Refactoring Network Functions Modules to Reduce Latencies and Improve Fault Tolerance in NFV

Muhammad Taqi Raza, *Member, IEEE*, Songwu Lu, *Fellow, IEEE*, Mario Gerla, *Fellow, IEEE*, and Xi Li

***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. This NFV-based software-centric approach cannot use dedicated mechanisms implemented over custom built boxes to reduce latencies and tolerate faults. We present a case study of IP multimedia subsystem (IMS), which is the most complex NFV instance, requires extremely low end-to-end latency (40 msec), and demands system availability as high as five nines. Through an empirical study, we discover that highly modular 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 failure, 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 IMS modules, and recover failed modules by reconfiguring their neighboring modules. We build our system prototype of open source IMS over OpenStack platform. Our results show that our scheme reduces latencies and failure recovery time up to 12× and 10×, respectively, when compared with the state-of-the-art virtualized IMS implementation.**

***Index Terms*— Network functions virtualization, LTE, IP multimedia subsystem, fault tolerance, software defined networking.**

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]. NFV allows network service providers to scale services up or down quickly to address changing service demands while reducing capital and operational expenditures (CAPEX and OPEX) [3], [4].

The move away from proprietary hardwares means virtualized network systems cannot take advantage of special mechanisms to tolerate faults and reduce packet processing latencies. The special mechanisms in proprietary hardware include (1) module redundancy by providing strong coupling between software and hardware [5], [6]; and (2) performing functional checks through proprietary software platforms [7], [8]. This motivates us to study how virtualized network functions (VNFs) handle latencies of growing number of users requests, and tolerate faults in the absence of vendor specific mechanisms.

We consider Virtualized IP Multimedia Subsystem (vIMS) as a case study of NFV based network system implementation. There are several reasons behind our choice of vIMS as NFV implementation use case. First, there are a variety of IMS based applications (such as Voice over LTE (VoLTE), Video over LTE (ViLTE), Conference Call, to name a few) used by millions of subscribers in operational LTE networks [9]. Second, IMS based applications stipulate stringent requirements on both low latency and high fault tolerance. For example, VoLTE call requires end-to-end voice packets latency to be less than 100 msec while continuing voice call during system faults [10]. Third, IMS procedure is well standardized by 3GPP [11] standard body restricting all IMS implementations to follow the standard. Fourth, through open source IMS implementation (OpenIMS [12]), we can test the working of standardized NFs and their modules.

We prototype vIMS by implementing OpenIMS over open source cloud platform (OpenStack [13]), and we ensure that vIMS implementation is in accordance to the IMS standard. Our prototype acts as a baseline vIMS implementation that we call state-of-the-art vIMS. We conduct an empirical study on our vIMS implementation and record the observations as follows:

1) IMS NFs are highly modular; 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

Manuscript received May 01, 2018; revised August 15, 2018; accepted August 28, 2018. Date of publication September 13, 2018; date of current version November 28, 2018. The work of M. T. Raza and S. Lu was supported by NSF under Grants 1423576 and 1526985. The work of X. Li was supported by NSFC under Grants 61772482 and 61272131. This paper was presented at ACM/IEEE/IFIP CNSM 2017. *(Corresponding author: Xi Li.)*

M. T. Raza, S. Lu, and M. Gerla are with the Department of Computer Science, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: taqi@cs.ucla.edu; slu@cs.ucla.edu; gerla@cs.ucla.edu).

X. Li is with the School of Software Engineering, University of Science and Technology of China, Hefei 230000, China (e-mail: llxx@ustc.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2869965

processor, controller and policy modules in different NFs that require realtime packet processing behavior to update control information and policies. For example, a controller module requires available bandwidth, and packets arrival rate from a processor module to adjust voice codec [14]. This results in packet processing latencies.

2) During media session setup, both control and data planes create device session states and transition from one state to the other as a call progresses. Device states at control and data planes 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*. This 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 [15] 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 correct future control instruction even for a small prediction error.

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, neighboring module detects a module failure and assumes the role of failed module by loading its execution logic as provided in the configurations, 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 OpenIMS [12] implementation over Openstack [13]. Our results show that (1) our system reduces media latencies upto 12×, and (2) resumes failed operation within 2.5 seconds, which is 10× better than current state-of-the-art vIMS design.

