Enabling Low Latency and High Reliability for IMS–NFV

Muhammad Taqi Raza* and Songwu Lu
Computer Science Department, University of California – Los Angeles (UCLA)
Email: {taqi, slu}@cs.ucla.edu

***Abstract*—Network Functions Virtualization (NFV) allows service providers to deliver new services to their customers more quickly by adopting software centric network functions implementation over commercial, off-the-shelf hardwares. IP Multimedia Subsystem (IMS) which is one of the most complex NFV instances requires extremely low end-to-end latency (up to 40 msec), and demands system availability as high as five nines. We discover that highly modular 3GPP standardized IMS network functions implementation over virtualized platform (1) incurs latencies, and (2) does not tolerate faults. NFV-based IMS modules incur high latencies by creating a feedback loop among each other while executing delay sensitive data-plane traffic. These IMS modules are also susceptible to failures, causing the control-plane to terminate the application session while keeping the data-plane to forward data packets. To address these issues, we propose to refactor network function modules. We reduce latencies by pipelining the communication between IMS modules, and achieve fault tolerance by reconfiguring their neighboring modules. We build our system prototype of open source 3GPP compliant IMS over OpenStack platform. Our results show that our scheme reduces latencies and failure recovery time upto 12X and 10X, respectively, when compared to the stat-of-the-art 3GPP compliant virtualized IMS implementation.**

I. Introduction

To meet exponentially increasing service demands and to even launch a new network service, a service provider often requires installing a new dedicated appliance that brings complexity of integrating and deploying it in a network. Moreover, purpose-built appliances rapidly reach end of life because dedicated hardwares life cycles are becoming shorter as innovation accelerates, reducing the return on investment of deploying new services [1]. Network Functions Virtualization (NFV) addresses these problems by implementing network functions (NFs) in a software that can run on general-purpose hardware servers [2]. IMS that provides multimedia services (i.e. voice, interactive video streaming, and rich communication services, etc.) to LTE subscribers is the leading use case of carrier network virtualization [3] [4]. IMS-NFV has been prototyped by a number network operators and vendors. It has been reported that their NFV implementations incur high packet processing latencies and provide weak fault tolerance [5]. Their solutions to reduce latencies [6] and improve fault tolerance [5] are generic and do not incorporate IMS domain-specific knowledge.

In this paper, we refer to IMS specifications as standardized by 3GPP [7] and study how well standardized IMS procedures tolerate faults and reduce media-plane latencies over virtualized platforms. This helps to highlight domain-specific issues that all vendors' virtualized IMS (vIMS) implementation (being 3GPP compliant) may face.

First, we find that IMS NFs are highly modular where one NF has many functional modules. During media plane processing, different modules implemented in three different NFs interact with each other in a loop; where a module in one NF uses delegation model to delegate the processing of data packets to a module in different NF, while retaining the overall control to itself. Similarly, policies are defined at a module in one NF, but are enforced at a module in different NF. As a result, a loop is created between processor, controller and policy modules, all residing in different NFs. Some of these modules require realtime packet processing behavior to update control information and policies. For example, a controller module requires certain information, such as available bandwidth and packets arrival rate, from a processor module to adjust voice codec [8]. This real-time interaction between controller and processor modules result in packet processing latencies.

Second, our study reveals that during media session setup, both control and data planes create respective device session states and transition from one state to the other as a call progresses. Device states at control and data planes must remain synchronized during call life cycle. However, we discover that a particular module failure causes device states to be desynchronized; the control-plane terminates the call but data-plane keeps device state as *Connected*. The hanging state machine phenomenon emerges when control-plane detects the failure and changes device state to *Morgue* before terminating the control-plane connection; whereas, the failure goes undetected at data-plane that keeps receiving downlink data packets (whose control-plane connection is aborted).

To address these issues, we propose to refactor IMS NFs modules by (1) pipelining data packets processing and fetching its control instructions, and (2) reconfiguring modules to recover from a failure. To reduce media plane latencies, our design predicts future packets and prefetches control instructions. Once future packets arrive, these control instructions are used to steer media plane execution. For prediction, we use exponential smoothing model [9] that weighs past observations using exponentially decreasing weights, i.e. recent observations are given relatively more weight in forecasting than the older observations. Because control instructions remain same for a range of metadata values (that represent media behavior), our design fetches accurate future control instruction even in

*Corresponding and student author.

978-3-901882-98-2

the presence of small prediction errors.

To improve fault tolerance, our design provides configurations for each module during the system setup phase by adding the back-up paths to their one-hop neighboring module. At runtime, a neighboring module detects the module failure and assumes the role of failed module by loading its execution logic as provided in the configurations. This neighboring module then connects with rest of the failed module's neighbors through a back-up link and resumes failed operation.

We evaluate our design and gather results from our 3GPP compliant OpenIMS [10] implementation over Openstack [11]. Our results show that (1) our system reduces media latencies upto 12X, and (2) resumes failed operation within 2.5 seconds, which is 10X better than current state-of-the-art vIMS design.

II. IMS Background

LTE provides best effort service to the users, with no guarantee on the amount of bandwidth a user gets for a connection and the delay experienced by the packets. Therefore, network operators implement IP Multimedia Subsystem (IMS) to provide guaranteed real-time multimedia services in their LTE network. IMS is an architectural framework for delivering IP multimedia services, such as Voice over LTE (VoLTE), Video over LTE (ViLTE), software as a service (SaaS), social sites, navigation, and many more.

IMS Architecture: IMS operations are categorized into control-plane and data-plane operations, as shown in Fig. 1.

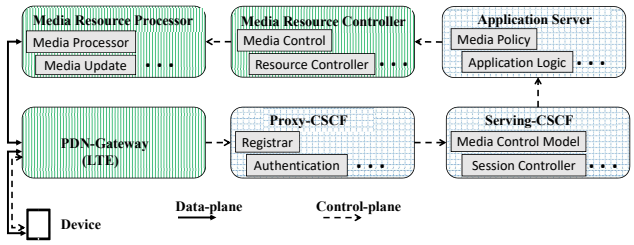


Figure 1: Standardized IMS architecture: An overview

Control-Plane supports media sessions control through Call Session Control Function (CSCF) NFs, and Application Server (AS) NF. The CSCF performs all the signaling operations, manages Session Initiation Protocol (SIP) sessions and coordinates with other NFs for session control, service control and resource allocation. It consists of two main NFs: the Proxy-CSCF (P-CSCF) and Serving-CSCF (S-CSCF). The AS, on the other hand, implements multimedia execution logic and policies, and coordinates with both CSCFs and data-plane NFs.

