HAPPY networks. In the P/S network, a link between two P/S brokers may be a route of multiple IP links and a shorter path on P/S networks may not necessarily be a route with a shorter message transmission delay. Within a HAPPY network, HAPPY agents are deployed on the nodes with multiple communication interfaces to facilitate communication among multiple physical communication networks.

3 RELATED WORK

This section discusses prior efforts closely related to this research, including work on information exchange protocols for distributed message delivery and time decoupling message delivery. Distributed message delivery avoids single-point and any-point failures. Time decoupling message delivery allows the message senders and receivers to be active in different time intervals. These capabilities are essential for information exchanges during disasters.

3.1 Publish/Subscribe model for message exchange

Publish/Subscribe is a messaging model that supports asynchronous and persistent message-oriented communication [7]. This model views the system and applications as consisting of three types of components. The first two types are publisher and subscriber: A publisher is a producer of information, and a subscriber is a consumer of information. The third component is the middleware, which is responsible for delivering information from publishers to subscribers.

Specifically, when a publisher produces a message containing information to be disseminated, it gives the message to the middleware. The middleware routes the message to subscribers of this and related messages. If some subscriber is not active at that time, the middleware buffers the message until the subscriber becomes active and ready for receiving the message. Consequently, in comparison with traditional messaging models, the distinguishing feature of publish/subscribe model is decoupling in the following three dimensions [7].

- Space decoupling: A publisher does not need to know the address of the subscribers.
- Time decoupling: A subscriber does not need to be active when the publisher is sending messages.
- Synchronization decoupling: The publisher and subscriber(s) are not blocked.

These decoupling properties make publish/subscribe messaging scalable and flexible and, therefore, suitable for the disaster management applications. They are the reasons that TRIPS supports publish/subscribe messaging model.

3.2 The Apache Qpid AMQP and Pubsubhubhub Protocol

The prototype P/S broker described in the next section runs on Apache Qpid. The Apache Qpid AMQP (Advanced Message Queue Protocol) distribution [8] is a widely deployed project and implements AMQP [9] according to publish/subscribe messaging model. AMQP is an open standard for message-oriented middleware (MOM) communication. Its primary goal is to enable better interoperability between MOM implementations. Since the inception of AMQP, several open-source messaging software distributions have emerged, including Apache Qpid AMQP. This distribution provides a broker federation option which enables workload sharing among a group of brokers linked with each other in decentralized architecture.

Qpid also has built-in fault-tolerance features, the most critical of which is High Availability Messaging Cluster. A cluster is a group of Qpid brokers with the same configurations for exchanges, queues, and other entities. The brokers in the cluster synchronize their events with each other and hence are in the same state. With replicated states, the publish/subscribe service does not fail unless all the brokers in the cluster fail. Cluster incurs high synchronization overhead. It is sustainable when used for message delivery in a small area where the network connecting brokers is stable and fast. This assumption is clearly not valid for wide-area networks and disaster management. Moreover, cluster requires redundant computation resources. This requirement is too demanding since information/communication support systems for disaster response and relief are typically resource poor. The self-healing mechanism for Qpid described in the next section incurs lower overhead than the cluster approach.

TRIPS also makes use of pubsubhubhub [10], which is a simple and flexible publish and subscribe protocol over the Internet. The protocol can provide subscribers with near instant notifications via web-hook callbacks when a feed URL subscribed by them is updated. The protocol does not incur the network traffic introduced by periodically polling the feed server as existing RSS/Atom services do. However, services speaking the protocol cannot recover from failures caused by disaster. The recovery mechanism described here can enhance the availability of such services.

3.3 Data distribution service and content centric networks

TRIPS also plans to support Data distribution service (DDS) and content-centric networking (CCN). DDA is a publish/subscribe protocol featuring QoS support. As pointed by A. Corradi *et al.* [11,12], current implementations of DDS suffer from scalability issue. A way to improve scalability of DDS is to divide the global space of support infrastructure into domains. Each domain has components responsible for copying or relaying data. With separated, domains and redundant services, scalability and availability of pub/sub messaging services can be enhanced. To prevent the services interruptions caused by partitioned networks, TRIPS take advantage of the domain concept: Computing devices are grouped into subnetworks according to their network connectivity and each group is managed individually so as to increase the scalability and availability.

