Building A Monitoring Overlay for Peer-to-Peer Streaming

Xing Jin Qiuyan Xia S.-H. Gary Chan
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{csvenus, xiaqy, gchan}@cse.ust.hk

Abstract— **Current peer-to-peer (P2P) streaming systems often assume that nodes are cooperative to upload and download data. However, in the open environment of the Internet, this is not true and there exist malicious nodes in the system. In this paper, we study malicious actions that can be detected through peer-based monitoring. We require each node to monitor the data received and to periodically send out monitoring messages about its neighbors to some trustworthy nodes. To efficiently store and search the messages among multiple trustworthy nodes, we organize the trustworthy nodes into a threaded binary tree. The trustworthy nodes also dynamically redistribute the monitoring messages among them to achieve load balancing. Our simulation results show that this scheme can efficiently detect malicious nodes with high accuracy, and that the dynamic redistribution method can achieve good load balancing among the trustworthy nodes.**

I. Introduction

With the popularity of broadband Internet access, there has been increasing interest in media streaming. Recently, P2P streaming has emerged to overcome limitations in traditional server-based streaming. In P2P streaming, cooperative peers self-organize themselves into overlay networks via unicast tunnels. They cache and relay data for each other, thereby eliminating the need of powerful servers from the system. Current P2P streaming tools have been shown to be able to support up to thousands of peers with acceptable quality of services [1], [2].

Most proposed P2P streaming systems focus on improving streaming quality and assume that all nodes cooperate as desired. However, this may not be true in the open environment of the Internet. Some nodes in the system may be selfish and unwilling to upload data to others. Some may have abnormal actions such as frequent rebooting which adversely affect their neighbors. More seriously, some nodes may cheat their neighbors, launch attacks to disrupt the service or distribute viruses in the overlay network. Following the notations in [3], we call these uncooperative, abnormal or attacking behavior *malicious actions* and the corresponding nodes *malicious nodes*.

In this paper, we focus on the malicious actions that can be detected through nodes' past performance and study a detecting scheme to identify malicious nodes in the system. In our scheme, each node keeps monitoring its neighbors and periodically generates monitoring messages. The monitoring messages are collected somewhere to compute node's reputation. Once a malicious action of a node is detected, the reputation of the node is decreased. In this way, nodes frequently conducting malicious actions are likely to have low reputation, and a node whose reputation is lower than a certain threshold can be identified as malicious. We consider two important issues in the design. Firstly, we study where the monitoring messages are stored. If the messages are stored at some normal nodes, the nodes may modify or forge the data. Therefore, we use a monitoring overlay formed by a set of trustworthy nodes to manage the messages. These nodes are certificated by a trusted third-party and are fully trustworthy (e.g., pre-deployed proxies). In the following, we call a trustworthy node in the monitoring overlay a *monitoring node*, and a normal node in the streaming overlay a *streaming node*. The monitoring nodes organize themselves into a threaded binary tree to ensure that an update or query to a streaming node's reputation can be quickly accomplished. The monitoring nodes also dynamically redistribute their management loads to achieve load balancing among them. Secondly, a streaming node may lie about the performance of its neighbors in the monitoring messages. Such a node can affect the evaluation as it desires. To detect node lying in the monitoring messages, we employ a traditional approach of monitoring *suspicious messages* [4]. In detail, to evaluate the performance of a data sender in a certain duration, both the sender and the corresponding receiver generate monitoring messages. If there is an obvious gap between the two messages, the messages are regarded as suspicious. Each streaming node is then assigned a credit value based on the number of suspicious messages it generates, which indicates to what extend a later monitoring message from the node can be trusted. We then integrate the credit values of nodes into reputation computing to address the node lying problem.

We have conducted simulations to evaluate our scheme. The results show that it can detect malicious nodes with low false positive and false negative rates, and the loads on monitoring nodes can be efficiently balanced by the dynamic redistribution method.

This work was supported, in part, by Competitive Earmarked Research Grant (HKUST6156/03E) of the Research Grant Council in Hong Kong, and Innovation and Technology Commission of the Hong Kong Special Administrative Region, China (GHP/045/05).