II. Migrating Carrier Grade NFs to NFV

Network systems heavily rely on advanced in-network processing for critical functions ranging from traffic management to VoLTE service. These Network Functions (NFs) are implemented over dedicated hardware boxes which are spread within the network. Recently, the motivation to reduce systems' operational and capital expenditures with exponentially increasing traffic growth inspire building network systems (NFs) using commodity compute/storage/network resources while reducing (or removing) dependency on specialized platforms. Migrating carrier grade boxes functionality towards virtualized middleboxes is called Network Function Virtualization, NFV. Such software only implementation of NF(s) decoupled from hardware also lead to more agile and flexible deployment models.

However, implementing task specific network systems over virtualized platform incurs latencies; The time critical applications suffer the most whose latencies were previously contained by dedicated box mechanisms. Latencies in NFV mainly come from virtualization layer, lack of system support for providing enough resources to time critical applications, service chaining between different NFs, and more. Furthermore, failures are common in virtualized environment [16], which are caused by single server or node failure, misconfigured NFs, software patch or upgrade glitches, and due to network congestion or overload [17].

To address these issues, both industry and researchers proposed very generic solutions, such as VNF placement according to application need [18], using baremetal approach to reduce latencies [19], adding 1:N redundancy [17] and others. However, we argue that such generic solutions do not guarantee application-specific QoS in terms of endured latencies and expected system performance. We advocate to apply domain specific knowledge in improving a particular system and deliver application requirements. Therefore, in this work, we focused on an NFV system by considering its associated application needs, the interactions between different NFs during the system operation, and system requirements stipulated by the standard.

III. Motivation of Using Virtualized IMS as NFV Implementation Case-Study

NFV is keenly followed by the telecom industry and its proof of concept implementations are already in process [20]. Most telecom operators are considering to support their IP Multimedia Subsystem (IMS) implementation over NFV [21]. IMS is an architectural framework for delivering IP multimedia services, such as VoLTE, ViLTE, software as a service (SaaS), social sites, navigation, and many more. 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). These multimedia services have stringent requirements on latency (end-to-end latency goes as low as 40 msec [10]) and fault tolerance (operator networks want to achieve 5 nines availability [22]). This motivates us to study how IMS can achieve such low latency by tolerating faults over virtualized infrastructure. We are also motivated to use IMS as a case-study because it already has an open source implementation (OpenIMS [12]), and its functionalities are well documented by 3GPP specifications [11]. Lastly, like LTE, IMS supports both control-plane and data-plane functions. That means, the lesson learned in IMS use case study can also be used in virtualizing LTE core.

IV. IMS Background

IMS runs on top of LTE that provides best effort service to the users, with no guarantee on the amount of bandwidth

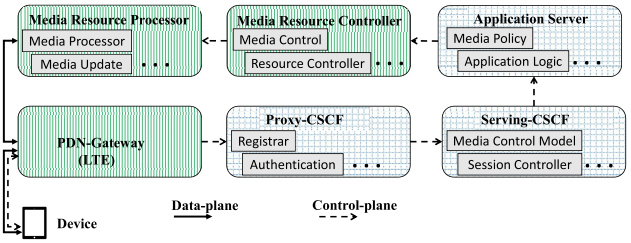


Fig. 1. IMS architecture: An overview.

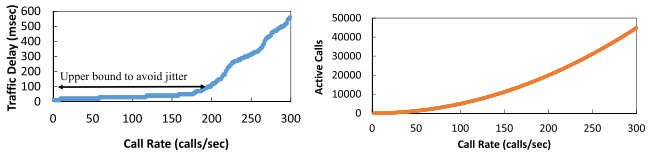


Fig. 2. Even under the moderate call rate, the packets latencies are beyond acceptable value. These latencies exponentially grow by adding fewer number of more calls and potentially clog the whole data-plane.

a user gets for a connection and the delay experienced by the packets. Therefore, IMS is the preferred choice of mobile operators to support real-time multimedia services. IMS uses Internet protocols and brings multiple media, multiple point of access and multiple modes of communication into a single network, enabling simultaneous voice and multimedia services for end users [23].

