End-to-end Verification of QoS Policies

Adel El-Atawy
Google Inc
Mountain View, CA
aelatawy@google.com

Taghrid Samak
Lawrence Berkeley National Laboratory
Berkeley, CA
tsamak@lbl.gov

***Abstract*—Configuring a large number of routers and network devices to achieve quality of service (QoS) goals is a challenging task. In a differentiated services (DiffServ) environment, traffic flows are assigned specific classes of service, and service level agreements (SLA) are enforced at routers within each domain. We present a model for QoS configurations that facilitates efficient property-based verification. Network configuration is given as a set of policies governing each device. The model efficiently checks the required properties against the current configuration using computation tree logic (CTL) model checking. By symbolically modeling possible decision paths for different flows from source to destination, properties can be checked at each hop, and assessments can be made on how closely configurations adhere to the specified agreement. The model also covers configuration debugging given a specific QoS violation. Efficiency and scalability of the model are analyzed for policy per-hop behavior (PHB) parameters over large network configurations.**

I. Introduction

Analyzing quality of service (QoS) policies across different devices is essential to large scale quality-sensitive environments. Different devices can honor the same quality request differently by imposing slightly incompatible sets of configuration parameters. Performing such analysis across devices (possibly across domains, if collaboration is encouraged) was limited in previous work to studying the possible conflicts of quality classes rather than the actual parameters used to satisfy these classes. On the practical side, administrators lacked the tools to analyze QoS policies without the actual deployment on a real network.

A service level agreement (SLA) is the contract by which requests for a certain QoS are specified between the user and the Internet service provider (ISP). Applications are given different treatments from source to destination based on SLA specifications. To achieve the requirements of an SLA, policies are configured on different network nodes along the path from source to destination. Guaranteeing performance stability and correctness between different configurations on multiple nodes is a critical issue. Misconfiguring policies on large domains can cause conflicts between policy parameters within different devices in the domain. The same problem happens due to variations in device capabilities and policy interpretation. These conflicts lead to performance instability, unpredictability and quality degradation. Moreover, inconsistent configurations handling the same traffic class can lead to violation of SLAs and unsatisfactory overall performance.

In this work, a model is presented that uses unbounded model checking, particularly computation tree logic (CTL), to verify the correctness and consistency of QoS configurations across multiple devices and/or domains. A query processing mechanism is also used to check for user defined violations. The system uses a symbolic implementation of model checking that facilitates the analysis of multiple nodes, flows and packets simultaneously with graceful handling of networks' increasing size and complexity. The implementation uses binary decision diagrams (BDD [1], [2]) to model different aspects of the model: states, transitions, and property verification intermediate and final results. A BDD is a very powerful structure that can be used to represent boolean expressions and sets in a concise/canonical symbolic form. Moreover, it defines boolean and set operations using theoretically proven and efficient algorithms.

The system is a significant addition to the configuration analysis tool: *ConfigChecker* [3]. *ConfigChecker* was used to analyze and verify the correctness of network reachability and security configurations. However, it does not have a way to model actual physical link capacities, or flow dynamic properties (*e.g.*, flow bandwidth, delay, quality imposed by successive routers, etc). This work adds the QoS aspects to this analysis tool making it possible to verify routing tables, access control lists, application layer policies, IPSec as well as QoS configurations in a single consistent and homogeneous model. The underlying engine for CTL query processing and model management had to be changed to accommodate the unique nature of quality properties of traffic. The quality experienced by a flow is an aggregate of the treatment it receives at all intermediate hops, which is completely different from the memoryless nature of basic routing and firewall operations analyzed in *ConfigChecker*.

The main contributions of this work are:

- CTL-based model for QoS configurations.
- Property-based (SLA-based) QoS verification.
- QoS policy debugging for a specific QoS violation.
- Answering what-if questions regarding policy changes.
- Incorporating QoS with general network reachability and security analysis in a single model.

A. DiffServ Environment

QoS guarantees can be specified by assigning values to certain traffic characteristics (*e.g.*, delay), link parameters (*e.g.*, bandwidth) or router configuration (*e.g.*, queue length). A configuration is a mapping from higher level SLA, to such parameters and characteristics. Three main traffic classes are addressed in the DiffServ domain; expedited forwarding (EF), best effort (BE), and aggregated forwarding (AF). Policy rules are specified in a way to control the behavior of each class. Those rules may differ across different nodes and domains.

Implementing DiffServ in a network domain requires supporting two main functionalities:

- *Traffic conditioning:* is implemented in boundary routers, and is responsible for classifying (*i.e.*, marking) packets into traffic classes.
- *Per hop behavior (PHB):* is implemented on all routers in the DiffServ domain, and specifies individual nodes' treatment of each class.

The classification part of the traffic conditioning process is similar to general router operations. Incoming packets are mapped to specific traffic classes (*e.g.*, EF, AF). The marking process chooses the classification that reflects the desired level of quality for these flows. Depending on device specification and vendors, the marking could be performed using different options; IP header fields (precedence or differentiated services code point, DSCP), or other manufacturer specific marking (*e.g.*, Cisco QoS group). The latter is used when networks need to define more classes than are supported by IP headers. Applying this step at boundary routers might cause packets to receive different classifications as they travel from one domain to another. Different ISPs might provide different classes of service, or even the same service with different markings. Even if all domains on the path support the same class of service defined in the IP header, specifying QoS groups on some domains might result in inconsistent behavior if the marking does not correspond to a supported behavior.

