Responsive Alert Delivery over IP Network

Y. Z. Ou, C. M. Huang,
C. T. Hu, J. W. S. Liu
Institute of Information Science
Academia Sinica, Taiwan
{yzou, charles.huang, tomhu
janeliu}@iis.sinica.edu.tw

E. T. H. Chu
Dept. of Computer Science
National Yunlin Univ. of Sci. and Tech.
Yunlin, Taiwan
edwardchu@yuntech.edu.tw

C. S. Shih
Dept. of CS and Info. Eng
National Taiwan Univ.
Taipei, Taiwan
cshih@csie.ntu.edu.tw

***Abstract*—Intelligent Guards against Disasters (iGaDs) are devices and applications deployed pervasively in future smart homes and environments. By receiving and processing standard-based machine-readable disaster alert messages from authorized senders and taking appropriate actions in response, they aim to help us better cope with natural disasters. The focus of this paper is on responsive delivery of alert messages over an IP network to a huge number of embedded iGaDs. The Asynchronous Message Delivery Service (AMeDS) is designed for this purpose. After presenting an overview of iGaDs and AMeDs, the paper describes an end-to-end scheduling mechanism used by AMeDS and discusses performance limitations in pushing time-critical alerts to iGaDs over the Internet.**

Keywords-end-to-end scheduling, intelligent things, message delivery service, smart environment, disaster preparedness

I. INTRODUCTION

In the recent decade, we have seen great advances in disaster prediction and detection technologies, as well as information and communication technologies (ICT) for disaster preparedness and response (e.g., [1-9]). Equally impressive are the wide deployments in developed regions of advanced ICT support infrastructures for the generation and distribution of disaster alerts/warnings (e.g., [10-14]). XML-based EDXL (Emergency Data Exchange Language) [15] messaging standards, including CAP (Common Alerting Protocol) [16], EDXL-DE (Distribution Element), and EDXL-RM (Resource Messaging), have enabled automatic reports by sensor systems to analysis centers, aggregation and correlation of warnings from multiple sources, and information exchanges between emergency information systems and services. Now, emergency alert authorities in typical developed regions can generate accurate warnings of devastating natural disasters seconds and minutes or more before they occur, encode alert messages in a standard machine-readable format (e.g., CAP) and broadcast the messages via all communication pathways, including digital radio broadcast, cellular networks and the Internet to public and commercial emergency alert systems and services. Indeed, alert authorities in US, Canada, and many parts of EU and Asia can warn people in this way.

To date, disaster warning messages, though in a machine-readable format, are consumed mostly by people. The limits in human reaction time limit the effectiveness of the warnings. A natural next step in the advancement of disaster preparedness and response technology is the emergence of smart things that can respond to warnings of imminent calamities with humanly impossible speeds. This is the motivation behind *Intelligent Guards against Disasters*, or *iGaDs* for short [17, 18]. The term iGaDs refers to a diversity of embedded devices, systems and applications that can authenticate and process standard-conforming disaster warning messages and respond by taking appropriate actions to help us prevent loss of lives, reduce chance of injuries and minimize property damages and economical losses when disasters strike.

For sake of concreteness, we suppose here that alert messages conform to the CAP standard and sometimes call iGaDs *CAP-aware* devices and applications. We will use the terms messages, alerts, warnings, and data updates in different context interchangeably. Being of an XML document of length a thousand bytes or so, each message fits in an IP packet.

As illustrative examples, Figure 1 shows three iGaDs and an earthquake scenario. The scenario starts when an authorized alert sender (e.g., the Taiwan Central Weather Bureau) issues a warning of a magnitude 8.1 earthquake a fraction of a second or more before people in affected areas feel ground movements. While the warning time is too short for people to react, it is long enough for CAP-aware elevator controllers in a smart building to stop the elevators and open their doors when they reach the closest floors. An iGaD in a smart home shuts down natural gas intake valve to prevent fire and open the front door to ease evacuation. A software iGaD running on a home computer displays on monitors in the home safe places to be.

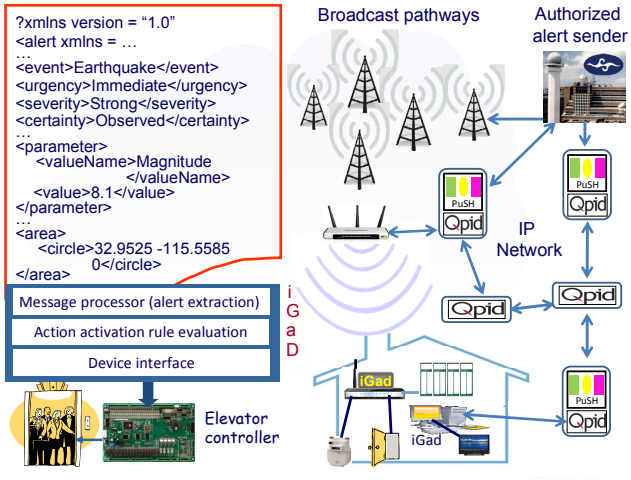

Figure 1. Examples of iGaDs and their connections to alert senders

Figure 1 also shows that multiple communication pathways may be used by alert senders to deliver warning messages to iGaDs. Some iGaDs are reachable via broadcast networks. Examples include CAP-aware applications running on smart phones and vehicular safety devices. These examples are not shown in the figure; we will not consider them hereafter.