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

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. LTE device (IMS client) first registers with LTE core network and then initiates IMS signalling over IMS control-plane. The P-CSCF is an access point for IMS and acts as a SIP proxy for all the user equipments. P-CSCF simply forwards all traffic to S-CSCF. S-CSCF is the core of the IMS and it is the point of control within the network that enables operators to control the entire service delivery process and all the sessions. S-CSCF forwards the request to AS that applies service logic in accordance to defined policy and replies back the modified session to S-CSCF for its delivery to destination network.

Data-Plane includes Media Resource Function (MRF) that processes, stores data and generates services for the subscribers. MRF functionality is further split into two NFs that perform its control-plane, and data-plane actions. Once user session has been established, the user data-plane traffic is sent to Media Resource Function Processor (MRFP) – performing data-plane action of MRF. 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) – performing control-plane action of MRF. Moreover, MRFC employs monitoring schemes to determine policy rules in real-time.

Note that, in order to improve the readability of the paper, we use the term MRF when discussing the data-plane functionality as a whole.

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 different modules (rectangular shaped) implemented within different NFs (rounded rectangular shaped).

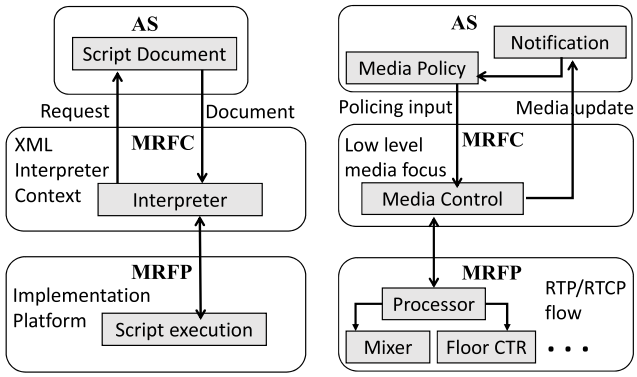
V. Empirical Study

Through an empirical study, we understand (1) how different VNFs' modules interaction adds latencies in media-plane (data-plane), and (2) key module failure impacts on-going multimedia traffic. We prototype open source vIMS implementation by making significant changes into OpenIMS [12] platform. We develop various software modules after studying tens of IMS standard documents [11], [24] and observe these modules' interaction with each other for on-going media flow. The implementation details of our prototype is discussed in Section VII. We use VoLTE as multimedia application, because VoLTE is a premier IMS application widely used in operational LTE networks worldwide [25]. Below, we describe each finding, its impact and then provide analysis of the finding.

A. Media-Plane Latencies Exponentially Increase With Call Rate

We find that media-plane latencies significantly rise as we add moderate number of simultaneous calls into the vIMS system. In reality, LTE network operators receive hundreds of thousands of multimedia requests (including VoLTE calls) per second [26]. Each of these media requests has stringent latency requirement as defined by LTE standard (refer to Table 6.1.7: Standardized QCI characteristics in [10]). 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. For testing latency requirements, we launched simultaneous call requests by varying call rate from 2 calls per second to 300 calls per second. Fig. 2(left) shows that the packets delay remain 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 reach 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 reaches its capacity to provide 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 [27]



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

Fig. 3. Frequent interactions between different modules in data-plane execution.

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) [28] 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 cause packets congestion at different modules and render these module non-responsive. IMS media-plane functionality is divided among 3 NFs (AS, MRFC and MRFP). 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 meta data (e.g. call arrival rate, bandwidth usage, available buffer size etc.) is generated. This meta data 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. 3b during media execution.

We find that on top of media execution loop, there exists script fetching loop, as shown in Fig. 3a. This is mainly because AS retains full media execution control by dynamically generating XML scripts [29]. IMS employs web model where media application behavior is defined in terms of markup languages/scripts (e.g. VoiceXML, SCXML, CCXML, and others)[1] [30], [31]. These scripts are located on the AS and

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

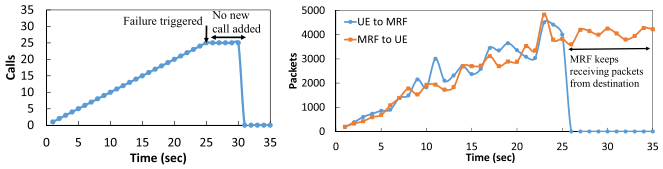


Fig. 4. All calls are dropped when *S-Incoming Leg Control* module failure is triggered (left). However, MRF keeps forwarding data packets to device even though the device does not maintain any connection (right).