Inside a DiffServ domain, a PHB is a collection of action parameters applied to a certain traffic flow. After the marking process, packets belonging to the same flow should get the same treatment from devices within the domain. This is performed by defining PHB action parameters. PHB specifications from QoS policy information model (RFC 3644 [4]) defines two main categories of PHB actions: actions controlling bandwidth and delay, and actions for congestion control.

The general setup is also applicable to other QoS protocols. For example, MPLS (multi-protocol label switching) can be implemented instead of internal DiffServ routers. In fact the case of MPLS is easier for configuration analysis where the main task would be checking labels over the paths instead of performing analysis over each individual PHB parameter.

The following section describes the state model and how policies are encoded. Section III explains how temporal analysis is represented in our model. QoS queries are mapped to CTL properties using temporal as well as quantifying operators to check whether these properties hold and at which nodes. Implementation details are presented in section IV. Evaluation of the system with respect to policy configuration parameters is presented in section V.

II. Configuration Model

Formal verification of network configurations regards the network as a Kripke structure [5], $K = \langle S, T, A, \sigma \rangle$, with the following components:

- S is a set of states defined on atomic propositions.
- $T \subseteq S \times S$ is a transition function between states.
- $A = 2^p$ is a finite alphabet using a set of p propositions.
- $\sigma : S \mapsto A$ is a valuation function from a state to a label.

A state describes a specific network condition at a certain point in time. Atomic propositions are basic domain-specific descriptors reflecting the configuration belonging to the state. For example, packet header fields when assigned specific values can be considered atomic propositions. Routing and security policy parameters can also correspond to atomic propositions. The temporal logic aspect of K is manifested by transitions from state to another state as time progresses. For example, if a state corresponds to a packet arriving at one router, then the next state would be the packet at the next hop.

The definition of the alphabet A over all possible atomic propositions is the formal way of describing that an expression based on the atomic constructs is satisfied at a certain state. Therefore σ is now mapping each possible state to the atomic expression that can be satisfied at this state.

A. State Representation

States in this model are defined in the context of a packet traversing a set of devices from source to destination. This involves all intermediate nodes and their configurations: routing tables, security policies and QoS specifications.

A single state is defined as a boolean expression encoding three components; packet header fields (Pkt), a certain network location (Loc), and a corresponding PHB parameters. The state expression is the conjunction of the boolean expression for each of the three components.

$$s = Pkt \wedge Loc \wedge PHB$$

We use the basic IP header fields representing packets (Pkt): Protocol: 8 bits, IP addresses: 32×2 bits for source and destination, Port numbers (16×2 bits source and destination), and IP precedences, or DSCP.[1].

The location is expressed as 32 bits for the IP address of the network node along with 16 bits for the specific application layer service. The PHB component of the expression specifies the set of QoS policy actions, reflected in parameter values (table I). Each parameter will be mapped to a boolean expression depending on its type; boolean, quantitative or range.

The number of bits needed to encode the PHB expression:

$$L = \mid V_B \mid + \sum_{Q_i \in V_Q} \lceil log_2(max(Q_i)) \rceil + \sum_{R_i \in V_R} \lceil log_2(max(R_i)) \rceil$$

where

- $V_B = \{B_1, B_2, \dots B_k\}$ is the set of boolean parameters,
- $V_Q = \{Q_1, Q_2, \dots Q_l\}$ is the set of quantitative parameters,
- $V_R = \{R_1, R_2, \dots R_m\}$ is the set of range parameters,
- and $max(x)$ is the maximum possible value of the parameter.

A single PHB response to a certain flow is encoded as a boolean expression combining all parameters involved in the policy definition with respect to that flow. For a parameter $P \in V_B \cup V_Q \cup V_R$, the expression is built as follows:

- A boolean parameter is mapped to a single bit expression; $B = b$.
- A quantitative parameter is mapped to the numerical encoding of the value; $Q = q_{log(max(Q))} \cdots q_1 q_0$.
- A range parameter is mapped to the disjunction of powers of two values spanning the range (*e.g.*, $[1, 3]$ over 2 bits is encoded as $R = \bar{r_1} r_0 \vee r_1 \bar{r_0} \vee r_1 r_0 = \bar{r_1} r_0 \vee r_1$.)

[1] In this work, DSCP, ToS, CoS, or IP precedence are used interchangeably.

Class	Property	Type
PHB	Max Packet size	Quantitative
Bandwidth	Forwarding priority	Quantitative
	Bandwidth units	Boolean
	Min/Max Bandwidth	Range
	Max Delay	Quantitative
	Max Jitter	Quantitative
	Fairness	Boolean
Congestion Control	Queue size units	Boolean
	Queue size	Quantitative
	Drop method	Boolean
	Drop Threshold units	Boolean
	Drop Threshold method	Boolean
	Min/Max Threshold value	Range

TABLE I
QoS POLICY INFORMATION MODEL (PQIM) PHB PROPERTIES MAPPING

B. State Transitions

Transitions in this model correspond to packets moving between locations in the network. The source location could be a general router or QoS router. The latter might be a boundary router that performs marking or an internal DiffServ enabled router. For each type of node, the next state will depend on the functionality of this node. General routers only forward packets. So, the next state will only differ in location, boundary QoS routers might change the DSCP value in the packet header field, and finally DiffServ routers might change any of the PHB parameters for that packet. For security and routing policies, we follow the same structure as in [3].