In a CCN, the packets are routed according to their content descriptor. According to a typical content delivery procedure in CCN, the client first expresses its interest in certain content. The CCN routers then forward the interest to the content providers to fulfill the request. In response to the request, the content is not only delivered but duplicated to the routers on the delivery path. The other clients requesting for the same content can retrieve the duplication from any of the routers instead of the provider. The idea of CCN is similar to content-based publish/subscribe. The difference between publish/subscribe and CCN is that the interest in CCN is transient while the subscription in publish/subscribe is persistent [13].

4 DESIGN AND IMPLEMENTATION

In the landmark-based service recovery framework described here, the PubSub network is divided into several disjointed subnets. Brokers are grouped into subnets. Each broker is included in only one subnet. The broker has a unique identification within the subnet (e.g. with an ID that combines its IP address and port number). Figure 3 illustrates the architecture of the PubSub network within the subnet. Dashed arrows represent flows of control messages; solid arrows represent flows of pub/sub messages. In addition to broker nodes, each subset has at least a *backbone node*, i.e. a service node responsible for transmitting messages among subnets. Each subnet also has a *landmark monitor*, or simply *landmark*. The term refers to a service process that is responsible for monitoring and recovering the message delivery service in case of failure and manages all the brokers in the subnet. Landmark should be deployed on a reliable node so that it has a very low failure rate or it can be recovered

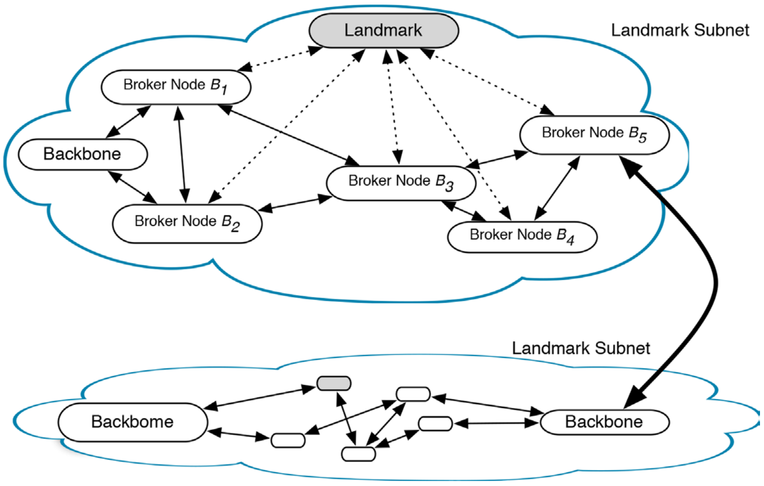

Figure 3: Pub/Sub networks with Landmark and backbone nodes.

automatically. Moreover, the landmark of a subnet is reachable from all the brokers in the subnet. In practice, one may deploy landmark and backbone on a single POS for the sake of simplicity. Hereafter, this article is confined to one pub/sub subnet. Broker, landmark, and backbone nodes refer to the nodes in one subset.

4.1 Overview of service recovery process

The rationale of landmark-based service recovery is to reduce the recovery overhead by collecting sufficient information before failure occurs. In the meantime, the impact of information collection should be minimized. During runtime, the landmark listens to the sent messages and state changes by each broker. Qpid/AMQP and several other Pub/Sub frameworks support this functionality. Specifically, when a broker service is active and health, its QMF (Qpid Management Framework) monitor transmits messaging activities conducted by the broker and changes in its queue configuration and queue state to the landmark monitor. The monitoring process on the landmark will be described shortly. The landmark monitor also duplicates the service objects on broker nodes to its local repository when the landmark monitor starts and when it detects a new broker. Qpid/AMQP and several other Pub/Sub frameworks support this functionality. When a broker learns that a link connected to or from itself is down, it sends a control message to the landmark monitor to report the link failure. Consequently, the landmark monitor can keep track of the activities and health conditions of all brokers in the subnet without actively probing the broker nodes. When the landmark learns that a broker service has failed, it starts the recovery process. The first step of the recovery process is to update the distance vector table which the landmark maintains to keep track of route lengths in terms of the average transmission delay or average bandwidth between every pair of broker nodes in the subnet.

