Proc. of the Intl. Conf. on Advances In Computing, Communication and Information Technology - CCIT 2014.

ISBN: 978-1-63248-010-1 doi: 10.15224/ 978-1-63248-010-1-34

An Algorithm to Detect Inconsistencies in Access Control Policies

Muhammad Aqib, Riaz Ahmed Shaikh

***Abstract*—Inconsistency in access control policies exists when two or more than two rules defined in the policy set lead to the contradictory decisions. It makes it difficult for the system to decide which rule is applicable to the current scenario and hence make the system vulnerable to the unauthorized use. Different inconsistency detection methods have been proposed by researchers. However, those suffer from various limitations. In this article, we propose an algorithm that detects the inconsistencies in the policies using decision trees and returns the inconsistent rules with contradictory attribute values.**

***Keywords*—access control, inconsistency detection, policy validation**

I. Introduction

Security of the software applications is a critical issue. For this purpose, different mechanisms are used to restrict the users of enterprise applications from the unauthorized use. One of such mechanisms is access control policies. The rules in these policies could be defined using different languages such as XACML [1]. In order to ensure that the application resources are secure and out of the reach of unauthorized users, these policies should be defined in such a way that there should be no error, ambiguity and overlapping of rules.

Defining policies in an error free manner is not a trivial task, especially for big organizations that contains large number of users and resources. The most common problem that they deal is the presence of inconsistent rules. Detecting inconsistent rules in large set of complex policies is a challenging task. Many researchers have proposed solutions [2][3][4][6][7] but they suffer from various limitations. For example, inefficient handling of Boolean expressions, and dealing of only discrete attributes etc.

In this paper, we are going to present an algorithm-based approach to detect the inconsistencies which is capable of handling both continuous and discrete attribute values. Furthermore, it is not only applicable to dynamic data but it also handles Boolean expressions that include contextual attributes such as time and date.

The rest of the paper is organized as follows. Section II defines the inconsistency. Section III gives a brief description of the proposed algorithm. Section IV contains qualitative comparison with existing validation methods. Finally, Section V concludes the paper.

II. Concepts and Definitions

Inconsistency in the policy set exists when any two rules in that policy set lead to the contradictory outcomes. For example, if a rule defined in the policy set allows a user to access some resources during a specific time span but there exists some other rule in the same policy set which deny the user to access the same resource during some other time slots. However, if these time slots are same or they overlap, then we say that these two rules lead towards the contradictory statements and therefore they are not consistent. Hence, the policy set is said to be inconsistent. The rules defined by the administrators consist of different attribute values and the values of these attributes lead them to some decision.

Let $S = \{s_{1,}s_2\, s_{3,} \dots., s_n\}$ is the set of subjects, $O = \{o_1, o_2, o_3, \dots, o_m\}$ is the set of objects, $C = \{c_1, c_2, c_3, \dots, c_l\}$ is the set of contexts, and $A = \{a_1, a_2, a_3, \dots, a_k\}$ is the set of actions. Let $D = \{permit, deny, undefined\}$ be the set of decision attributes. In access control policy, a rule can be defined in four tuple form: $(s, o, a, c) \rightarrow d$. Let R is the set of rules. Two rules $r_i\ and\ r_j\ \in R$ such that $i \neq j$ are said to be inconsistent if $s_i = s_j$, $o_i = o_j$, $a_i = a_j$, $c_i = c_j$, and have contradictory decisions i.e. $r_i \rightarrow\ d_x$ and $r_j \rightarrow\ d_y, x \neq y$.

III. Inconsistency Detection Algorithm

In this section, we present a new inconsistency detection algorithm for access control policies. It works in two phases. In the first phase, it takes a decision tree as an input and divides it into sub-trees based upon the number of decision attribute values. In the second phase, algorithm takes sub-trees as an input and compares them recursively to detect inconsistencies.

