© 2015 IJSRSET | Volume 1 | Issue 5 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Science and Technology

A Study on Monitoring of Visitors in Dynamic Network

Dr. M. Sivasakthi[1], M. Nester Jeyakumar[2]
[1]Asst. Professor & Head, Department of MCA, MASS College of Arts and Science, Chennai, Tamilnadu, India
[2]Asst. Professor, Department of Computer Science, Loyola College, Chennai, Tamilnadu, India

ABSTRACT

Wireless Network is very broad area, which include set of nodes those communicate through radio waves. Dynamics and Portability are important aspect of Wireless Network. In this paper we present a monitoring system for a dynamic network, in which a set of domain nodes shares the responsibility for producing and storing monitoring information about a set of visitors. This information is stored persistently when the set of domain nodes grows and shrinks. Such a system can be used to store traffic or other logs for auditing, or can be used as a subroutine for many applications to allow significant increases in functionality and reliability. The features of the system include authenticating visitors, monitoring their traffic through the domain, and storing this information in a persistent, efficient, and searchable manner. From a theoretical outlook our system performs fighting fit, but it would certainly be interesting to see how it would perform in real life.

Keywords: Survivable Monitoring, Network Intrusion Detection, Emergency Communication

I. INTRODUCTION

In dynamic network, network configuration being rearranged at every time when subscriber moves into different base station. Dynamic overlay networks have recently attracted a lot of attention due to the enormous interest in peer-to-peer systems and wireless ad-hoc networks.

We present a monitoring system which collects and stores information about visiting participants in a network. The information is made available upon request and can be subsequently analyzed and used for any purpose by an administrator. Methods for authentication ensure that visitor nodes are identified before being allowed to communicate; message encryption within the network ensures that no node can impersonate a domain node or send messages through the domain nodes except through the proper monitoring process.

Depending on how the information collected by the system and it is being used, there are several applications such as persistent audit logs, network intrusion detection, and emergency systems.

1.1 Problem description

We assume that there is two different kinds of nodes, visitors and domain nodes, and that the visitors are untrusted and the domain nodes are trusted. The task of the domain nodes is to monitor all activities of the visitors which involve the network. They also store a distributed database containing recorded monitoring information for all visitors. There are three components to this monitoring process:

- Traffic of the visitors has to be cached.
- The intercepted traffic must be processed to produce relevant monitoring information.
- This information must be stored permanently. We focus primarily on the last of these, studying a distributed database and algorithms for the storage of this information. The requirements of such a database are as follows:

Authentication: The system must be able to identify visitors accurately to ensure that stored information can be correctly matched to a visitor.

Search ability: The database must be searchable, in the sense that an administrator must be able to acquire all

information about a particular visitor wishing to connect to the network.

Persistence: The database must be persistent, in the sense that no entries in the database can be lost by network disruptions.

Efficiency: The algorithms for maintaining and using the database should run with minimal communication and computational overhead.

II. METHODS AND MATERIAL

A. Related Work

Emergency communication systems are becoming increasingly popular but, none of the existing and proposed systems operates over a dynamic network and provides an open access policy that allows visitors to communicate; see [20] for a survey of existing emergency systems. Intrusion detection for distributed systems is a very active research area and several systems have been proposed that can be classified based on the approach employed by the detector.

For instance, DIDS [17] and NSTAT [9] are systems based on the centralized analysis approach where audit data is collected on individual nodes and then reported to a centralized location where the intrusion detection analysis is performed. In GrIDS [18] and EMERALD [7], systems based on the hierarchical analysis approach, audit data is collected and analyzed by each node and the results of the analysis is reported according to some hierarchical structure.

We study the load balancing and recovery mechanisms built on top of the overlay network SPON [14], which was designed for reliable broadcasting in dynamic networks. Extensive research has been recently carried out on the design of overlay networks that support arrivals and departures of nodes. Recent systems projects on such networks include Freenet [4], Ohaha [11], Archival Intermemory [3], and the Globe system [1].