Figure 4 illustrates the process carried out collaboratively by all the nodes in the subnet to update distance vector table. To support the process, each broker node executes a ping daemon which sends a response to ping requests from other broker nodes and the landmark. The landmark monitor starts the process by sending a distance update request (i.e. Step ❶ in Fig. 4) to brokers. When the daemon of a broker node receives the request from the landmark monitor, it sends a ping message to the corresponding broker node (i.e. Step ❷). When the daemon on the corresponding broker node receives a ping request, it echoes a null message back to the sender (i.e. Step ❸). After receiving the echo message, the sender calculates the distance based on the remained TTL (Time-To-Live). TTL is a counter in the packet header that limits the number of lifespan of a packet in the network, and is usually represented by the number of hops in the packet header (Step❹ in the figure). When the landmark learns the remaining TTLs from a pair of broker nodes, it calculates the number of hops between them and estimates the route length. Note that there is no need to update the distance vector table for all pairs of broker nodes in the subnet and only the distance of the routes connected to the

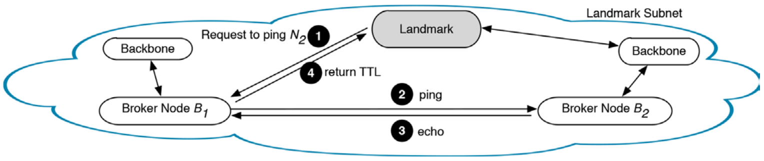

Figure 4: Procedure for Ping daemon to collect route distance.

neighbours of the failed node require updates. Moreover, the distance vector table only keeps track of the distance between brokers, which are monitored by the landmark node, and the messages will not be forwarded to neighbouring subnet by backbone nodes.

4.2 System architecture for QpidR

The landmark-based service recovery mechanism is implemented on Qpid. The mechanism is called Qpid Recovery or QpidR for short. QpidR is designed as an application in Qpid and does not modify the core services of Qpid. Hence, the landmark-based recovery protocol can also be implemented on other message publication and subscription services. Figure 5 shows the system architecture of QpidR to illustrate how QpidR interacts with typical message publish and subscription systems. In the figure, Qpid serves the underlying message publish/ subscribe services. Landmark-based service recovery mechanism is realized by three types of servers in QpidR: landmark, servicing brokers, and backup brokers. As discussed earlier, there is only one landmark node in the subnet to monitor the status of brokers and to recover failed services. Servicing brokers are brokers that send and receive messages, and thus provide messaging services. Backup brokers are the candidate broker nodes to recover failed servicing brokers when there is any. A backup broker does not send and receive messages before it is selected to recover failed service.

Landmark node starts the landmark service when the node starts and does not stop until the node terminates. Landmark service conducts three operations: *monitoring brokers, scoring backup brokers,* and *recovering failed brokers*. To monitor broker nodes, landmark service sends heartbeat messages to a list of servicing brokers and backup brokers periodically, shown as solid lines in Fig. 5. The returned messages from service brokers and backup brokers are received and stored in `GET_READY_TIMEOUT` file descriptor.

To choose a backup broker to recover a broken servicing broker, the landmark service requests each backup broker to evaluate its own capability of being a broker and return the evaluation results. The capability of being a broker depends not only on the static hardware configuration but also on the run-time parameters such as CPU workload and network connectivity. In fact, the evaluation is conducted repeatedly. The evaluation schedule can be configured for either periodically renewals or event driven renewals. The messages for scoring are represented by dashed lines in Fig. 5. When detecting a failed broker node, landmark service chooses a backup broker node which has best evaluation scores to recover the failed broker. The recovery process consists of reconfiguring the message routing tables, updating the list of servicing brokers, and selecting additional backup brokers. Landmark service coordinates the Qpid services in the network to accomplish the work, which are represented by dotted lines in Fig. 5.

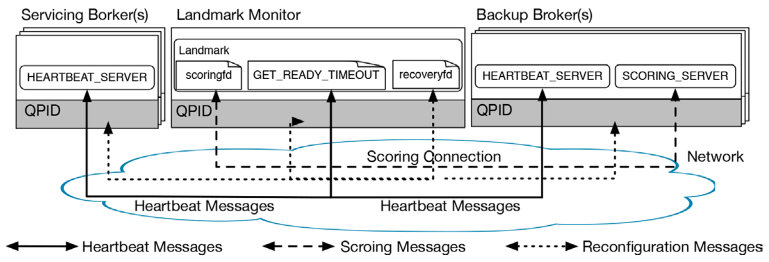

Figure 5: System architecture for QpidR.