A. *Decision Tree Hierarchy*

In the tree, the root node is at the first level of the decision tree whereas the decision attribute (d) is on the top of the attributes hierarchy that is the child of the root node and exists on the second level as shown in the Fig. 1. These nodes include the action attributes in their child attribute list so the action attributes are on the third level in the tree hierarchy. Object attributes are the direct children of the action attributes and exist in the children attribute list of the action attributes. So they are on the fourth level in this hierarchy. In this tree hierarchy, the subject nodes are on the fifth level and they exist in the children attribute list of the objects which are the parents of subject attribute nodes. Subject attribute nodes in turn contain the contextual attributes in their children attribute lists and exists on the sixth level of this hierarchy and they are

Muhammad Aqib, Riaz Ahmed Shaikh
Computer Science Department,
Faculty of Computing and Information Technology,
King Abdulaziz University,
Jeddah, Saudi Arabia.
aqib.qazi@yahoo.com, rashaikh@kau.edu.sa

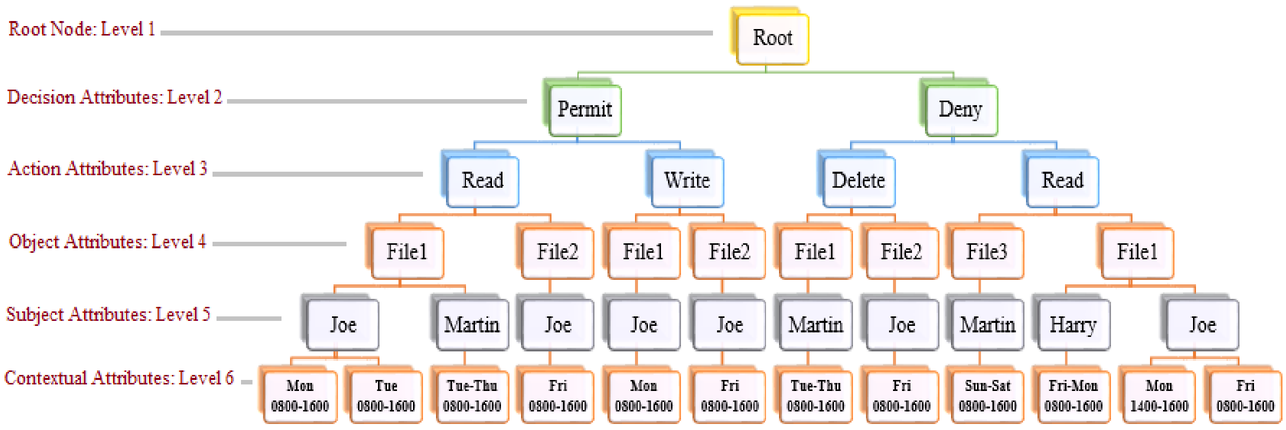
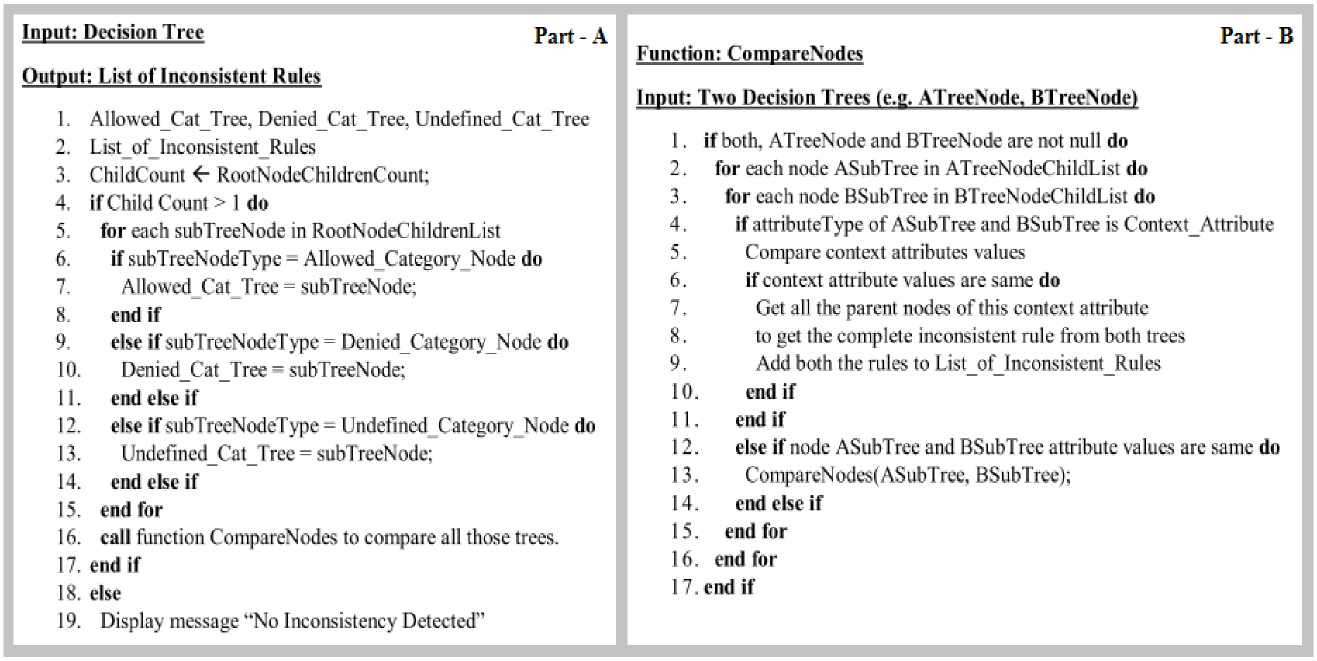