This paper is concerned with iGaDs that are reachable by senders only via the Internet: Some of them are reachable directly via the Internet; they are exemplified by CAP-aware applications running on a computer in a smart home in Figure 1. Others are reachable indirectly via local connections from aggregation servers. Examples include CAP-aware elevator controllers, smart gas valves and front doors; these embedded iGaDs receive alerts relayed by a smart building (home) management system that is connected to the Internet. A problem for such iGaDs, especially for embedded devices that do not have a web interface, is how to deliver to them in real-time warning messages asynchronously over the Internet. A solution is the Asynchronous Message Delivery Service (AMeDS) presented in [19]. We will present an overview of the service, its structure and components in a later section.

We focus our attention here to the real-time performance of AMeDS in pushing time-critical alerts to iGaDs over the Internet. Specifically, we present here an end-to-end scheduling mechanism needed by AMeDS for prioritized delivery of time-critical messages and assess the end-to-end delays of messages under typical network load conditions. Clearly, iGaDs and people can take protective actions in preparation of an imminent calamity only when they receive warnings about the calamity in time. This means that the end-to-end delay of earthquake warning messages should be seconds or less, and delay for tornado warnings a minutes or a few minutes. On the other hand, wide-area floods often take days to develop. End-to-end delay in order of an hour may be acceptable for warnings about this and similar types of disasters.

Following this introduction, Section II presents background and related work. Section III presents an overview of AMeDS. Section IV presents the above mentioned end-to-end priority scheduling mechanism and the scheduling schemes it can support. Section V describes two experiments done to assess the performance limitation of AMeDS and measurement data obtained from the experiments. Section VI presents our conclusions and discusses future works.

II. Background and Related Works

This paper builds on our previous work on iGaDs [17-19]: The earlier papers [17, 18] described many examples of iGaDs and discussed how they can be effective for preparedness and earlier response purposes in various disaster scenarios. These papers also presented architecture and key components of diverse iGaDs, cost effective ways to make iGaDs configurable and customizable while in use, hardware enhancements to keep energy consumption of battery operated iGaDs low and opportunities and challenges in making iGaDs pervasive elements of future disaster prepared smart home and environments. We considered in these papers only the case where messages warning of all types of disasters are delivered to iGaDs via cellular cast and broadcast. In this case, a major component of the end-to-end delay from issuance of an alert to the time an iGaD receives and interprets the alert and takes actions accordingly is the propagation delay of the broadcast signal. It is typically a fraction of a second, sufficiently small even for earthquake alerts.

Each node on the edge of a network that supports the Asynchronous Message Delivery Service (AMeDS) [19] mentioned above uses as components a PubSubHubbub (PuSH) hub [20] and a Qpid node [21]. PubSubHubbub is a simple and flexible server-to-server, web-hook-based publish and subscribe protocol. Servers speaking the PuSH protocol can get near instant notifications via web-hook callbacks when a *topic* (i.e., a feed URL) subscribed by them is updated. Qpid implements Advanced Message Queuing Protocol (AMQP) [22], which can deliver messages by Publish and Subscribe message protocol. The hub and the Qpid components are linked by a data bridge. A node thus created combines PuSH and Qpid protocols so that the applications can take advantage of the HTTP-based application interface provided by PuSH and the prioritized asynchronous message delivery provided by Qpid. In particular, alert senders can publish via PuSH hub interface warnings on imminent disasters as updates of data on specified topics and have the updates pushed asynchronously and on priority basis over the Internet to numerous iGaDs. Hereafter, we call such a node a *PQ node* when it is necessary to be specific. Our previous paper [19] focused on the design of the data bridge. Issues related to scheduling data transfers between the hub and Qpid components were not addressed.

The mechanism for end-to-end scheduling of messages within each PQ node makes use of Qpid prioritized message queuing facility, together with priority queuing within the data bridge. Together, they support consistent prioritization for sequencing messages from (to) the hub to (from) Qpid. The mechanism can support almost all fixed priority schemes, including the well known deadline-monotonic (DM) scheme [23]: The DM scheme assigns priorities to messages according to their (end-to-end) relative deadlines (i.e., response times), the smaller the deadline, the higher the priority.