4.2.1 Implementation of landmark service

As mentioned above, landmark service monitors servicing brokers and recovers failed services. The service makes use of Qpid management framework (QMF), the general purpose management bus built on Qpid messages. To handle concurrent events and reduce run-time overhead, utility `select` is used in a single thread. In the thread, the following three file descriptors are used to handle events for three types of services.

- `recoverysfd`: This file descriptor receives messages from specific functions called by `SessionManager` within QMF. When it starts, landmark registers with `ConsoleListener` into `SessionManager`. Callback functions implemented in `ConsoleListner` include `object-Props`, `event`, and `brokerDisconnected`. When `brokerDisconnected` function is called, it will send the message `BROKER-DISCONNECTION` to `recoverysfd` to delete the broker. When `object-Props` function is called, it will send the message `OBJECTPROPERTY` to the `recoverysfd` to add the broker into the broker listening list. When called with event name `brokerLinkDown`, the event function will send the message `LINKDOWN` to the `recoverysfd`. `Recoverysfd` will check if the service IP is still alive. If the service is disconnected, `recoverysfd` will search for one broker from listening list and then use the function `listenAddressChange` to reroute the service. Function `listenAddress-Change` will send the request to `querysfd` to retrieve the broker's information and then start rerouting from `linkdown` broker to a new one.
- `querysfd`: When the function `listenAddressChange` is called, it sends a message to `querysfd`. Landmark will retrieve the name embedded in the message, received in file descriptor `querysfd`, to locate the servicing broker and reply with the broker's information.
- `GET_READY_FD_TIMEOUT`: This file descriptor is needed for checking whether the broker node is shut down. When a broker node is broken, it will be replaced by a backup broker. This function is different from the message `LINKDOWN` in `recoverysfd` which only checks the specific serving broker for rerouting but not all serving brokers. `GET_READY_FD_TIMEOUT` needs to monitor all serving and backup brokers' internet connections.

To collaborate with servicing brokers and backup brokers, Landmark services interact with `heartbeat_server` on servicing brokers and backup brokers, and `scoring_server` on backup brokers. `Heartbeat_server` on each broker waits for the requests from the landmark. Landmark sends a message to `heartbeat_server` at either certain constant time intervals or predefined events. A response timeout is also configured to detect a failed broker node. (Note that a service may fail for different reasons such as broken connection and failed broker messaging service. Broken connection can be detected by either QMF, as discussed above, or heartbeat message.) If landmark service does not receive the response within the predefined timeout, it assumes that the service is not reachable and will start the recovery process. `Scoring_server` on backup brokers wait for the request from landmark. Receiving the request, `scoring_server` will create a thread to evaluate the backup brokers according to the predefined scoring criteria.

The scoring procedure [14] is conducted in two steps: *collect the resource usage from multiple resources*, and *compute the overall score for each backup broker*. Below shows the scenario for using CPU utilization, available memory, batter level, and network bandwidth to evaluate a backup broker. On UNIX-like systems, one can utilize system APIs to collect the resource usages. For example, one can use the command `cat /proc/cpuinfo | grep MHz` to query CPU frequency and, then, use `mpstate` to query CPU usage. After having the usage parameters, one can combine these values and estimate CPU utilization by eqn (1). Similarly, one can use the command `free -b` to query memory usage and estimate memory utilitization by eqn (2). For battery, one can use command `acpi -i` to query battery lifetime and, then, use eqn (3) to estimate the battery utilization. When `scoring_server` receives the request to estimate bandwidth, it first queries all the IPs connected to the broken service and then use the PBProbe tool [15] to estimate each IPs bandwidth and average them by eqn (4).

$$U_C = \exp(\{[C_e * (1 - C_u) - \mathrm{C}_n, 0]\}) \tag{1}$$

$$U_M = \exp(\{[M_e * (1 - M_u) - M_n, 0]\}) \tag{2}$$

$$U_{BA} = \exp(\{[BA_e - BA_n), 0]\}) \tag{3}$$

$$U_{BW} = \exp(\{[\sum_{i=1}^{m} BW_{ei}/m - BW_n), 0]\}) \tag{4}$$

The second step is to compute the overall score of a backup broker. Specifically, each `scoring_server` assigns weights to different resources, where the sum of weights must be equal to one, and computes the overall score by eqn (5). Again, the users can define their different evaluation criteria in QpidR. The example shown here was a default setting for QpidR and was used in our experiments.