Figure 1. Sample hierarchy of the decision tree



```
Input: Decision Tree                                              Part - A
Output: List of Inconsistent Rules

1.  Allowed_Cat_Tree, Denied_Cat_Tree, Undefined_Cat_Tree
2.  List_of_Inconsistent_Rules
3.  ChildCount ← RootNodeChildrenCount;
4.  if Child Count > 1 do
5.    for each subTreeNode in RootNodeChildrenList
6.      if subTreeNodeType = Allowed_Category_Node do
7.        Allowed_Cat_Tree = subTreeNode;
8.      end if
9.      else if subTreeNodeType = Denied_Category_Node do
10.       Denied_Cat_Tree = subTreeNode;
11.     end else if
12.     else if subTreeNodeType = Undefined_Category_Node do
13.       Undefined_Cat_Tree = subTreeNode;
14.     end else if
15.   end for
16.   call function CompareNodes to compare all those trees.
17. end if
18. else
19.   Display message "No Inconsistency Detected"
```

```
                                                                  Part - B
Function: CompareNodes
Input: Two Decision Trees (e.g. ATreeNode, BTreeNode)

1.  if both, ATreeNode and BTreeNode are not null do
2.    for each node ASubTree in ATreeNodeChildList do
3.      for each node BSubTree in BTreeNodeChildList do
4.        if attributeType of ASubTree and BSubTree is Context_Attribute
5.          Compare context attributes values
6.          if context attribute values are same do
7.            Get all the parent nodes of this context attribute
8.            to get the complete inconsistent rule from both trees
9.            Add both the rules to List_of_Inconsistent_Rules
10.         end if
11.       end if
12.       else if node ASubTree and BSubTree attribute values are same do
13.         CompareNodes(ASubTree, BSubTree);
14.       end else if
15.     end for
16.   end for
17. end if
```

Figure 2. Proposed algorithm to detect inconsistencies in access control policies

also the leaf nodes of the policy tree. It then starts the validation process and returns the inconsistent rules in case inconsistencies found in the policies.

B. *Inconsistencies Detection Process*

As discussed above, the proposed algorithm consists of two parts that are clearly shown in Fig. 2. Description of both the parts is given below.

Step 1: In this step, the main tree will be divided into the sub-trees equal to the number of decision attributes. For this purpose it will count the number of decision attribute nodes that are the children of the root node (Part A, Line: 3). If there is only one decision attribute node in the children node list of the root node (Part A, Line: 4), then the algorithm will stop and it will display no inconsistency found message (Part A, Lines: 18, 19). In another case, the main tree is divided into the sub-trees equal to the number of decision attributes in the children attributes list of the root node (Part A, Lines: 5-15). Suppose there are two decision attributes, permit and deny as shown in the Fig. 1. In this case the main tree is divided into the two sub-trees as shown in the Fig. 3. All the policies with category attribute value "permit" are presented in the first tree. Similarly, all the other rules are presented in the second tree with category attribute value "deny" as the root node.