The aim is to define members of the transition functions $T \subseteq S \times S$. For two states, $s = Pkt \wedge Loc \wedge PHB$, and $s' = Pkt' \wedge Loc' \wedge PHB'$, $T(s, s') = true$ iff a packet satisfying s at any point in time, will satisfy s' in the next time instance. These transitions are usually derived from the domain policy definition: routing, security or QoS policies. In order to construct the transition function two copies of all variables are used: current state, and next state.

Example: For a small network of three nodes, assume the locations and IPs are expressed as 2-bit integer values. The routing policy for node 0 is defined such that packets with destination 3 should be routed to node 2. To express the state of the configuration performing a transition from state $s = (Loc = 0) \wedge (Pkt.dst = 3)$ to the next state $s' = (Loc' = 2) \wedge (Pkt'.dst = 3)$, the following condition should be satisfied.

$$(\bar{l_1} \wedge \bar{l_0}) \wedge (d_1 \wedge d_0) \wedge (l_1' \wedge \bar{l_0'}) \wedge (d_1' \wedge d_0')$$

where l_1, l_0 are location variables, and d_1, d_0 are destination variables. For every variable v, v' is the next state variable corresponding to v.

The overall transition relation T is the disjunction of the partial transition functions obtained from each device separately. Formally, one can write $T = \bigvee_{d \in D} \zeta(d)$, where $\zeta(d)$ is the set of transitions relevant to a specific device d in the overall set of devices D.

For each router type, the transitions will be defined in terms of state expressions s and s' as well as the defined policy configurations P. A policy is specified as a sequence of rules with condition and actions.

$$Rule := Condition \Rightarrow Action$$

Conditions are defined as $(f_1 = v_1) \wedge (f_2 = v_2) \ldots$, where f is either packet field or location. For example, routing a packet will be based on matching a rule with common prefix. The condition is specified on Pkt for the set of packets satisfying current state to be $Pkt.dst = P.prefix$. P is used instead of the actual triggered policy rule to simplify the notation. It is assumed that each packet will match the highest priority rule according to the policy.

Actions depend on node type, whether it is just forwarding (next location), a transformation (changing header values), or applying QoS parameter (change PHB). For each node type, the following subsections describe state transitions and how the policy definition will affect changes at each state.

1) General router: Routing policies are described at each location with destination prefixes and next hop information. A general router does not change packet header fields, so in the next state, information will be the same for all packets. A rule matching "$this$" location will check the packet destination prefix, $Pkt.dst$. If the destination matches, all Pkt fields are transferred to the next state and the location of the next state is changed to the next hop according to the policy.

$$(Loc = this) \wedge (Pkt.dst = P.prefix)$$
$$\wedge (Loc' = P.next) \wedge (Pkt' = Pkt)$$

Writing $Loc = this$ is a way of writing an expression over the variables of Loc that is only satisfied by the value of "$this$". For example, $v_2v_1v_0 = 6$ is equivalent to the expression $v_2v_1\overline{v_0}$. Also, $Pkt' = Pkt$ is a shorthand for the longer expression that restricts all variables of the next state to be equivalent to those of the current state. For example, $X = Y$, where X and Y are two 2-bit integers is identical to writing: $(x_1 \Leftrightarrow y_1) \wedge (x_0 \Leftrightarrow y_0)$

2) Boundary router: Since those routers are responsible for marking packets, the state will change the DSCP field of the header. So, the transition to the next state will require changing the location, and the DSCP.

$$(Loc = this) \wedge (Pkt = P.pkt)$$
$$\wedge (Loc' = P.next) \wedge (Pkt'.dscp = P.dscp) \quad (1)$$

In general, the device can decide which mark to assign to a flow based on any field in the packet header, including the previous mark or DSCP value (*i.e.*, the condition $P.pkt$ in the above expression).

3) Internal DiffServ router: This network component is different as it specifies PHB at each node. As proposed in [6], actions controlling PHB might change from device to device. Each policy is enforcing its own interpretation of SLA parameters given local capabilities, or even hardware constraints. When parameter value changes at one node for a traffic flow, this means a new constraint has been added to the treatment of this flow. The final PHB will be restricted by the different specifications along the path. For example, if audio streaming is given a bandwidth range of $20 - 40\%$ at one node, and another node is enforcing $30 - 50\%$, then the overall path will have to enforce the overlapping range of $30 - 40\%$. This range, is the range of bandwidth in which the audio stream can operate "safely". In other words, if the original audio stream was sent using, say, 35%, then all nodes will work with conformance with the need of such stream. In this case, when a packet makes a transition from Loc to Loc', PHB parameter values will be transformed reflecting the more restrictive treatment of the two locations.

Each parameter type; boolean, quantitative and range, will be transformed according to an aggregation function reflecting the different behaviors of the parameter at successive hops. The behavior of each parameter at Loc' will be computed from PHB specification at both Loc and Loc', reflecting what the final treatment will be after passing both states.