Theoretically well-founded peer-to-peer networks have also been presented, such as Pastry [15], Tapestry [10], Chord [19] along with SPON. With the exception of SPON, the topologies of these networks are based on DNS-like, hyper cubic, or random constructions, which are either not useful or far too complex .Recently, a new backup system based on peer-to-peer overlay networks has been proposed in [5], similar to an approach previously suggested in other works, including, for example, [2, 6, 7, 13, 16]; the scope of these systems is to backup entire file systems. The storage component of the system studied here is designed solely to store monitoring information, allows us to fulfill our requirements while achieving provable efficiency, which more expensive systems cannot.

B. System Overview and Its Components

The monitoring information could be exchanged, but this would generate significant communication overhead. The monitoring information could be left at the domain node that collected it, and collected only when needed; this saves unnecessary message passing, but can cause load imbalances and can exceed the capacity of domain nodes. Layers of Secure Monitoring protocol are presented herewith.

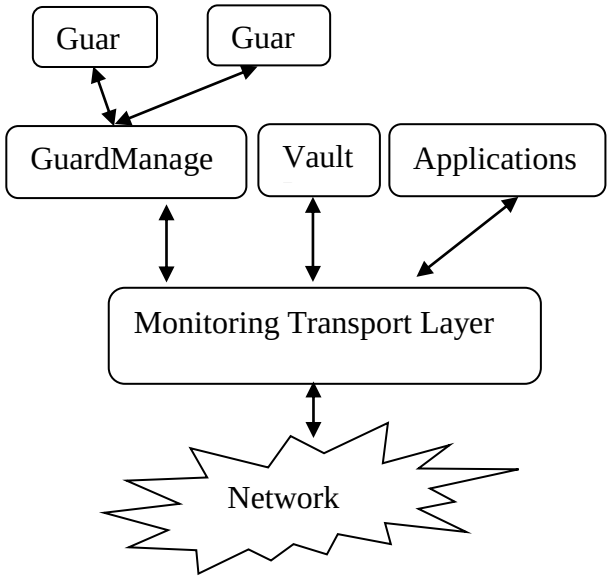


Figure 1. Layers of Secure Monitoring protocol

1. Visitors:

The visitor is responsible for authenticating its messages by signing them, so that a domain node receiving them can properly match the traffic to the node. Any unsigned messages from a visitor are ignored.

sign**()**: authenticate itself in its message.

2. Domain Node

The monitoring transport layer receives all messages arriving from the network. It passes messages which are not valid domain messages to the guard process, and routes valid domain messages to the guard and vault processes and to any applications in use according to their destination. It also signs all messages from the node, from any process, to mark them as valid domain messages.

Sign**():** sign all outgoing messages as valid domain messages

Route(): route incoming messages to appropriate processes

The guard process verifies the identity of a visitor and clears it with the supervisor when it first connects. On the first and subsequent connections, the guard forwards the visitor's messages into the network, and also produces monitoring information about the visitor's messages. We use a single guard manager in the domain node which spawns independent guard processes for each visitor connecting through it.

Interface of guard manager

New**():** Spawn a new guard process to handle a new visitor

Delete**()**: delete a guard process.

Interfaces of guard

Check():query supervisor regarding a visitor

Monitor():produce monitoring information

Forward():send a visitor's message

Page():request a node to send monitoring information

Upload():send monitoring information to a vault.

The vault process is responsible for the storage of monitoring information assigned to it.

Interfaces of Vault

Join():join a heap

Leave(): leave a heap

Page(): request a node to monitoring information to

Move(): move data to another vault

Heapify(): rearrange with neighbours in heap

Search(): search locally stored information for a specific node

Write(): write monitoring information locally

3 Supervisor

The supervisor is a single domain node known by all other domain nodes, and is also a process running on that node which performs the supervisor functions.

Interfaces of supervisor

Insert(): add a vault to the backup heap

Remove():remove a vault from its heap

Check(): see if a visitor is in the blacklist

Update-list():update a blacklist when told

Get-lightest():return the top vault in the active heap.