$$U_T = W_C * U_C + W_M * U_M + W_{BW} * U_{BW} + W_{BA} * U_{BA} \tag{5}$$

where
W_C: weight for CPU utilization
W_M: weight for memory utilization
W_{BW}: weight for bandwidth estimation
W_{BA}: weight for battery usage
$W_C + W_M + W_{BW} + W_{BA} = 1$

4.2.2 Distributed testing and configuration

To allow the users to evaluate QpidR in different distributed environments, the mechanism provides an installation package to reduce the burden of configuring landmark-based recovery and distributed message publish and subscription. For the sake of testing and illustration, ESTINET and virtual machines are used. ESTINET is the emulation environment for network services which provide different network topologies and configuration settings for bandwidth and packet delay. Thus, the tool can be used to emulate distributed environments having ten to hundred nodes, as described below:

Configuration Files The installation package of QpidR requires three configuration files: *network configuration*, *Qpid configuration*, and *QpidR configuration*. Figure 6 shows how these three configuration files are imported into the services. Network configuration file is imported into the network simulator to configure the network topology. If QpidR is installed on a physical machine, network configuration file is not needed. Qpid configuration files are

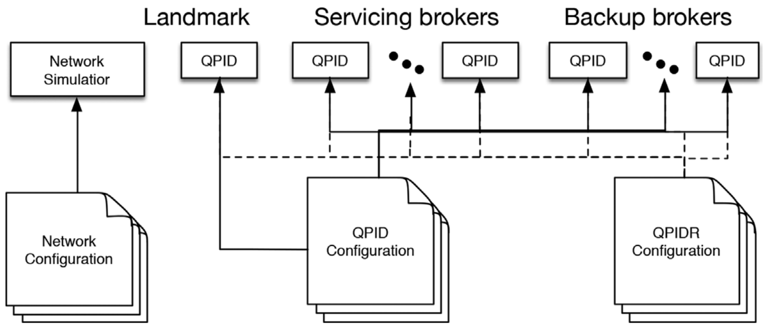

Figure 6: QpidR Configuration.

imported into all the Qpid nodes for landmark, servicing brokers, and backup brokers to configure the connection among Qpid broker nodes. QpidR configuration files are imported into all brokers to initialize heartbeat messages, scoring service, and recovery services.

Supported Commands In current implementation, QpidR supports three recovery configurations: (1) static nodes serve as servicing brokers and landmark, (2) mobile nodes serve as servicing brokers, and (3) mobile nodes serve as backup brokers. The commands to start the services are listed as follow:

- `qpidd -p PORT_NUM --no-data-dir --auth no`
  This command starts Qpid services in background. For servicing brokers, `qpidd` provides service for message clients. For backup brokers, `qpidd` starts in background to recover failed servicing brokers. `PORT_NUM` is the port number for client to connect. `--no-data-dir` configures that the application will store the information in the memory. `--auth no` configures that authentication is not required.
- `heartbeat_server`
  On landmark, servicing brokers and backup brokers, the above command is used to start the heartbeat server. The process will send, receive, and respond to heartbeat messages.
- `landmark BROKERLIST_FILE BACKUP_LIST_FILE`
  The above command is the main function for the recovery mechanism. Users provide the list of all serving services in the subnet and the backup file for backup services in the file. Landmark will send heartbeat messages to the brokers listed in file `BROKERLIST_FILE`. In addition, landmark reads IPs about backup brokers from `BACKUP_LIST_FILE` and choose the proper broker for recovery once the serving broker is broken.
- `scoring_server scoringServer.ini`
  The above command starts `scoring_server` services, according to the configuration specified in file `scoringServer.ini`. File `scoringServer.ini` defines the resources and parameters for scoring. For example, one can specify CPU utilization, memory utilization, battery, and bandwidth. `Scoring_server` will read the configuration file, do the calculation and, then, send the score to the landmark. Figure 7 shows an example configuration file for scoring. In the file, `RunOnExecution` specifies the path for the application. `INPUTFILE` specifies the file for application to read. `OUTPUTFILE` specifies the output file for the application. Scoring server will use the value from `OUTPUTFILE` to compute the utilization. `WEIGHT` is the weighting for the application. In Figure 7, the weighting factor for CPU is 1/(1 + 1 + 50 + 60).

```
[CPU]
RunOnExecution = $HOME/qpidd/CPU
   INPUTFILE=$HOME/CPUINPUT
   OUTPUTFILE=$HOME/CPUINPUT
   WEIGHT=1

[MEMORY]
RunOnExecution = $HOME/qpidd/MEMORY
   INPUTFILE=$HOME/MEMORYINPUT
   OUTPUTFILE=$HOME/MEMORYINPUT
   WEIGHT=1

[BATTERY]
RunOnExecution = $HOME/qpidd/BATTERY
   INPUTFILE=$HOME/BATTERYINPUT
   OUTPUTFILE=$HOME/BATTERYINPUT
   WEIGHT=50

[BANDWIDTH]
RunOnExecution = $HOME/qpidd/BANDWIDTH
   INPUTFILE=$HOME/BANDWIDTH INPUT
   OUTPUTFILE=$HOME/BANDWIDTH INPUT
   WEIGHT=60
```