Data-Plane includes media-gateway NFs that process and store data, and generate services for the subscribers. Once user session has been established, the user data-plane traffic is sent to Media Resource Function Processor (MRFP). The MRFP connects LTE core domain (via PDN gateway - PGW) with IMS domain for multimedia service and converts between different transmission and coding techniques as controlled by Media Resource Function Controller (MRFC). Moreover, MRFC employs monitoring schemes to determine policy rules in real-time.

IMS NFs are highly modular where different modules handle different functionality such as execution logic, processing, policy, security, session states, resource control and more. Fig. 1 shows few modules (rectangular shaped) implemented within different NFs (rounded rectangular shaped).

III. Migrating Carrier Grade IMS to NFV

Network equipment vendors have spent decades understanding telecom requirements for the high availability, low latency and security of different types of network functions. They have crafted solutions to these functions' mission-critical demands into their network equipment, with both hardware and software elements within vendor appliances playing a part in meeting carrier-grade requirements. Examples include Ericsson's Blade Systems (EBS) [12] and Alcatel-Lucent's Element Management System (EMS) [13]. These systems continue to provide the required functioning despite occasional internal components and modules failures, either transient or permanent. Their software designs, such as Ericsson's ERLANG [12] and Alcatel-Lucent's NVP [14], ensure redundancy, both for error detection and error recovery.

However, recently, the concept of Network Function Virtualization (NFV) emerges that aims in replacing dedicated network appliances – such as network processing functions and gateways – with software running on commercial off-the-shelf servers. NFV has been keenly followed by the telecom industry and its proof of concept implementations are already in process [3] [15]. Most telecom operators are considering to support their IP Multimedia Subsystem (IMS) implementation over NFV [4] [16]. Operators can benefit from virtualized IMS (vIMS) implementation; where they can quickly scale up virtualized NFs (VNFs) to support growing demand of multimedia-rich applications and services (e.g. VoLTE and ViLTE) by using low cost commodity servers.

These multimedia services have stringent requirements on latency (end-to-end latency goes as low as 40 msec [17]) and fault tolerance (operator networks want to achieve 5 nines availability). Telecom vendors and operators have indeed faced a lot of challenges in reducing latencies and achieving fault tolerance. Startus Technologies warn that in a virtualized environment, transparent fault tolerance hasn't been possible, making full-blown NFV a risky proposition for telecom operators [5]. They provide Stratus solution in which an application virtually runs on secondary server while it is running on primary server. In case of failure, the application transparently continues to run on the secondary server, with the same state as the primary application. Juniper introduces the concept of virtual reliability [18]. Their solution monitors the health and performance of virtualized network objects to detect and isolate faults. To address the latency, Ericsson proposes the concept of Network Slicing, in which the network is sliced to meet traffic characteristics [6]. Resources for the network slices can be set up based on various service characteristics e.g. bandwidth demand, latency demand etc. Huawei introduces traffic scheduling management technique to reduces incoming/outgoing messages queue [19].

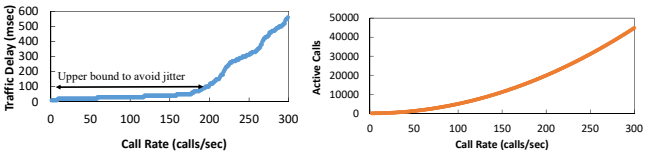


Figure 2: Even under the moderate call rate, the packets latencies are beyond acceptable value. These latencies exponentially grow by adding fewer number of more calls and potentially clog the whole data-plane.

However, all above solutions are generic and may fail when domain-specific issues lead to higher latency and failures. This motivates us to study 3GPP standardized IMS specifications and to discuss how well IMS standards handle latency and fault tolerance requirements. Moreover, our study is also useful to vIMS community in general. This is because IMS is 3GPP standardized and vendors and operators vIMS implementation must comply the standardized specifications.

IV. 3GPP Standardized IMS

The service requirements, architecture and protocol functionalities of IMS are standardized by 3GPP. The 3GPP standard specifications ensure that the device and IMS network elements comply to established procedures for their control and user plane functionalities. These specifications describe different IMS functions and their implementation framework. These include IMS access network [20], core network [21], call control functions [22], application functions [23], media functions [24], charging function [25], IMS interactions with circuit switched network [26], IMS security [27] and many more [7].

In the absence of vendor specific mechanisms, IMS NFV implementation relies on 3GPP standardized procedures to meet multimedia traffic requirements and to provide IMS fault tolerance. We put forward two simple questions. First, how well 3GPP standardized procedures meet multimedia quality of service (QoS) requirements in vIMS? and (2) how quickly these procedures recover vIMS failure? To find answers to these two questions, we prototype open source 3GPP compliant vIMS implementation by making significant changes into OpenIMS [10] platform. We develop various software modules as described by 3GPP IMS specifications [7] and observe these modules' interaction with each other for on-going media flow.