Quantitative transformation: PHBs parameters that reflect quantitative values (*e.g.*, queue length, packet size, etc...) are restricted by the minimum of all values involved in computing an overall packet treatment. At a current state, s, a quantitative parameter Q holds the aggregated value so far from all previous states (network locations). The transition from s to s' will change the value of Q if the policy at the new location is more restrictive. The transition takes the form:

$$(Loc = this) \wedge (Pkt = P.pkt)$$
$$\wedge (Loc' = P.next) \wedge (Pkt' = Pkt) \wedge (Q' = min(Q, P.Q))$$

Range transformation: The final behavior of range parameters across different devices corresponds to the intersection of parameter values for nodes passed from source to destination. To model single transition, range value R for state s should be the combined overlapping range from all previous states. For example, bandwidth is defined as a range of values, and expressed as conjunctions to reflect possible range decomposition. When packet moves from device to device, s to s', the aggregated range, R', is computed from the current aggregated range R and the range enforced by the policy $P.R$. The equivalent transformation will take the form:

$$(Loc = this) \wedge (Pkt = P.pkt) \wedge$$
$$(Loc' = P.next) \wedge (Pkt' = Pkt) \wedge (R' = R \wedge P.R))$$

Boolean transformation: The case of boolean variables is simpler than quantitative and range, where parameter values need to be the same for all nodes on the path from source to destination. It can be viewed also as describing if a certain condition exists or does not hold between states. In this case, either all nodes will enforce the parameter ($true$), or some of them do not care about this parameter ($false$). For example, the transition should make sure that if the current state requires priority to be enforced, all previous transitions must have asked for priority. The aggregation will now translate to whether all values are $true$ along the path.

$$(Loc = this) \wedge (Pkt = P.pkt) \wedge$$
$$(Loc' = P.next) \wedge (Pkt' = Pkt) \wedge (B' = B \wedge P.B)$$

III. QoS Property Verification

Property verification for QoS configuration aims to check whether a specific SLA requested by customers are being enforced at each router in DiffServ domain from source to destination. This maps to querying the model for satisfying certain properties. The overall SLA can be checked, or specific questions regarding certain parameters can be asked.

The model regards a property as a boolean formula, and the aim is to check if the formula is satisfied at a particular state or a set of states. A property ϕ is a function expressed in the model variables $\phi = \pi(Loc, Pkt, PHB)$. For checking properties, the temporal aspect of the model needs to be defined along with an initial set of states s_0. The following are the basic temporal operators in CTL (computational tree logic) [5], defined for a property/formula ϕ and ψ:

- $\mathbf{X}\phi$: ϕ holds in the next point of time.
- $\phi\mathbf{U}\psi$: ϕ has to hold until ψ becomes true.
- $\mathbf{F}\phi$: ϕ will eventually be true.
- $\mathbf{G}\phi$: ϕ has to hold forever (globally true).

In addition to the temporal operators, two quantifiers are also defined:

- $\mathbf{A}\phi$: ϕ holds for all possible computation paths (*i.e.*, all next hops satisfies ϕ).
- $\mathbf{E}\phi$: there exists a computation path satisfying ϕ.

Complex properties can be expressed using temporal operators with quantifiers. For example, the model can check if the bandwidth for traffic class EF is in the range $20 - 40\%$ by:

$$\phi = (Pkt.DSCP = EF) \wedge (PHB.bw \in [20, 40])$$

Given an initial state s_0, the property can be checked at all paths starting from s_0. If the system requires that the bandwidth range of $20 - 40\%$ should be satisfied at all nodes serving flow EF, the CTL formula can be formed as $\mathbf{AG}\phi$. This means all states that can be reached from s_0 will have a non-empty bandwidth intersection with $20 - 40\%$, hence satisfy this property.

A more precise query would check if ϕ is satisfied from source to destination. In this case, the initial state will describe the source location with packet information $s_0 = (Loc = src)$. Another property ψ can be defined as reaching the desired destination, $\psi = (Loc = dst)$. The overall CTL formula checking whether traffic belonging to EF class will face bandwidth $20 - 40\%$ from src to dst will be, $\phi\mathbf{U}\psi$.

In the following, different families of queries are shown. In each category, one or more examples will be discussed.

a) SLA enforcement verification questions: To verify that certain aspects of the SLA are honored by the policies in place, the administrators have to check that certain categories of flows are given some predefined levels of quality treatment. Examples of this type of questions:

- No traffic from domain X (one of the ISP clients) should receive bandwidth less than b Mbps or have packet size restricted to less than s bytes.
- Voice traffic passing through our network should have available bandwidth in the range $[x, y]$ Kbps as long as it is between two specific partner ISPs.

b) Administrative and debugging queries: This family of queries are more concerned with a specific flow. For example:

- A specific server is complaining from a drop in video quality because of excessive delay. Using a few test clients, is there any intermediate hops that impose low quality restrictions on its flows?
- There is a high bias in the quality of flows received by a client depending on the server (and all servers are known not to be overloaded). What is the range of PHB parameters for web traffic reaching this specific client?

c) What-if scenarios and provisioning queries: These queries are the most sophisticated and they use the unique features in the proposed model. What-if scenarios are evaluated starting by creating two identical versions of the model. Then, one of the models are modified at one or more nodes by replacing original state-state transitions with others that reflect the proposed modifications to the network. Two identical queries can then be executed over each of the models, and the results can be analyzed to study the effect of the proposed changes. The calculation steps are as follows:

1) Calculate original transition relation: T.
2) Create a copy of T: T_{new}.
3) Remove changed devices ($d \in D'$): $T_{new} = T_{new} \wedge \neg \bigvee_{d \in D'} (loc = d.loc)$ where D' is the set of updated devices.
4) Add updated devices ($d \in D'$): $T_{new} = T_{new} \vee \bigvee_{d \in D'} \zeta(d)$
5) Query comparison: Evaluate the same CTL/Boolean expression (*e.g.*, $\phi(.)$) using both models: $\phi(T)$ and $\phi(T_{new})$, and compare the two results.

This process is efficiently implemented, and does not require a rebuild of the system to adapt to the changes in few devices. Examples include:

- If at a given node, the PHB parameters corresponding to a specific class have changed to new values, what will happen to the quality of a specific set of flows?
- What if device x has its QoS module turned off (*i.e.*, changed into a regular router)? What are the implications of this change on a specific set of flows or a domain?