Figure 7: Configuration for `scoring server`.

5 PERFORMANCE EVALUATION

5.1 Experiment environment

To evaluate the design described in the previous sections, the discrete-time simulator EstiNet [16] was used. Again, EstiNet is capable of simulating network interfaces, network devices, topologies, and protocols for the applications. The users can also configure numerous parameters of the environment such as bandwidth, latency, link down, and node failure. In addition, every node in the simulated environment has a synchronized clock so as to calculate the interval between events on different nodes by logging their timestamps. The EstiNet simulator was installed on a virtual machine. The guest system is Fedora Linux 14. The CPU on the host machine is Intel Xeon E5620 @ 2.4 GHz. A CPU core and a 2 GB-RAM are allocated to the virtual machine.

5.2 Evaluation results

This section presents the results for verifying the correctness of recovery operation, evaluating transmitted data size during recovery and monitoring and measuring the latency of recovery and initialization for monitoring.

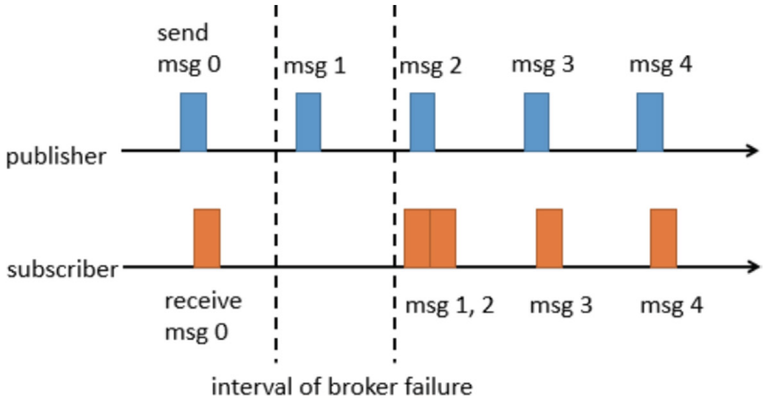

Figure 8: Time sequence of correctness test.

5.2.1 Correctness

Here, correctness refers to the fact that service recovery indeed prevents the subscribers from losing messages. Synthetic workload is designed to verify correctness of our implementation. In the experiments, messages are published periodically. They are transferred from one broker to another and finally received by the subscriber. Besides, the brokers temporarily store the messages until the messages expire. Figure 8 shows the time sequence of a publisher and a subscriber. When there is no broker failure, the messages are delivered to the subscriber in a short time. During broker service failure, the delivery path is interrupted. The subscriber cannot receive any message. With the service recovery mechanism, the failed broker service is replaced by a backup service. When the failed routes are reconnected to the backup service, the non-expired messages are transferred to the backup service and its succeeding brokers. The messages published during broker failure are delivered to the subscriber right after recovery. Hence, in the experiments, brokers will be temporarily disconnected from the network. The mechanism is correct if the periodic messages are all received by the subscriber.