A. *Media-plane latencies exponentially increase with call rate*

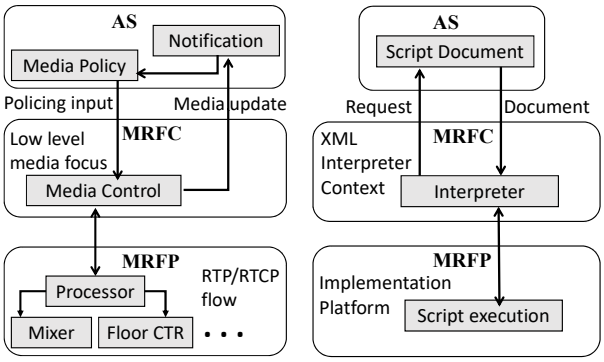
In reality, LTE network operators receive hundreds of thousands of multimedia requests (including VoLTE calls) per second [28]. Each of these media requests has stringent latency requirement as defined by LTE standard (refer to Table 6.1.7: Standardized QCI characteristics in [17]). For example, voice, video, push to talk voice, and IMS signaling have latency bounds of 100 msec; whereas interactive gaming and mission critical jobs have latency bounds of 50 msec and 60 msec, respectively. To test how well standardized implementation of vIMS meets latency requirements, we launched simultaneous call requests by varying call rate from 2 calls per second to 300 calls per second. We find that media-plane latencies significantly rise as we add moderate number of simultaneous calls into the vIMS system. Fig. 2(left) shows that the packets delay remains under 30 msec for first 100 calls per second. The latency doubles to 60 msec when we add as few as 25 more simultaneous calls; and reaches upto upper bound of VoLTE call's acceptable value (100 msec) for only 180 calls per second. The delay increases up to 4 times of VoLTE call's acceptable value (400 msec) by adding 0.5 times the existing number of calls (270 calls/sec). In our experiment, we did not terminate the already established calls because in reality IMS system should accept more number of calls on top of already on-going calls. vIMS provides seamless voice service under 15,000 active calls in total, as shown in Fig. 2(right).

Impact: This experiment explains that state-of-the-art vIMS design fails to meet QoS requirements on media traffic in operational LTE network. Therefore, operators are required to install many more NFs instances to meet current subscribers demand. This will increase their capital and operational expenditures (CAPEX and OPEX) which is against the NFV philosophy. High latencies may also cause failures, when vIMS completely stops responding because of system overload (by throwing too busy error) [29] and requires NFs reboot.

Analysis: Frequent loops between media plane modules The root cause of above issues is due to frequent interactions between different modules residing in different NFs. These interactions form a loop and packet processing latencies soar to the level where handling of media packets is no longer meaningful (i.e. voice jitters with large packet delays). Even worse, further increase in call rate causes packets congestion at different modules and renders these module non-responsive. IMS media-plane functionality is divided among 3 NFs (AS, MRFC and MRFP), as described by 3GPP specifications TS23.333 [24], TS23.218 [30] and TS24.147 [31]. When AS NF receives originating call notification, it sets-up media policy and informs MRFC to prepare network resources at MRFP. MRFC fetches media execution script documents, located at AS and forwards it to MRFP script execution engine. When media (VoLTE data packets) starts arriving at MRFP, media packets are processed and call metadata (e.g. call arrival rate, bandwidth usage, available buffer size etc.) is generated. This metadata is fed back to MRFC that adjusts call execution logic (e.g. codec bit rate, codec sample size and codec interval etc.) and informs MRFP. MRFP adjusts the number of packets that need to be transmitted every second (i.e. PPS = (codec bit rate) / (voice payload siz)) and the bandwidth (total packet size * PPS). This loop between different modules of MRFC and MRFP continues, as shown in Fig. 3a, during media execution.

We find that on top of media execution loop, there exists script fetching loop, as shown in Fig. 3b. This is mainly because AS retains full media execution control by dynamically generating XML scripts to be used by MRFC and MRFP (as explained in 4.3.1 *Delegation Model* of 3GPP document 24.880 [32]). IMS employs web model where media application behavior is defined in terms of markup languages/scripts (e.g. VoiceXML, SCXML, CCXML, and others)[1] [33] [34].

[1]Voice Extensible Markup Language (VoiceXML), State Chart extensible Markup Language (SCXML), and Call Control eXtensible Markup Language (CCXML)



(a) Multimedia execution loop **(b)** Script fetching loop

Figure 3: Frequent interactions between different modules

These scripts are located on the AS and retrieved by the MRFC using HTTP protocol. Scripts running on the MRFP provide media behavior notifications to AS (via MRFC), and receive media control updates from AS. This delegation model generates the loop between AS, MRFC and MRFP for media execution.

B. Media packets keep forwarded to device whose control-plane is aborted

Certain modules act as bridges between NFs. 3GPP specification TS23.218 [30] describes these modules that include Incoming/Outgoing Leg Control, Incoming/Outgoing Leg State, Registrar, Notifier, and others. Failure of these bridging modules results in control-plane termination. However, data-plane being decoupled from control-plane stays connected via different set of NFs. We dial originating VoLTE calls and slowly increase the call rate (adding 1 call per second). We then trigger *S-Incoming Leg Control* module failure at 25^{th} second (i.e. on setting up 25^{th} call), as shown in Fig. 5. On module failure, P-CSCF does not accept any new call and makes 5 retries (with retry interval of 1 second) to receive a response from S-CSCF. On 5^{th} unsuccessful retry, P-CSCF drops all calls by sending SIP BYE message to originating device (as shown in Fig. 5(left)). However, this SIP BYE message from P-CSCF was not forwarded to terminating devices because of the failure of only bridging module (*S-Incoming Leg Control*) between originating and terminating devices. Also, MRF does not receive this call disconnect message from AS (via failed *S-Incoming Leg Control* module) and maintains data-plane connection with LTE PGW.

Terminating devices keep generating uplink media packets that are forwarded to MRF through data-plane. Fig. 5(right) shows number of media packets received at different time interval. As shown, MRF forwards the terminating devices packets towards originating device, but does not receive any packet from originating device. In short, terminating devices keep the VoLTE call intact and experience speech mute issue, where they do not receive a response from originating device.

