PGRIP: PNNI Global Routing Infrastructure Protection*

Sabrina De Capitani di Vimercati
Dipartimento di Scienze dell'Informazione
Università di Milano
20135 Milano, Italy
decapita@dsi.unimi.it

Patrick Lincoln Livio Ricciulli Pierangela Samarati†
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
{lincoln,livio,samarati}@csl.sri.com

Abstract

We describe a system for achieving PNNI (Private Network-Network Interface) Global Routing Infrastructure Protection (PGRIP). We give details of PGRIP's system-level design and identify some conditions to rigorously guarantee the distributed fault tolerant detection of anomalies. PGRIP detects integrity compromises of PNNI routing by enforcing rules that characterize topology information updates that are anomalous (or uncommon) with respect to the network status, past events occurred, or statistical measures. Rules are expressed in a flexible and expressive, yet simple, language using a tree structure to organize and reference topology information maintained at each node. We introduce a powerful notation to identify data objects contained in the PNNI topology database and statistical operators to examine the history of topology database updates accumulated during PNNI operation. Using the given notation, we give heuristical rules to illustrate how some anomalous database operations can be detected.

1 Introduction

Global routing networks require novel security mechanisms to protect control information spanning multiple untrusted administrative domains. Proper protection of the global data structures necessary for the operation of large distributed networks is necessary because (1) compromises of the topology information can have catastrophic effects for the operation of the network, thus being a good target for denial-of-service attacks and (2) user-level security mechanisms can be made less effective by compromising the underlying data transport layers (for example, by preempting the updates of the cryptographic keys, one could force a user to use less secure, expired keys).

Several designs based on cryptography have been proposed to secure routing infrastructures (see, for example, [4, 14, 17, 18, 20]). These designs typically rely on a key management infrastructure that must be as large as the routing network itself and that is resilient to faults. The key management problem has not been completely solved even for relatively small networks. Therefore, performing secure and fault tolerant key management for global networks can be considered a substantial obstacle. Another shortcoming of cryptographically secure routing protection designs is that they focus on protection against malicious faults, and do not directly address spontaneous accidental failures. In fact, most designs have no direct handling of nonmalicious faults, and leave these simpler fault tolerance issues to other network protocols. These limitations result in possible conflicts between the built-in mechanisms that reconfigure the system upon failure and the security mechanisms that overlay the routing protocols. For these reasons, other methods of achieving global network integrity are required. Our design overcomes the abovementioned limitations by protecting the routing infrastructure against both malicious and nonmalicious faults in a unified manner by replicating network resources and using Byzantine fault tolerant protocols to identify failures.

Another limitation of cryptographic protection come from the fact that it is preventive: the router nodes (nodes or routers, for short) protect the integrity of the system by preventing unauthorized modification of the state through encryption. This approach, if correctly implemented, offers very strong guarantees that no integrity is lost because of malicious activities. This guarantee, however, may come at a very high price in performance for having to continuously encrypt and decrypt a large amount of redundant topology information.[1] Our

*This work has been supported by Sprint Contract No. CK5005116JMD and National Science Foundation under grant ECS-94-22688.

†On leave from Università di Milano.

[1]In most cases, to protect against reply attacks, identical information needs to be made unique through the use of time-stamps.

approach is reactive: we leave the routing protocols in the clear, thus releasing the burden of encryption, and replicate enough resources to guarantee that, if an anomaly can be detected by a heuristical rule present in the system, it will be detected shortly after it happens. In our reactive approach, once an anomaly is detected and correctly diagnosed, predefined protocols (perhaps cryptographically secure) intervene in resolving the anomaly. The advantage of a reactive approach is that it treats the common case more efficiently (i.e., in the absence of anomalies the topology information propagates un encrypted), and introduces policing actions only in the rare cases when they are needed.[2]

By replicating both processing and communication resources according to rigorous and well-understood rules derived from fault tolerant distributed system theory [9], PGRIP is able to detect and, in some cases, tolerate large classes of fault or attack scenarios, either intentionally or unintentionally provoked, relying on cryptography only when absolutely necessary. Ideally, every router could be equipped with PGRIP so that all nodes could independently perform detection and resolution of anomalies. This requirement, however, may be too strong. We will show how detection, evaluation, and anomaly resolution can be correctly performed even by assuming the enforcement of our control (i.e., the implementation of PGRIP) in a relatively small subset of the nodes. Each PGRIP node evaluates changes to the local database and executes appropriate actions when anomalous operations on the database are observed. PGRIP's monitoring does not require interaction with the routing protocols but only monitors the resulting operations, and therefore could be thought of as residing in the network management layer.

PGRIP's design merges ideas from intrusion detection, network management, fault tolerance, and database security to result in a distributed fault tolerant system that maps extremely well to modern routing systems [5, 11], and does not require modifications to the routing protocol standards. In addition, because PGRIP does not directly interact with the underlying protocols, it could be coupled in an orthogonal way with other more conventional protection measures based on encryption, thus independently providing additional security. We detail our design and we show how it can be naturally integrated into the Private Network-Network Interface (PNNI) ATM routing infrastructure without requiring modifications to the standard and without requiring the maintenance of any additional topology information. Although PGRIP's system-level architecture in the context of PNNI is intimately tied to this particular routing standard, we believe that some of these core ideas could be reused to achieve integrity protection for other kinds of similar routing systems.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of the PNNI standard. Sections 3 and 4 describe the PGRIP system-level and node-level architectures. Sections 5 through 8 describe the different components of the PGRIP system with particular emphasis on the language and the knowledge representation. Section 9 discusses previous work. Finally, in Section 10 we give our concluding remarks.

[2] This argument assumes a routing system to be stable and therefore not to exhibit a large number of anomalies. We do not consider unstable routing systems because they would not be viable and would not be worth protecting.

2 PNNI overview

We outline the main concepts of the PNNI standard [5], focusing on the network topology information, maintained at each node, whose protection is the goal of our work.

2.1 Basic concepts

The Private Network-Network Interface (PNNI) is based on the link-state routing technique [5]. To reduce the amount of network connectivity information that each node must store and maintain and for efficient routing, the PNNI protocol uses a hierarchical organization. Nodes are organized into peer groups, each having a unique peer group identifier (PGID). Each group has a leader (PGL) that abstracts and represents the group at the next higher level of the hierarchy. This organization is recursive and at each higher level of the hierarchy, PGLs are organized into peer groups in which leaders are defined and abstracted again at a subsequent higher level. The leader representing a peer group at the next level of the hierarchy is elected by the nodes of the peer group in a priority-based election process (the node with the highest leadership priority in a peer group becomes the PGL). Figure 1 illustrates an example of PNNI hierarchy. The exemplified network is composed, at the lowest level, of eight nodes organized into three peer groups, $PG(A.1.1)$, $PG(A.1.2)$, and $PG(A.2)$, which are represented, at the subsequent level, by nodes $A.1.1$, $A.1.2$, and $A.2$, respectively. The protocol distinguishes different types of links connecting nodes: *horizontal links*, which connect nodes within the same peer group; *outside links*, which connect nodes (called border nodes) belonging to different peer groups; and *uplinks*, which are derived links that connect a border node to a node representing a peer group to which the node is con-