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

We briefly review previous work on P2P reputation as follows. Trust-Aware Multicast (TAM) computes a level of trust for each node according to their past performance and builds a multicast tree based on the trustworthiness of nodes [5]. However, the trustworthiness, or reputation of nodes is maintained at the root, which limits the system scalability. Our scheme considers a set of trustworthy nodes instead of a single one. We accordingly design the cooperation and load balancing mechanisms among the trustworthy nodes. In NICE trust model, each node holds the reputation of its transaction partners according to the quality of transactions [6]. All the nodes further form a trust graph according to the reputation values. Later on, an overlay path between two nodes is selected as the most trustworthy path between them in the trust graph. P-Grid trust model and EigenTrust use DHT-like (Distributed Hash Table) systems to store and search node reputation [7], [8]. However, none of these approaches have considered secure reputation computing. A message with reputation/trust values may be intercepted, modified or discarded in transmission. A malicious node may lie to others or forge messages to affect the reputation computing as it desires. Our scheme uses a set of trustworthy nodes to manage reputation. These nodes are fully trustworthy and never cheat in reputation computing. Furthermore, we organize the trustworthy nodes into a threaded binary tree instead of a DHT network. This is because DHT has large setup and maintenance costs, and the management loads on DHT nodes are often unevenly distributed. Instead, a binary tree is much easier to maintain, and we can use a dynamic load redistribution method to balance the loads on nodes.

The rest of the paper is organized as follows: In Section II we discuss the design of the system. In Section III we present the simulation results. Finally, we conclude in Section IV.

II. System Design

A. Design Overview

As discussed above, we are interested in the malicious actions that can be detected by monitoring nodes' past performance. Examples include Eclipse attack, resource-consuming attack and distributing corrupt data [3]. To detect such actions in a streaming system, we build a reputation system among streaming nodes based on their history performance. The more malicious actions a streaming node conducts, the less reputation it gets. Note that there are many ways to use the reputation results. In one case, if node A wants to set up a connection with another node B, A first queries B's reputation. If B has low reputation (i.e., a potential malicious node), A blacklists B and does not connect to it for a while. In another case, if some node is identified as malicious, this information is broadcasted to other nodes so that they may block it. In the following, we assume that most streaming nodes in the system are well-behaved, and that node behavior is consistent for a considerably long time.

We consider a set of trustworthy nodes in the system. A trustworthy node always behaves as desired and never modifies or forges data. The trustworthy nodes form a monitoring overlay to store and maintain monitoring messages from streaming nodes. Each trustworthy node holds a certificate issued by a trusted certification authority (CA). With the certificates, two trustworthy nodes can authenticate each other and set up a secure connection as in SSL (Secure Socket Layer) [9]. Similar to SSL, data transmission between two trustworthy nodes is secure and cannot be attacked by end systems. In fact, a trustworthy node can be a pre-deployed proxy or an authenticated end system. The selection of trustworthy nodes has been out of the scope of this paper. Interested readers can refer to [10].

Each streaming node S is associated with a monitoring node T_S. All the messages about the performance of S are forwarded to T_S. A streaming node can send two types of messages to the monitoring overlay:

- *UPDATE* message: A streaming node keeps checking the data it receives. It periodically sends an *UPDATE* message to report the performance of its parent. An *UPDATE* message contains the data sender's IP address, the data receiver's IP address, information about the data sender's performance (e.g., the amount of corrupt data detected) and a timestamp. Here we assume a streaming node can be uniquely represented by its IP address.
- *QUERY* message: A streaming node can send a *QUERY* message to query the reputation of any other streaming node.

On the other hand, a monitoring node A is responsible for a certain IP range $[A_l, A_r)$. A streaming node with IP address within this range will be associated with A. Clearly, the IP ranges of the monitoring nodes should not overlap. We further define $L(A)$ as the load on a monitoring node A, i.e., the number of IP addresses that have been associated with A.