Proc. of the Intl. Conf. on Advances In Computing, Communication and Information Technology - CCIT 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-010-1 doi: 10.15224/ 978-1-63248-010-1-34

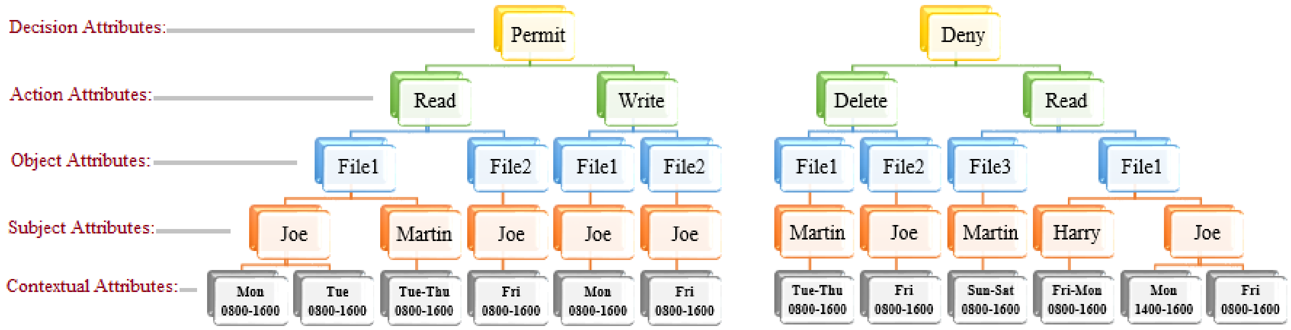


Figure 3. Sub-trees generated with decision attribute as the root node.

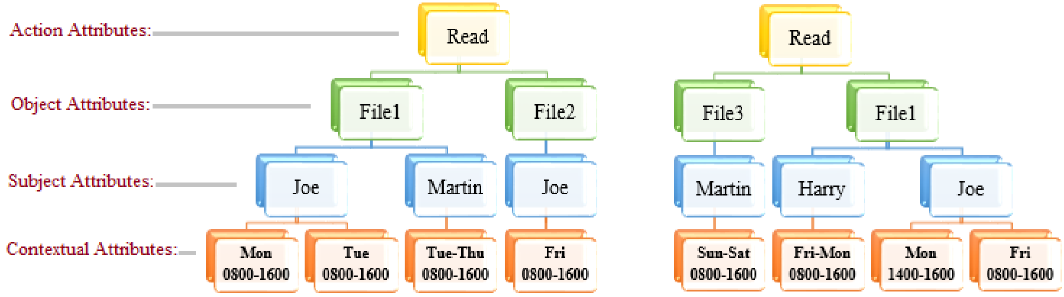


Figure 4. Sub-trees generated with action attribute as the root node.

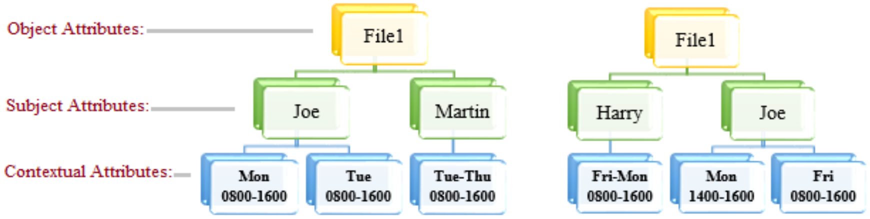


Figure 5. Sub-trees generates with object node as the root node.

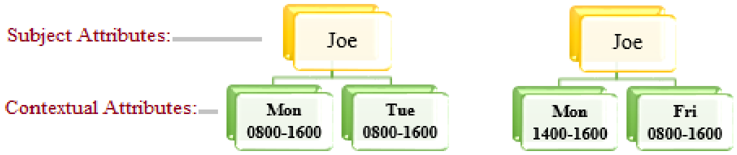


Figure 6. Sub-trees generated with subject node as the root node.

Step 2: After having separate trees for each decision node as shown in the Fig. 3, our algorithm will start comparing two sub-trees using the *CompareNodes* function (Part A, Line: 16). It will compare only if both of the trees are not null (Part B, Line: 2, 3). If the child node type in both trees is action and the node values are also same, then it will pick those nodes and will call the *CompareNodes* function again (Part B, Lines: 12-14). In Fig. 3, the child node of decision attribute node is action node and its value “Read” is same in both sub-trees. Now the action node will become the root node of both the trees passed to the *CompareNodes* function as shown in Fig. 4.