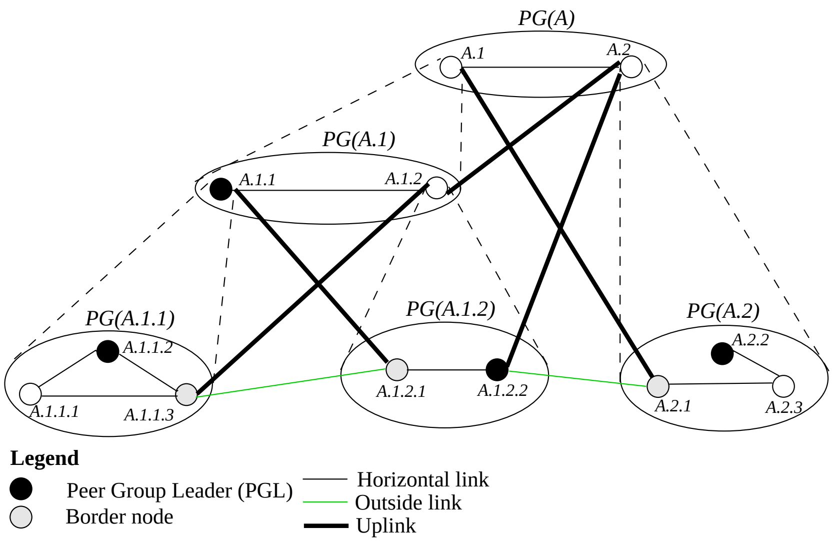

Figure 1: PNNI network example

nected. With respect to routing, user data transmission between end systems belonging to different peer groups is logically routed through higher-level logical nodes.

2.2 Network topology databases

To determine routing paths of packets to be transmitted, each node must maintain some information regarding the nodes in the network and their connectivity. The hierarchical organization of the network and the corresponding routing protocol do not require nodes to maintain information on every other node, but only on a subset of them. In particular, each node maintains a *topology database* containing information regarding the node itself, all the nodes belonging to its peer group, and all ancestor nodes. For instance, with reference to Figure 1, $A.1.1.1$ will maintain information on nodes $A.1.1.1$, $A.1.1.2$, $A.1.1.3$, $A.1.1$, $A.1.2$, $A.1$, and $A.2$. Information maintained *for each node* is organized into *information groups* (IGs) as follows.

- *Nodal Information* (NI) describes the state of a node. It includes address, priority information, and administrative flags regulating the participation of the nodes in routing and election protocols.
- *Nodal State Parameter* (NSP) provides information on properties of the node. It is used when the node represents its peer group at the next higher level of the hierarchy. It includes all metrics and attributes for the given input-output port pair.
- *Horizontal Link* (HL) contains information on the connections between nodes. For each link, it includes node ID and port ID of the nodes it connects, and all metrics and attributes associated with the link.
- *Uplink* (UPL) contains information on the uplinks. For each uplink, it includes node ID and port ID of the border node, and all metrics and attributes associated with both directions of the link.
- *Internal and External Reachable ATM Address* (IRA and ERA) contain information on the end systems directly reachable from the node.

Metrics and attributes in the information groups described above are also organized into (sub)information groups, called *Resource Availability Information Groups* (RAIGs). Examples of metrics are the administrative weights to be used in routing decisions and the data transmission delay. Examples of attributes are the maximum and available data transmission.

The information groups stored at a node reflect the node's view of the network. Modifications to the network structure are communicated by transmission of collections of IGs, called PNNI Topology State Elements (PTSEs), grouping together information groups of the same type. A node's topology database consists of a collection of all PTSEs received. If a node has all the PTSEs for all nodes in its peer group, it has the complete topology and can compute routes to any address in that peer group. PTSEs are transmitted to nodes by means of the *flooding* and *database synchronization* protocols. The flooding protocol is a reliable transmission protocol by which packets (PTSEs) are transmitted between nodes. The *database synchronization* protocol uses flooding to efficiently exchange topology information between directly connected nodes in the same peer group. In addition to manual administrative changes, information stored at each node can be modified by other PNNI protocols: the *hello* protocol, by which nodes establish and control connectivity, and the *PGL election* protocol, by which nodes of the same peer group establish their leader.

3 PGRIP system-level architecture

Necessary and sufficient conditions must be satisfied in the design of PGRIP's system-level architecture in order to provide correct identification and resolution of anomalies. We only require that a subset of the nodes, called *Peer Group Core Group* (PGCG), in each PNNI peer group be equipped with PGRIP. The PGCG nodes must include the PGL and must meet specific connectivity requirements.

We will now both motivate and characterize the conditions under which PGRIP must operate with regard to the peer groups' topology, the amount of replication of PGRIP's resources, and the location of PGRIP nodes within peer groups. All these conditions are assumed to be recursively applied at all levels of the hierarchy.

3.1 Peer Group Topology

Nodes belonging to the same peer group share common views of the state of the network (link states and reachability information) by periodically flooding the peer group with messages that synchronize their topology databases. The flooding protocol is such that a node receiving a packet that changes any information in its own topology database automatically relays the change to all the other nodes directly connected to it (minus the node from which the change was received or nodes residing in other peer groups). PGRIP's anomaly detection and diagnosis requirements map very naturally to this mechanism and can fully exploit this inherent feature. The only requirement for PGRIP's proper operation is to keep the peer group fully connected at all times, so that messages flooding a peer group reach every node in the group, even during multiple failures.

We assume Byzantine faults (faults can be arbitrarily bad), and therefore we must assume that faulty nodes may be able to drop, modify, or forge malicious packets that travel throughout a peer group. In particular, because of our Byzantine assumption, there is no guarantee that legitimate messages traversing a peer group can reach every node in the peer group. To provide this guarantee, and to prevent m faulty nodes from partitioning the peer group, we formulate the first condition under which PGRIP must operate.

Condition 1 *Given a maximum of m simultaneous faulty routers, peer groups must be $(m+1)$-connected, that is, any two routers in any peer group must have at least $m+1$ distinct paths, to communicate with each other, that do not traverse any other node twice.*

This condition, analogous to the one proposed by Perlman [14], guarantees that, even if the m faults block m routes, there exists at least one other route through which legitimate message transmissions can take place. Notice that the $(m+1)$-connected requirement only imposes a precise and rigorous amount of redundancy in the communication links, which can be easily introduced at the planning stage of the network topology. The network illustrated in Figure 2, where nodes are 2-connected, satisfies this condition for $m = 1$ at all levels of the hierarchy.

3.2 Byzantine Agreement

By judiciously picking a subset of the nodes (PGCG) to independently carry out fault detection tasks, one can tolerate a failure in any nodes in the system, including one or more of the nodes performing the diagnosis. The theoretical results arising from the formulation of Byzantine agreement algorithms (first appearing in [9] and later refined in [19]) can be applied to provide necessary and sufficient conditions for determining how many PGCG nodes must be employed in each peer group.

Condition 2 *In order to tolerate the correct diagnosis of m arbitrary (including Byzantine) faults, there must exist at least $3m+1$ routers, in the system, which independently perform the diagnosis.*

By employing at least $3m+1$ nodes of the peer group in performing the diagnostic functions, we ensure that the nodes will reach a consensus on the diagnosis[3] even if m or fewer PGCG nodes conspire in an attempt to confuse the diagnosis.

Securing the fault detection system through cryptographic means by using techniques like the ones in [15]

[3]Note that it is also possible to reach a consensus on the impossibility of a correct diagnosis.