Switch-heap():active the backup heap

4 Administrator

The exact functioning of the administrator is beyond the scope of this paper. In general, the administrator initiates data collection through broadcasts through the domain, in order to retrieve all monitoring information about a set of visitors. If broadcasting is not a primitive in the domain, a strategy such as [29] can be used to perform reliable broadcasting using a unicast primitive.

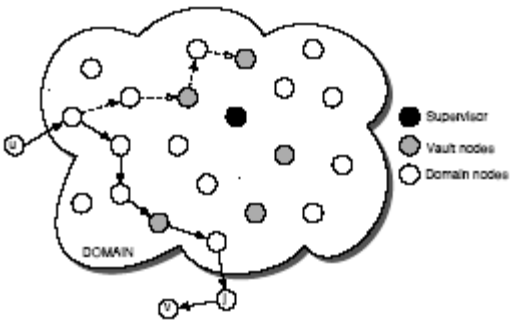


Figure 2: The flow of message from a visitor through the network to another visitor. The Solid path is message path and doted path is the path of some monitoring information

Flow path of messages

Messages can be freely exchanged between domain nodes. A message from a visitor node is stopped at the first domain node it reaches (which may change over time as the visitor and domain nodes move around), and the node determines whether or not to let the visitor send to the network by contacting the network supervisor. The domain node monitors the traffic of the visitor after

it is cleared by the supervisor. Monitoring information is distributed through the domain by being sent piecemeal through the network to vaults, and can be accessed and used by an administrator, for example to update the network's acceptance policies for visitors. A sample overview of the flow of a message is given in Figure 2.

III. RESULTS AND DISCUSSION

A. Algorithm

1 Cryptographic Algorithm

Effective monitoring is only possible if untrusted nodes cannot create multiple or false identities, and if the complete traffic to and from an untrusted node is monitored, filtered, and stored. Central requirements for a monitoring system are:

1. Untrusted nodes must be uniquely identifiable.

This can be achieved via a wide range of standard authentication techniques, from password-based systems to digital certificates that bind node identifiers to public keys. (This is similar to the unique network identifier in intrusion detection systems [33, 18].)

2. Domain nodes should be able to communicate securely.

Domain nodes should be able to communicate so that outsiders cannot read, modify or inject messages. This can be achieved via standard techniques although techniques based on public-key cryptography should be kept at minimum whenever domain nodes are mobile, since mobile nodes often rely on battery power which can be consumed rapidly by CPU-intensive operations. Depending on network conditions there are a number of standard solutions to these requirements, including public-key cryptography, shared keys, and group key communication protocols. we discuss a set of solutions to these issues, designed for a single application. This section by no means represents the only way to implement the general system.

2 Data Management Algorithms

2.1 Guards, pages of logs, and temporary page storage

As the guard monitors the visitor, it stores this information in a temporary fixed size page of storage space; when this page is filled, the guard requests a destination from the supervisor through page(), receives a network address, and calls upload() to send the page to the address to be stored in that node's vault process. The guard's temporary page can then be erased and reused. Collecting the data into pages improves the efficiency of the supervisor, since each store operation includes a certain overhead cost independent of the amount of data being stored. But if a page is too large, or if all data is stored at the guard, then the load can become unbalanced.

2.2 Vaults and SPON-based heaps

For permanent storage of pages, vaults are organized into structures based on the SPON network developed in [14] and discussed in section 4. The SPON topology is a rooted tree structure consisting of multiple trees of varying depths similar to a binomial heap; it tolerates single node insertions and removals through replacement in constant time per operation under the assumption that the roots of all trees are stored in an array at a supervisor node.

On top of this topology a heap is maintained, where heaping is performed according to the maximum space available at a node, such that each node has at least as much free space for storage as its children. This is done through the heapify() calls of each node, which need to occur only when a node joins or leaves (when its replacement is inserted) or when a node is given additional load. An inserted node queries its new parent and children (as applicable).