5.2.2 Data size

To determine the size of transmitted data during monitoring and recovery, the network traffic was measured by recording the I/O flows on the simulated network interface. The network is a subnet consisting of a number of brokers and one centralized recovery service. In order not to count any background traffic, the subnet contains no clients.

In the first experiment, the data rate of control messages is measured. Even when there is no failure, the recovery service has to exchange control messages, including events and statistical data, with the monitored broker services. These data are generated either by QMF or recovery protocol. Except for events such as broker disconnection, link down, and object property change, the period and size of control messages are fixed. Here, objects refer to message queues, broker names, broker ports, et al. This information is stored by the recovery service, and does not change.

Table 1 shows the relationship between number of monitored brokers and data rate of periodic control messages. The experiment result shows that the data rate grows linearly with number of monitored brokers. It takes approximately 2.3 KB/s of bandwidth to monitor a broker.

In the second experiment, the amount of network I/O during recovery is measured. The subnet used for experiment consists of a recovery service, a failed broker, and a backup

Table 1: Data rate of periodic control message.

Number of brokers	Average data rate (KB/sec)	Standard deviation
10	23.97	1.67
20	42.68	2.97
30	68.92	3.27
40	89.49	6.68

Table 2: Traffic size of recovering queues.

Number of queues	Average traffic size (KB)	Standard deviation
0	496.38	4.51
10	595.80	5.16
20	704.23	5.48
30	800.05	6.09
40	907.87	5.80

Table 3: Traffic size of recovering routes.

Number of routes	Average traffic size (KB)	Standard deviation
5	627.87	3.99
10	690.02	5.30
15	751.34	6.19
20	810.38	6.91

broker. The experiments record the network I/O on the recovery service during recovering the failure. The amount of network I/O is related to the type and number of objects on the backup broker.

In the second experiment, the amount of network I/O during recovery is measured. The subnet used for experiment consists of a recovery service, a failed broker, and a backup broker. The experiments record the network I/O on the recovery service during recovering the failure. The amount of network I/O is related to the type and number of objects on the backup broker.

Table 2 shows the relation between data size and the number of objects. Without loss of generality, only queues and routes are shown. It takes about 10 KB to create a queue and 12 KB to create a route. In the case where there are no objects, the recovery service does not perform any operation on the backup broker. The I/O data is used for initializing a QMF session, receiving events, and terminating the session. Table 3 shows the relation between data size and the number of routes. When the number of routes increase, the network traffic increases at the rate of approximately 12 KB per route. Compared to the traffic of service recovery, the increase is negligible.

Table 4: Centralized recovery latency in a small network.

Number of failures	Average latency (sec)	Standard deviation	Max	Min
1	2.92	0.34	3.43	2.39
2	5.50	0.64	6.78	4.35
3	8.52	0.77	9.56	6.99
4	11.46	0.82	12.63	9.93
5	14.37	0.87	15.71	12.37

5.2.3 Average latency

The experiments also evaluated recovery latency of centralized framework in different network environments. In the experiment, the parameters are network size, bandwidth, and number of concurrent failures. The size of network is represented by the number of hosts. The available bandwidth is not only affected by the link bandwidth, but also by the background traffic, including client messages and control messages produced by non-failed brokers.

The first simulation environment was done in a small network in which there are 24 hosts. The bandwidth between the host and the router is 10 Mbps which is relatively large to the background traffic. In centralized service recovery, every host runs a broker service and one of the hosts is the landmark. In distributed framework, every host runs a broker service and a recovery service. Table 4 shows the latency of centralized recovery.

In the case of concurrent failures, the latency of centralized recovery increases linearly. This is mainly caused by the sequential operations on the recovery service.

6 SUMMARY

Autonomous service recovery is critical and desirable for information exchange service for disaster management. This article presents a landmark-based service recovery mechanism to restore the failed message publishing and subscription services. The mechanism requires a central server, i.e. the landmark server, to collect service configuration and monitor the states of broker services. Given the collected status data, the landmark server chooses a backup service node to recover failed service. The developed mechanism consists of status monitoring component, service evaluation component, and online reconfiguration component. The experiment results presented above confirm the aforementioned properties for a publish/subscribe network in a reasonable size. Specifically, the services can be recovered within 15 seconds up to five concurrent failures. Future work includes the extension of the central mechanism into a distributed service recovery mechanism to support large sized publish/subscribe networks.