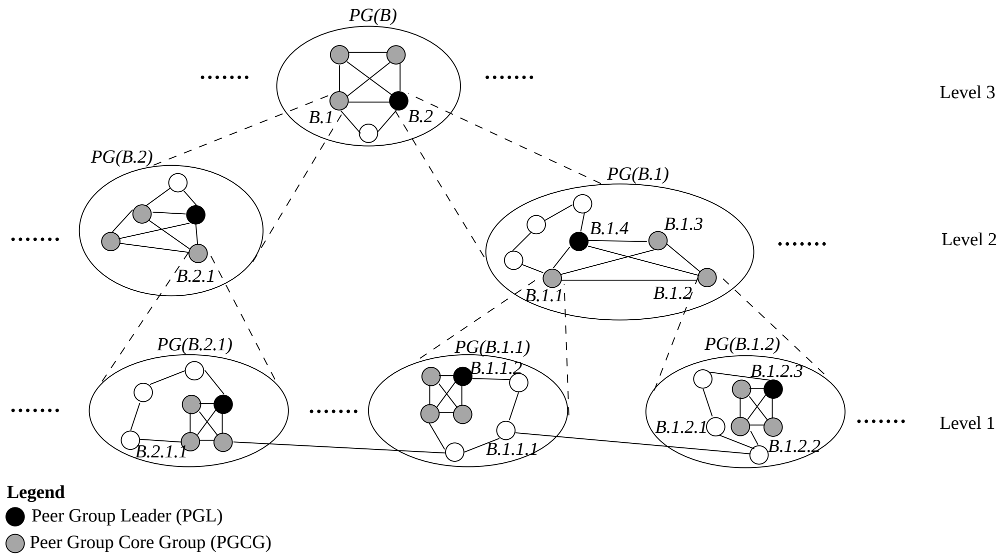

Figure 2: An example of PNNI network equipped with PGRIP

would require a key management hierarchy and non-portable cryptographic algorithms. Instead, link state routing protocols like PNNI provide quality of service guarantees that can be exploited in the fault detection process to achieving Byzantine agreement with standard algorithms without requiring cryptographic authentication. To achieve this, PGCG nodes must have additional redundancy in the connectivity among them. In fact, another well-known result from [9] translates to the following:

Condition 3 *If the* $3m+1$ *PGCG nodes are connected through point-to-point connections (as in a PNNI network), the topology of the* $3m+1$ *nodes must be at least 3m-connected.*

This condition prevents malicious PGCG nodes from affecting the consensus by intercepting and changing messages while the nodes perform the diagnosis agreement algorithm. The topology illustrated in Figure 2 is tolerant to one fault ($m = 1$) and satisfies both Condition 2 and Condition 3. At any level of the hierarchy, four completely connected PGCG nodes are responsible for reaching a consensus on fault diagnosis.

3.3 Partitioning Requirements

In Section 7.3 we discuss a situation in which the PGCG nodes must actively partition a peer group in order to prevent malicious nodes from becoming peer group leaders. In another instance, PGCG nodes (as discussed in Section 8.2) may agree to preempt a particular node that exhibits faulty behavior. In this case also, the PGCG routes must be able to block malicious messages so that they do not interfere with the preemption process.

Given these requirements, Condition 4 ensures that the nonfaulty PGCG nodes can block unauthorized messages and therefore can partition a peer group in such a way that no one partition contains more than one-third of the nodes in the group.

Condition 4 *All possible combinations of* $3m$ *PGCG nodes out of the* $3m+1$ *can partition the peer group's topology in subgraphs such that any subgraph contains less than one-third of the total nodes in the group.*

With reference to Figure 2, Condition 4 is trivially satisfied by having a large proportion of PGCG nodes.

4 PGRIP node-level architecture

PGRIP's anomaly detection and processing system is installed on the PGCG nodes. As shown in Figure 3, PGRIP's node-level architecture is composed of four modules:

- The *Anomaly Detection* module monitors all changes to the *Topology Database* to determine suspicious or anomalous modifications. Based on *anomaly detection rules*, which may be customized at each node, the Anomaly Detection module

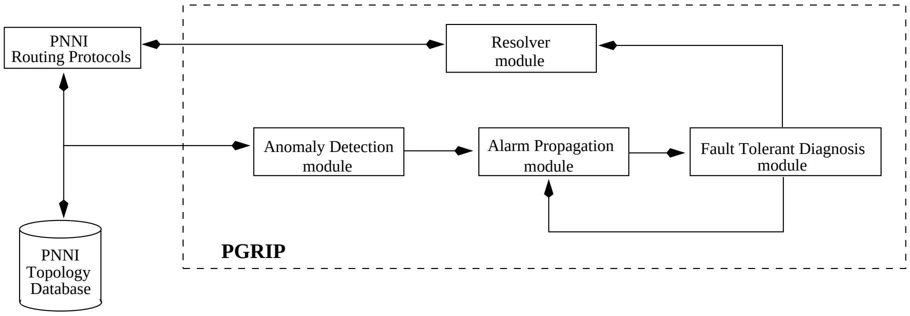

Figure 3: PGRIP architecture

can generate alarms that characterize anomalous changes to the database.

- The *Alarm Propagation* module receives alarms, generated by the anomaly detection rule evaluation, from either the local node or remote nodes. It can further propagate the alarm to other remote nodes or to its local *Fault Tolerant Diagnosis* module.

- The *Fault Tolerant Diagnosis* module uses interactive consistency agreement protocols of the kind proposed in [9, 19] to reach consensus on the actions to take. The most basic form of action is to simply log a diagnosis for further review by an operator. Other possible actions are to further propagate the alarm through the Alarm Propagation module or to feed the diagnosis to the *Resolver* module.

- The *Resolver* module logs results coming from the diagnosis module and in some extreme cases initiates a specialized protocol to remedy the diagnosed fault.

We are currently finalizing the detailed design and specification of the Anomaly Detection module and are planning to perform detailed analysis of its computational requirements. We are also planning the development of a prototype to be used in the field to log anomalous topology updates for review by an operator. We believe that this functionality alone could be very useful in a production environment to spot and resolve inconsistencies in the system configuration that could lead to integrity violations. The key technology for the prototyping and final realization of the Anomaly Detection module is available today [3] and could be relatively easily ported to PGRIP. The Alarm Propagation module should be relatively easily implemented, given its simplicity. An effective Fault Tolerant Diagnosis module to analyze the anomaly reports can be realized only with substantial additional research and development. By automatically correlating and summarizing relevant information in a distributed and fault tolerant manner, PGRIP will greatly amplify the effectiveness of the Anomaly Detection module. We plan to adopt technologies, such as the one described in [8], to provide a starting point for this module. The Resolver module will be developed as the last stage to automate the diagnostic system response mechanisms. This module requires the formulation of a set of specialized protocols (see Section 8) designed to remedy specific faults. The realization of the Resolver module, therefore, requires a limited degree of standardization. The main motivation for providing automatic response services through the Resolver module is to greatly reduce the response time for remedying integrity compromises.

In the remainder of this paper we describe each of the four main components of PGRIP's node architecture and show how their design can be integrated into the existing PNNI standard. In particular, we concentrate on the Anomaly Detection module that, for the moment, is our main focus.