Impact: Media plane packets keep flowing from terminating devices towards originating device, and IMS does not have any mechanism (e.g. timers) to fully terminate the media connections. During this failure, IMS relies on human intervention

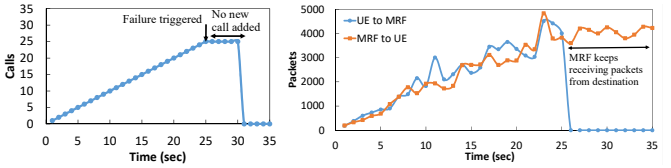
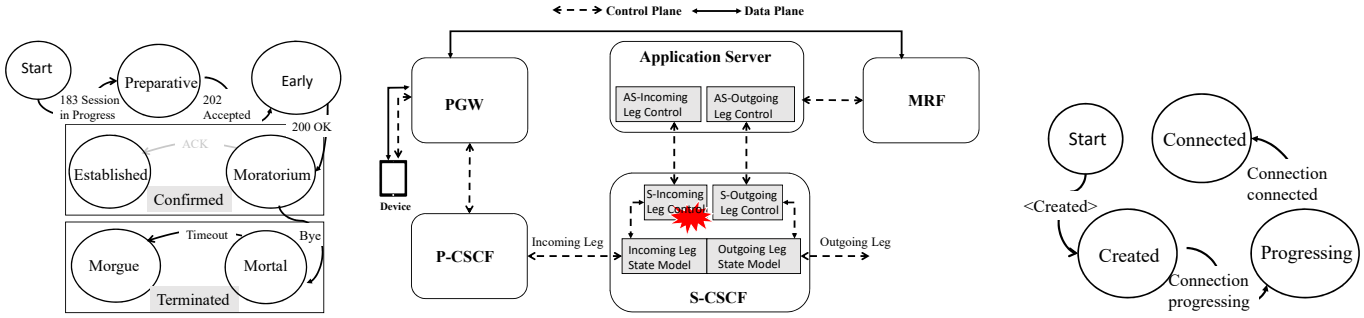


Figure 5: All calls are dropped when *S-Incoming Leg Control* module failure is triggered (left). However, MRF keeps forwarding data packets to device(right)

to stop the call (expecting the called party hangs-up the call after speech mute issue).

We find that the impact of this issue can be reduced (upto 30 seconds) when originating and terminating devices are located behind the NAT. In this case, STUN (Session Traversal Utilities for NAT) Binding Requests are used by these devices as a *keep-alive* mechanism to maintain NAT bindings for signalling and media flows [35]. If a device does not receive a STUN reply (within 30 seconds), it considers the flow and any associated security associations invalid and performs the initial registration procedures. However, operational LTE network operators are using IPv6 and do not install NAT in their network [36]; as a result, our finding has greater impact for operational IMS systems. Furthermore, in accordance to 3GPP requirements [35], device does not implement *keep-alive* mechanism when a NAT is not present (i.e. given battery considerations for wireless devices).

Analysis: Control-plane termination does not stop data-plane flow Both P-CSCF and MRF maintain dialogue states to keep track of user states in control-plane and data-plane, respectively [22]. The issue arises when MRF moves to *Connected* state, whereas P-CSCF fails to move from *Moratorium* state to *Established* state because of *S-Incoming Leg Control* module failure. This results in hanging state machine at MRF. MRF being in *Connected* state believes that control-plane connection has been established between originating and terminating devices and multimedia traffic can flow at any time instance. However, P-CSCF has not completed its control-plane connection establishment procedure yet; and finds *S-Incoming Leg Control* module being unresponsive. On detecting failure, P-CSCF first tries to reconnect by sending *Reconnect* control message towards S-CSCF, where S-CSCF tries to forward the message to *S-Incoming Leg Control* module for message delivery to AS. Meanwhile, P-CSCF times-out and declares S-CSCF to be busy by generating 600 BUSY EVERYWHERE message code. P-CSCF then acts as a User Agent (UA) and generates SIP BYE message towards originating device and S-CSCF, and transitions into *Mortal* state. When originating device receives SIP BYE message, it terminates both control and data plane connections; whereas S-CSCF tries to forward SIP BYE to AS so that the ongoing multimedia connection with the terminating device and MRF be stopped. However, S-CSCF drops the packet because it could not forward it to AS due to constant *S-Incoming Leg Control* module failure. As a result, terminating device does not receive SIP BYE message and keeps its control and data planes sessions. When terminating device generates its media packets, it then forwards them to MRF. MRF finds originating device dialogue state as *Connected* and forwards the received



(a) Dialogue state machine at P-CSCF **(b)** Module failure causes hanging state machine **(c)** Dialogue state machine at MRF

Figure 4: When *S-Incoming Leg Control* module fails, the control-plane communication between originating and terminating device breaks. The originating device connection is aborted by P-CSCF (where P-CSCF state transitions to *Mortal* state). However, this abort control signal does not reach to terminating device. As a result, terminating device keeps forwarding media packets to media-plane (MRF)

data packets to PGW. However, these packets are dropped at PGW because PGW cannot reach the originating device.
Note that *S-Incoming Leg Control* module failure goes undetected by AS because S-CSCF keeps replying layer 3 keep-alive message to AS NF.

V. Design

We put forward two design goals: (1) reducing media latency, and (2) improving system fault tolerance. At high level, we refactor NFs modules by (1) pipelining media plane processing and media control commands, and (2) quickly isolate faulty module by reconfiguring its neighboring modules. Fig. 6 gives an overview of our design. To reduce media plane latencies, *Media Update* module receives media metadata from *Media Processor* module and predicts future metadata values. It then requests control information for these predicted metadata values from MRFC. In other words, MRFC prefetches control information from MRFP for future purpose and steers *Media Processor* module accordingly. This prefetching of control information and processing of media packets are done in parallel, unlike serially in the state-of-the-art vIMS implementation. This is achieved because MRFC can likely predict future media behavior as the media processing conditions change. Similarly, MRFC prefetches media execution scripts from AS by predicting media execution conditions.

Our design reconfigures each module by adding back-up path with each of one-hop neighboring module. As shown in Fig. 6, *Session Controller* module at S-CSCF adds a back-up link to its one-hop neighboring module (*Incoming Leg Control* module) at AS. When *Incoming Leg Control* module at S-CSCF fails, then *Session Controller* assumes the role of failed *Incoming Leg Control* module and connects with *Incoming Leg Control* module of AS through back-up link. *Session Controller* loads failed module's execution logic and device session states upto to saved check-point. It then replays some of the already executed commands to resume failed operation.

