Flexible QoS Provisioning for SIP Telephony over DVB-RCS Satellite Networks

Teemu Kärkkäinen and Jörg Ott
Helsinki University of Technology, Networking Laboratory
{teemuk|jo}@netlab.tkk.fi

***Abstract*— DVB-RCS satellite networks pose a challenging environment for IP telephony due to the link conditions of long delays and low bandwidth as well as the diversity of network scenarios. We have designed an architecture for providing flexible QoS provisioning under these conditions. The architecture addresses flow detection, admission control and user notification in this heterogeneous environment where it is not always possible to take active part in call signaling and endpoint cooperation cannot be relied upon. We have developed and tested a prototype implementation of parts of the architecture which indicated that scalability will not be limited by processing speed or memory usage up to the data rates encountered in satellite links.**

I. Introduction

Prioritization of interactive, real-time traffic flows over other types of traffic is often crucial for ensuring the flawless operation of IP telephony (voice over IP, VoIP) applications. Without prioritization, other traffic competing for the same transmission resources may lead to increasing queuing delays and packet losses for UDP-based real-time flows. This is especially important in networks with challenging links conditions, such as long delays and low bandwidth. DVB-RCS is one concrete link layer technology that exhibits such conditions.

DVB-RCS is an ETSI standard for providing a return channel via satellite, typically used in conjunction with DVB-S forward channels. It defines PHY and MAC layer mechanisms including capacity requests and resource allocation. The mapping of application layer QoS requirements onto the PHY and MAC layer mechanisms is still an open research problem. Higher layer issues of the DVB-RCS architecture are handled by an independent, non-profit organization SatLabs. SatLabs has addressed the issue of higher layer QoS requirements by developing a mapping strategy between DiffServ Per-Hop Behaviors (PHB) and lower layer capacity request and resource allocation mechanisms [1], [2]. This mapping serves as a basis for the architecture presented in this paper, allowing a design based on the standard DiffServ mechanisms.

While numerous solutions exist for prioritizing IP packets in order to provide different levels of QoS (see section III), a complete architecture for implementing QoS over DVB-RCS links has not been proposed.

The IST FP 6 project VIVALDI addresses this gap still existing between the QoS requirements of VoIP applications and the lower layer DVB-RCS mechanisms by developing an architecture for providing QoS to (SIP-based) VoIP calls in DVB-RCS networks. This paper presents a high level architecture designed in the VIVALDI project in order to address the identification of critical flows (e.g., SIP and RTP), call admission control and user notification. Section II presents the target scenarios and the resulting requirements and section III selected related work on QoS. Section IV introduces the VIVALDI architecture, a prototype implementation of which is presented in section V. Section VI reviews our findings and hints at future work.

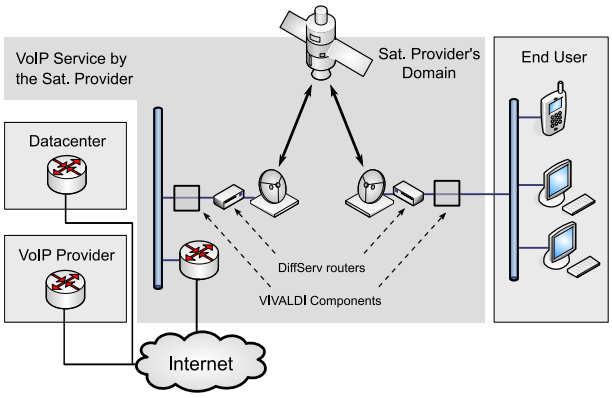

Fig. 1. Network layout

II. Scenarios and Requirements

A. Usage Scenarios

The basic usage scenarios can be subdivided into three cases depending on who provides the IP telephony service and where the VoIP server(s) are located as shown in Figure 1. Each scenario imposes different requirements on the architecture.

1) The satellite service provider is also the VoIP service provider and operator of the VoIP server. In this case, the VoIP service comes bundled with the satellite Internet service and the operator controls all the network elements.

2) The VoIP service is offered by a separate service provider that operates the server in their own network. An example of this case is a consumer who purchases satellite Internet access from a satellite service provider and VoIP service from a dedicated VoIP service provider.