5 Anomaly Detection module

The Anomaly Detection module monitors all changes to the topology database stored at a node to detect possible anomalous updates. The evaluation is based on rules characterizing modifications that may result as anomalous with respect to the current network status, events occurred, or statistical measures, and that could compromise the integrity of the topology information. In addition to supervising the consistency of the database state, this module exploits statistical knowledge accumulated during the operation of the nodes to detect database operations that are unexpected. An anomaly detection rule describes when a database modification is to be considered suspicious. The specification and evaluation of anomaly detection rules require a means of addressing topology information stored at a node, characterizing operations, and expressing anomalies. Their

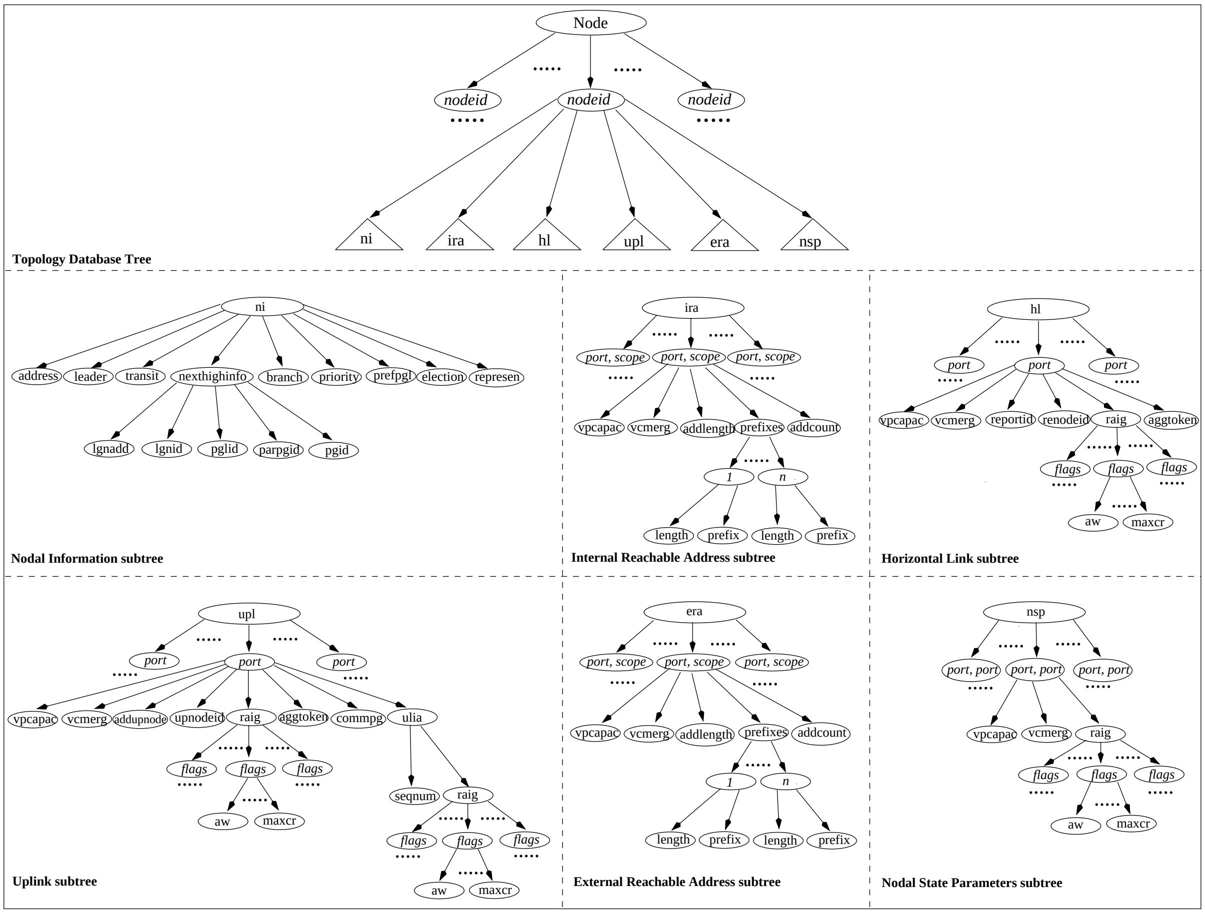

Figure 4: Graphical representation of the PNNI topology database

treatment is the focus of the remainder of this section.

5.1 Topology information representation

An important issue we faced in the design of the Anomaly Detection module was the choice of a representation for the topology information stored in a topology database. As illustrated in Section 2, topology information is received and stored by means of PTSEs collecting information groups, referred to a given node, of the same type. For instance, some PTSEs will contain Nodal State Parameter information groups, other PTSEs will contain Horizontal Link information groups, and so on. Information describing a node's local topology may therefore be dispersed in different PTSEs. We provide an abstraction of the topology database that collects IGs referring to the same node. Intuitively, each information group is a record. However, the structure is not as regular. Some fields are optional (e.g., the outgoing and/or incoming RAIG within the Internal Reachable ATM Address IG); other fields appear several times (e.g., in the Horizontal Link IG, the outgoing RAIG is repeated for each service category). Moreover, as PNNI is still under standardization, new information groups might be added. To provide for future extensibility, it is important that the chosen representation is simple enough, and yet flexible and powerful enough, to describe topology information without modifying the database structure. Thus, by borrowing some ideas from graph rewriting theory [3] and, more recently, semistructured data management [1, 2, 13], we abstract the topology information maintained at each node by viewing it as a tree, called the *topology database tree*.

A topology database tree has labeled vertices. The label of a vertex is either the *name* of a property in an information group or a base *value* (such as a node identifier, or a port identifier) of a property. The parent-child relationship of the tree reflects the structural organization of the information. Figure 4 illustrates the structure of the topology database tree. For readability, the subtrees corresponding to the different information groups are represented separately. Italic labels denote vertices that correspond to base values. For simplicity, leaf vertices containing the values of an IG's fields are not reported. The root of the tree, named "Node", has one child for each router for which information is stored, where each router is identified by the corresponding router identifier. For instance, with reference to Figure 1, the tree at node $A.1.1.1$ will have seven children, namely, $A.1.1.1$, $A.1.1.2$, $A.1.1.3$, $A.1.1$, $A.1.2$, $A.1$, and $A.2$. Each vertex corresponding to a router has six children, one for each information group referred to it. Each vertex corresponding to an information group, in turn, has children and descendants representing the topology information it contains. Vertices with base values are needed to univocally identify entities (such as nodes or links) to which information is referred. In particular, as is visible from Figure 4, Nodal State Parameter IGs are identified by the pair ⟨input port, output port⟩; Internal and External Reachable ATM Address IGs are identified by the pair ⟨port,scope[4]⟩; Horizontal Link IGs and Uplink IGs are identified by their ports; Resource Availability IGs are identified by the flags indicating the service categories to which the metrics and attributes apply.

Given the tree organization of the topology information, every piece of data maintained at a router can be referenced by means of the corresponding path expression. A *path expression* is a dot-separated sequence of labels of the form $\mathrm{Node.label_1.label_2 \ldots label_n}$ representing the path from the root to vertex $\mathrm{label_n}$. To avoid confusion between dots separating the different elements of a node identifier and dots separating different labels in a path expression, we report node's identifier between parenthesis. For instance, with reference to Figure 1, the path expression "Node.(A.1.1.2).ni.priority" denotes the field priority within the $A.1.1.2$'s Nodal Information IG.

By adopting ideas taken from graph rewriting theory [3], we allow path expressions to contain variables representing generic vertex labels, thus greatly increasing the expressive power of path expressions. In the following, we use uppercase single letters to denote variables. Variables provide a powerful mechanism for representing information in the topology database that matches to different values, possibly bounded by some conditions. For instance, the path expression "Node.X.hl.Y" denotes the set of all horizontal links of every node X in the topology database. The expression "Node.X.ni.priority.Z, X=A.1.2.1" denotes the priority Z of node $A.1.2.1$.

[4]The scope defines the highest level of the hierarchy at which the address will be visible.