B. Construction of the Monitoring Overlay

To efficiently process the messages from streaming nodes, we organize the monitoring nodes into a *threaded binary tree*. A threaded binary tree is a binary search tree in which each node maintains a $Pred$ link pointing to the node's in-order predecessor and a $Succ$ link pointing to its in-order successor [11]. Figure 1 shows an example of a threaded binary tree. Each quadrangle is a monitoring node, and the numbers in a quadrangle indicate the IP range maintained by the node. Here we use numerical values to represent IP addresses. Suppose that a node K is responsible for range $[K_l, K_r)$. The ranges maintained by nodes in the left subtree of K are all smaller than K_l, and the ranges maintained by nodes in the right subtree of K are larger than or equal to K_r. The directed dash lines in the figure indicate the in-order successors of nodes. Starting from the leftmost leaf node of the tree (the node responsible for $[0, 10)$ in the figure) and following the dash lines, all the nodes can be traversed in an ascending order according to their ranges. The $Pred$ links are not shown in the figure, since they can be tracked through the reverse of the $Succ$ links. In other words, if node A is node B's successor, B is A's predecessor.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

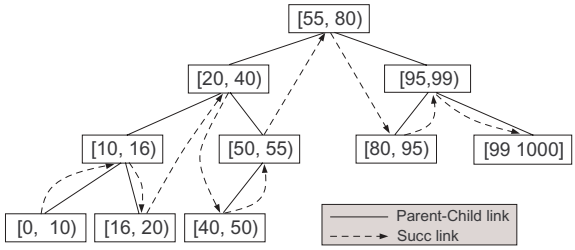


Fig. 1. An example of a threaded binary search tree.

Nodes in the tree periodically exchange $KeepAlive$ messages with their parents and children. With node joining or leaving, the tree needs to be accordingly updated. We do not discuss the operations upon node joining or leaving here. Interested readers can refer to [11].

C. Access and Maintenance of Reputation

A message is processed in the monitoring overlay as Fig. 2 shows. Suppose that streaming node B is streaming node A's child in the streaming overlay, and B prepares to submit an *UPDATE* message to report A's performance. If B has not sent an *UPDATE* message about A before, B first sends A's IP address to R, the root of the threaded tree. R then searches in the threaded tree to identify the monitoring node whose range covers A's IP address (node T in this case). T then sends a response message to B as well as its certificate of trustworthiness. After B confirms the trustworthiness of T, it sends its *UPDATE* message about A to T. In the following periods, B will directly send the *UPDATE* messages about A to T.

The IP ranges maintained by the monitoring nodes should be carefully computed so that the loads on the nodes are balanced. Initially, the root of the tree maintains the whole IP range while all the others maintain an empty range. A monitoring node A periodically compares its own load with the loads of its predecessor A_- and successor A_+ (if any). If the gap between $L(A)$ and $L(A_-)$ or the gap between $L(A)$ and $L(A_+)$ is larger than a certain threshold δ, the three nodes redistribute their loads and IP ranges to equally share the loads. If a monitoring node leaves the tree, its load is distributed to its predecessor and successor before it leaves. To prevent data loss due to unexpected node failure, each node sends a copy of its data to its predecessor and successor, and periodically updates them.

Figure 3 shows an example of load redistribution among nodes. Initially, all the IP addresses are maintained by the root R, and all the other nodes have empty ranges. With the insertion of new IPs, when R finds that its loads $L(R)$ is larger than its successor R_+'s load $L(R_+)$ (or its predecessor R_-'s load $L(R_-)$) by δ, R moves one third of its load to R_- and another one third to R_+. R further accordingly redistributes the ranges among the three nodes as Fig. 3(b) shows.