The end-to-end delay of an alert message from an alert sender directly to an iGaD is the sum of three terms:

$$D_{PQ}(s) + D_{PQ}(d) + k\,D_N \quad (1)$$

The first two terms are the lengths of time the message spends in the PQ node serving the sender and in the destination PQ node serving the iGaD, respectively. The last term D_N is the one-way delay suffered by the data packet or a protocol packet in the IP network. We will provide the multiplying factor $k > 1$ later. We measured $D_{PQ}(s)$ and $D_{PQ}(d)$ for various priority schemes, but use available data on Internet performance, including data reported in [24-26], for the values of D_N.

III. Asynchronous Message Delivery Service

As stated earlier, AMeDS is designed to enables real-time asynchronous deliveries of alert messages over the Internet to

iGaDs. It is a topic-based publish/subscribe service. Each topic is defined by a known feed URL. For sake of concreteness, we assume that authorized alert senders publish warning messages about different types of disasters to different topics. By subscribing to a topic for warnings of a disaster type, an iGaD, as well as other subscribers of the topic, receives all the warning messages published to the topic.

A. *Hub, Data Bridge and Qpid*

Figure 2 shows the structure of a PQ node: It has a PubSubHubbub (PuSH) hub on the top, a Qpid broker at the bottom and a data bridge linking them. The right part of Figure 1 shows how PQ nodes are used together with Qpid nodes: The former are deployed on the edge to provide web-based services to publishers and subscribers. The latter are deployed inside the network core to route messages.

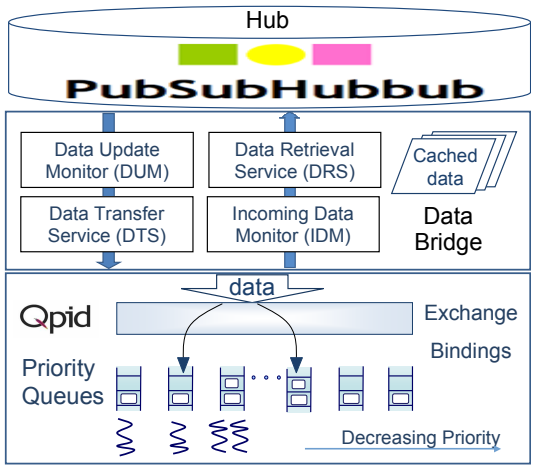

Figure 2. Structure of Hub, Data Bridge and Qpid

Specifically, PuSH provides a HTTP-based interface using which publishers can publish data via topics defined by feed URLs and subscribers can subscribe to selected topics: When a publisher wants to publish the latest of its data via a PuSH hub, it sends to the hub a HTTP POST request along with a callback URL to inform the hub that it has new data to publish. (For our application, new data or data updates are new alert or revised alert messages, each of which fit in an IP packet.) When thus notified, the hub retrieves the data from the publisher.

As Figure 2 shows, the Qpid component has a set of priority queues: Data in the queues are sent to network in priority order by worker threads. Queues may be dedicated to topics on different types of disaster. We are experimenting with this scheme, i.e., assigns priorities to alert messages/data on the basis of disaster types within the alerts. This is the priority assignment described in [19]: Data queues dedicated to topics on earthquake, tsunami, tornado, and so on have decreasing priorities. We plan to experiment with the DM scheme also. In that case, each data queue is dedicated to messages/data that have identical (or similar) relative deadlines.

The Qpid component also has an exchange, which routes data from the data bridge to data queues. The exchange is of topic type. It gives us the flexibility of routing each alert message to one or more data queues based on a message routing key (e.g., disaster type or relative deadline), and/or matching of other patterns. So, it is easy for us to experiment with different priority schemes.

The middle part of Figure 2 depicts the structure of the data bridge middleware. The rectangular boxes and arrows in the figure represent the functional modules and the directions of data flow, respectively. The bridge has two parts. The first part consists of modules named Data Update Monitor (DUM) and Data Transfer Service (DTS). It is responsible for moving data downward from the hub served by the data bridge to the Qpid component below. The second part is responsible for moving data arriving from the network upward, from Qpid data queues to the hub. This part consists of the Incoming Data Monitor (IDM) and the Data Retrieval Service (DRS). We will present the queue structures of these parts in the next section.

B. *Interactions between PuSH, Data Bridge and Qpid*

The sequence diagram in Figure 3 explains the operations of the data bridge, as well as the interactions between a publisher, hub, the data bridge, Qpid and subscribers. The dashed rectangle encircles the components of a PQ node. Vertical arrows in the diagram indicate the direction of time flow. Horizontal and slanted arrows labeled by ***A1***, ***A2*** and so on represent the series of actions taken by parties participating in the interactions triggered by a publisher with new data to publish. Arrows labeled by ***B1*** and ***B2*** represent actions taken by the components when a data packet is forwarded to the Qpid from other nodes in the network.