5.2 Update operations and recording

Another important issue we addressed in the design of our system was the identification and characterization of updates to topology information. Consideration of modifications at the level of protocol (e.g., database synchronization) or at the level of whole PTSEs, however, does not seem to be sufficient to enforce meaningful controls. Although update operations are at such large granularity, we consider information at a finer, more semantically meaningful, granularity level. In particular, we consider topology information changes (additions, updates, or deletions) referred to specific information groups. With respect to the interface with PNNI operation, reception of a given PTSE will therefore be considered as the execution of a set of operations (one for each information group contained in the PTSE). The operation are `Add` if the information group is not presented, `Upd` (for update) if the information group already exists and therefore is replaced, and `Del` (for delete) if the information group is deleted. It is important to note that the update to an information group may actually change only some of the properties within it. For instance, only the weight associated with a link might have changed. In accordance with the PNNI approach, we do not consider operations as executed at such fine level of granularity (i.e., update of a property) but we consider the operations at the level of the whole information group. We illustrate in Section 5.4 how our language allows the evaluation the specific changes entailed within the update of an information group. In the remainder of this paper we often refer to the occurrence of an operation on an information group as an *event*. Events are characterized as `op`(*path_exp*) describing the execution of an operation `op` on an object *path_exp*.

To allow the evaluation of anomaly conditions based on previous operations executed, or on the previous status of the network, each node records all operations executed. Operations are recorded as triples of the form ⟨time, IG_before_image, IG_after_image⟩, denoting the snapshot of the information group on which the operation is executed before and after the operation and the time at which the operation occurred. The IG_before_image (IG_after_image resp.) is null in case of an insert (delete resp.) operation. To avoid the history log to grow indefinitely, pruning operations can be executed removing records that do not need to be considered further.

5.3 Basic operators of the language

Our language provides some basic operators that allow us to refer to summaries, aggregates, and statistical measures derived from the recorded history. Operators fall into three classes of measures.

- *Count* measures count the number of occurrences of events. They include the following operator.
  - `count`(*result,event,time_int*) returns the number *result* of events *event* executed over the time interval *time_int*.
- *Time* measures keep track of the time interval between two distinct, successive operations. They include the following operators.
  - `avgtime`(*result,path_exp,time_int*) returns the average time interval *result* between any two requests on object *path_exp* over the time interval *time_int*.
  - `timeint`(*result*,$event_1$,$event_2$,*time_int*) returns the time interval *result* between the last two events $event_1$ and $event_2$ occurring within time interval *time_int*.
- *Aggregation* measures combine a set of past operations into useful abstractions to be used to decide whether the effect of a given change operation appears consistent with historical observations.
  - `freq`(*result,event,time_int*) returns the *frequency* with which a given *event* has occurred over a given period of time *time_int* (e.g., number of updates originated by a node per unit time in the last hour).
  - `avgval`(*result,path_exp,time_int*) returns the *average* value *result* of the metric/attribute denoted by *path_exp* over the time interval *time_int* (e.g., the average value of the available cell rate on a horizontal link).
  - `max`(*result,path_exp,time_int*) returns the *maximum* value *result* assigned to the metric/attribute denoted by *path_exp* over the time interval *time_int* (e.g., the maximum value of the available cell rate on a horizontal link).
  - `min`(*result,path_exp,time_int*) returns the *minimum* value *result* assigned to the metric/attribute denoted by *path_exp* over the time interval *time_int* (e.g., the minimum value of the available cell rate on a horizontal link).

5.4 Anomaly detection rules

The syntax of the language to express rules is reported in Figure 5. Each rule is composed of the following four fields.

- *Operation:* a description of the operation on the topology database tree. This field is composed of an event and an optional path expressions. Path expressions refer to the database topology tree that would result after the operation is executed. These expressions, therefore, allow the anomaly detection rules to evaluate the effect of the operation on the database.
- *state:* a description of the state of the local PNNI database before the operation is executed. This field may contain statistical measures on past events and path expressions referring to the topology database tree before the execution of the operation.
- *condition:* a logical expression of conditions on the variables bounded in both the operation and the state fields. It always evaluates to either True or False.
- *alarm:* a unique identifier for a type of anomaly to be raised whenever the condition evaluates to True.

Intuitively, the rule semantics reads as follows. Upon the request to execute the *operation*, evaluate the expressions in *state* and, if there exists an instantiation of variables such that the *condition* evaluates to True, then raise the *alarm*. Figure 6 reports some examples of rules, where, for space reasons the alarm field is not specified. Anomalous situations controlled by those rules can be classified as follows.

Suspicious modifications Some database entries should be modified very seldom and their modification may be considered anomalous and require further investigation. For instance, the leadership priority of a node is expected to change rarely if not accompanied by a corresponding change of the leadership status (leader to non-leader or vice versa). This situation can be controlled by Rule 1. Another example of relatively static information is the weight associated with a link, which affects the PNNI routing decisions. Rule 2 raises an alarm whenever the weight associated with a link is modified.

Monitoring of the status of the network Other anomalies may correspond to improbable (sequence

```
⟨rule_definition⟩ ::= ⟨operation⟩ ⟨state⟩ ⟨condition⟩ ⟨alarm⟩
⟨operation⟩       ::= operation: ⟨event⟩ [,⟨path_exp_list⟩]
⟨event⟩           ::= Add(⟨path_exp⟩) | Upd(⟨path_exp⟩) | Del(⟨path_exp⟩)
⟨state⟩           ::= state: ⟨complex_state⟩
⟨complex_state⟩   ::= ⟨simple_state⟩ | ⟨simple_state⟩,⟨complex_state⟩
⟨simple_state⟩    ::= ⟨path_exp⟩ | count(⟨variable⟩,⟨event⟩,⟨time_int⟩) | avgtime(⟨variable⟩,⟨path_exp⟩,⟨time_int⟩) |
                      timeint(⟨variable⟩,⟨event⟩,⟨event⟩,⟨time_int⟩) | avgval(⟨variable⟩,⟨path_exp⟩,⟨time_int⟩) |
                      min(⟨variable⟩,⟨path_exp⟩,⟨time_int⟩) | max(⟨variable⟩,⟨path_exp⟩,⟨time_int⟩)|
                      freq(⟨variable⟩,⟨event⟩,⟨time_int⟩)
⟨condition⟩       ::= condition: ⟨complex_cond⟩
⟨complex_cond⟩    ::= ⟨simplex_cond⟩ | ⟨complex_cond⟩ ⟨bool_op⟩ ⟨complex_cond⟩
⟨simplex_cond⟩    ::= ⟨variable⟩ ⟨comp_op⟩ ⟨variable⟩ | ⟨variable⟩ ⟨eq_op⟩ ⟨variable⟩
⟨alarm⟩           ::= ⟨string⟩
⟨path_exp_list⟩   ::= ⟨path_exp⟩ | ⟨path_exp⟩,⟨path_exp_list⟩
⟨path_exp⟩        ::= ⟨label⟩ | ⟨label⟩.⟨path_exp⟩
⟨bool_op⟩         ::= ∧ | ∨
⟨comp_op⟩         ::= > | < | ≥ | ≤
⟨eq_op⟩           ::= = | ≠
⟨time_int⟩        ::= [⟨variable⟩|⟨number⟩,⟨variable⟩|⟨number⟩]
⟨variable⟩        ::= ⟨string⟩
⟨label⟩           ::= ⟨string⟩ | ⟨number⟩
```

Figure 5: Syntax of the language to express rules

of) updates. Anomalous situations can be, for instance, a sharp increase or decrease in the bandwidth associated with a link; a node or link that goes up and down frequently; or a short interval between two requests originated by the same node (note that PNNI requires a minimum time between two requests from the same node). Examples of rules controlling these anomalies are Rules 3 through 7. Rule 3 raises an alarm whenever the available cell rate of a RAIG differs considerably from the mean values of the rates observed until now. Rule 4 raises an alarm whenever the time between two update requests from the same node is smaller than a specified threshold. Rules 5 and 6 raise an alarm whenever there are two requests (`Add` and `Del`, or vice versa) for the same link within an interval of time smaller than a specified threshold. Rule 7 raises an alarm whenever there is disagreement about the identities of the nodes at each end of a link.