A. Pipelining control instructions with media plane execution

MRFP processes media packets based on instructions it receives from MRFC and AS in real time. Ideally, these latencies do not go beyond few μseconds, but they significantly

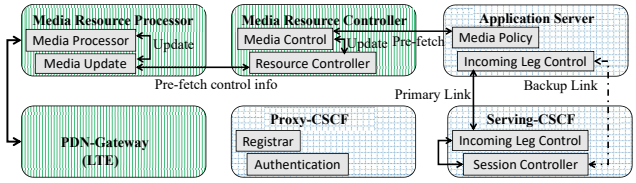


Figure 6: Design overview

increase under high call rate and fine-grained script execution control. To address this issue, we propose pipelining media processing and its control instructions request. This allows us to convert serial operations of processing of media packets at MRFP and fetching control instructions from MFRC into parallel operations. The control instructions are provided based on media behavior which can be determined through media metadata. When MRFP processes the packets, it also calculates the metadata (e.g. voice payload size, packets arrival rate or packets per second, available and consumed bandwidths and more) for these packets. To pipeline control instructions, we are required to predict future metadata generated by future media packets and then prefetching control instructions for these packets. We use a simple prediction algorithm, where we first calculate the deviation of received metadata value from its previous value and add this deviation into current value, as explained by algorithm 1. These new metadata values become our predicted medtadata for which we request control instructions from MRFC. Because control instructions remain same for a range of metadata values, our prediction does not prefetch wrong control instructions for most of the cases. For example, voice codec G.711 is applied for all packets with voice payload size in between 160 to 240 Bytes, and jitter rate in the range of 30 to 50 packets per seconds [8]. In other words voice processing instructions have built-in tolerance range that we exploit in our favor. However, as actual metadata comes closer to tolerance range, our algorithm may prefetch wrong control instruction by predicting wrong metadata. We address this issue by predicting batch of metadata that also optimizes our prefetching algorithm.

Optimization using batch prefetching: Prefetching future control instruction, although, helps to run packet processing in parallel; it does not reduce the control instruction fetching

Algorithm 1: Prefetching algorithm with/without optimization

procedure PREFETCHING
Predict:
    metadata ← receive *metadata* from Media Processor
    ϵ_i ← calculate prediction error from received*metadata*
    for *all metadata values m* **do**
        Δm = m - $m_{previous}$
        m_{new} = Δm + m + ϵ_i
        metadata$_{predicted}$ += m_{new}
    end for
Predict with Optimization:
    metadata$_h$ ← historical values of *metadata*
    α_i ← smoothing constant, where $0 < \alpha \leq 1$
    for *all metadata*$_h$ *values m* **do**
        $m_{new,t}$ = $\alpha m_{previous,t-1}$ + (1 - α)$m_{new,t-1}$
        metadata$_{predicted}$ += m_{new}
    end for
Send(metadata$_{predicted}$*)*
ReceiveFrom() ← *receive control info from MRFC*
Update(Control) / Update Media Processor */*
end procedure

loop. We propose generating batch of metadata by taking historical metadata measurements into account, and then requesting their control instruction. To achieve this, we use exponential smoothing model [9], described in algorithm 1, to forecast series of future metadata values. This model weighs past observations using exponentially decreasing weights, i.e. recent observations are given relatively more weight in forecasting than the older observations. Using exponential smoothing model, we generate a batch of predicted metadata. This batch contains 5 (also configurable) sets of metadata, where each metadata set contains several metadata values. Then this batch is forwarded to MRFC and respective control instructions are received. Similarly, MRFC provides script documents against predicted metadata sets that it fetches from AS. Thereafter, when *Media Update* module receives actual metadata values from *Media Processor* module, it immediately replies respective control instruction (as closest as possible) from prefetched control instructions set. *Media Update* module also observes the deviation of predicted and actual metadata values and requests or delays prefetching future control instructions by generating new batch of metadata. This procedure helps in reducing request/reply loop between MRFC and MRFP.

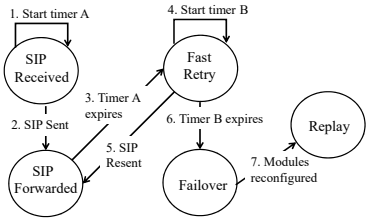


Figure 7: Failure detection and recovery procedure through FSM

B. Fault isolation and module reconfiguration

At the heart of failure recovery in our design is fault detection, its isolation and module reconfiguration. We show failure recovery procedure through Finite State Machine (FSM) diagram, as shown in Fig. 7. In the following, we explain failure detection and recovery procedure through SIP INVITE message example. Note that our design can also detect and recover from failure when failure occurs during call setup phase, (such as during INVITE, RINGING, TRYING SIP message failure), or after call setup phase (such as NOTIFY, UPDATE, CANCEL SIP message failure).

Failure detection procedure: When *Session Controller* of S-CSCF receives the INVITE request from P-CSCF, it enters into *SIP Received* state and starts the timer A, as shown in Fig. 7. Timer A is configurable timer, which is set to be 1 second in our implementation. Thereafter, INVITE message is forwarded to the next hop, i.e. towards *S-Incoming Leg Control* module, and *Session Controller* moves its state to *SIP Forwarded* sate. If *S-Incoming Leg Control* module does not reply to SIP INVITE message and Timer A expires, then *Session Controller* module enters into fast-retransmit stage – which is failure detection stage. The rationale of fast-retransmit is to quickly recover from failure by reducing the retry interval. *Session Controller* module starts Timer B, and resends SIP INVITE message after moving to *Fast Retry* state. In *Fast Retry*, SIP INVITE message is resent at an interval of 200 msec until Timer B expires, which is set to be 1 second in our implementation. In other words, we propose 5 retries of the failed SIP message. Assume, *S-Incoming Leg Control* still does not reply to SIP request; as a result Timer B expires and *S-Incoming Leg Control* is declared to be failed.

Failover procedure: After detecting failure, we perform fail-over procedure. When Timer B expires, *Session Controller* module declares *S-Incoming Leg Control* module out-of-service and takes charge of the non-responding module. In this process, *Session Controller* first deactivates the link between itself and *S-Incoming Leg Control* module. Then *Session Controller* module loads the failed module's executable through preloaded configurations. Next, it announces module failure to failed module's neighbors via backup link and declares itself being in-service module serving the failed module's execution. The recipient neighboring modules update their routing path and connects to *Session Controller*. Once the connection is setup, the *Session Controller* module replays the failed messages (i.e. SIP INVITE message in our example) at newly setup module and resumes the control-plane operation.