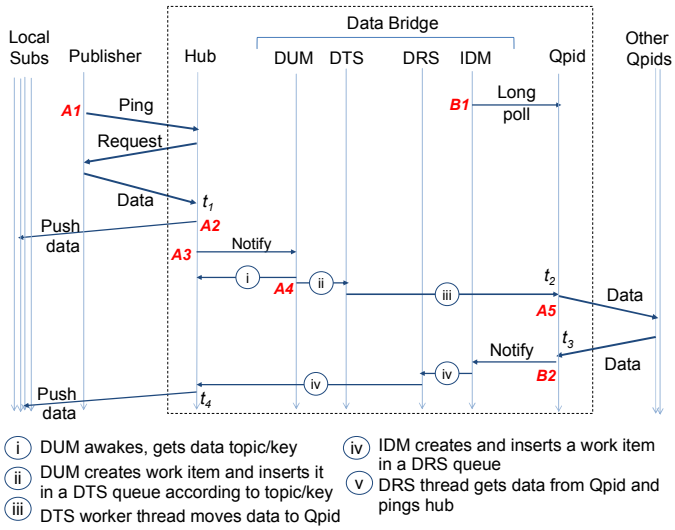

Figure 3. Sequece Diagram of Publisher,

The series of actions represented by the three slanted arrows beginning from ***A1*** starts from when a publisher pings the hub by sending to it a HTTP POST and a callback URL in accordance with the HTTP-based interface of the hub. Thus notified of the fact that new data on a topic is available, the hub retrieves the new data from the publisher according to the PuSH protocol. Figure 3 labels the time instant when the hub has acquired the new data as t_1.

Upon acquiring the new data, the hub performs two actions. First, it pushes the new data to local subscribers (***A2***) of the topic. By local subscribers, we mean subscribers who have previously registered with the hub, and hence their URLs are maintained by the hub. Second, the hub notifies the data bridge by calling the notification API function of DUM with the updated topic (disaster type in our case) as an input parameter. This action, labeled ***A3*** in Figure 3, triggers the series of actions, labeled ***A4,*** by DUM and DTS components of the data bridge to move the new data downward and have it routed by the Qpid exchange to one (or more) of the Qpid data queue(s). We will explain further these actions in the next section. Once the data is routed to a data queue in the Qpid, it is forwarded by Qpid to other Qpid(s) or other types of nodes according to the routing table of the Qpid. The time instant when the data leaves the PQ node is labeled as t_2 in the figure.

Horizontal arrows labeled ***B1*** and ***B2*** illustrates the actions of the components when a new message/data reaches the local Qpid. As ***B1*** indicates, IDM uses a monitoring thread and the long polling mechanism to monitor each data queue of the local Qpid for new data arrivals to the queue from the network. Upon awaken by the arrival of a new data to a Qpid data queue (at t_3), the monitoring thread of the queue creates and inserts a work item into a work queue of the DRS. When a DRS thread executes the work item, it retrieves incoming data from the Qpid data queue and ping the hub. In essential, the data bridge acts as a publisher. Its notification to the hub triggers actions similar to ***A1*** locally within the PQ node and subsequently the delivery the new data to subscribers (at t_4).

C. End-to-End Delay of Alert Message

Again, the *end-to-end delay* of an alert message is the length of time from the instant when an alert sender (i.e., a publisher of a PQ node) posts an update (i.e., a new alert message) to the instant when the message is pushed to an iGad or an aggregation server (i.e., a subscriber). It is given in general by the expression (1). We now use the sequence diagram to help us define the terms more precisely and determine the value of k in the expression.

$D_{PQ}(s)$ is the length of time from the instant when the PQ node serving the publisher acquires the new data to the instant when the data is forwarded by the Qpid to another node. These instants are labeled as t_1 and t_2, respectively, in Figure 3. Hence, $D_{PQ}(s) = t_2 - t_1$. $D_{PQ}(s)$ is called the *residence time* of messages through the PQ node at the source.

$D_{PQ}(d)$ is the residence time of the message through the PQ node at the destination: It is the length of time from the arrival instant of a new data item to a data queue of the Qpid component to the time instant when the hub pushes the data to local subscribers. In other words, $D_{PQ}(d) = t_4 - t_3$.

The last term in expression (1) is an integer k times D_N, *network delay* per traversal of the Internet. Figure 3 shows that from when a publisher notifies its PQ node of the new update to when the update is pushed to the subscribers, there are five network traversals by the data packet or protocol packets: 3 starting from the one labeled ***A1***, one between ***A5*** and ***B2*** and one labeled ***A3***. (Again, a realistic assumption is that every alert message fits in one IP packet.) Hence, $k = 5$.

IV. End-to-End Scheduling Mechanism

We describe in this section the structures of the data bridge components for transferring data between the hub and Qpid, specifically, the queuing facilities provided by the components to support different fixed priority scheduling schemes. Except for where it is stated otherwise, we assume that the data bridge and Qpid components are configured to use the same number of priority queues and the same scheme for assigning priorities to the queues. By doing so, we ensure consistent prioritization of message transfers within and across the components. The current version of PQ node [19] assigns priorities to data on the basis of disaster types (hence topics of messages carrying alerts for the disaster types) in both Qpid and data bridge.