3) The VoIP service is run by the end-user's organization and the VoIP servers are distributed to the end-user's remote locations or centralized in a data-center. A typical example of this case is a corporation that connects remote sites to the head office through satellite links.

In case 1, the VoIP service and the VIVALDI architecture are both operated by the same entity. This allows having elements of the VIVALDI architecture take an active part in the signaling which is a prerequisite for *deterministic* operation and allows the architecture's admission control process to have explicit control over the call setup procedure.

In cases 2 and 3, the VoIP service and the VIVALDI architecture are operated by separate entities so that it is not possible to define architectural elements involved in the call signaling. The satellite service provider is merely a bitpipe. As explicit control of the call setup procedure is not possible in these cases, nor can cooperation by the end-points be relied upon, a *probabilistic* approach is required.

B. Requirements

The core requirements of the VIVALDI architecture are derived from the need to support all the above scenarios:

1) Probabilistic detection of real-time flows. The probabilistic approach to flow detection attempts to associate packets with flows based only on the information gathered by inspecting the packets. As this approach does not rely on being able to take active part in the call signaling process it can be used even in scenarios 2 and 3. The probabilistic approach may attempt to detect signaling messages that precede the media flows (e.g., SIP INVITE messages if not encrypted) or to detect media flows directly (e.g., RTP).

2) Deterministic detection of real-time flows. Deterministic detection of flows is possible when the architecture takes active part in the call signaling process and therefore has explicit knowledge about the established calls. Scenario 1 is the only case where deterministic detection is feasible.

3) Call Admission Control (CAC). The DiffServ architecture only delivers higher QoS to traffic classes inside the DiffServ domain, not to individual flows. A CAC procedure is required to admit or deny the entry of individual flows into the traffic classes in such a way that the desired QoS level can be reached without negatively impacting the QoS level of previously admitted flows. Beyond this basic requirement, different scenarios might include further variables that affect the CAC procedure (e.g., pre-paid calling credits).

4) Indication of busy conditions. During congestion periods some of the flows cannot be admitted to the higher QoS classes. This appears to the end-users as lower than expected perceived quality. Under these conditions the VIVALDI architecture should be able to prevent the call from being set up (*deterministic* approach) or to signal the end-user that the lower than expected perceived quality is caused by congestion (*probabilistic* approach).

III. Related Work

We can distingush two different approaches for *explicitly* prioritizing selected packets in IP networks:[1] (1) With *integrated services* [3], signaling is used to reserve resources [4], [5] for a particular flow which is confirmed or rejected by the network. Routers identify IPv4 flows by the quintuple of source and destination address and port and the protocol id and then treat them according to their reservations. (2) With *differentiated services* [6], traffic flows are assigned to diffserv classes (typically but not necessarily by the source) and *marked* using the corresponding *DiffServ Code Point (DSCP)* in the TOS bits in the IP header. Routers handle packets according to their DSCPs: they may perform relative prioritization among the classes or limit the available resources per class and *unmark* packets exceeding service level agreements, treating them as best effort traffic. DiffServ does not give guarantees and does not provide explicit feedback to a source about the (un)availability of resources.

It is usually the endpoint's responsibility to obtain the necessary resources or determine which DSCPs to use. Both types of mechanisms can be integrated with SIP signaling: explicit resource reservations are supported by means of the QoS precondition framework [7], [8], whereas DSCPs can be signaled using recent SDP extensions [9].

Numerous architectures for QoS-enabled IP telephony have been designed based upon these basic building blocks, often for largely closed networks, such as the PacketCable DCS architecture [10], [11], 3GPP IMS [12], or TISPAN's use of IMS for the fixed network [13], where often the network and the IP telephony service provider are identical.[2] If not all parts of the network are under service provider control (e.g., when an ISP interfaces to a customer's private network), provider-controlled devices—referred to as *Session Border Controller (SBC)* [14]—are used for policy enforcement on packet forwarding, e.g., to improve or degrade QoS of flows.

In the above cases, it is either the endpoints' or the service providers' responsibility to determine the (un)availability of resources for a call and accept or reject the call—which is achievable as the relevant devices are under common control. In a heterogenous environment like VIVALDI, where different deployment scenarios may co-exist, unknown equipment may be used on the telephony service provider and user side, and even different telephony signaling protocols may be used, these properties do no longer hold.