Consistency of the topology database Rules can also be used to verify the consistency of the information stored in a topology database. For instance, the maximum cell rate (maxCR) of a node must always be greater than or equal to the corresponding available cell rate AvCR; maxCR must also be smaller than or equal to the AvCR associated with the input/output ports of the link; maxCR associated with input and output ports with the same link should be equal. Rules 8, 9, and 10 can be used to control satisfaction of these conditions.

6 Alarm Propagation module

As a result of applying anomaly detection heuristics, nodes can generate alarms that are propagated throughout the PNNI routing infrastructure. The PNNI architecture is hierarchical. The nodes are arranged in groups that share common views of the state of the network (link states and reachability information) by constantly flooding the peer group with messages (called PNNI Topology State Packets-PTSPs) that synchronize the different nodes' databases. In addition, PGL nodes aggregate and summarize local information and make it available to higher-level peer groups, thus implementing the PNNI hierarchy.

PGRIP's alarm propagation requirements map very naturally to the PNNI data flow organization and can fully exploit it. After the distributed diagnosis phase executed among the PGCG nodes (see Section 7), the PGL[5] uses the alarm propagation module to take one or more of three actions: (1) log the diagnosis locally and take no further action, (2) employ specific countermeasures thorough the Resolver module, or (3) use higher-level binding information to flood the alarm in its higher-level PNNI group.

Action 1 should always be followed, modulo some filtering to avoid redundant log. Action 2 is followed when the diagnosis module reaches a conclusion and therefore recommends precise response actions (see Section 8).

[5] If the PGL is not preempted because it is believed to be non-faulty.

Rule 1	**operation:**	`Upd`(Node.X.ni), Node.X.ni.priority.P, Node.X.ni.leader.L
	state:	Node.X.ni.priority.O, Node.X.ni.leader.M
	condition:	$P \neq O \wedge L = M$
Rule 2	**operation:**	`Upd`(Node.X.hl.Y), Node.X.hl.Y.raig.Z.aw.A
	state:	Node.X.hl.Y.raig.Z.aw.B
	condition:	$A \neq B$
Rule 3	**operation:**	`Upd`(Node.X.Y.Z), Node.X.Y.Z.raig.F.avcr.C
	state:	`avg`(V,Node.X.Y.Z.raig.F.avcr.W,[0,now])
	condition:	$C < V \vee C > V$
Rule 4	**operation:**	`Upd`(Node.X.Y.Z)
	state:	`timeint`(V,`Upd`(Node.X.W.P),`Upd`(Node.X.Y.Z),[0,now])
	condition:	V < threshold
Rule 5	**operation:**	`Add`(Node.X.hl.Y)
	state:	`timeint`(V,`Del`(Node.X.hl.Y),`Add`(Node.X.hl.Y),[0,now])
	condition:	V < threshold
Rule 6	**operation:**	`Del`(Node.X.hl.Y)
	state:	`timeint`(V,`Add`(Node.X.hl.Y),`Del`(Node.X.hl.Y),[0,now])
	condition:	V < threshold
Rule 7	**operation:**	`Upd`(Node.X.hl.Z), Node.X.hl.Z.renodeid.R, Node.X.hl.Z.reportid.P
	state:	Node.R.hl.P.renodeid.Y
	condition:	$Y \neq X$
Rule 8	**operation:**	`Upd`(Node.X.nsp.(Y,Z)), Node.X.nsp.(Y,Z).raig.F.avcr.A, Node.X.nsp.(Y,Z).raig.F.maxcr.M
	state:	
	condition:	$A > M$
Rule 9	**operation:**	`Upd`(Node.X.hl.Q), Node.X.hl.Q.raig.W.avcr.C
	state:	Node.X.nsp.(P,Q).raig.W.maxcr.M, Node.X.hl.P.renodeid.R, (R and X remote nodes)
		Node.X.hl.P.reportid.O, Node.R.hl.O.raig.W.avcr.A (O and P ports connecting R and X)
	condition:	$C > M \vee A > M$
Rule 10	**operation:**	`Upd`(Node.X.hl.Z), Node.X.hl.Z.raig.F.maxcr.C, Node.X.hl.Z.renodeid.R, Node.X.hl.Z.reportid.P
	state:	Node.R.hl.P.raig.F.maxcr.M
	condition:	$C \neq M$

Figure 6: Example of rules

Action 3 results in delegating higher-level PGCG nodes to perform a more global analysis of the anomaly. Each time the alarm goes up a level in the hierarchy, following action 3, the PGL of each higher level possesses more and more global information and perhaps can correlate alarms coming from different lower-level sources to make more informed decisions. At the same time, this mechanism removes traditional bottlenecks arising from the aggregation of several alarms at a unique alarm correlation service (as it is commonly done today). Propagation of alarms upward in the hierarchy results in good distribution of the alarm correlation, transmission, and processing load.

The Alarm Propagation module implementation is straightforward within the PNNI design because it can simply reuse the flooding protocol and hierarchy information already present in the node, interpret the results coming from the diagnosis module, and take appropriate actions according to a small set of statically defined deterministic rules.

7 Fault Tolerant Diagnosis module

The Fault Tolerant Diagnosis module should satisfy two basic requirements. First, it should be able to correctly interpret anomaly reports (alarms) so that appropriate action can be taken in case of significant failures. In particular, it should avoid false positive and false negative diagnoses. Second, it should be itself resilient to faults. The first objective is more challenging. The second objective can be obtained by using well-known results derived from fault tolerant theory (see Section 3). We describe three possible fault scenarios and recommend a way to improve decidability. Moreover, we illustrate a case in which the PGL is found to be faulty.

7.1 Fault diagnosis

Researchers in the field of distributed network management have long been investigating techniques for performing diagnosis of network malfunctions through alarm correlation (for an overview see, for example, [16]). The rationale is that, given a set of symptoms represented by a variety of distinct alarm messages, an expert system should be able to correlate the symptoms and diagnose the underlying problem. We describe three different kinds of alarm processing scenarios in the context of PNNI. For simplicity, we describe these scenarios with respect to the PGRIP topology depicted in Figure 2.