In order to replay the lost messages and to resume the service, in-service module should have access to the session states of the failed module. But these session states are also lost during module failure. To address this, we exploit the fact that both *Session Controller* and *S-Incoming Leg Control* modules communicate in a feedback loop of request and response. The requester can always know the session states at responder when it receives the reply. For example, device initiates call request by sending SIP INVITE message that ultimately reaches at *S-Incoming Leg Control* module of S-CSCF. S-CSCF then forwards it to *AS-Incoming Leg Control* module of AS. On receiving SIP message, AS creates device session that includes user identities, charging function address, and device authentication information etc. AS then

modifies the SIP INVITE message and sends it to S-CSCF. On receiving modified SIP INVITE message, S-CSCF modules store updated session along with checkpoint. Now assume, the failure occurred during next SIP message transmission (i.e. SIP PROGRESSING message). On the failure, the in-service module (which takes charge of failed module) replays the SIP messages starting from stored checkpoint. That is replaying all the SIP message upto the checkpoint over newly launched module (residing locally).

VI. Implementation

The details of our implementation efforts are as follow.

A. State-of-the-art implementation of IMS

OpenIMS provides basic IMS implementation where it does not implement all of the modules as defined by 3GPP specification [7]. We modify OpenIMS source code to add many more modules (such as breaking incoming and outgoing connection through *Incoming/Outgoing Session Control Leg* modules). Moreover, OpenIMS has coupled all IMS NFs by implementing them over single virtual machine (e.g. VMware) that provides optimal performance when hundreds of users are accessing IMS network at the same time. For NFV deployment, we first decouple IMS NFs into separate VMs. Then these VMs are bridged through virtual network interface. These stand-alone VMs are deployed over OpenStack to achieve state-of-the-art vIMS implementation. We also provide 1:1 redundant copy of IMS NFs to achieve minimum industry requirement for NFV. We use default timers as specified by IMS and OpenStack documents [37] [38]. We consider this 3GPP compliant implementation as state-of-the-art vIMS with which we compare our design.

B. Implementation of proposed vIMS

Pipelining control instructions with media plane processing To support pipelining, we first break the dependency between *Media Processor* and *Media Update* modules and setup two interfaces among them. *Media Processor* uses first interface to send metadata towards *Media Update*, and receives control instructions over second interface. It implements callback function to receive control instructions from *Media Update* module.

Failure detection procedure: We detect failure by observing state transitions in an FSM (as shown in Fig. 7). In FSM implementation an operation must start from an initial state and should transition to another accepted state. To achieve this, we create FSM transition table in which a given state transitions to a new state when either the response is generated for a request (i.e. no failure case) or its guard timer has expired (i.e. failure happens).

Fail-over procedure: We keep record of on-going device session (before failure) through an hash table. When failure occurs, the FSM transitions to *Failover* state. In *Failover* state, in-service module (that detected failure) retrieves last stored device session information from the hash table. Then in-service module updates the network configurations at incoming and outgoing interfaces and takes charge of failed module's operations.

VII. Evaluation

We evaluate how our proposed design reduces latencies and improves vIMS fault tolerance. The state-of-the-art vIMS described in Section VI-A serves as the baseline of our experiment with which we compare our design. We run our tests on a local network of servers with Intel Xeon(R) ES-2420 V2 processor at 2.20GHZ x 12, 16M Cache size, and 16GB memory. For each VM, we use Ubuntu Server 14.04.3 LTS with the OpenIMS Core. Each IMS NF is implemented over a separate server to make cluser of VMs in OpenStack. These VMs also share their resource with OpenAirInterface (an open source LTE platform) elements used for bridging IMS NFs with LTE.

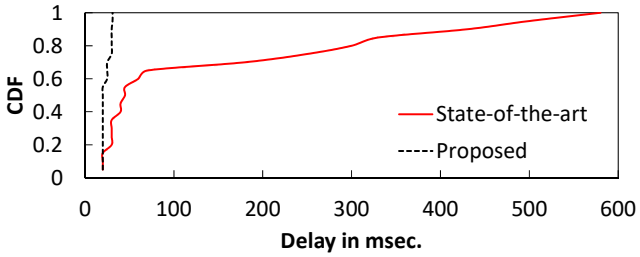


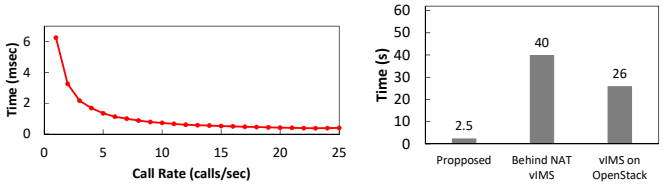
Figure 8: Comparing state-of-the-art vIMS media plane latencies with proposed vIMS

A. Reducing latencies

To evaluate how our proposed design reduces media-plane latencies compared to state-of-the-art vIMS design, we launch a number of voice calls and inject voice packets over established media connection between caller and callee. These calls are launched in systematic way, where our script increases the call rate by adding one call every second. In first second, 2 calls are dialed (2 calls per second), then 3 calls per second and so on, upto 300 calls per second.

In state-of-the-art design, as the call rate increases the media-plane latencies start increasing, as shown in Fig. 8. Only 60% of the traffic remains below 100 msec (upper bound of voice QoS requirement), where rest of 40% traffic incurs as high latency as 600 msec (6 times the upper bound of voice QoS requirement). The main reason behind this was the frequent interaction between *Media Processor* and *Media Control* modules. Media control module exercises fine-grained control on how voice packets are processed. It also interacts with *Media Policy* module to fetch call execution XML, using which *Media Processor* processes the packets.

Fig. 8 shows that proposed design significantly reduces media-plane latencies. These latencies remain under 50 msec even when MRF is processing around 40,000 simultaneous calls (when all active calls add-up, starting from 2 calls/sec to 300 calls/sec). This significant improvement was achieved because *Media Processor* module does not fetch control instructions and simply processes the packets as soon as voice packets arrive. The control instructions are sent to *Media Processor*



(a) Triggering of failure detection procedure **(b)** Failure recovery

Figure 9: Failure detection and recovery

over a separate interface, without meddling media packets execution.