IV. VIVALDI Architecture

Figure 2 shows the layout of the VIVALDI architecture designed to support the various usage scenarios and their combined requirements. The architecture is based upon dividing the required functionality into separate elements with clean interfaces and by adding elements supporting the overall operation. Furthermore, the performance-critical parts are separated from other functions. This is done to promote low coupling and high cohesion, which in turn allows for robust and extensible implementations.

Probabilistic flow detection (1) is fulfilled by the *flow modules*, deterministic flow detection (2) is done in the *Signaling Server*, CAC (3) is handled by the *Admission Control Unit* and busy condition signaling (4) is the responsibility of

[1] We deliberately exclude any detailed discussion of queuing and forwarding schemes part of any QoS-enabled router.

[2] Similar architectures may be found in closed corporate networks.

the *Pushback Signaling Unit*. Further elements were added to support centralized configuration (*Configuration Source*), and to simplify Flow Module management and the interfaces (*Module Controller*).

The layout describes logical boundaries for implementations. The elements can be implemented separately and even run on separate hardware connected via fast links. However, implementations do not have to strictly follow this separation and all the elements can be implemented in the same network element if required.

Instances of the architecture are deployed independently on both sides of the satellite link (see Figure 1) and process the data before it enters the satellite segment. Only the Flow Modules must lay directly on the path taken by the traffic entering the satellite segment. All other elements can be placed elsewhere as long as they are connected via low delay links.

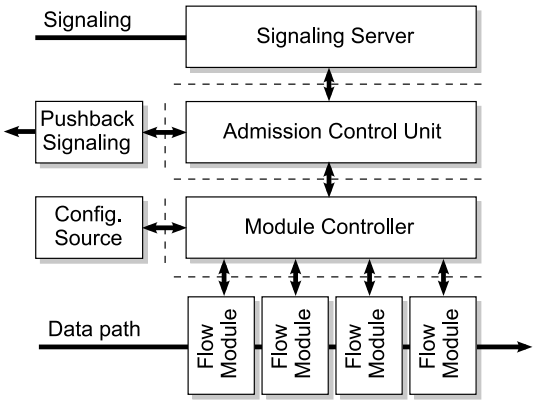

Fig. 2. Architecture overview

A. Flow Modules

Flow Modules are the lowest level functional elements. They are located in the data path and directly manipulate the IP packets. The Flow Modules behave as *detectors* and *actuators*. The *detector* functionality is implemented by an arbitrary heuristic capable of probabilistically detecting the beginning of new flows and the termination of existing flows (thus addressing requirement 1). The bandwidth usage of individual flows must also be continuously measured. The Flow Modules are also capable of acting as *actuators* by manipulating the DSCP values in the IP headers and optionally by dropping certain packets in order to perform simple traffic shaping (e.g., when a flow exceeds its bandwidth allocation), thereby supporting requirements 1 and 2. Flow Modules continuously act as *detectors* but only act as *actuators* when instructed to do so by the Module Controller.

Implementations of the Flow Modules may range from simple port-number-based detection to RTP header detection to SIP signaling interception and even to detection based on statistical models (e.g., when trying to detect encrypted media flows based on their statistical characteristics). We envision jointly operating specialized flow module implementations, each optimized for different detection tasks.

Flow Modules are the most performance critical elements of the architecture as they are required to process and manipulate all the IP packets at line speeds. This leads to a critical trade-off decision between detection accuracy and processing speed.

B. Module Controller

The Module Controller is responsible for all communications with the Flow Modules, for soliciting admission control decisions for new flows, and for storing flow states including addresses, UDP port numbers, responsible modules, and the decisions concerning the flows. All the Flow Modules communicate with the Module Controller via unicast IPC.

The primary function of the Module Controller is to control which flow module is the actuator for a given flow. It is possible that multiple Flow Modules of different type detect the same flow. Therefore, in order to avoid unpredictable behavior due to multiple Flow Modules interfering with each other, the Module Controller will instruct one of the flow modules to handle flow. This decision can be changed at any time in order to have a different Flow Module take over the responsibility for a flow (e.g., in case of Flow Module failures).