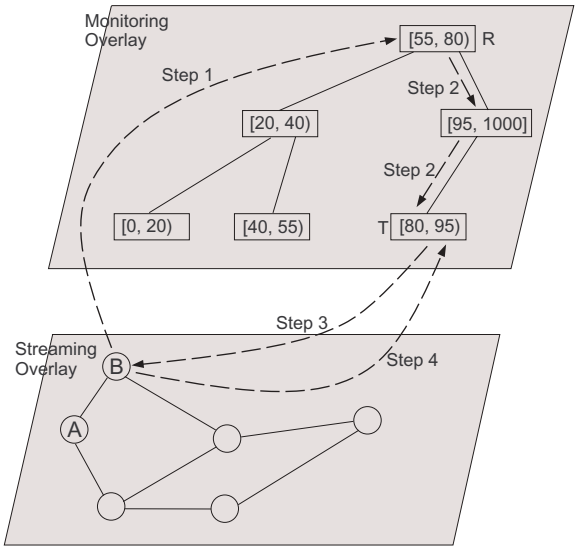
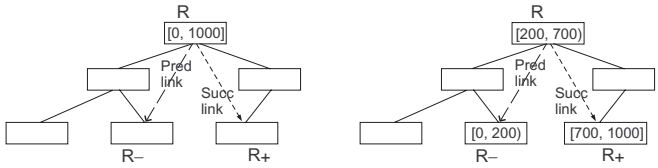


Fig. 2. The process of submitting an *UPDATE* message about a streaming node A by its child B for the first time.
Step 1) B sends A's IP address to R. Suppose that A's IP address is represented by a numerical value 88; Step 2) R searches in the threaded binary tree to identify the monitoring node that manages the value 88 (node T in this case); Step 3) T responses to B with its certificate; Step 4) After confirming the trustworthiness of T, B sends its *UPDATE* message about A to T.



(a) Initially, root R maintains the whole IP range;

(b) R redistributes its load and range to the predecessor R_- and the successor R_+.

Fig. 3. Load and range redistribution among monitoring nodes.

D. Reputation Computing

We now discuss how to compute the reputation of a streaming node A, $REP(A)$. For ease of illustration and brevity, we take one of the malicious actions, i.e., distributing corrupt data, as an example. That is, a node may send out corrupt data that do not conform to the stream format.

Although the monitoring overlay is secure, a dishonest streaming node may submit forged *UPDATE* messages to affect the reputation computing. To address this problem, we monitor *suspicious messages* as in [4]. When a data receiver submits an *UPDATE* message, the corresponding data sender also sends its own version of the message to the monitoring overlay. If there is an obvious gap between these two messages, the messages are regarded as suspicious.

Define $N_s(A)$ and $N_n(A)$ as the number of suspicious messages and non-suspicious messages generated by A. We further define $Credit(A) = \frac{N_n(A)}{N_s(A)+N_n(A)}$ as the credibility level of A, which indicates to what extend we should believe

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

the messages generated by A. Since dishonest nodes often have low credit values, their opinion on the evaluation of others should be accordingly reduced. We hence compute $REP(A)$ as

$$REP(A) = Average\left\{ Credit(X) \times \left(1 - \frac{Corrupt(A,X,t)}{Total(A,X,t)}\right), \forall\, t \in valid\ periods; \right\}$$

where $Corrupt(A,X,t)$ and $Total(A,X,t)$ are the amounts of corrupt data and total data received by X from A in period t as reported by X, respectively. The valid periods exclude outdated messages.

III. Illustrative Numerical Results

In this section we present simulation results on our detection scheme. We randomly put a group of $(1000-9000)$ streaming nodes into the network. Nodes form a streaming overlay as follows: Each node randomly selects multiple nodes as its parents, and a node can have at most 10 children. A node may also dynamically change some of its parents. The average duration of a streaming connection is 25 updating periods.

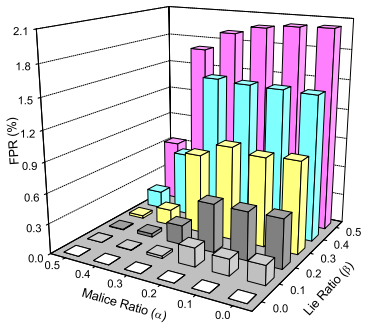
We simulate the example of distributing corrupt data. A non-malicious streaming node distributes a negligible amount of corrupt data, while a malicious streaming node distributes corrupt data with a probability uniformly distributed between $[0.4, 1]$. We define *malice ratio (α)* as the number of malicious streaming nodes divided by the total number of streaming nodes. Furthermore, a node may lie in its *UPDATE* messages. We define three types of lies in our simulation: (I) The node monitored is not malicious, but its dishonest neighbor reports its action as malicious; (II) The node monitored is malicious, but its dishonest neighbor reports its action as non-malicious (i.e., conspiracy); (III) A dishonest and malicious node reports its own action as non-malicious. In the system, a streaming node is either honest or dishonest. An honest node never lies in its messages. For dishonest nodes, the type-I and type-III lies occur with a probability uniformly distributed in $[0.4, 1]$, and the type-II lies occur with probability 0.05. We define *lie ratio (β)* as the number of dishonest streaming nodes divided by the total number of streaming nodes. Note that the behavior of malice and lying are independent.