A. Structure of DTM and DTS

Figure 4 shows the structure of DUM and DTS. As stated above, DTS uses its priority queues to hold work items. Like Qpid, DTS may also use multiple worker threads to execute the work items. (For example, Figure 4 shows that DTS has 4 worker threads. They are busy executing 4 work items in the 3 highest priority queues; none is waiting in the pool.) When executed, each work item moves the message specified by a field within the data structure of the work item from the outgoing data buffer (called a message queue) of the hub to an incoming data buffer in Qpid. The arrival of a new message to Qpid data buffer triggers the exchange to create a work item with a link to the location of the message in the data buffer and insert the item in one of priority data queues. The message is then forwarded under the control of a Qpid worker thread.

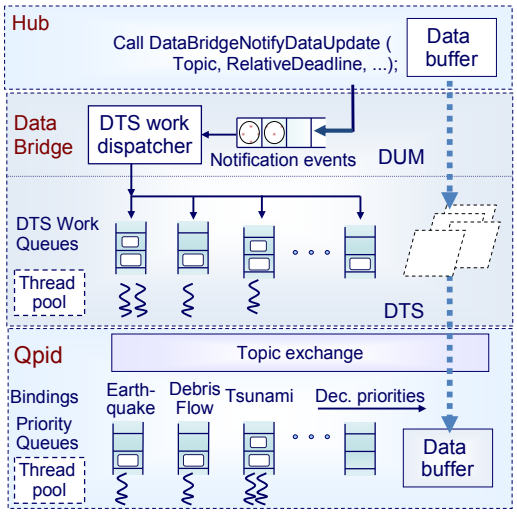

Figure 4. Queuing facility for outgoing messages

In its simplest form, DUM has only one thread, called *work dispatcher* in Figure 4. After initialization, the work dispatcher waits on notification event(s). Such an event is raised by the API function DataUpdateNotification(): The hub calls this function when it has a new data update/message to be pushed. The input augments of the function include the topic of the message and, possibly, other routing keys. (For example, the figure shows RelativeDeadline of the message as an augment.) The function wraps the input augments in an event augment and raises an event to be handled by the work dispatcher. When

awakened by a notification event, the work dispatcher retrieves from the event augment the topic of the message, creates a data transfer work items, insert the work item in the DTS work queues according to the message topic.

We note that the simply structure described above works whenever it is not necessary for DUM to read each message at least partially in order to determine at what priority the message is to be transferred to Qpid. For example, if relative deadline of the new message is also passed by the hub as an augment along with the topic of the message when it calls the data update notification API, then the DM priority scheme or schemes based on both the topic and relative deadline of the each message can be supported in the same way as prioritization according to message topic.

One can foresee the need for priority assignments based on parts of message content (e.g., the urgency and affected area specified in CAP messages). To do so without additional modification of the hub code requires that each message is read partially by DUM and may even be cached within the data bridge. This enhancement may use a multi-threaded DUM, having its own threads or sharing worker threads with DTS and is a part of our future work

B. Structure of IDM and DRS

The previous section mentioned that incoming data monitor module has a thread per Qpid data queue. The thread for a queue sends Qpid a long poll request and then waits for Qpid to respond. When notified by Qpid of an arrival of new message to the queue, the thread creates a data retrieval work item and inserts the item into the corresponding work queue of the DRS module and has the item executed by a DRS worker thread. This design does not require any modification of Qpid, but is in general an expensive design. (By default, Qpid supports 10 priority levels, and hence, 10 threads are needed to monitor the 10 queues using this method.)

Figure 5 shows the current structure of IDM and DRS. It uses only one monitor thread, called IDMonitor thread, but requires a new Qpid API function, called QpidLongPollAll (QueueIdentifier). The function is called by a caller (in our case, the IDMonitor thread) when the caller wishes to monitor all the Qpid priority queues and wants to be notified by an event when data arrives to any of the queues. The function raises an event for each signaled queue and returns as part of the event augment the identifier signaled queue.

Like DTS, DRS also uses a set of prioritized work queue and a pool of worker threads to execute work items in the queue. IDMonitor thread in Figure 5, like the work dispatcher thread in DUM, calls the above mentioned Qpid API function after it is initialized and goes to wait for incoming data event. When awakened by an event, it creates an DRS work item for transferring data from the Qpid queue specified by the event augment and inserts the work item in the DRS prioritized work queue. The IDMonitor thread continues to process events as long as some is waiting in the incoming event queue. When the queue becomes empty, the thread calls QpidLongPollAll() again and goes back to wait for incoming data event.