If it has more free space than its parent, they exchange places by exchanging adjacent node information and informing their neighbors as well; this requires O(1) rounds and messages. Then the node continues to move itself up the tree querying its new parent and exchanging until a terminal location is found; each round requires O(1) messages and rounds of communication, and the supervisor does not need to be involved in any of the operations. If an inserted node or a node given additional load has less free space than its children, it exchanges places with the child of most free load, and continues to

query its new children and exchange until it is in place. In this way the root of the rightmost tree is always the node with the most room.

2.3 Types of vaults and heaps

There are three types of vaults: old, spare, and active. There are also two separate heaps maintained in the system, the active heap and the backup heap. Active vaults are connected to the active heap, and spare vaults are connected to the backup heap; The root node of the active heap is sent by the supervisor to the next guard node to request storage for a page.

B. HEAP Structure

We study a tree-based network called SPON which manages group updates and supports efficient broadcasting. SPON uses a supervisor peer to maintain the network during node arrivals and departures and routes broadcasts using direct connections between nodes. SPON is capable of performing reliable broadcasting in unreliable networks.

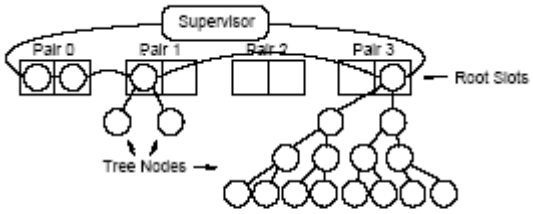


Figure 3: A sample network

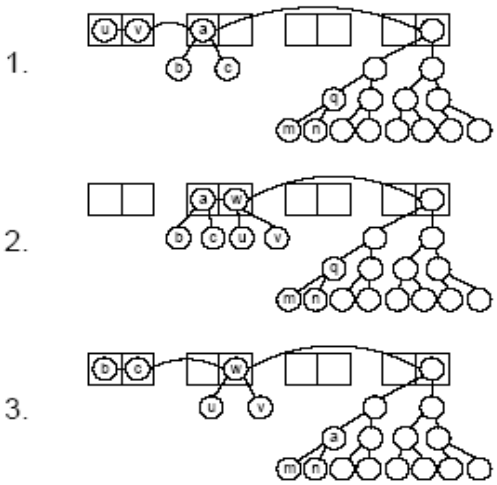
Any root node in a slot of pair i is the root of a complete binary tree of nodes of depth i. At most one slot pair is fully occupied, and below this pair there is no occupied slot. Furthermore, every root node maintains a link to the closest root node to the right and to the left of the array of root slots, and the leftmost and rightmost root nodes maintain a link to the supervisor as shown in Figure 3.

Thus, every root node in the SPON structure has a degree of at most 4. Tree nodes maintain links to a parent and to a left and right child (when appropriate) in the tree, and thus have degree at most 3

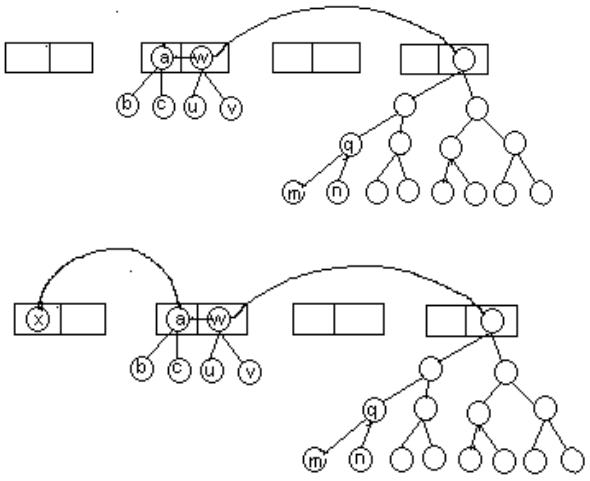
1 Join and leave

A join request can be sent to any node in the system by a new node wishing to join the network. This request is then forwarded to the supervisor, who then processes the request through a function called Integrate(v) to insert a new node v into the data structure. When some node w leaves the system, it performs a function called Replace(w,N[w]) so the supervisor can replace it with a new node reconnected to N[w].