A node is evaluated as malicious if its reputation is smaller than the average reputation value of all the non-leaf nodes and a given threshold 0.85.

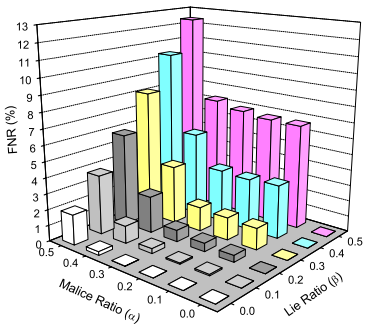
We further define the following metrics for evaluation:

- *False positive rate (FPR)*: defined as the number of non-malicious nodes evaluated as malicious divided by the total number of non-malicious nodes.
- *False negative rate (FNR)*: defined as the number of malicious nodes evaluated as non-malicious divided by the total number of malicious nodes.

Figure 4 shows the FPR and FNR values after 30 updating periods with different lie ratios and malice ratios. The larger



(a) FPR;



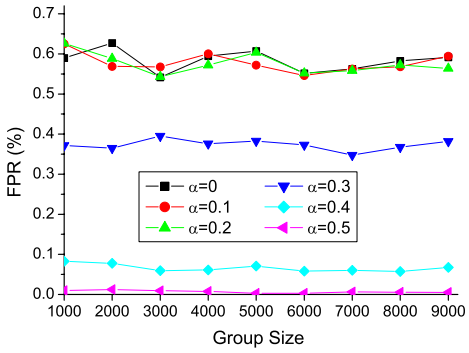
(b) FNR.

Fig. 4. Performance with different malice ratios and lie ratios (after 30 updating periods, with group size 5000).

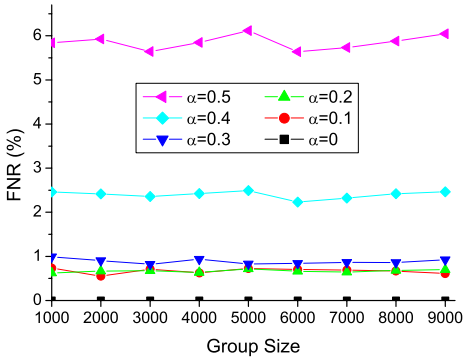
the lie ratio, the larger FPR and FNR. In Fig. 4(a) FPR is kept below 2.1%, even with 50% dishonest nodes. In the best case of $\beta = 0$, FPR is always 0. We note that the scheme achieves large FPR when malice ratio α is small. When $\alpha = 0$, the FPR value is almost the largest. This is because our judgment is based on the average reputation value. With small malice ratios, a large portion of nodes are non-malicious and have similar reputation values, which are also close to the average. In this case, a small perturbation to the evaluation may lead to incorrect judgment. While in the case of large malice ratios, the gap between the average reputation and the reputation of a non-malicious node is large, therefore non-malicious nodes are unlikely to be evaluated as malicious. As compared to the FPR values, we achieve much larger FNR values as shown in Fig. 4(b). This is partially due to the small evaluation threshold we set, because it is more important to protect non-malicious nodes than to detect malicious ones.

Figure 5 shows the FPR and FNR values with different group sizes. The FPR and FNR values are almost independent of the group size. This is because a node's reputation is determined by its neighbors, and the selection of neighbors in our simulation does not depend on the group size. In Fig. 5(a) the FPR value is large when the malice ratio is small. The

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.



