**ADAPTIVE MEASUREMENT-BASED QoS MANAGEMENT IN CONVERGED NETWORKS**

S. Y. Yerima, G.P. Parr, S. McCLean, P. J. Morrow

School of Computing and Information Engineering, University of Ulster, Northern Ireland, UK
{s.yerima, gp.parr, si.mcclean, pj.morrow}@ulster.ac.uk

**ABSTRACT**

Fixed and wireless networks are increasingly converging towards common connectivity with IP-based core networks. Providing effective end-to-end QoS management in such complex heterogeneous converged network scenarios requires unified, adaptive and scalable solutions that integrate and coordinate diverse QoS mechanisms of different access technologies with IP-based QoS. Policy-based management is one approach that could be employed to address this challenge. Hence, a policy-based framework for end-to-end QoS management in converged networks, CNQF (Converged Networks QoS Management Framework) has been designed within our project. This paper explores the application of the framework for adaptive measurement-based QoS control driven by high-level policy decisions. Based on CNQF architecture, a prototype has been developed in Java and evaluation tests undertaken on our testbed demonstrate the viability of the measurement-based CNQF QoS management approach.

**KEYWORDS**

QoS, Policy-based network management, converged networks, resource management

**1. INTRODUCTION**

Next generation networks are increasingly being characterized by the convergence of heterogeneous fixed and wireless access network technologies towards interconnectivity with IP-based core networks. The management and control of network resources in such converged domains so as to provide the QoS that will in turn deliver satisfactory end user QoE poses significant technical challenges, given the complexity introduced by the integration of diverse QoS control mechanisms that may be deployed within each access technology on the end-to-end transport path. A promising network management approach that can be employed to overcome the challenges is the Policy-Based Network Management (PBNM) paradigm. PBNM allows for configuration and control of the network as a whole thus eliminating the need to configure and manage each network entity individually. PBNM can simplify administration of complex operational characteristics of a network, including QoS, access control, network security, and IP address allocation [1]. PBNM systems generally use hierarchical structures and will facilitate the management of next generation networks [2]. Moreover, PBNM provides the means for application transparency across existing and emerging access technologies which permit applications to be transport layer-agnostic when deployed [3]. The aforementioned pros of the PBNM approach and its potential to facilitate QoS management in converged next generation networks motivated its adoption as the underlying design paradigm for our proposed QoS management framework, CNQF.

CNQF (Converged Networks QoS Management Framework) is a PBNM framework designed to provide homogeneous, unified, end-to-end QoS management over heterogeneous access technologies, together with scalable, context-aware adaptive QoS control. Our previous work presented the proposed CNQF architecture, detailing the subsystems, elements, and their interaction whilst also exemplifying some use case scenarios for context-driven QoS control [4].
In this paper we focus on adaptive measurement-based QoS management in converged networks, based on the CNQF framework architecture. Within our project, a CNQF prototype has been developed in Java and is evaluated using a Linux-based experimental testbed. The paper discusses the implementation of the CNQF elements and explores CNQF-driven measurement-based adaptive QoS management using real traffic flows on the testbed.

The rest of the paper is organised as follows. Section 2 reviews related literature. Section 3 summarises the CNQF framework architecture and description of its constituent entities. Section 4 describes the CNQF prototype implementation and testbed configuration. Section 5 discusses adaptive QoS management strategy with CNQF while experiments and validation tests are presented in section 6. Concluding remarks are given in section 7.

2. RELATED WORK

Standards bodies such as 3GPP, ETSI’s TISPAN and IETF have defined a number of policy-based control architectures which can be found in [5], [6] and [7] for example. Although the standards’ policy frameworks define function blocks, interfaces and protocols to facilitate interoperability of the standards-compliant products from different vendors, no details are provided on how the various functionalities would be implemented. Our work aims to bridge this gap by not only leveraging the aligned policy-based models to develop scalable solutions for QoS management and control of converged networks but also introduces important novel extensions particularly context-aware functionality support, in order to provide value-added intelligence to the PBNM system.