The operation UpdateRootLinks() in these functions makes sure that at the end the links between the root nodes satisfy Invariant 4.1. For a possible outcome, see Fig. 4. Next we show that the algorithms Integrate and Replace indeed preserve Invariant 4.1.



1. A sample network containing 20 nodes.
2. Node w joins; the supervisor assigns u and v to be its children.
3. Node q leaves; the supervisor selects a as its replacement and sends a's children b and c to the open slots in level 0.



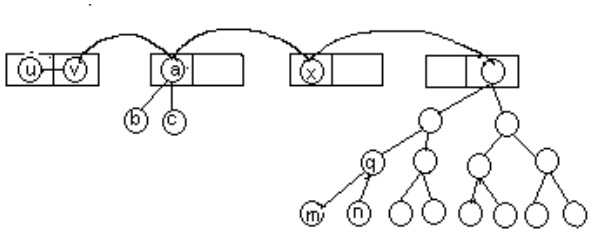


Figure 5:
1. A sample network containing 21 Nodes
2. Node x joins the supervisor place it in open slot pair 0
3. Node w leaves; the supervisor remove w from its slot and place w's childrean in pair 0.the existing root node shifted to pair 2.

C. Analysis

1. Control messages

We analyze the cost of control messages through the following comparison to the cost of data movements.

Lemma 6.1.1: Except for messages to process nodes joining and leaving heaps, control message cost is at most a constant multiple of data movement cost. Proof. Other than nodes joining and leaving heaps, control messages are triggered by two types of events: visitor communication and page movement.

The initial communication of a visitor to a guard causes the guard to check with the supervisor to see if the node is okay; this requires constant work. An optimal algorithm still must send the message from the visitor to its destination. Hence, for the visitor communication the algorithm only creates a constant overhead and is therefore constant competitive. Page movement control messages are identical regardless of whether a page is moved from a guard to a vault or from one vault to another: the source process must request a destination node from the supervisor, which responds, and after moving the data, the destination node may need to heapify itself.

This total process requires up to O(log n) messages. But a page of data is moved in this process, and since we assumed above that a page of data is by at least a logarithmic factor larger than a control message, the cost of sending control messages in this case is within a constant factor of sending pages of data, which completes the proof of the lemma.

Notice that node join and graceful leave operations can be processed with O(1) control messages in SPON. Hence, we can ignore the cost of control messages in our competitive analysis.

2. Data movements

Recall that OPT denotes any algorithm with an optimal cost for every sequence of operations. When data is written to the active heap in our algorithm, the optimal algorithm OPT may instead write the data to a different node in the active heap or to a node in the backup heap. Let us consider a sub optimal extension of OPT, SUB, which always writes the data first to a node in the active heap; this is always possible since the active heap is by definition not full, since if it fills it stops being active. If OPT would have assigned that data to a node currently in the backup heap, then SUB moves the data to that node when its first node fails.

From these rules it follows that the cost of SUB is at most twice the cost of OPT under any circumstances, because it moves any set of data at most twice as often as OPT.

Lemma 2.1: The amount of data in the vaults in the backup and active heaps in the algorithm is at most the amount in the same vaults in SUB.

Proof. This holds because in the algorithm all nodes not in the backup and active heaps are full (in the sense of having less than a page free), and consequently must be holding at least as much information as SUB and OPT can hold in these nodes.

Lemma 2.3: Data movements caused by departures of vaults not in the active heap is constant competitive to SUB.

Proof. At time t, let A be the set of vaults in the active heap, B the vaults in the backup heap, S the set of all old vaults (not in either heap), and V the entire set of vaults, so that V = A∪ B ∪ S. Of the load stored in A ∪ B in both algorithms, some will have been first placed in A ∪ B and some will have been moved in when a vault in S departed. Because SUB places data first in the currently active set, the amount of load in A ∪ B placed in A∪ B to begin with is the amount of load placed in A to begin

with, which is the same in both algorithms since both first place all load in A. According to Lemma 5.2, SUB must have at least as much load in A ∪ B as our algorithm. Therefore SUB must have moved at least as much data into the active heap from vaults not in the active heap as in the algorithm.