B. Improving fault tolerance

Our design improves fault tolerance by (1) quickly detecting the failure, and (2) running failover procedure after isolating faulty module. In our experiment, we trigger *S-Incoming Leg Control* module failure during control-plane operation (i.e when call is being established) and let our system detect the failure and perform failover procedure.

How quickly failure detection mechanism triggers: In our design, we are not using keep-alive mechanism to detect the failure. Therefore, we are interested to observe how quickly the failure detection mechanism starts once the failure has occured. We argue that fault tolerance is only important for running system, i.e. when control-plane and data-plane messages are flowing through the system. Faults in idle systems do not have any impact on user applications. Fig. 9a shows that failure detection mechanism triggering time sharply drops from 6 msec to 1 msec when the call rate increases from 2 calls per seconds to just 7 calls per second. This is mainly because a call request triggers at least 4 SIP messages (i.e. INVITE, PROGRESSING, RINGING, and 200OK) that arrive within the call establishment time (on average 26 msec). This means roughly every 6 msec, one SIP message is processed at IMS – that is also the gap between failure occurrence time and start of failure detection procedure time. However, this gap significantly reduces to 1 msec as soon as fewer than 10 calls are added to the system. In other words, we can say that our design is efficient in triggering failure detection procedure, which is almost real time.

Failure detection: Failure detection in our proposed IMS, Behind NAT IMS (that uses Session Traversal Utilities for NAT (STUN) protocol [39]), and OpenStack implementations is based on timers. In our proposed design, failure is detected when both timers, Timer A and Timer B, expire after 2 seconds. Behind NAT IMS implementation declares failure after default value of 30 seconds, whereas OpenStack implementation takes 16 seconds at minimum to detect the failure [40].

Failover: Once the failure is detected, the failover procedure starts. Fig. 9b shows the time different systems take to restore the service after failure. Our proposed design simply reconfigures modules and loads the failed module's executable, and takes only 500 msec to recover from failure (after failure detection). However, both Behind NAT and OpenStack take roughly 10 more seconds after failure detection (8 seconds to prepare backup NF and restores the service, and 2 seconds to register device and establish call).

VIII. Related Work

Our work is in contrast with other recent efforts on NFV, vIMS and middle boxes' fault tolerance space.

NFV: [41] provides general purpose NFV platform. [42] and [43] make use of software and hardware choices to meet specific service demands. [44] [45] and [46] discuss NFV integration in mobile network. But these works do not discuss how NFV can provide same level of latency and fault tolerance as that of original carrier grade solutions.

vIMS: Recent works [47] [48] [49] study IMS-NFV. [47] [48] discuss IMS performance over virtualized instances (both at core and edge). [49] provides robust IMS over cloud through redundant modules. In contrast, we discuss vIMS latency and failure issues for modular vIMS implementation, and do not require any module redundancy. [50] provides dynamic resource allocation algorithm for vIMS, [50] discusses merits of deployment strategies of vIMS. [51] enhances vIMS features for M2M. But these efforts do not discuss latency and fault tolerance aspects in vIMS.

Fault Tolerance: [52] and [53] propose logging NF states during normal operations and reconstructing them after a failure. Their approaches cannot address real-time and transitory NF sessions recovery. [54] [55] and [41] discuss fault tolerance in non-IMS (SIP based) voice over IP applications. [56] discusses general load balancing strategies in vIMS and does not discuss vIMS working during faults.

Latency: [57] and [58] imply dynamically scheduling schemes to meet changing traffic demands in NFV. [59] and [60] propose mobile edge computing designs to reduce latencies. [61] proposes trading the latency off with other performance metrics.

All of above approaches fail to reduce system latencies which come from virtualizing system and its components interactions. They do not guarantee failure recovery when a particular software module stops working. Our design refactors system modules that not only improves fault tolerance but also reduces latencies.

Acknowledgement

We thank anonymous reviewers for their insightful comments. This work is partly supported by NSF grants (CNS-1422835 and 1528122).

IX. Conclusion

We show that highly modular vIMS design incurs latencies when different NFs modules interact with each other over a chain of operations. Also, such design, relying on cloud platform's failure detection mechanisms, does not tolerate module's faults. This results downlink data-plane operations to carry-on, even though device control-plane has been terminated. Our design, refactors NFs modules and reduces latencies by pipelining control and processing operations. It also reconfigures modules to tolerate faults by first isolating faulty module and then resuming the failed operation.

REFERENCES

[1] What is NFV. http://www.etsi.org/images/files/ETSITechnologyLeaflets/NetworkFunctionsVirtualization.pdf.

[2] F. Networks. Virtual solutions for your nfv environment. In *Technical Report*.

[3] AT&T raises SDN network transformation goal to 55% for 2017. http://www.fiercetelecom.com/telecom/at-t-raises-sdn-network-transformation-goal-to-55-for-2017.

[4] Successful virtual IMS pilot testing announced by Iskratel ahead of demonstration at Mobile World Congress 2017. http://www.realwire.com/releases/Successful-virtual-IMS-pilot-testing-announced-by-Iskratel-ahead-of-MWC.

[5] Deploying Telco-Grade Cloud Solutions and NFV. http://www.stratus.com/assets/WP-Telco.pdf.

[6] Network Slicing. https://www.ericsson.com/networks/topics/network-slicing.

[7] List of IMS-related Specifications. http://onlinelibrary.wiley.com/doi/10.1002/9780470695135.app1/pdf.

[8] Quality of Service Design Overview. http://www.ciscopress.com/articles/article.asp?p=357102.

[9] Forecasting with Single Exponential Smoothing. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc432.htm.

[10] OpenIMS – The Open Source IMS Core Project. http://www.openimscore.org/.

[11] OpenStack Open Source Cloud Computing Software. https://www.openstack.org/software/.

[12] Ericsson Blade System (EBS) for IMS. http://archive.ericsson.net/service/internet/picov/get?DocNo=266/03819-FAP130506&Lang=AE&HighestFree=Y.

[13] Alcatel-Lucent End-to-End IMS Solution. https://www.alcatel-lucent.com/solutions/communications-collaboration.