The decision logic for the choice of the responsible Flow Module in the case of probabilistically detected flows can be arbitrary, but is limited by two factors: 1) the Module Controller has no knowledge about the specific capabilities of any module (e.g. the protocol that the module detects) unless it is statically configured, and 2) the Module Controller's information about each flow is limited (IP addresses, UDP port numbers, and bandwidth usage). In the case of a deterministically detected flow, the Module Controller should be preconfigured to always select the same Flow Module with known capabilities.

C. Admission Control Unit

The Admission Control Unit (ACU) holds the logic required for making admission control decisions. In practice this means deciding which flows are admitted to higher QoS classes and which ones are given only basic best-effort service.

The main responsibility of the ACU is to ensure that the higher QoS classes are not saturated. This may be done by an arbitrary CAC algorithm that tracks the resource usage and the overall resource allocation, and decides to admit or deny new flows into the DiffServ QoS classes. The decisions are made when new flows are detected or when the characteristics (e.g., bandwidth usage) of existing flows change.

The information in the ACU (which flows have been given higher QoS at what point in time) can also be used for providing Call Detail Records for billing, efficiency measurements, quality monitoring, and similar purposes. And the ACU algorithm may be designed to support nomadic and prepaid services if desired.

D. Pushback Signaling Unit

The Pushback Signaling Unit (PSU) is an optional unit that can be used to provide feedback to the users about the current resource usage status and the admission control decisions in

order to fill the requirement 4. Since the information required by the PSU is contained in the ACU the two are connected directly. The ACU will inform the PSU of all its decisions (e.g., the admission of a new flow) as well as periodically announce the overall state of resource usage.

The range of possible PSU functions is large. For example, it could be used to control an indicator light or a small client application based on the current resource status, or in a more complex case, to implement a presence server that users can add to their VoIP client's user list or to which the VoIP server could subscribe to. However, pushback signaling is always limited by the fact that, unless specifically configured, the PSU has no knowledge which functionality is available on the endpoints.

E. Signaling Server

The Signaling Server in the VIVALDI architecture refers to a standard server for a given signaling protocol (such as SIP or H.323) with additional logic added for communicating with the ACU. It is an optional element that can be used when a high level of control over the service is desired (scenario 1). This is achieved by having the Signaling Server take active part in the call setup signaling, with the ability to explicitly get information about the flows (requirement 2). Furthermore, if the admission of the flows is rejected, the Signaling Server may explicitly signal this to the users and stop or delay the call setup process, thus also acting as a PSU (requirement 4).

F. Configuration Source

The Configuration Source is a central location for maintaining all the configuration information required by the Flow Modules. The configuration data contains mandatory information such as policy decisions (e.g., which DSCP is used for detected flows before a decision is available), as well as static, module specific configuration information such as the known protocol ports which a port based detection Flow Module should track. The configuration information will be pushed to the Flow Modules enabling automatic configuration of new Flow Modules.

G. Interactions between the Elements

The Flow Modules, the Module Controller and the Admission Control Unit all maintain state information about the active flows. The flow modules need the flow state in order to make quick decisions for each packet. The Module Controller needs to maintain flow information in order to track which Flow Modules are responsible for which flows, and to keep track of the decisions made by the Admission Control Unit. The Admission Control Unit needs flow state in order to track the resource usage and to make decisions about admitting or denying new flows. In the deterministic case, the signaling server will also hold state about the active flows.

In all inter-element interactions the flow states are abstracted to a 5-tuple of source and destination IP addresses and UDP ports and the measured bandwidth usage. UDP ports are assumed since real-time media transmission protocols typically use UDP. If support for other transport protocols is required, the flow state abstraction can be extended to include a protocol indicator (e.g., the protocol number). Furthermore, support for multiple flows multiplexed within a single transport layer connection can be added by including a flow identifier.

Although the state information in the different architectural elements is not identical (for example, the Admission Control Unit does not need the same amount of detail as the Flow Modules), the states should be held consistent between all the elements. Therefore, the state information should be modified at the same time in all the elements. However, only rough state consistency is required (in the order of seconds) which allows the frequency of inter-element signaling to be orders of magnitude lower than the packet frequency of the underlying media flows. This can be reduced further by aggregating multiple flow state information units into a single message.