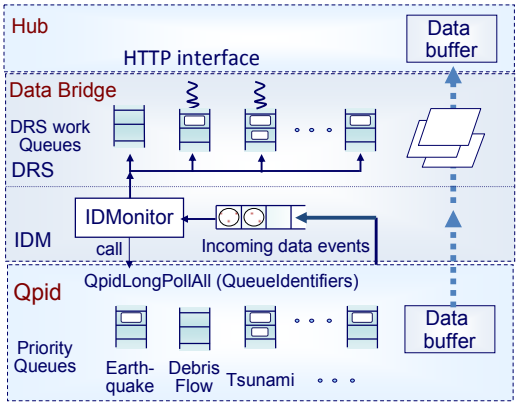

Figure 5. Queuing facility for incoming messages,

When a work item in a DRS work queue is executed, the new data in the specified Qpid data queue is transferred to the data bridge under the supervision of the DRS worker thread. Then the worker thread pings the hub and handles the handshake with the hub according to HTTP-based interface of the hub. After the hub has successfully picked up the data, the worker thread go back to look for work in the work queues and executes the highest priority work item if one is found or goes back to wait for work if it finds no work item is waiting.

V. Real-Time Performance

From expression (1), one can see that the end-to-end delay of an alert message is dominated by the term $5D_N$. the total delay suffered by data and protocol packets associated with a message as they traverse the Internet. Published data [24-26] on maximum network delay D_N under normal traffic conditions ranges from under 10 ms when the nodes are local (e.g., in the same building and building complex), to 20 to 40 ms when nodes are nearby (e.g., within a geographical region in a country), to 200 – 400 ms when the nodes are far apart (e.g., across a large country and multiple continents). For all likely scenarios, disaster alerts are sent by regional authorities, targeting areas within the region. Time critical alerts warning of strong earthquakes, tsunamis, flash floods, etc. are sent when the disaster is still imminent, network connectivity has not yet been disrupted, and network traffic is still normal. For this reason, a conservative estimate is $5D_N = 200$ ms.

One expects that when the PQ nodes at both the sending end and destination end execute on modern desktop computers, residence times $D_{PQ}(s)$ and $D_{PQ}(d)$ are very small compared with D_N. Moreover, the nodes should have sufficient bandwidth to meet the total bandwidth demands of all authorized alert senders in common disaster scenarios. To be sure that this is indeed the case, we are experimenting with several alternative configurations of PQ nodes (including different resource allocations to hub, modules of the data bridge and Qpid; numbers of worker threads, and so on). We plan to measure residence times of messages of different priorities and total throughputs of outgoing messages and incoming messages for different combinations of configuration parameters.

A. Experiment Set Up

The data reported here was collected from the first of the experiments. This experiment aims to answer the fundamental question of whether time-critical disaster alerts (e.g., strong earthquake alerts) requiring a fraction of a second delay can be sent asynchronously over the Internet. The answer clearly depends on whether the end-to-end delay of the highest priority alerts is a small fraction of a second (e.g., 200-300 ms and in the worst case 0.5 second). Giving that $5D_N$ can be as large as 200 ms, the residence time of the highest priority messages through PQ nodes indeed must be negligibly small.

The experiment focused on the performance of the PQ node serving alert senders: The scenario assumed by the experiment is that a dedicated PQ node is used by a regional disaster management office and emergency operations center (EOC) to serve disaster alert authorities who are responsible for publishing disaster alerts to people and iGaDs in the region. The node is not used for collection of sensor data, message exchanges and communication among agencies, and so on: Other PQ nodes (or simply Qpid nodes) are used to support the subscriptions of the EOC to data and messages (e.g., weather forecast, road conditions, and news) from other publishers. Consequently, the PQ node on which performance data were taken has no incoming messages. This simplifying yet realistic scenario allows us to ignore complicating factors, including how residence time and throughput depend on allocations of resources to components that handle incoming and outgoing traffic through the data bridge.

Table 1 describes the platform on which the tested PQ node ran. Intel Core i7 has four hyper-threaded physical cores, hence 8 virtual cores. The experiment did not use the hypervisor: Push hub, data bridge middleware and the Qpid broker ran on the same operating system, contenting for resources under the control of the OS. In the experiment, the data update monitor (DUM) is singled threaded. The data transfer service module either uses only one worker thread or 7 worker threads.

TABLE I. CONFIGURATION OF TEST PLATFORM

Hardware Platform	Intel Core i7 with 32GB DDR III RAM, 500GB 7200 rpm Hard Disk
Hypervisor	VMWare Esxi
Common Resource	No CPU reservation, 4GB memory, 50GB virtual hard disk, virtual Gigabit LAN, Fedora OS
VM1	PuSH Hub, Data bridge, and Qpid

B. Minimum Residence Time

In the first part of the experiment, we measured the minimum residence time of messages of various sizes. This is the residence time of a highest priority message (e.g., a strong earthquake alert) that arrives at the PQ node when the node has no equal priority messages to process. (In other words, the queuing time of the message is zero.) The results are plotted in Figure 6. By 1+1 (or 1+7) threads, we mean that the data bridge is configured to use one monitor thread and one worker thread (or 7 worker threads).