Decidable Non-fault In the first scenario, a lower-level node $B.1.1.1$ floods an anomalous database update regarding the fact that $B.1.1.1$'s connection to peer group $PG(B.1.2)$ has changed to $PG(B.1.3)$. The anomaly is detected by the members of the PGCG with the rule

operation:	`Upd`(Node.X.upl.P), Node.X.upl.P.upnodeid.U
state:	Node.X.upl.P.upnodeid.V
condition:	U $\neq$ V

The PGCG nodes at this point may come to a consensus that the anomaly is significant but should be resolved at a higher level. Therefore, the inconclusive diagnosis is fed back to the Alarm Propagation module, and the PGL $B.1.1.2$ of group $PG(B.1.1)$ passes the alarm up to the higher level of the routing hierarchy. At the same time in group $PG(B.1.2)$, the switching from $PG(B.1.2)$ to $PG(B.1.3)$ also causes an anomaly to be reported in the group's PGCG. As in group $PG(B.1.1)$ the PGL $B.1.2.3$ of group $PG(B.1.2)$ floods the anomaly up to the higher-level group. At the higher level, logical nodes $B.1.1$, $B.1.2$, $B.1.3$, and $B.1.4$ therefore receive two alarms and can independently come up with a diagnosis. After a consensus is reached, the two alarms may be either sinked or combined into a single alarm, perhaps called *topology change*, that is recursively passed up by node $B.1.4$ to the next higher level, where it may be simply logged.

Not Decidable Fault The scenario just described can result from normal network behavior. In a more interesting case, peer group $PG(B.1.2)$, although equipped with PGRIP, does not generate any anomaly. This situation is more serious in that either $B.1.1.1$ or $PG(B.1.2)$'s PGL may be malicious because either $B.1.1.1$ is lying about $B.1.2.2$ having changed peer group or $PG(B.1.2)$'s PGL is spoofing $PG(B.1.3)$ with $B.1.1.1$. Unfortunately, it impossible to tell who is malicious just from examining the anomaly. The only solution is to pass the significant anomaly, perhaps called *unresolved topology change* up to the higher-level group for logging and for an operator's evaluation.

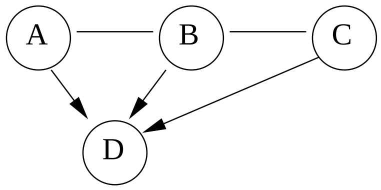

Figure 7: Diagnosis of Byzantine Fault

Decidable Fault A more productive scenario is given by the case in which peer group $PG(B.1.2)$ detects that node $B.1.2.2$ advertises to be part of a different peer group $PG(B.1.3)$. In peer group $PG(B.1.2)$, this anomaly can be detected with the rule

operation:	`Add`(Node.X.ni), Node.X.ni.address.A
state:	Node.X.ni.address.B
condition:	A $\neq$ B

When both peer groups detect an anomaly, higher-level nodes $B.1.1$, $B.1.2$, $B.1.3$, and $B.1.4$ can deduce that $B.1.2.2$, identified by X in the above rule, is faulty because it reports being part of a new peer group. As a result of this more definitive diagnosis, nodes $B.1.1$, $B.1.2$, $B.1.3$, and $B.1.4$ may further concur to employ preemption of node $B.1.2.2$. The Resolver in peer group $PG(B.1.2)$ (located in the PGL of peer group $PG(B.1.2)$) can then use the mechanism outlined in Section 8 to flood necessary packets through its own peer group and delete node $B.1.2.2$ from the routing hierarchy.

7.2 Improving decidability

A major assumption existing in current alarm correlation software is that the faults are nonmalicious. Fault diagnosis in a Byzantine environment is much harder because in some situations a malicious node can be smart enough to generate alarms that inhibit the correlation functions. Although the limitation of fault diagnosis in a Byzantine environment has been long recognized, recent work by Lincoln et al. [19] has demonstrated that, under reasonable fault modeling assumptions, by recording the history of anomalous events, one can construct an algorithm that converges to satisfactorily high levels of accuracy of diagnosis even of Byzantine faults.

The idea is illustrated in Figure 7. Consider nodes A, B, C, and D. Suppose that A tells D that B generated an anomaly and that B tells D that A generated an anomaly. D can detect that there is a fault but does not have enough information to determine which node is lying. D records the anomaly and does not take any corrective action. Suppose now that, some time later, C tells D that B generated the anomaly and B makes a counter claim as before. This time, D can use its history and deduce that it is unlikely that A and C are both conspiring against B, and therefore heuristically declare B faulty on the basis of multiple correlated anomaly reports. Although algorithms such as the one described in [19] could be used for improving PGRIP's fault diagnosis behavior in the Byzantine environment, the fundamental limitation in diagnosis of Byzantine must be recognized and acknowledged in the design to accommodate the case in which anomaly reports cannot resolve the source of a fault. For this reason, as outlined in Section 6, the Alarm Propagation module can simply log an undecidable diagnosis without taking further action.

It is important to realize that the extent of this limitation depends on the accuracy of the Anomaly Detection module and its ability to correlate anomalous behavior with past events, and it is therefore not easily quantifiable. A more common and easily solvable case (which is directly addressed by standard alarm correlation software) is the case in which anomaly correlation can be performed on the basis of multiple concurrent anomaly reports coming simultaneously from different routers. As described in Section 7.1, by gathering enough global information the diagnosis system may immediately and decisively identify faulty nodes and pass the information to the Resolver module.

7.3 Faulty PGL

Assuming that the PGCG satisfies both Condition 2 and Condition 3, by running an interactive consistency algorithm of the kind proposed in [9] or [10], the nodes can agree on a common diagnosis and implicitly allow the PGL to respond with the right action. An interesting situation arises when the PGCG nodes find that the PGL is faulty. In this case, it is necessary to demote the PGL and elect a new leader to carry out appropriate fault recovery mechanisms.[6] The PGL could be malicious and therefore it is not sufficient to start a standard PGL election phase, because the malicious PGL may interfere with the election process by granting itself arbitrary priorities. This problem has two solutions. The first one requires modification to the baseline PNNI standard. The second one requires a certain topological assumption and employing changes only in the PGCG nodes' flooding algorithm. Our design favors the latter solution because of its lesser impact. The idea is for $3m$ of the PGCG nodes to block any leader election messages coming from the faulty node so that no consensus may be reached for its election. The PNNI standard requires that at least two-thirds of the nodes in the peer group must acknowledge the election of a PGL for it to be legal. Condition 4 ensures that the (non-faulty subset of) PGCG nodes can block unauthorized election messages and inhibit the election of a faulty node.

[6] This case also presents itself when a non-PGCG malicious node unilaterally decides to become PGL without authorization.

8 Resolver module

The Resolver is activated in the PGL node after the PGCG nodes agree on a diagnosis. The Resolver module should answer, with very specific countermeasures, only those threats that are particularly severe. The Resolver module should be used carefully or not used at all because it can affect the network's operation. If misused, the Resolver can introduce instability or side effects that may be worse than the original fault.

We see the Resolver module as being capable of offering additional protocols to the PNNI standard so that (1) routing information can be verified, thus exploiting the redundancy and replication of information in the PNNI hierarchy and (2) the PGCG can preempt some nodes by cutting them out of the routing hierarchy until an operator can assess and remedy potential integrity compromises. Both mechanisms could be implemented, as the rest of the PGRIP system, without cryptography but, as we will see, their effectiveness would be greatly increased through the use of cryptographic signatures. In PNNI v2.0 the nodes of the routing hierarchy benefit from the key management hierarchy based on x.500. In this standard, nodes have group-wide asymmetric cryptographic keys that can be used for establishing cryptographically secure chains of trust. In PNNI v2.0 the keys are used during the hello handshake to authenticate the logical addresses of neighboring nodes. The key management infrastructure provides the necessary functionality to distribute and maintain the public/secret key pairs in the network.

8.1 Database verification

In the PNNI standard, nodes that have established connectivity through the Hello protocol synchronize their databases by directly exchanging topology information. In the database verification mechanism, the protocol used by nodes that are directly connected is used also by nodes that are indirectly connected when they suspect that a directly connected neighbor is faulty. In this mechanism, as a result of detecting a faulty node, a PGL node can establish a special connection to a third-party node and explicitly request topology information.