IV. Discussion and Implementation Issues

In the model presentation, there were various aspects of the system and its implementation that have been abstracted for the sake of clarity. The following points are some of these aspects that are needed to understand the practicality of the system and the rigor behind its design.

A. Layer-by-layer verification:

The evaluation and property verification system was implemented on top of *ConfigChecker* [3]. This system was originally implemented for the purpose of analyzing reachability and security policy consistency across devices. Therefore, it is a legitimate concern that some of the results obtained when investigating QoS properties are affected by reachability or security misconfigurations that are not QoS in nature. To separate the analysis of each layer, a "QoS-skip" variable is added to the model. When this variable is $true$, QoS policies are converted into a no-operation (NOP) device. In other words, QoS markers and PHB parameters do not affect the packets, and all packets pass through as if passing through a repeater. This is implemented by modifying eqn. 1 to be:

$$(Loc = this) \wedge (Loc' = P.next) \wedge$$
$$[qosSkip \vee ((Pkt = P.pkt) \wedge (Pkt'.dscp = P.dscp))]$$

Once the analysis of a complex network shows that the network reachability requirements are intact, the "QoS-skip" flag can be switched off and QoS-relevant queries can be performed. Another apoproach is to build two versions of the system: with and without the QoS layer. However, this requires rebuilding the transition function, where the bulk of the computation cost takes place.

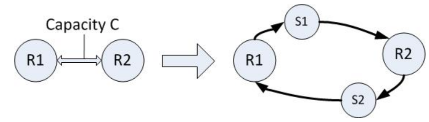


Fig. 1. Capacity Model

B. Modeling link capacities:

Another limitation for the original *ConfigChecker* model is that it did not include any notion of physical link capacities or any other of their properties. Using the presented extensions, it is possible to model any link as a strict traffic shaper that enforces a bandwidth in the range $[0, C)$ where C is the link capacity. A connection between two regular routers will be broken into two links with a QoS router in between (as shown in figure 1. Its policy will simply be a single line stating that for any packet, regardless of its properties and DSCP, the PHB will be that $BW \in [0, C]$.

To investigate the maximum possible bandwidth between any two nodes in the network, one can execute this simple query: $(Loc = source) \wedge AF(Loc = dest)$. Finding the range of BW for all satisfying assignments of the answer to the query can be obtained efficiently using BDDs (*i.e.*, existential quantification to extract the BW variables, followed by a single root-to-leaf traversal via the $true$ branches).

C. Modeling markers and shapers:

In order to simplify implementation without loosing expressiveness, the QoS devices are designed as two successive components: a marker (or DiffServ component) followed by a generic router. Therefore, the QoS logic is separated from that of traffic routing. This simplifies the expressions used to build the transitions, and enables us to disable the QoS component separately (*i.e.*, "QoS-skip"). Also, it adds more control when building what-if queries.

V. System and Model Evaluation

A. Evaluation Methodology

1) Topology Generation: To evaluate domain level conflicts (PDB policies), a large set of test network topologies is needed. A network is modeled as a graph with machines as vertices and links as edges. The following parameters are used to control the generated configurations:

- *Network size n:* The number of nodes in a domain.
- *Connectivity bias δ:* The average degree per graph node (used to estimate how many connections are needed between graph nodes). The total numbers of edges will be $e = (n - 1) \times \delta/2$

After generating graph nodes, e random edges are added. To guarantee that the graph is connected, the connected components of the graph is computed after adding the initial edges. Random nodes are then selected from each component and extra edges are added to connect the network clusters. After generating the topology, policies are then added to each node in the network.

2) Policy Generation: Test cases of policies were generated based on the behavior of real policies analyzed in [7]. In most enterprises (60%), the same combinations of traffic classes are defined in all nodes in the network. For example, if real time traffic is defined in one node, all other nodes must define a

Parameter	Values
Network size	$n = 10 - 10,000$
Connectivity	$\delta = 2 - 10$
Node bias	$B_n = 0.1 - 0.9$
Parameter bias	$B_p = 0.1 - 0.9$

TABLE II
TOPOLOGY AND POLICY GENERATION OPTIONS.

rule for real time traffic. A very small percentage of domain policies will have different classes at each node.

The generation parameters handle two option sets:

- *Node bias B_n:* This controls the fraction of the network nodes that will have the same set of flow classes defined in the policy. The observations of [7] are taken into consideration when choosing this parameter. A random variable controls the rest of the nodes. For a network of size n, $(n \times B_n)$ nodes will have similar policies. Random parameter values are generated for the remaining nodes.
- *Parameter bias B_p:* This controls the behavior of individual policies on each network node. Each PHB parameter can take different values according to a generation bias. The subset of nodes that have the same class combination will have similar PHBs. Random variables are used to generate PHB parameters for the remaining nodes.

For each traffic class, a policy rule is generated at each node to specify all PHB parameters from table I. Table II summarizes different generation parameters used in the evaluation. Both node and policy parameter biases will affect the overall complexity. Diverse values for parameters might result in more complicated expressions and added complexity to BDD operations. On the other hand, any common terms in the BDD tree will be shared between subexpressions.