IV. CONCLUSION

We deliberated an efficient monitoring system for dynamic networks. The system produces and stores monitoring information in a persistent manner about visiting nodes in the network. The information is searchable and available to system administrators. Here a novel data reallocation mechanism that ensures that no monitoring information is lost even if several nodes depart ungracefully. The storage process is O(log n)-competitive in the number of network messages with respect to an optimal offline algorithm and this is as good as any online algorithm can be. Hence, from a theoretical perspective the monitoring system performs well. The monitoring system can be used as a building block for the collection of persistent audit logs, network intrusion detection, and in emergency systems.

V. REFERENCES

[1] A. Bakker, E. Amade, G. Ballintijn, I. Kuz, P. Verkaik, I. van der Wijk, M. van Steen, and A. Tanenbaum. The Globe distribution network. In Proceedings of the 2000 USENIX Annual Conference (FREENIX Track), pp. 141-152, 2000.

[2] C. Batten, K. Barr, A. Saraf, and S. Trepetin. pStore: A secure peer-to-peer backup system. Technical Report, MIT Laboratory for Computer Science, December 2001.

[3] Y. Chen, J. Edler, A. Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos. A prototype implementation of archival intermemory. In Proceedings of the 4th ACM Conference on Digital Libraries, pp. 28–37, 1999.

[4] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous infor- mation storage and retrieval system. In Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, 2000. http://freenet.sourceforge.net.

[5] L.P. Cox and B.D. Noble. Pastiche: making backup cheap and easy. In the Fifth USENIX Symposium on Operating Systems Design and Implementation, December 2002, Boston, MA.

[6] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proceedings of the 18th ACM Symposium on Operating Systems Principles.October 2001.

[7] S. Elnikety, M. Lillibridge, M. Burrows, and W. Zwaenepoel. Cooperative backup system. In the USENIX Conference on File and Storage Technologies, Monterey, CA, January 2002.

[8] Jinho Ahn, Fault-tolerant Mobile Agent-based Monitoring Mechanism for Highly Dynamic Distributed Networks, International Journal of Computer Science Issues, Vol. 7, Issue 3, No 3, May 2010

[9] R. A. Kemmer. NSTAT: A Model-based Real-Time Network Intrusion Detection System. University of California-Santa Barbara Technical Report TRCS97-18, Nov. 1997.

[10] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea,H.Weatherspoon, W.Weimer, C.Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), pp. 190–201,2000.

[11] Ohaha, Smart decentralized peer-to-peer sharing. http://www.ohaha.com/design.html

[12] P.A. Porras and P.G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In 19th National Information System Security Conference (NISSC), 1997.

[13] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. IEEE Internet Computing, 5(5):40-49, Sept. 2001.

[14] C. Riley, C. Scheideler. Guaranteed Broadcasting Using SPON: Supervised Peer Overlay Network. Technical report, Johns Hopkins University, 2003. http://www.cs.jhu.edu/~chrisr/papers/spon tr.ps.gz.

[15] Rowstron A and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Sys- tems Platforms. Germany, Nov. 2001.

[16] Rowstron A and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In Proceedings of the 18th Symposium on Operating Systems Principles (SOSP '01), pp. 188–201, 2001.

[17] S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and An Early Prototype. In Proceedings of the 14th National Computer Security Conference, October 1991.

[18] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS-A Graph Based Intrusion Detection System for Large Networks. In 20th National Information System Security Conference (NISSC), October 1996.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001), ppp. 149–160, 2001.

[20] Murray Turoff. Past and Future Emergency Response Information Systems. In Communica-tion of the ACM, April 2002/Vol. 45, No.4. Pages 29-32.