This mechanism reuses the existing PNNI database synchronization protocol and permits an additional level of verification mechanism without much standardization. The mechanism does not require cryptography, but relies exclusively on redundancy. Cryptography, if available, could make this verification mechanism even more effective when the topology does not satisfy Condition 1 and the suspected faulty node can mount a man-in-the-middle attack and affect the database synchronization.

8.2 Node preemption

The preemption mechanism allows the PGL to effectively eliminate a node from the peer group databases so that it cannot adversely affect the group. Preemption can be implemented in a fault tolerant way by reusing the PGL election algorithm. Condition 4 was introduced to allow the preemption of the PGL if the PGCG diagnoses the PGL as faulty. The properties introduced by Condition 4 can be exploited once more to guarantee that if a node founding faulty by the PGCG cannot influence, with malicious messages, more than two-thirds of the nodes in the group and thus prevent consensus. Once the group reaches a consensus on preempting a particular node, all paths through the node must be deleted and any message originating from the preempted node must be dropped. In the absence of cryptographic tools, this mechanism could guarantee the proper operation of only two-thirds of the nodes in the peer group and would still permit a malicious node (through the forging of spoofed messages) to be active in one-third of the group and even elect itself as PGL. However, because our two-connection assumption should be obeyed at all levels of the hierarchy, this anomalous condition would certainly be detected by the higher-level peer group. At the higher-level the peer group containing the malicious node will appear to behave inconsistently and/or to have two PGLs. Upon detection of this anomaly the entire peer group may therefore be preempted.

As in the case of database verification, cryptographically securing the node preemption algorithm would render it much more effective. Once a node is diagnosed to be faulty, properly signed messages can be broadcast by the PGL node to implement the preemption throughout the entire per group. In this case a malicious node could not affect the preemption process; thus, a resolution of the fault could be performed in a much more expeditious manner.

9 Related work

Most routing infrastructure protection mechanisms that have been proposed (see [4, 14, 17, 18]) differ from our approach because they are preventive in that they use cryptographic services to secure the routing protocols. Hauser et al. [6], while still proposing a preventive methodology based on the work in [14], propose to reduce the cost of cryptographic protection by optimizing the authentication of the routing messages that carry redundant information though hash chains. The work closest to ours is the one by Wu et al. [7]. They designed an intrusion detection system that can detect anomalies in the OSPF routing protocol. In their design, properly modified SNMP agents and protocol analyzers are distributed throughout the OSPF routing hierarchy to detect anomalous behavior. Although the idea is in principle similar to PGRIP, there several crucial differences. The most relevant difference is that PGRIP does not analyze the protocol packets but operates at a higher abstraction level. This allows PGRIP to be decoupled from the actual routing transport mechanism and therefore to be more portable. In particular, PGRIP can transparently coexist with different security measures employed at the protocol level without modification. Another important differences are that Wu et al. concentrate on intrusions; they do not seem to allow the cooperation of different distributed nodes in the diagnosis process, and their design does not seem to handle spontaneous failures. Moreover, their detection module allows the real-time interception and manipulation of routing packets, thus introducing additional complexity in the verification of the routing protocols thus compromising their resilience.

10 Conclusion

We have presented a novel approach for securing global routing infrastructures and have instantiated our ideas for the design of PGRIP. The PGRIP design can be used to augment the current PNNI standard and to offer a high level of integrity protection without requiring significant changes to the standard, and without entirely relying on cryptography. PGRIP handles both malicious and nonmalicious faults in a unified manner and can therefore be used as an additional level of assurance for the proper operation of large communication networks. PGRIP's effectiveness is intimately tied to its ability to properly detect and processing anomalies and making precise and informative diagnoses. It is therefore necessary to validate its effectiveness with hands-on experiments that can reproduce numerous fault scenarios and help in accumulating enough heuristical experience for the specification of effective anomaly detection rules. In addition, the substantial complexity of Byzantine faults diagnosis should be addressed as one of our primary future research efforts.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1), 1997.

[2] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *ACM SIGMOD International Conference*, pages 505–516, Montreal, Canada, June 1996.

[3] S. Eker. Fast Matching in Combinations of Regular Equational Theories. In J. Meseguer, editor, *First Intl. Workshop on Rewriting Logic and Its Applications*, pages 90–108. Elsevier Science, September 1996.

[4] D. Estrin and G. Tsudik. Security Issues in Policy Routing. In *Proc. of the 1989 IEEE Symposium on Security and Privacy*, pages 183–193, Oakland, CA, May 1989.

[5] ATM Forum. Private Network-Network Interface Version 2.0 Specification. Technical Report BTD-PNNI-02.00, 1997.

[6] R. Hauser, T. Przygienda, and G. Tsudik. Reducing the Cost of Security in Link-State Routing. In *Proc. of the 1997 Symposium on Network and Distributed System Security (NDSS'97)*, pages 93–99, San Diego, CA, February 1997.

[7] Y.F. Jou and S.F. Wu. Architecture Design of a Scalable Intrusion Detection System for the Emerging Network Infrastructure. Technical Report, DARPA E296, April 1997.

[8] S.K. Kilger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A Coding Approach to Event Correlation. In A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, *Integrated Network Management IV*, page 266. Chapman & Hall, 1995.

[9] L. Lamport, R. Shostak, and M. Pease. The Byzantine General Problem. *ACM Transaction on Programming Languages and Systems*, 4(3):382, 1982.

[10] P. Lincoln and J. Rushby. A Formally Verified Algorithm for Interactive Consistency under a Hybrid Fault Model. Technical Report SRI-CSL-93-02, April 1993.

[11] J. Moy. OSPF Version 2. Technical Report RFC 2178, 1997.

[12] S.L. Murphy and M.R. Badger. Digital Signature Protection of the OSPF Routing Protocol. In *Proc. of the 1996 Symposium on Network and Distributed System Security (NDSS'96)*, San Diego, CA, February 1996.

[13] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *IEEE International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[14] R. Perlman. *Network Layer Protocols with Byzantine Robustness.* Technical Report, MIT LCS TR-429, Massachusetts Institute of Technology, October 1988.

[15] M.K. Reiter. Secure Agreement Protocols: Reliable and Atomic Multicast in Rampart. In *2nd ACM Conference on Computer and Communication Security*, pages 68–80, Fairfax, Virginia, November 1994.

[16] L. Ricciulli and N. Shacham. Modeling Correlated Alarms in Network Management Systems. In *Communication Networks and Distributed Systems Modeling and Simulation*, 1997.

[17] K.E. Sirios and S.T. Kent. Securing the Nimrod Routing Architecture. In *Proc. of the 1997 Symposium on Network and Distributed System Security (NDSS'97)*, pages 74–84, San Diego, CA, February 1997.

[18] B.R. Smith, S. Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance-Vector Routing Protocols. In *Proc. of the 1997 Symposium on Network and Distributed System Security (NDSS'97)*, pages 85–92, San Diego, CA, February 1997.

[19] C.J. Walter, P. Lincoln, and N. Suri. Formally Verified On-Line Diagnosis. *IEEE Transactions on Software Engineering*, 23(11):684, 1997.

[20] S.F. Wu, F. Wang, B.M. Vetter, R. Cleaveland, Y.F. Jou, F. Gong, and C. Sargor. Intrusion Detection for Link-State Routing Protocols. Presented in the 5 Minute Talks session at *1997 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.