Policy based management have been studied in different contexts such as tactical networks [8], virtualization environments [9], MPLS Networks [10], Virtual Private Networks [11], IMS-enabled networks [12], etc. Unlike most of the existing research, our work is aimed at policy based management support for unified end-to-end QoS control in IP based converged networks with heterogeneous fixed and wireless access domains. The incorporation of context management functionality as a building block of the overall architecture is also a distinguishing design feature of CNQF which makes our proposal different from the aforementioned works.

3. CNQF ARCHITECTURE

In [1], a distinction is made between QoS mechanisms which support service differentiation (that determine how to achieve performance objectives for heterogeneous flows) and QoS management and control (concerned with who should be entitled to preferential treatment from the network). CNQF mainly addresses the latter (QoS management/control) through its framework architecture which can be adapted to support heterogeneous underlying QoS mechanisms (such as DiffServ, MPLS, WiMAX QoS, UMTS QoS etc.) that may be deployed in the access and core segments of the transport layer. From the operational point of view, CNQF comprises of distributed Policy Decision Points (PDPs) which handle various management and control decisions driven by high-level declarative policies and enforced at policy enforcement points (PEPs) such as routers, switches, gateways, etc. in the transport plane of converged networks. Three logical subsystems make up the CNQF framework: Resource Management Subsystems (RMS), Measurement and Monitoring Subsystem (MMS) and Context Management and Adaptation Subsystems (CAS). As shown in Fig. 1, CNQF has distributed functional entities whose instances coordinate the resources of the transport network to enable closed-loop, scalable, end-to-end QoS control and resource management in converged networks.

3.1. Resource Management Subsystem (RMS)

The two key elements that make up the RMS are the Resource Brokers (RBs) and the Resource Controllers (RCs). The distributed instances of RBs which could be Wireless Access (WARB), Fixed Access (FARB) or Core Network (CNRB), perform policy-based resource management
roles depending on the segment of the converged transport plane the RB oversees (see Fig. 1). The RBs act as PDPs and communicate with one or more RCs which are located at various PEPs (i.e. routers and gateways). The policy decisions in the RBs invoke policy actions which translate to a set of configuration and control functions undertaken by the RCs.

![Diagram of CNQF operational entities in a converged network](image)

**Figure 1. CNQF operational entities in a converged network [4].**

### 3.2. Measurement and Monitoring Subsystem

The MMS is key to providing closed-loop, adaptive, measurement-based control within CNQF architecture. MMS consists of distributed Network Monitors (NMs) interfacing with the policy enforcement points and providing active and passive measurement capabilities. Each NM is designed to collect and process measurement data locally and can then be invoked via a central measurement and monitoring element (CMM), to provide averages/summaries or asynchronous instantaneous measurement data. CMM could provide high level summaries that may be used to gauge network-wide health status through GUIs in a centralized management station.

### 3.3. Context Management and Adaptation Subsystem

As shown in Fig. 1, CNQF provides context-aware functionality through its Context Management and Adaptation Subsystem (CAS). CAS consists of distributed Context Acquisition Function blocks (CAF) instantiated in each access network. The CAFs are PDPs that execute context-aware or context-driven policies within the CNQF system. Each CAF elements has associated Adaptation Servers (ADs) which are function blocks that configure/reconfigure PEPs directly affected by context-driven policy decisions in the CAF.

### 4. CNQF Prototype and Testbed Implementation

Based on the proposed CNQF architecture, a working prototype has been implemented in Java. Presently, distributed entities in the RMS and MMS subsystems which provide capability for adaptive measurement-based QoS management based on CNQF architecture have been built in the netbeans environment. Furthermore, a testbed for evaluating the CNQF prototype has been implemented using Linux machines. Brief descriptions are outlined in this section.