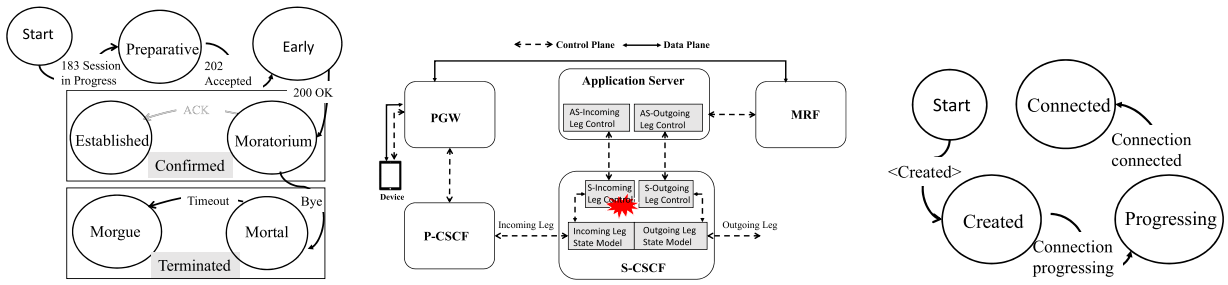
retrieved by the MRFC using HTTP protocol. Scripts running on the MRFP provide media behavior notifications to AS (via MRFC), and receive media control updates from AS. Furthermore, AS can exercise more fine-grained control of the media behaviour by defining smaller scripts for the MRFC to execute and requiring MRFC to retrieve further scripts from AS. These scripts are dynamic where techniques (such as JSP, ASP and Servlets) are used to dynamically generate script documents and are transmitted to MRFC over HTTP protocol. In short, the media flow and presentation state is delegated to the MRFC and MRFP, while the AS retains overall control since scripts are defined (and can be dynamically generated) by the 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. Failure of these bridging modules result 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 (a bridging module between S-CSCF and the AS) failure at 25th second (i.e. on setting up 25th call), as shown in Fig. 4. 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 5th unsuccessful retry, P-CSCF drops all calls by sending SIP BYE message to originating device (as shown in Fig. 4(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 UL media packets that are forwarded to MRF through data-plane. Fig. 4(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: The impact of *S-Incoming Leg Control* module failure is quite severe, because media plane packets from terminating devices towards originating device keep flowing



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

Fig. 5. 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).

and system does not have any mechanism (e.g. timers) to fully terminate the media connections. During this failure, IMS relies on human intervention 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 [32]. 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 [33]; as a result, our finding has greater impact for operational IMS systems. Furthermore, in accordance to 3GPP requirements [32], 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 The root cause of above issue is the hanging state machine at MRF. On *S-Incoming Leg Control* module failure, P-CSCF updates device state to *Mortal* but MRF stays in *Connected* state.

We now explain how dialogue states are created at control and data planes and how these states transition that leads to hanging state at data-plane on module failure. Once the device is registered, it initiates multimedia request (i.e. originating VoLTE call) by sending SIP INVITE request towards P-CSCF (via LTE PGW). The P-CSCF creates a dialogue for requested multimedia service with a *Start* state (shown in Fig. 5a) and forwards the request to S-CSCF by transitioning to *Preparative* state. Through dialogue, P-CSCF keeps track that which SIP message is for which device and for which multimedia session. On receiving the INVITE message from P-CSCF, S-CSCF creates device session and states for incoming leg at its *S-Incoming Leg Control* module and then forwards the INVITE request to *AS-Incoming Leg Control* module. AS then processes the INVITE request and prepares MRF to handle imminent multimedia traffic. MRF also creates a dialogue for requested multimedia service, with a *Start* state (shown in Fig. 5c), and prepares required multimedia resources for media application. Note that both P-CSCF and MRF maintain dialogue states to keep track of user states in control-plane and data-plane, respectively. MRF transitions to *Created* state by sending an acknowledgement regarding data-plane dialogue setup to AS. *AS-Outgoing Leg Control* module modifies the INVITE request and forwards the request to *S-Outgoing Leg Control* module of S-CSCF, which then forwards it to terminating device and waits for a provisional response from terminating device (such as 180 RINGING). Once S-CSCF receives the provisional response, it forwards it to MRF (via AS) and P-CSCF. Both P-CSCF and MRF move to *Early* and *Progressing* states, respectively. Similarly, P-CSCF and MRF transition to *Moratorium* and *Connected* states, respectively, on receiving successful multimedia setup response (such as 200 OK) from terminating device.