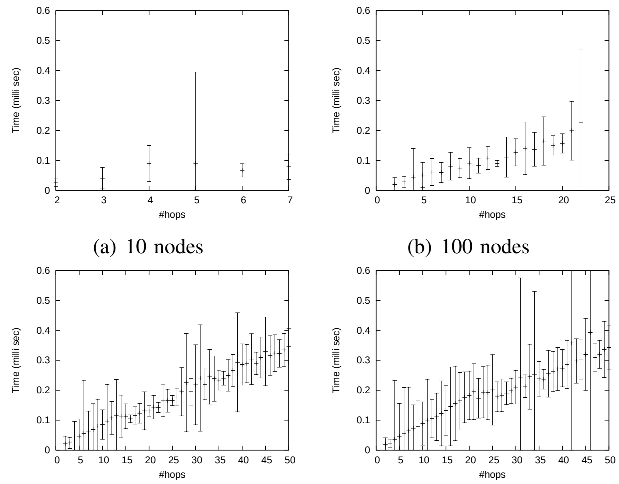
B. Experimental Results

1) Query processing for property verification: In this seciton, we focus on the performance of individual queries after the system has been initialized. After checking end-to-end reachability and making sure that routing is valid (analysis can be found in [3]), QoS queries regarding certain PHB need to be answered. Table III summarizes results for all policy parameters described in section II (table I). For different network sizes, generic end-to-end queries on different parameters are performed. The query specifies source and destination devices, and checks a certain flow for specific desired PHB. The average is computed over all source-destination pairs for each network size. From the generation parameters, different nodes in the same network might enforce similar or different policies (*i.e.*, node bias). For each parameter, similar or completely different values might be applied (*i.e.*, parameter bias).
From table III, the most complicated query processing time was 0.09 msec for a network of size $10K$ nodes. It is also evident that parameter type affects the overall trend. Quantitative parameters need more processing time per query because they involve more manipulation than usual bit operations optimized by BDDs. The fact that min and max calculation is embedded into state transition for those parameters makes extra processing time unavoidable.

Individual end-to-end queries also depend on the number of hops between source and destination. Once source and destination are specified, the complex transition graph is pruned to include only those relevant to the flow. Because

	100	$1K$	$2K$	$5K$	$10K$
Quan	0.067	0.072	0.075	0.079	0.093
Range	0.049	0.052	0.054	0.057	0.067
Bool	0.041	0.046	0.048	0.052	0.061

TABLE III
PROCESSING TIME ($milli\,sec$) COMPARISON BASED ON PARAMETER TYPE



(a) 10 nodes (b) 100 nodes

(c) 1000 nodes (d) 10000 nodes

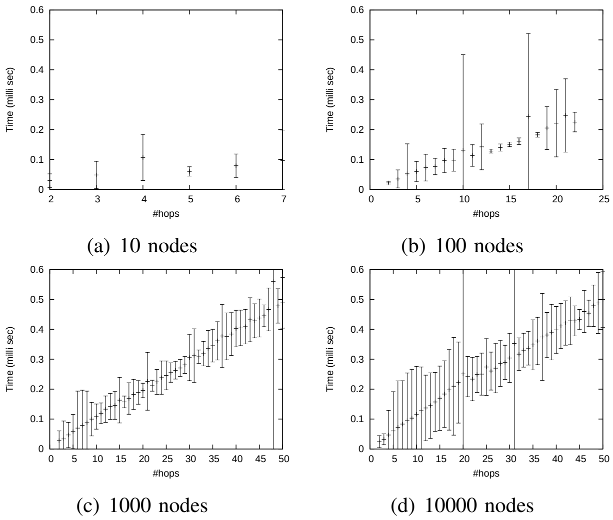
Fig. 2. Bandwidth query average time versus number of hops

this is a symbolic reduction process, it is highly efficient and does not require any exhaustive state traversal. Hence, the main factor affecting query processing time is the path length from source to destination. For each parameter query, the average time is compared versus the number of hops on the path.

Queries involving all parameters are evaluated. Results are shown here for bandwidth, delay and fairness as representative for range, quantitative and boolean parameters respectively. For different network sizes, $10-10K$, number of hops is used as the independent variable. The average time for evaluating queries with source-destination pairs are calculated. Figures 2, 4 and 3 show the average time versus number of hops. The figures are plotted with time scale of $milli$ seconds. Standard deviation is also depicted. The overall trend shows that longer paths will take more time to be evaluated. On the other hand, the maximum time for evaluation never exceeded 0.6 msec, even for large networks. The standard deviation trend seems to be consistent, with some extreme jumps. As the number of hops on the path increases, the effect of small variations between individual nodes becomes negligible with respect to the average. For short paths, each node contributes more to the final evaluation and time complexity.

Figure 5 compares the time needed to evaluate queries involving different parameter types. Again, the same three parameters are compared (bandwidth, delay and fairness). The graph comparing individual parameters supports the summary of table III, where quantitative parameters suffer the worst time performance. This is attributed to the numerical value comparisons to evaluate minimum and maximum.