#### 4.1. CNQF prototype development

#### 4.1.1 RMS implementation
A resource broker (RB) has been built where resource management decision logic are implemented. High level policies entered into the GUI policy editor are mapped into a set of commands within the decision logic in the RB (the logic implementation will be different for CN-, WA-, and FARBs). These commands are Java code segments that invoke the services of instances of other CNQF function blocks/entities (such as network monitors and resource controllers) that are installed and running at the PEPs in the network. For instance, a high level CNQF policy that has an action part: Configure Edge Router will be mapped to the following RB Java code:

```java
new ResCon
ResCon.ConfigureEdge()
```

This creates an instance of the Resource Controller class and calls its ConfigureEdge() method that provides edge router configuration services for the RB policies. The RC instances are installed on the PEPs (Linux-based routers on our testbed). The installed RC opens an RMI port (Remote Method Invocation, TCP port 1099) to listen for incoming RB commands.

### 4.1.2 MMS implementation

In order to enable closed-loop QoS control within CNQF prototype, the MMS network monitoring element NM described in section 3.2 has been implemented as a netmon class. The main feature of netmon class is the provision of functions to enable SNMP based measurements using available Management Information Blocks (MIBs). Thus CNQF can create and install instances of netmon at various PEPs and communicate with SNMP agents at the PEPs via the netmon instance. Two configurations are possible with the current implementation. A centralised netmon instance can be used to poll measurements throughout the network, using the IP address of the interface on the routers (CMM mode). Alternatively, each router can run a netmon instance which can pre-process and send measurement data on demand to other CNQF entities thereby minimizing the amount of control/management traffic traversing the network.

#### 4.1.3 Implementing bandwidth monitoring and bandwidth utilization monitoring

In order to monitor attributes such as bandwidth on a router interface, the netmon instance is supplied with the IP address and MIB OIDs (Object Id) representing the required attributes as well as the SNMP community string. MIB OIDs employed within the MMS netmon class for bandwidth monitoring are obtained from IETF RFC 1213. From the MIB attributes, netmon calculates the bandwidth using:

\[
BW (\text{bits/s}) = \frac{(O(t) - O(t-\Delta t)) * 8}{\Delta t}
\]

Where \(\Delta t\) is the interval between two SNMP get operations which are used to read the MIB values \(O(t)\) which is basically a counter indicating the number of octets sent (ifOutOctets) or received (ifInOctets) on the network interface. Since the MIB variables are stored as counters, two poll cycles are taken by the netmon instance and the difference is calculated to get the bandwidth. Utilization is calculated using:

\[
BWU (%) = \frac{(O(t) - O(t-\Delta t)) * 8 * 100}{\Delta t * ifSpeed}
\]

Within netmon class, the average bandwidth is also tracked using the exponentially weighted moving average:

\[
BW(t) = (1-\alpha) * BW (t-\Delta t) + \alpha * BW(t)
\]

These are employed by resource management policies that involve adaptive measurement-based configurations driven by state changes that are enabled by decision thresholds in the RBs.

### 4.2. Testbed Implementation

The testbed used for development of CNQF and evaluation of its functionalities is implemented using Linux systems. The configuration is shown in Fig. 2. The testbed consists of two Linux-
based edge routers and a Linux-based core router. These elements constitute the PEPs each having an instance of CNQF RC (ResCon) that interacts with the Linux router kernel to set various parameters that enable configuration/reconfiguration of QoS management strategies stipulated in the high-level policies processed by the RB. The Linux TC (traffic control) utility in the kernel provides commands for implementing packet marking, classification, queuing disciplines, and policing of flows (enabling transport layer QoS mechanisms). In the testbed we developed, the RCs employ TC commands for low level configuration which have equivalent mappings to RB Java code that implement the high level policy actions. The testbed elements include a CNQF management station which houses central CNQF management application with the GUI policy editing tool and RMS CNRB implemented in Java which invokes policy actions via remote RCs (ResCon instances) installed at the PEPs (routers). The ntools traffic generator is used to generate multi-client traffic with different flow characteristics including constant bit rate (CBR), On-Off traffic, and variable bit rate (VBR) traffic.

![CNQF development and evaluation testbed](image)

Figure 2. CNQF development and evaluation testbed.