Connected state at MRF means that control-plane connection has been established between originating and terminating devices and multimedia traffic can flow at any time instance. However, P-CSCF requires an acknowledgement from device to transition from *Moratorium* state (a substate of the *Confirmed* state) to *Established* state.

S-Incoming Leg Control module failure coupled with device failure at this point (when P-CSCF has not yet transitioned to *Established* state) results into hanging state machine scenario 1. Because of failure, P-CSCF is not able to receive an acknowledgement from device and transitions to *Mortal* state by sending call clear (SIP BYE) message towards S-CSCF. However, MRF does not receive the call clear message from S-CSCF via AS and does not terminate the data-plane dialogue. We assume that there are no device failures and originating and terminating devices are successful to establish multimedia connection. At later stage, *S-Incoming Leg Control* module failure happens that results into hanging state machine scenario 2. 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, as shown in Fig. 6. 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

Protocol	Info
SIP	Status: 600 Busy everywhere - Forwarding to S-CSCF failed |
SIP	Status: 600 Busy everywhere - Forwarding to S-CSCF failed |
SIP	Request: sip:scscf.open-ims.test:6060 (1 binding) |
SIP	Status: 600 Busy everywhere - Forwarding to S-CSCF failed |
SIP	Status: 600 Busy everywhere - Forwarding to S-CSCF failed |
SIP	Request: BYE sip:alice@open-ims.test |
SIP	Request: BYE sip:alice@open-ims.test |
SIP	Status: 403 Forbidden - Originating subsequent requests outside dialog not allowed |
SIP	Status: 403 Forbidden - Originating subsequent requests outside dialog not allowed |

Fig. 6. Wireshark logs at P-CSCF: P-CSCF tries to connect to S-CSCF and times out because S-CSCF is unable to forward the request to AS. P-CSCF updates S-CSCF status as busy and sends a BYE message to device by acting as an user agent (UA). Thereafter, any modification to the dialogue is not allowed.

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. 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 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 replies layer 3 keep-alive message to AS NF [34].

C. Discussion

We discuss how latencies are controlled and fault tolerance is provided in current dedicated boxes, and why virtualization platforms cannot match the performance of carrier grade solutions.

Purpose-built hardware platforms have been developed which can tolerate faults and reduce latencies. They continue to provide the required functioning despite occasional internal components and modules failures, either transient or permanent. Examples of hardware platforms are Ericsson's Blade Systems (EBS) [5], Alcatel-Lucent's Element Management System (EMS) [35], and Huawaei's ATCA [36] that employ internal redundant hardware modules and provide NF availability even during failures and maintenance at any time without disturbing traffic. At software side, NF equipment vendors provide strong coupling between their software and hardware. Software fault tolerance is achieved by software design that ensures redundancy, both for error detection and error recovery. As the system operates, functional checks are made on the acceptability of the results generated by each piece of software component. Software platforms include Ericsson's ERLANG [5], Alcatel-Lucent's NVP [8], and Huawaei's Fusion [37] that use various software techniques for scalable real-time systems with requirements on concurrency, distribution and fault tolerance.

On the other hand, vIMS is implemented over cloud platforms (such as OpenStack [13]), and relies on cloud platform's recovery procedure for fault tolerance that uses heart-beat mechanism to detect failures. These cloud systems can only detect fail-stop failures at NF level, and not at the module-level granularity. Moreover, these system takes tens of seconds to reallocate a NF [38].

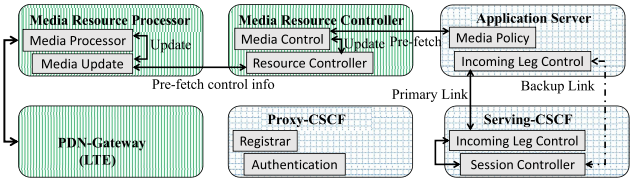


Fig. 7. Design overview.