Both the trees shown in Fig. 4 are not null (Part B, Line: 1), it will get the child nodes of the root node (action node is root node here) and the object attribute nodes are the child nodes at this step (Part B, Lines: 2, 3). Now it will compare the values of object attributes and will call the *CompareNodes* function again if they have the same values in both trees (Part B, Lines: 12-14). As shown in the Fig. 4, object nodes having “File1” are same in both the trees so now sub-trees will be having them as root nodes. The Fig. 5 shows the resulting trees passed to the *CompareNodes* function in result of this comparison.

Proc. of the Intl. Conf. on Advances In Computing, Communication and Information Technology - CCIT 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-010-1 doi: 10.15224/ 978-1-63248-010-1-34

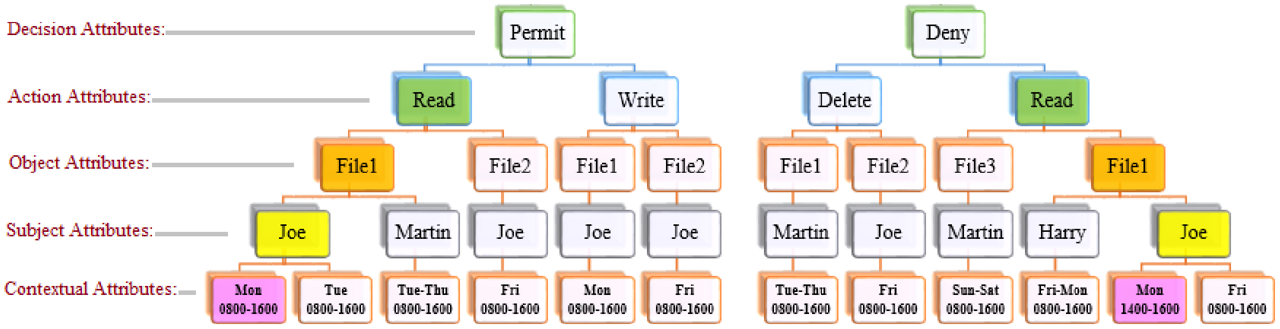


Figure 7. Rules with contradictory decisions identified.

TABLE 1. QUALITATIVE COMPARISON OF EXISTING AND PROPOSED ACCESS CONTROL POLICY VALIDATION TECHNIQUES

Approach		Inconsistency Detection	Boolean Expression	Continuous Data Handling	Dynamic Data Handling	Contextual Attributes
Our Proposed Method	Decision Tree based Algorithm	Yes	Yes	Yes	Yes	Yes
V.R. Karimi & D.D. Cowan [2]	Model checking Alloy	Yes	No	No	No	No
Ma *et al.* [3]	Model Checking SPIN	Yes	No	No	No	No
Bravo *et al.* [4]	DTD graph, algorithms	Yes	No	Yes	No	No
Shaikh *et al.* [5]	Data classification	Yes	Yes	Yes	Yes	Yes
Fisler *et al.* [6]	Decision diagrams MTBDD	Yes	Yes	No	No	No
Bauer *et al.* [7]	Association rule mining approach	Yes	Yes	No	Yes	No

The *CompareNodes* function will compare the trees shown in Fig. 5 where object attribute node is the root node. It is clear that the child node type is subject node and “Joe” is the same attribute value in both the trees. So *CompareNodes* function will be called again and this time the subject attribute node will be the root node in both the sub-trees passed as parameters. The Fig. 6 shows the resulting sub-trees with subject attribute nodes as the root nodes.

These trees will be passed to *the CompareNodes* function and they have contextual attributes as their child nodes. So this time the *CompareNodes* function will not be called again and contextual attributes will be compared in step 3 of the algorithm.

Step 3: As mentioned above, if the child node type in both the trees is context node, the *CompareNodes* function will not be called because these are the leaf nodes of the decision tree. It also indicates that all the other attributes are same. Now, it will start comparing the contextual attribute values (Part B, Lines: 4, 5). If the contextual attributes have the same values, it means both these rules are same. In Fig. 6, we can see that there is a contradiction in time attribute. The user is permitted to access the resource on Monday from 0800 to 1600 but on the same day, he cannot access the resource from 1400 to 1600. So it will get all the parent nodes of those contextual attributes to get those rules (Part B, Lines: 6-8) as shown in Fig. 7. Here all attribute-values of both the rules are same, it means they are inconsistent and hence they will be stored in the list of inconsistent rules (Part B, Line: 9). The Same process will be repeated until all the sub-trees generated during step 1 are compared with each other.