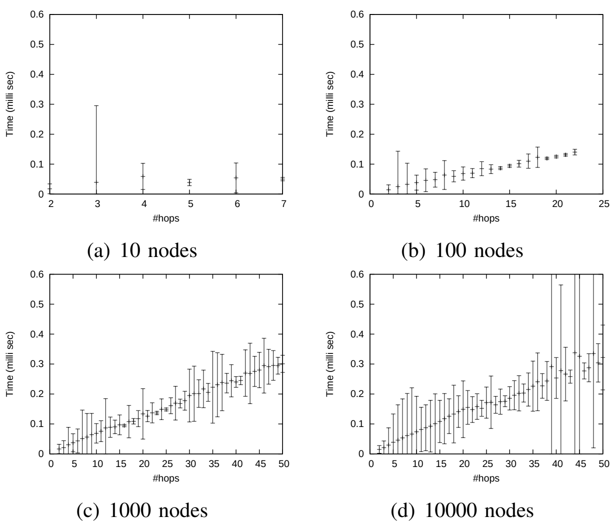
2) Time and space complexity: Evaluating the space and time complexity of the QoS model requires studying both types of devices; boundary routers (markers) and DiffServ routers (PHB enforcers/shapers). Building the transitions emerging from QoS states (*i.e.*, where QoS policy and parameters are being encoded) is the most expensive step in the



(a) 10 nodes (b) 100 nodes

(c) 1000 nodes (d) 10000 nodes

Fig. 3. Delay query average time versus number of hops



(a) 10 nodes (b) 100 nodes

(c) 1000 nodes (d) 10000 nodes

Fig. 4. Fairness query average time versus number of hops

overall modeling and verification framework. While the logic in DiffServ routers is more complicated, the policy size at each node is bounded by the different markers (*i.e.*, different ToS, CoS or DSCP values) that can be assigned. Therefore, worst case time/space overhead can be guaranteed in terms of the number of DiffServ nodes in the system. On the other hand, boundary routes can have complex and lengthy policies to assign marking values to different flows (which depends on header field combinations). Therefore, the focus will be on the space and time requirements to encode boundary routers.

In figure 6, the average time required to encode marking policies of a specific size is shown. It is clear that the time trend is linear in nature, which can be understood by studying the logic by which the transition expression is built. For every rule, a local expression is constructed, and then, the overlap with previous rules is excluded by Boolean manipulation, and so on with the rest of the policy. On the other hand, the space requirement is shown to plateau after a certain level. The reason behind this is the fact that BDD can share parts of expressions that are common to more than one expression. For example, if two CoS values have almost the same PHB,

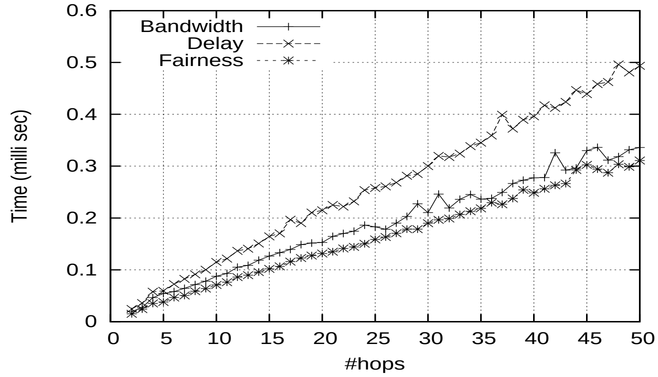


Fig. 5. Average time for evaluating queries versus the number of hops between source and destination

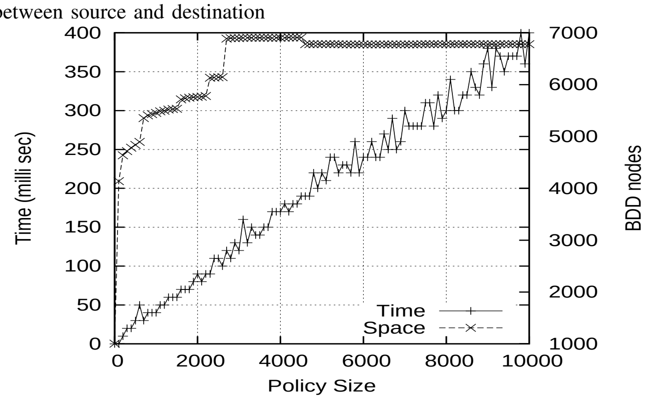


Fig. 6. Initialization time and space complexity of boundary routers (markers)

their expressions will be merged in memory saving valuable space. The plateau level depends on the number of bits defined in each field, and the number of PHB parameters used by the implemented standard. The size is shown in terms of BDD nodes (20 bytes each).

VI. Related Work

End-to-end network policy modeling has been investigated in different contexts, considering different configuration levels. In [7], the analysis is performed in the context of class of service configurations, particularly VPNs. The analysis focused on modeling QoS policies to detect class of service conflicts. The focus was mainly on maintaining the same marking (*i.e.*, class) for a certain traffic class on the end-to-end path from source to destination. The work did not address the actual configuration that each traffic class will face after properly being marked with CoS. Our configuration model and the general property verification presented can be considered a super set of the MPLS VPN configuration analysis model.

Using binary decision diagrams (BDDs),[1], [2], to model policies has attracted many researchers, especially for modeling security policies. The main focus was on discovering policy anomalies and rule conflicts. Firewall policy anomalies were presented in [8], [9] where a classification of rules conflicts is presented. The analysis was further extended to include multiple firewalls and IPSec policies [10], [11]. In [12], BDDs were also used to model distributed firewalls and discover policy anomalies, with improved complexity.

Other frameworks have been also proposed that used different policy mappings. In [13], SAT solvers were used to model firewalls. Configuration conflicts are mapped to model properties that can be checked by finding a satisfying assignment. The main advantage of using BDD over SAT solvers is the canonical representations of the policy expressions, resulting in efficient comparison and aggregation.

General policy models that incorporate security and routing information were addressed in [14], [3]. Both used formal modeling to check network reachability and general configuration queries. Conflicts can be formed as queries presented to the system. Both models use static analysis, and did not consider QoS parameters.