(a) FPR;



(b) FNR.

Fig. 5. Performance with different group sizes (after 30 updating periods, $\beta = 0.25$).

reason has been explained before. Similar to Fig. 4, the scheme achieves much larger FNR than FPR as shown in Fig. 5(b) .

Figure 6 shows the management loads on the monitoring nodes. We set the number of streaming nodes and the number of monitoring nodes to 9000 and 50, respectively. The IP addresses of streaming nodes are uniformly distributed within a given range. The redistribution threshold δ is 8. Note that the monitoring nodes have been sorted in an ascending order according to their ranges by following the $Succ$ links in the threaded binary tree. They are further assigned node IDs in that order. As shown in the figure, the maximum load is only 26.5% larger than the minimum load. The loads are well balanced among the monitoring nodes. Note that the loads on nodes first increase and then decrease with the increase of node IDs. This is because the load redistribution is propagated along the tree in a top-down manner. The root and its close neighbors (with IDs from 20 to 35) hence have the heaviest loads. After the ranges have been redistributed among all the nodes, we believe the load distribution is more related to the IP distribution instead of the distance to the root.

IV. Conclusion

Most proposed P2P streaming systems assume that nodes are cooperative to cache and relay data. However, this may not be true in the open Internet. In this paper, we study how to detect malicious nodes in a P2P streaming system. We require each node to keep monitoring its neighbors and to periodically generate monitoring messages. The messages are collected and analyzed at some trustworthy nodes in a monitoring overlay. We study several key components in this framework, including efficient structure of the monitoring overlay, load balancing among the trustworthy nodes and reputation computing in the presence of node lying in the monitoring messages. Our simulation results show that this scheme can efficiently detect malicious nodes with low error, and that our load redistribution method can achieve good load balancing among the trustworthy nodes.

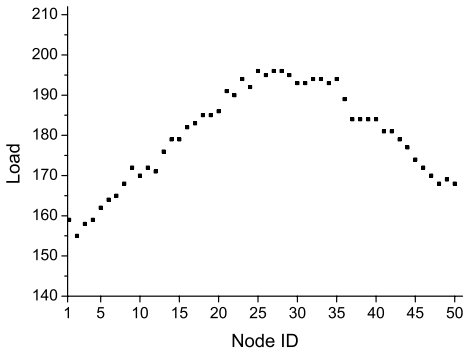


Fig. 6. Load distribution on monitoring nodes.

Acknowledgement

The authors would like to thank Yongqiang Xiong from MSRA, Qian Zhang, Yajun Wang and Zhen Zhou from HKUST for their helpful discussions.

References

[1] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming," in *Proc. IEEE INFOCOM'05*, March 2005.

[2] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Inter-overlay optimization based p2p live streaming system," in *Proc. IEEE INFOCOM'06*, April 2006.

[3] X. Jin, S.-H. G. Chan, W.-P. K. Yiu, Y. Xiong, and Q. Zhang, "Detecting malicious hosts in the presence of lying hosts in peer-to-peer streaming," in *Proc. IEEE ICME'06*, July 2006.

[4] L. Mekouar, Y. Iraqi, and R. Boutaba, "Detecting malicious peers in a reputation-based peer-to-peer system," in *Proc. IEEE CCNC'05*, 2005.

[5] S. Jun, M. Ahamad, and J. Xu, "Robust information dissemination in uncooperative environments," in *Proc. IEEE ICDCS'05*, 2005.

[6] S. Lee, R. Sherwood, and B. Bhattacharjee, "Cooperative peer groups in NICE," in *Proc. IEEE INFOCOM'03*, 2003.

[7] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *Proc. ACM CIKM'01*, 2001.

[8] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigen-Trust algorithm for reputation management in P2P networks," in *Proc. WWW'03*, 2003.

[9] "Introduction to SSL." http://docs.sun.com/source/816-6156-10/contents.htm.

[10] R. Chen and B. Yeager, "Poblano: A distributed trust model for peer-to-peer networks." http://www.jxta.org/docs/trust.pdf.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2001.

1-4244-0357-X/06/$20.00 ©2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.