[14] N-Version Programming. http://www.inf.pucrs.br/~zorzo/cs/n-versionprogramming.pdf.

[15] NFV booms as Verizon, AT&T, NTT, Deutsche Telekom, SK Telecom invest. http://www.telecomlead.com/telecom-equipment/nfv-booms-verizon-att-ntt-deutsche-telekom-sk-telecom-invest-66784.

[16] DoCoMo to Offer NFV-Based LTE in 2016. http://www.lightreading.com/carrier-sdn/nfv-(network-functions-virtualization)/docomo-to-offer-nfv-based-lte-in-2016-/d/d-id/709223/.

[17] 3GPP. TS 23.203: Policy and Charging Control Architecture, 2013.

[18] NFV Solutions for the Telco Cloud. https://www.juniper.net/assets/kr/kr/local/pdf/solutionbriefs/3510513-en.pdf.

[19] Unleashing IT Values, Keeping Telco DNA. http://e.huawei.com/us/publications/global/ict_insights/201509091408/Features/201509091519.

[20] 3GPP. TS23.981: Interworking aspects and migration scenarios for IPv4-based IP Multimedia Subsystem (IMS) implementations, 2012.

[21] 3GPP. TS23.228: IP Multimedia Subsystem (IMS);Stage 2, 2012.

[22] 3GPP. TS 23.229 – Release 12: IP multimedia call control protocol based on Session Initiation Protocol (SIP), 2014.

[23] 3GPP. TS29.334: IMS Application Level Gateway (IMS-ALG) - IMS Access Gateway (IMS-AGW), 2014.

[24] 3GPP. TS23.333: Multimedia Resource Function Controller (MRFC) and Multimedia Resource Function Processor (MRFP); Procedures descriptions, 2014.

[25] 3GPP. TS32.260: IP Multimedia Subsystem (IMS) charging.

[26] 3GPP. TS29.163: Interworking between the IP Multimedia (IM) Core Network (CN) subsystem and Circuit Switched (CS) networks, 2014.

[27] 3GPP. TS33.203: Access security for IP-based services, Sep. 2012.

[28] Call volume statistics. http://www.arcep.fr/index.php?id=10519&L=1.

[29] E. Bauer and R. Adams. *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[30] 3GPP. TS23.218: IP Multimedia (IM) session handling; IM call model, 2014.

[31] 3GPP. TS24.147: Conferencing using the IP Multimedia (IM), 2014.

[32] 3GPP. TR24.880: Media server control using the IP Multimedia (IM).

[33] Call Control eXtensible Markup Language. https://www.w3.org/TR/ccxml/.

[34] Voice Extensible Markup Language. https://www.w3.org/TR/voicexml20/.

[35] 3GPP. TS124.229: Internet Protocol (IP) multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP).

[36] C.-Y. Li and et al. Insecurity of voice solution VoLTE in LTE mobile networks. In *ACM CCS*, 2015.

[37] 3GPP. TS186.008–2: IIMS Network Testing: IMS Configurations and Benchmarks.

[38] Default openstack timers. http://docs.openstack.org/kilo/config-reference/content/cinder-conf-changes-kilo.html.

[39] P. M. J. Rosenberg, R. Mahy and D. Wing. Session Traversal Utilities for NAT (STUN), 2008. RFC 5389.

[40] Apache ZooKeeper. https://zookeeper.apache.org/.

[41] S. Palkar and et al. E2: a framework for NFV applications. In *ACM SOSP*, 2015.

[42] R. Mahindra and et al. A practical traffic management system for integrated LTE-WiFi networks. *ACM Mobicom*, 2014.

[43] M. T. Raza and et al. Rethinking LTE Network Function Virtualization. In *IEEE ICNP*, 2017.

[44] Costa-Requena and et al. SDN and NFV integration in generalized mobile network architecture. In *IEEE EuCNC*, 2015.

[45] Krishnaswamy and et al. An open NFV and cloud architectural framework for managing application virality behaviour. In *IEEE CCNC*, 2015.

[46] Akyildiz and et al. Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: an overview and qualitative evaluation. *Computer Networks*, 93:66–79, 2015.

[47] A. Sheoran and et al. Contain-ed: An NFV Micro-Service System for Containing e2e Latency. In *ACM HotConNET workshop*, 2017.

[48] A. Sheoran and et al. An empirical case for container-driven fine-grained VNF resource flexing. In *IEEE NFV-SDN Conference*, 2016.

[49] M. T. Raza and et al. Modular Redundancy for Cloud based IMS Robustness. In *ACM Mobiwac*, 2017.

[50] Carella and et al. Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures. In *IEEE ISCC*, 2014.

[51] M. Abu-Lebdeh and et al. Cloudifying the 3GPP IP multimedia subsystem for 4G and beyond: A survey. *IEEE Communications Magazine*, 54(1):91–97, 2016.

[52] J. Sherry and et al. Rollback-recovery for middleboxes. In *ACM SIGCOMM*, 2015.

[53] Rajagopalan and et al. Pico Replication: A high availability framework for middleboxes. In *ACM Symposium on Cloud Computing*, 2013.

[54] Bozinovski and et al. Fault-tolerant SIP-based call control system. *Electronics Letters*, 39(2):254–256, 2003.

[55] Pant and et al. Optimal availability and security for IMS-based VoIP networks. *Bell Labs Technical Journal*, 11(3):211–223, 2006.

[56] F. Lu and et al. A virtualization-based cloud infrastructure for IMS core network. In *IEEE CloudCom*, 2013.

[57] P. Duan and et al. Toward Latency-Aware Dynamic Middlebox Scheduling. In *IEEE ICCCN*, 2015.

[58] R. Mijumbi and et al. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *IEEE NetSoft*, 2015.

[59] B. Yang and et al. Seamless Support of Low Latency Mobile Applications with NFV-Enabled Mobile Edge-Cloud. In *IEEE Cloudnet*, 2016.

[60] D. Lake. Enhancing Mobile Services with Edge Computing Capabilities. *Newsletter*, 2015.

[61] R. A. Hamada and et al. An IMS-Based LTE-WiMAX-WLAN architecture with efficient mobility management. In *IEEE MELECON*, 2016.