Each architectural element that needs information about the flow states maintains a flow state table that contains all the relevant state information about the known flows. The entries in the flow state tables are soft-state; entries are removed from the table after certain time-period unless they are refreshed by an incoming message. We have developed a UDP-based protocol for each of the interfaces shown in fig. 2 to facilitate the exchange of state information.

The state refreshes (as well as the initial detection messages) are triggered from different sources depending on whether the flow is detected probabilistically or deterministically. In the case of probabilistically detected flows, the Flow Modules are responsible for the initiation of the refresh message; whereas in the case of deterministically detected flows the ACU is responsible for initiating the refreshes.

V. Implementation and Validation

We have developed a prototype implementation for validation purposes covering admission control, probabilistic flow detection, and centralized configuration; the deterministic approach is not included in the initial prototype but we are presently investigating using third-party SIP servers and a standardized admission interface. We implemented the prototype as three separate applications: Admission Control Unit, Module Controller with a Configuration Source, and a Flow Module.

We have tested the prototype implementation on a Linux-based desktop machine with multiple Ethernet interfaces and enough resources so that processing speed did not become a bottleneck. Test loads in the order of a hundred simultaneous calls were generated with a dedicated traffic generator.

The prototype implementation includes a Flow Module containing an RTP detection state machine. The RTP detection mechanism resulted in the first packet of a new RTP flow being missed, the second and third packets were detected but an admission control decision from the ACU was not available, and from the fourth packet onwards the flow was detected and an admission control decision from the ACU was available. This detection pattern is independent of the offered load, but does depend on the communication delay between the ACU,

Module Controller and the Flow Modules. In the typical case of 20 ms of voice data per IP packet, the results imply that the first 20 ms of a new call will not receive higher QoS, the following 40 ms will receive QoS based on a static policy decision and from 60 ms onwards the QoS will depend on the admission control decision. This is unlikely to lead to noticeable clipping and will also not have a severe impact on the jitter reporting by the receiver.

The additional processing done by the prototype increased the average jitter of the RTP flows by less than 1 ms and the average delay of all packets by no more than 0.1 ms. The initial testing also indicates that the architecture is reasonably scalable, with the prototype implementation capable of processing at least 16,000 packets per second (average total processing time per packet of 0.06 ms). Significantly shorter processing times are likely to be achievable through more optimized implementations. But even our prototype would be able to support more than 100 simultaneous flows at 50 packets/s (i.e., 20 ms intervals) at a total data rate of some 1–6 Mbit/s depending on the codec.[3]

The architecture appears also scalable in terms of the signaling traffic between the architectural elements. The total combined inter-element signaling traffic was 1 KB/s for 20 simultaneous calls, 1.7 KB/s for 40 simultaneous calls and thus would also scale to hundreds of calls. As the prototype was not highly optimized with respect to the inter-element signaling traffic these figures can also be lowered with more optimized implementations. We therefore foresee no problems in covering a satellite link.

The impact of the architecture on the media flow transmission characteristics such as delay, jitter and packet loss were deliberately left out of scope of the testing. These are entirely dependent on the scheduling and queuing algorithms used by the DiffServ enabled network elements which have been extensively studied before. The impact of various scheduling and queuing algorithms can be found, e.g., in [15], [16].

VI. Conclusion

This paper presented scenarios in the context of DVB-RCS satellite networks that motivate the design of a flexible QoS provisioning architecture for VoIP. The architecture presented allows for the required flexibility in the implementations needed to meet the requirements of the diverse scenarios and the heterogeneous environment where end-point cooperation cannot be relied upon.

Initial prototype testing results indicate that the architecture meets the requirements in the areas of probabilistic flow detection and admission control. Furthermore, the results indicate that the architecture can potentially be scaled beyond the relatively low bandwidth environment of DVB-RCS networks.