In addition to minimum residence time, Figure 6 also shows how service times of messages depend on their sizes. The *service time* of a message is the time taken by the data bridge to move the message from the message queue of the hub to one of the priority queues of the Qpid broker. To simulate CAP messages in the experiment, each message was divided into multiple lines of 100-byte text. The larger the message, the more time is required to load the whole text file line by line into memory, and consequently, the larger the service time. According to the OASIS CAP [16] white paper, sizes of typical CAP message are less than 10K bytes. On the tested PQ node, their service times are 200 microseconds or less.

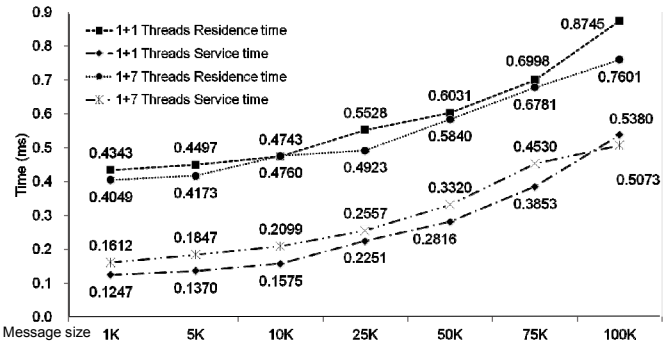

Figure 6. Minimum residence time in PQ node

The difference between the residence time and the service is the *latency overhead* of the PQ node, i.e., the time spent by the data bridge and Qpid broker to create work items, insert the items in queues, remove them from queues and so on. Figure 6 shows that the latency overhead is essential independent of message size as one expects. Similarly, as one expects, the latency overhead is smaller when 7 worker threads are used: Figure 6 shows a decrease of 50% or less.

In contrast, when there are 7 worker threads, messages of all sizes have larger service times: for some sizes, by as much as 70 microseconds. Consequently, the gain in residence time achieved by using more worker threads is small. The increase in service time is likely due to the overhead in managing thread priorities and lock contention, which we will work to improve in later versions of PQ node.

C. Throughput Per Core

In the second part of the experiment, we measured the throughput of PQ node. In this experiment, we send messages to priority queues of the data bridge at a higher rate than the middleware can actually process. Specifically, the message queue of the PuSH hub and priority queues of the data bridge are never empty.

Figure 7 shows the throughput of the tested PQ node in which DTS uses only one worker thread (i.e., with 1+1 threads configuration.) Again, typical CAP messages are 10K bytes or shorter. (Disaster alert messages in the CAP standard format contain only metadata (e.g., URLs) on large multimedia files, not the files themselves.) This means that the PQ node can push over two thousand CAP messages per second even when the data bridge uses only one worker thread to move data from the hub to Qpid. This throughput is much more than adequate for pushing disaster alerts in CAP format by a regional office in most likely disaster scenarios.

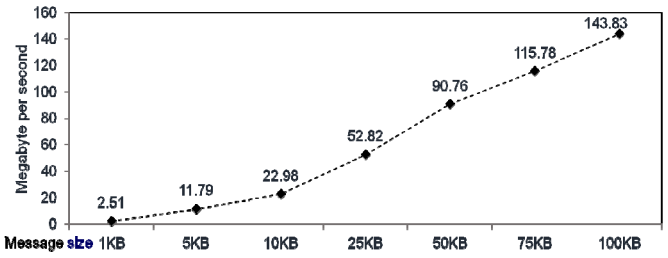

Figure 7. Throughput of PQ node (1+1 threads)

VI. Summary and Future Work

Asynchronous Message Delivery Service (AMeDS) [19] was proposed as way to deliver disaster alert messages to hundreds and thousands of iGaDs via the Internet in real-time. AMeDS uses PQ nodes at the edges of the network. Such a node is composed of a PuSH hub [20] and a Qpid broker [21]. These components of the node are linked by a data bridge, which is responsible for moving incoming and outgoing data/messages from (to) the message queue of the hub to (from) the prioritized data queues of Qpid.

The structures of the data bridge modules responsible for moving data/message between the hub and Qpid broker were described in detail in Section IV. One can see from the description there, the data bridge provides data/messages in each direction with a set of priority queues and monitor and worker threads to support end-to-end priority scheduling of their transits through the PQ node.

The previous sections also presented performance data obtained from measurements done on a proof-of-concept PQ node. The preliminary data on residence time of highest-priority messages through the PQ node serving the publishers show that the delay introduced by the node is very small (in order of 100 microseconds) and the end-to-end delay of alert messages through the Internet is within a small fraction of a second. Thus, the data removed our doubt on whether AMeDS can meet end-to-end delay requirements of time-critical alerts warning of imminent strong earthquakes, flash floods, etc.