ACKNOWLEDGEMENT

This work was partially supported by National Science Council under Grants 102-2219-E-002-022 and by Academia Sinca under Grants AS-102-SS-A04.

REFERENCES

[1] Chiu, W.T., Arnold, J., Shih, Y.T., Hsiung, K.H., Chi, H.Y., Chiu, C.H., Tsai, W.C. & Huang, W.C., A survey of international urban search-and-rescue teams following the Ji Ji earthquake. *Disasters*, **26(1)**, pp. 85–94, 2002. doi: http://dx.doi.org/10.1111/1467-7717.00193

[2] United States Department of Transportation, Next generation 9-1-1 (accessed January 2012), available at http://www.its.dot.gov/ng911/

[3] United States Coast Guard, Rescue 21 (accessed January 2012), available at http://www.uscg.mil/acquisition/rescue21/

[4] Chanson, H., The impact of Typhoon Morakot on the southern Taiwan coast. *Shore & Beach*, **78(2)**, pp. 33–37, 2011.

[5] NBC news services, Rescue crews pull 2 more from Haitian market (accessed 17 January 2010), available at http://www.nbcnews.com/id/34829978/ns/world_news-haiti/t/rescue-crews-pull-more-haitian-market/

[6] Chen, L.-J., Li, C.-W., Huang, Y.T. & Shih, C.-S., A rapid method for detecting geographically disconnected areas after disasters. *Technologies for Homeland Security (HST) (Nov 2011)*, pp. 501–506, 2011. doi: http://dx.doi.org/10.1109/ths.2011.6107919

[7] Eugster, P.T., Felber, P.A., Guerraoui, R. & Kermarrec, A.-M., The many faces of publish/subscribe. *ACM Computing Surveys*, **35(2)**, pp. 114–131, 2003. doi: http://dx.doi.org/10.1145/857076.857078

[8] Apache Qpid, http://cwiki.apache.org/qpidl

[9] Advanced Message Queuing Protocol, http://www.amqp.org

[10] Fitzpatrick, B., Slatkin, B. & Atkins, M., Pubsubhubbub core 0.3 working draft. Technical Report, 2010.

[11] Corradi, A. & Foschini, L., A DDS-compliant P2P infrastructure for reliable and QoS-enabled data dissemination. *Proceedings of 2009 IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009)*, pp. 1–8, 23–29, 2009, doi: 10.1109/IPDPS.2009.5160957

[12] Corradi, A., Foschini, L. & Nardelli, L., A DDS-compliant infrastructure for fault-tolerant and scalable data dissemination. *Proceedings of 2010 IEEE Symposium on Computers and Communications (ISCC 2010)*, pp. 489–495, 2010, doi: 10.1109/ISCC.2010.5546756.

[13] Carzaniga, A., Papalini, M. & Wolf, A.L., Content-based publish/subscribe networking and information-centric networking. *Proceedings of the ACM SIGCOMM workshop on Information-centric Networking*, pp. 56–61, 2011. doi: http://dx.doi.org/10.1145/2018584.2018599

[14] Chen, L.-J., Sun, T., Chen, B., Rajendran, V., & Gerla, M., A smart decision model for vertical handoff. *The 4th International Workshop on Wireless Internet and Reconguability (ANWIRE04)*, May 2004.

[15] Chen, L.-J., Sun, T., Wang, B.-C., Sanadidi, M.Y. & Gerla, M., PBProbe: A capacity estimation tool for high speed networks. *Journal of Computer Communications*, **31(17)**, pp. 3883–3893, 2008. doi: http://dx.doi.org/10.1016/j.comcom.2008.05.047

[16] EstiNet, http://www.estinet.com/

[17] Lin, K.J., Zhang, J., Zhai, Y. & Xu, B., The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA. *Service Oriented Computing and Applications*, **4(3)**, pp. 157–168, 2010. doi: http://dx.doi.org/10.1007/s11761-010-0063-6

[18] Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H. & Braynard, R.L., Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 1–12, 2009. doi: http://dx.doi.org/10.1145/2063176.2063204

[19] Wu, Q. & Li, P., Study and implement of dynamic routing based on QoS in enterprise service bus. *Journal of Computational Information Systems*, **6(7)**, pp. 2093–2098, 2010.