5. MEASUREMENT-BASED ADAPTIVE QoS MANAGEMENT

The elements within the CNQF subsystems described in section 3 interoperate under the supervision of a central policy-based CNQF management application to co-ordinate the transport layer resources. The distributed entities exchange messages in response to events that trigger policy decisions which facilitate adaptive QoS management strategies. The network monitors allows for a closed-loop system enabling adaptive QoS configuration policies (and other functionalities such as measurement-based admission control) to be implemented. As static QoS configurations may not always meet service performance requirements in converged networks due to highly dynamic network conditions, adaptive measurement-based control supported by the CNQF architecture can provide efficient network resource utilization and maintain service flow QoS requirements.

5.1 Initial configuration policies

For a QoS domain under CNQF administration, the initial configuration policies are entered into the GUI policy tool. These policies govern the configuration of the PEPs (routers) in the transport network with specific QoS provisioning capabilities. Current CNQF implementation supports DiffServ QoS mechanisms; initial/adaptive runtime management policies that utilize DiffServ QoS mechanisms can therefore be specified by an administrator. Extension of CNQF prototype to support others QoS technologies such as MPLS is ongoing. Hence, the GUI allows the DiffServ Code Points (DSCP) codes intended for marking packets for service differentiation (in the ingress edge router) to be specified along with the queuing discipline for implementing the intended per-hop-behaviours (PHB) for the QoS domain. Once these are entered by the
admin, the relevant RB instance is called to process the information and will in turn invoke the relevant ResCon instances to configure the routers with the selected configuration parameters. Once the PEPs have been configured, the ResCon instance returns a status message which is visible in the GUI’s status pane. Next, the policies relating to how specific flows, flow types or group of flows identified by identifier are to be handled (initial static QoS provisioning) are entered in the GUI. The format used is:

\textit{If (flow==identifier) then (policy action)}

Where the identifier can be a flow’s IP address or a range of IP addresses (for group of flows) or other unique flow identifiers. This will normally be used for configuring flows for specific preferential QoS treatment (e.g. for Expedited Forwarding PHB in DiffServ domain). If not configured, the default best effort treatment is applied to a flow. Each of these policy rules adds an entry to a look-up table specifying which flows have which DSCP and therefore a corresponding PHB. Through the use of the look-up table each flow’s current QoS configuration state is maintained. The RC employs the look-up table to add Linux TC filter parameters to a script which the kernel employs to configure the router to implement the required DiffServ parameters.

5.2 Runtime adaptive policies

CNQF adaptive policies are those that are triggered by time-varying/temporal events in the network. With the run time adaptive policies, the RB can invoke reconfigurations of QoS management via the ResCon instance, but will require monitoring reports which are provided by the netmon instance running on the PEP (routers). In a particular test scenario, when bandwidth utilization on a link is high (indicative of congestion situation), the PEPs are reconfigured via a runtime policy to mark a specific flow with EF DSCP (0x2e). This run time policy is specified in the GUI policy tool as follows:

- IF \textit{bandwidth utilization} ==high THEN (mark flow_id with DSCP=0x2e).

The runtime policies for CNQF measurement-based adaptive QoS configuration are specified in the following format:

- IF \textit{temporal	extunderscore event} == state \textit{action}

Where the \textit{temporal	extunderscore event} is a monitored event which may be mapped to specific \textit{netmon} measurement ranges; while predefined RB-compliant commands invoke specific set of procedures in the ReSCon instances to implement the required policy \textit{action} when the condition \textit{state} is fulfilled. At the same time the look-up table is updated with the changed flow’s QoS specification. A \textit{temporal	extunderscore event} such as bandwidth utilization is monitored by the \textit{netmon} instance in the PEP and the state is reported back to the RB via RMI protocol when a trigger threshold is exceeded. RB can also poll the \textit{netmon} instance using the \textit{netmon.getEWMA()} RMI (Remote Method Invocation) call to determine whether the policy condition is satisfied upon which the policy action (reconfiguration) commands are sent to the ResCon. ReSCon then returns the status of the configuration action as success or failure which can be seen in the GUI status pane.