The preliminary data on throughput taken from the PQ node prototype show that the throughput demands of all authorized alert senders in likely scenarios can be met even with the data transfer service module uses only one worker thread. We will investigate further in the future whether it is indeed possible to thus simplify the configuration of the data bridge. As stated earlier, we plan to measure residence times of messages of different priorities and total throughputs of outgoing messages and incoming messages for different combinations of configuration parameters. We will also experiment with ways to allocate resources to major components with a PQ node, including the use of multiple VM's to protect the components from ill effects of resource contentions with each other.

Acknowledgement

This work was supported by the Taiwan Academia Sinica Open Information Systems for Disaster Management project.

References

[1] "Case study on developing new technologies to increase tornado warning time," Georgia Tech, Severe Storms Research Center, http://www.gtri.gatech.edu/casestudy/twister-technology

[2] D. Coulter and T. Phillips, "New GOES-R to give more tornado warning time," http://ephemeris.sjaa.net/1108/d.html

[3] T. C. Shin, et al., "Strong motion instrumentation programs in Taiwan," in Handbook of Earthquake and Engineering Seismology, W. H. K. Lee, H. Kanamori and P. C. Jennings, Ed. Academic Press, 2003

[4] Focal mechanism, http://en.wikipedia.org/wiki/Focal_mechanism

[5] G. P. Hayes, L. Rivera, and H. Kanamori, "Source inversion of the W-Phase: real-time implementation and extension to low magnitudes," *Bull. Seism. Soc. Am.*, 80, 2009.

[6] N. C. Hsiao, et al., "Development of earthquake early warning system in Taiwan," *Geophys. Research Letters,* 36, 2009.

[7] C. Buratti, A. Conti, D. Darkari, and B. Verdone, "An overview on wireless sensor networks technology," Sensors, 2009

[8] R. Sherwood and S. Chien, "Sensor Web: a new paradigm for operations," in *Proceedings of International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations*, June 2007

[9] SANY - an open service architecture for sensor networks, edited by M. Klopfer and I. Simons, http://sany-ip.eu/publications/3317, 2009.

[10] IPAWS-OPEN (Integrated Public Alert and Warning System – Open Platform for Emergency Networks), at http://www.fema.gov/emergency/ipaws/about.shtm

[11] Global Disaster Alert and Coordination System, GDACS, http://w3.gdacs.org/

[12] European Public Warning System (EU-Alert) Using Cell Broadcast, ETSI TS 102900 V1.1.1 Technical Specification, ETSI, 2010

[13] Worldwide PWS Initiative, http://www.one2many.eu/en/portfolio/emergency-alerts/worldwide-initiatives

[14] "Survey on Public Warning Systems in Europe," publication of European Emergency Number Association, December 2012

[15] EDXL-DE: Emergency Data Exchange Language Distribution Element, V1.0, at http://www.oasis-open.org/committees/download.php/17227/EDXL-DE_Spec_v1.0.html

[16] CAP: Common Alerting Protocol, V1.2, http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.html

[17] J. W. S. Liu, E. T.-H. Chu and C. S. Shih, "Cyber-physical elements of disaster prepared smart environments," IEEE *Computer,* February 2013.

[18] W. P. Liao, Y. Z. Ou, E. T. H. Chu, C. S. Shih, and J. W. S. Liu, "Ubiquitous Smart Devices and Applications for Disaster Preparedness," *Proceedings of The 2012 International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies,* September 2012

[19] Y. Z. Ou, et al., "An asynchronous message delivery service for iGaDs," to appear in Proceedings of nternational Workshop on Extending Seamlessly to the Internet of Things, July 2013.

[20] Bradfitz, Bslatkin, Andyster and Bradfitzgoog, "pubsubhubbub," https://code.google.com/p/pubsubhubbub/ , 11/27/2012.

[21] The Apache Software Foundation, "Apache QpidTM Open Source AMQP Messaging," http://qpid.apache.org/index.html , 11/27/2012.

[22] The Apache Software Foundation, "AMQP Messaging Broker (Implemented in C++)," http://qpid.apache.org/books/0.16/AMQP-Messaging-Broker-CPP-Book/pdf/AMQP-Messaging-Broker-CPP-Book.pdf , 11/27/2012.

[23] Deadline-monotonic scheduling, http://en.wikipedia.org/wiki/Deadline-monotonic_scheduling .

[24] MIT King Data Set at http://pdos.csail.mit.edu/p2psim/kingdata/ and King Blog Data at http://www.eecs.harvard.edu/~syrah/nc/

[25] M. Kalman and B. Girod, "Modeling the delays of successfully transmitted packets", *Proceedings of IEEE International Conference on Multimedia and Expo*, 2004.

[26] G. Hooghiemstra and P. V. Miegham, "Delay distributions over mixed Internet paths," Delft University of Technology, Technical Report 20011031, 2001.