[3]Probes from two links of the FUNET backbone in May 2006 have shown that these figures are reasonable: on a 1 Gbit/s link (some 50% utilization) some 80,000 packets/s were observed at an average size of some 700 bytes, with a fraction of 2-5% being UDP traffic (less than than 7,000 UDP packets/s with a mean size of some 300 bytes). Our prototype can support such a traffic mix (some 16,000 packets) at satellite forward channel data rates of up to 100 Mbit/s both in terms of processing speed and memory requirements.

While the architecture was motivated by DVB-RCS satellite networks and SIP telephony in particular, the concepts developed can be applied in other scenarios (e.g., prioritization of traffic prior to entering a VPN) and for other types of real-time media flows.

Further work is required in the area of *pushback signaling* and interactions with VoIP *security mechanisms*. How the users are informed about link congestion and admission control decisions in the case of heterogeneous end-points is still an open problem. Furthermore, various security mechanisms, such as the use of VPN tunnels and encryption of signaling and media flows, hide much of the information that could be used for probabilistic flow detection. We will explore whether this problem can be overcome by using statistical traffic analysis for flow detection.

References

[1] Stephane Combes and Stephane Pirio, "SatLabs System Recommendations Part 2 - Quality of Service Specifications," November 2006.

[2] Harald Skinnemoen, Anca Vermesan, Augustin Iuoras, Geert Adams, and Xavier Lobao, "VoIP over DVB-RCS with QoS and Bandwidth on Demand," *IEEE Wireless Communications*, vol. 12, no. 5, pp. 46–53, October 2005.

[3] Robert Braden, David Clark, and Scott Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, June 1994.

[4] Robert Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin, "Resource ReSerVation Protocol (RSVP) – Functional Specification," RFC 2205, September 1997.

[5] Robert Hancock, Georgios Karagiannis, John Loughney, and Sven Van den Bosch, "Next Steps in Signaling (NSIS): Framework," RFC 4080, June 2005.

[6] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss, "An Architecture for Differentiated Service," RFC 2475, December 1998.

[7] Mark Handley, Van Jacobson, and Colin Perkins, "SDP: Session Description Protocol," RFC 4566, July 2006.

[8] James Polk, Subha Dhesikan, and Gonzalo Camarillo, "Configuring the Differentiated Services Codepoint of Session Description Protocol Established Media Streams," Internet Draft draft-polk-mmusic-qos-mechanism-identification-02.txt, Work in progress, October 2006.

[9] James Polk, "Configuring the Differentiated Services Codepoint of Session Description Protocol Established Media Streams," Internet Draft draft-polk-mmusic-dscp-attribute-00.txt, Work in progress, October 2006.

[10] Bill Marshall, Matt Osman, Flemming Andreasen, and Doc Evans, "Architectural Considerations for Providing Carrier Class Telephony Services Utilizing Session Initiation Protocol (SIP)-based Distributed Call Control Mechanisms," Internet Draft draft-dcsgroup-sipping-arch-01.txt, Work in progress, January 2003.

[11] PacketCable, "Distributed Call Signaling Specification," PKT-SP-DCS-D03-000428, available from ftp://ftp.cablelabs.com/pub/pkt-sp-dcs-d03-000428.pdf, 2000.

[12] 3GPP, "TS 23.228: IP Multimedia Subsystem (IMS); Stage 2 (Release 5)," 3GPP 23.228, September 2002.

[13] ETSI, "Website of the TISPAN project," http://www.etsi.org/tispan, 2006.

[14] Jani Hautakorpi, Gonzalo Camarillo, Robert F. Penfield, Alan Hawrylyshen, and Medhavi Bhatia, "Requirements from SIP (Session Initiation Protocol) Session Border Control Deployments," Internet Draft draft-camarillo-sipping-sbc-funcs-05.txt, Work in progress, October 2006.

[15] Zesong Di and H. T. Mouftah, "Performance Evaluation of Per-Hop Forwarding Behaviors in the Diffserv Internet," in *Proceedings of the Fifth IEEE Symposium on Computers and Communications*, 2000.

[16] Günther Stattenberger, Torsten Braun, Matthias Scheidegger, Marcus Brunner, and Heinrich J. Stüttgen, "Performance evaluation of a Linux DiffServ implementation," *Computer Communications*, vol. 25, no. 13, pp. 1195–1213, August 2002.