IV. Qualitative Comparison

In Table 1, we have compared our method with existing different inconsistency detection methods [2-7]. We have compared all these methods on the basis of the parameters defined in this table. These include inconsistency detection, inconsistency resolution and use of Boolean expressions. In addition we also have tested whether the proposed methods support the continuous data values or it is limited to the discrete values only. Similarly, the handling of dynamic data in addition to the static data and the use of contextual attributes has also been compared in qualitative comparison of these proposed methods.

V. Conclusion

In this article, we have proposed an algorithm to detect inconsistencies in the access control policies. It provides a solution to validate the access control policies especially those

ISBN: 978-1-63248-010-1 doi: 10.15224/ 978-1-63248-010-1-34

which involve contextual attributes and expressions. By supporting Boolean expressions, continuous attribute values and contextual attribute values, our proposed algorithm reduces the number of rules. However, this approach also has some limitations. For example, this algorithm supports bounded continuous attribute values and does not provide any solution for detection and resolution of incompleteness problem. So in the future, we are planning to address these issues and also to improve the performance in terms of computational complexity.

References

[1] E. Rissanen, “eXtensible Access Control Markup Language (XACML) Version 3.0 OASIS Standard.” http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf, Jan 2013. Accessed: 2014-02-03.

[2] V. R. Karimi, and D. D. Cowan, "Verification of Access Control Policies for REA Business Processes", *33rd Annual IEEE Int. Computer Software and Application Conf.*, 2009, p 422-427.

[3] J. Ma, D. Zhang, G. Xu, and Y. Yang, "Model Checking Based Security Policy Verification and Validation", in Proc. of the 2nd International Workshop on Intelligent Systems and Applications (ISA) , IEEE, 2010.

[4] L. Bravo, J. Cheney, and I. Fundulaki, "ACCOn: Checking Consistency of XML Write-Access Control Policies", In proc. of the *11th Int. Conf. on Extending Database Technology: Advances in Database Technology*, EDBT, 2008, pp. 715-719.

[5] R. A. Shaikh, K. Adi, L. Logrippo, S. Mankovski, "Inconsistency Detection Method for Access Control Policies", in Proc. of *Sixth Int. Conf. on Information Assurance and Security*, 2010, pp. 204-209.

[6] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, “Verification and change-impact analysis of access-control policies,” in Proc. of the *27th Int. Conf. on Software engineering*, NY, USA, 2005, pp. 196–205.

[7] L. Bauer, S. Garriss, and M. K. Reiter, “Detecting and Resolving Policy Misconfigurations in Access-Control Systems”, *SACMAT*, June 2008, USA.

About Authors:

Muhammad Aqib is a student in King Abdulaziz University, Jeddah, Saudi Arabia. He is doing his master and is attached to the Department of Computer Science in Faculty of Computing and Information Technology. His research interest includes privacy, security and database management.

Riaz Ahmed Shaikh is an Assistant Professor at Computer Science Department in the King Abdulaziz University, Jeddah, Saudi Arabia. He obtained his Ph.D. from Computer Engineering Dept., of Kyung Hee University, Korea, 2009, M.S. in Information Technology from the National University of Sciences and Technology, Pakistan, 2005, and B.Sc. in Computer Engineering from Sir Syed University of Engineering & Technology, Pakistan, Feb. 2003. His research interest includes privacy, security, trust management, wireless sensor networks, and vehicular networks. He is a reviewer/ editorial board member of various international journals, e.g., IEEE Transaction on Parallel and Distributed Systems, IEEE Computer Journal, Elsevier International Journal of Systems, Control and Communications, Elsevier Mathematical and Computer Modeling, Journal of Communications and Networks, Oxford Computer Journal, Transactions on Emerging Telecommunications Technologies, International Journal of Internet and Distributed Systems, and many more. He is a member of the ICST and the ACM. For more information please visit http://sites.google.com/site/riaz289/ .