Most work on QoS policy analysis focused on conflict detection. A general classification of conflicts in policy-based distributed system management was introduced in [15]. This general taxonomy has been used in different domains to help classify conflicts depending on the functionality and parameters. Static conflict analysis in [16] was presented and addressed domain specific conflicts. Dynamic conflicts analysis, addressed in [17], handles conflicts arising from current state of the system. Those conflicts occur due to either inconsistencies between statically defined policies and system state, or interactions between different modules in the system. This part corresponds to our fuzzy analysis of conflicts. The performance evaluation of both static and dynamic models was discussed in the extended work [18]. In both approaches, event calculus is used to represent policies. Global configuration verification has not been addressed.

VII. Conclusion

In this work, a model is presented for the analysis and verification of QoS policy configurations. The proposed model is capable of incorporating flow QoS markers as well as the actual PHB parameters used. Using model checking and temporal logic queries (via CTL), it is shown that different types of queries can be performed to help administrators identify quality related problems as well as SLA violations. The system also provides an efficient way to investigate what-if scenarios without actual deployment or dry-runs of the modified configurations.

The model is evaluated using a wide range of QoS configurations, and network topology characteristics. The complexity of building the model using BDD implementation of QoS routers was analyzed. The system initialization time never exceeded 0.4 seconds per device for policy sizes up to $10K$ rules with memory requirements less than $140Kbyte$ for the largest policy. Evaluating individual QoS queries was also performed for source-destination pairs of large networks. For any source-destination pair, evaluating the path quality parameters was performed in less than $0.65ms$ in all cases.

A complete tool that can be deployed in a production environment is not far from reach, but it still requires a few components for commercial deployment. A module for processing QoS policies in vendor-specific syntax is needed. Designing a user interface that enables users not familiar with CTL queries to use the system can be crucial for wider deployment. From the model design side, it is possible to add different variants of aggregation mechanisms for specific PHB parameters to better imitate actual working environments. Also, incorporating live feedback from network monitors to modify available capacities and link status can be extremely valuable for dynamic environments.

Acknowledgment

Thanks to Wilfredo Marrero of DePaul University and Ehab Al-Shaer of UNCC for their invaluable discussions and feedback on early drafts of the paper.
This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract DEAC02- 05CH11231.

References

[1] R. E. Bryant, “Symbolic boolean manipulation with ordered binary-decision diagrams,” *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

[2] J. Lind-Nielsen, “Buddy - a binary decision diagram package,” http://buddy.sourceforge.net/, 2004.

[3] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. Elbadawi, “Network configuration in a box: Towards end-to-end verification of network reachability and security,” in *17th annual IEEE International Conference on Network Protocols, ICNP*, 2009, pp. 123–132.

[4] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore, “Policy QoS Information Model,” RFC 3644 (Proposed Standard), Nov. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3644.txt

[5] C. Wang, G. D. Hachtel, and F. Somenzi, *Abstraction Refinement for Large Scale Model Checking*, ser. Series on Integrated Circuits and Systems. Springer US, Sep 2006, ch. 2: Symbolic Model Checking.

[6] T. Samak, E. Al-Shaer, and H. Li, “Qos policy modeling and conflict analysis,” *Policies for Distributed Systems and Networks, IEEE International Workshop on*, vol. 0, pp. 19–26, 2008.

[7] Y.-W. E. Sung, C. Lund, M. Lyn, S. G. Rao, and S. Sen, “Modeling and understanding end-to-end class of service policies in operational networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 219–230, 2009.

[8] E. S. Al-Shaer and H. H. Hamed, “Firewall policy advisor for anomaly discovery and rule editing,” in *Integrated Network Management, IM*, 2003, pp. 17–30.

[9] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” *IEEE Journal on Selected Areas in Comm.*, vol. 23, no. 10, pp. 2069–2084, 2005.

[10] E. S. Al-Shaer and H. H. Hamed, “Discovery of policy anomalies in distributed firewalls.” in *The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, 2004.

[11] H. Hamed, E. Al-Shaer, and W. Marrero, “Modeling and verification of ipsec and vpn security policies,” in *13TH IEEE International Conference on Network Protocols (ICNP'05)*, 2005, pp. 259–278.

[12] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, “Fireman: A toolkit for firewall modeling and analysis.” in *2006 IEEE Symposium on Security and Privacy (S&P 2006)*, 2006, pp. 199–213.

[13] A. Jeffrey and T. Samak, “Model checking firewall policy configurations,” in *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY'09)*, 2009.

[14] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. G. Greenberg, G. Hjálmtýsson, and J. Rexford, “On static reachability analysis of IP networks,” in *Proc. IEEE Computer and Communications Societies*, 2005, pp. 2170–2183.

[15] E. C. Lupu and M. Sloman, “Conflicts in policy-based distributed systems management,” *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, November/December 1999.

[16] M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, and J. Rubio-Loyola, “Policy Conflict Analysis for Quality of Service Management,” in *6th IEEE Workshop on Policies for Distributed Systems and Networks (Policy 2005)*, 2005.

[17] M. Charalambides, P. Flegkas, G. Pavlou, A. Bandara, N. Dulay, E. Lupu, J. Rubio-Loyola, A. Russo, and M. Sloman, “Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management,” in *IFIP/IEEE Network Operations and Management Symposium (NOMS 2006)*, April 2006.

[18] M. Charalambides, G. Pavlou, P. Flegkas, J. Loyola, A. Bandara, E. Lupu, A. Russo, N. Dulay, and M. Sloman, “Policy conflict analysis for diffserv quality of service management,” *IEEE Transactions on Network and Service Management (TNSM)*, vol. 6, no. 1, March 2009.