6. Validation Tests and Experiments

In order to validate the CNQF prototype for adaptive QoS management we performed some tests on the Linux-based testbed where the prototype was deployed. High level policies were entered into the policy editing tool and the effects on the underlying network traffic were observed. The behaviour of selected traffic flows under observation indicated whether the adaptive measurement-based policies were successfully applied or not. First, some initial configuration policies (static) were applied via the GUI policy tool. This set of policies was intended to configure the network with DiffServ capabilities. Next, another set of policies for runtime configuration to govern QoS control of selected flows in response to network measurements were entered. These allowed us to validate the CNQF prototype capability for
DiffServ configuration as well as adaptive measurement-based QoS management capability of the architecture.

6.1. Initial configuration validation

In this test, the DiffServ configuration parameters were entered via the CNQF policy editing tool and when the RB was invoked it successfully processed the request and configured the edge routers and the core router successfully. These were indicated by the ‘success’ status message returned by the remote ResCon instances running on routers A, B and C (see Fig. 2), which were displayed on the GUI’s status pane. Note that this configuration needs to be successful for any runtime adaptive QoS management policies to work.

6.2. Runtime adaptive QoS control validation

In order to validate this aspect, a policy which stipulated that the QoS parameters in the network elements should be reconfigured to provide EF DiffServ treatment to a given CBR test flow was entered into the CNQF GUI tool. The condition given was that bandwidth utilization state of 80% on the ingress edge router outbound interface should be exceeded (i.e. high traffic load). Without an adaptive CNQF policy, increasing traffic load from other flows will starve the test flow of resources leading to rapid QoS degradation. To alleviate this, the policy allows preferential QoS treatment to be applied to the test flow at all nodes in the DiffServ domain so that its QoS requirements can be maintained.

Fig. 3 illustrates increasing traffic load generated from the ntools traffic generator which injects new flows into the DiffServ domain with Poisson inter-arrival time intervals and exponentially distributed average lifetime of 120s. As seen from the corresponding measurement in Fig. 4 (a), a 1Mbit/s CBR test flow traversing the same DiffServ domain is initially unaffected when the traffic entering the DiffServ network is low. As the total traffic increases and reaches the point where the load causes the stipulated 80% bandwidth utilization threshold to be exceeded, the adaptive policy kicks in allowing the test flow’s packets to be marked with DSCP of 0x2e and treated with EF PHB thus restoring the its QoS through priority queuing and scheduling. It can be seen from figure 4 (a) that the reconfiguration takes place at time 70s because the packet measurements filtered on DSCP == 0x2e (EF DSCP) in the observation tool suddenly shoots up and coincides with the test flow measurement (filtered on the test flow IP address).

The same experiment is undertaken without applying an adaptive QoS management policy for the test flow. The same ntools traffic generation of background flows shown in Fig. 3 was also applied. It can be seen in Fig. 4 (b) that the bit rate degrades considerably from around 70s due to the absence of adaptive policy to mitigate this phenomenon as the total traffic load increases.

Figure 3. Increasing volume of traffic flow entering the tetsbed’s DiffServ network.
Figure 4. (a) Effect of adaptive QoS reconfiguration on test flow. (b) Effect of increasing traffic load on test flow (without QoS reconfiguration).

7. **CONCLUSION AND FUTURE WORK**

This paper described a proposed PBNM system for end-to-end QoS control in converged networks (Converged Networks QoS Management Framework) CNQF. The implementation of a CNQF prototype is presented and the feasibility of adaptive measurement-based QoS management based on the architecture is explored using the prototype deployment on a Linux testbed. Test results showed that the CNQF approach was effective in implementing adaptive measurement-based QoS management policies. In future work, elements for context management subsystem will be built into the prototype for experimenting with context-aware policies to drive QoS control strategies encompassing wireless/fixed access and core networks in a larger scale converged networks scenario.

**REFERENCES**