VI. Design

We put forward two design goals: (1) reduce media latency, and (2) improve 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. 7 gives an overview of our design. To reduce media plane latencies, *Media Update* module receives media meta-data from *Media Processor* module and predicts future meta-data values. It then requests control information for these predicted meta-data values from MRFC. In other words, MRFC prefetches control information from MRFP for future purpose and steers *Media Processor* 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 neighbouring module. As shown in Fig. 7, *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 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. We achieve this by pipelining next control instructions for future media packets while processing received media packets. 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, availble 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 [14]. In other words voice processing instructions have built-in tolerance range that we exploit in our favour. 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.

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**
    $\Delta \mathrm{m} = \mathrm{m} - \mathrm{m}_{previous}$
    $\mathrm{m}_{new} = \Delta \mathrm{m} + \mathrm{m} + \epsilon_i$
    $\mathrm{metadata}_{predicted} += \mathrm{m}_{new}$
  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**
    $\mathrm{m}_{new,t} = \alpha \mathrm{m}_{previous,t-1} + (1 - \alpha)\mathrm{m}_{new,t-1}$
    $\mathrm{metadata}_{predicted} += \mathrm{m}_{new}$
  end for
Send($metadata_{predicted}$*)*
ReceiveFrom()← *receive control info from MRFC*
Update(Control) ▷ $UpdateMediaProcessor$
end procedure

1) Optimization Using Batch Prefetching: Prefetching future control instruction, although, helps to run packet processing in parallel; it does not reduce the control instruction fetching 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 [15], 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.

2) Complexity: The algorithm 1 contains two parts: (1) In the first part, the prediction of metadata value, $\mathrm{metadata}_{predicted}$, is done by iterating through all metadata values m. This iteration is performed once which is linear with complexity of O(m). (2) The prediction with optimization part considers historical values, $\mathrm{metadata}_h$, into account and iterates through all metadata values m. This iteration is also done once and it is linear in time. Because, either part 1 or part 2 is executed, thus, the time complexity of the algorithm is linear in m.

Algorithm 2 Module Reconfiguration Algorithm

procedure RECOVERY PROCEDURE
Start:
  SIP ← ReceiveFrom()
  Start timer A
  SendTo (SIP)
  if Timer A expires And SendTo(SIP) is Failed **then**
    goto *FastRetry()*
  else if SendTo(SIP) is Success **then**
    Break
  end if
FastRetry():
  Start timer B
  SendTo (SIP)
  if Timer B expires And SendTo(SIP) is Failed **then**
    goto *Failover()*
  else if SendTo(SIP) is Success **then**
    Break
  end if
Failover():
  Announce(): failure to failed module's neighbors
  Reconfigure(): links with failed module's neighbors
  goto *Start()*
end procedure

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 diagram, as shown in Fig. 8, and capture the steps through algorithm 2. In the following, we explain failure detection and recovery procedure through SIP INVITE message example. However, note that our design can detect and recover from failure when failure occurs during call setup phase, (such as during INVITE,

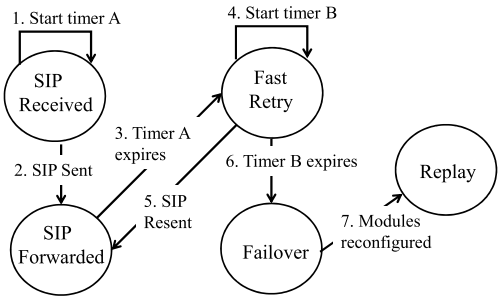


Fig. 8. Failure detection and recovery procedure through state transition diagram.

RINGING, TRYING SIP message failure), or after call setup phase (such as NOTIFY, UPDATE, CANCEL SIP message failure).

1) Failure Detection Procedure: When *Session Controller* receives the INVITE request from P-CSCF, it enters into *SIP Received* state and starts the timer A, as shown in Fig. 8. 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.

We should also highlight the rationale of not using keep-alive mechanism to detect the failure. First, such mechanism can burden internal NFs modules by unnecessarily pinging all neighboring modules. Second, IMS system capable of handling hundreds of thousands of subscribers voice requests per day [26] is processing at least tens of SIP messages per seconds. These SIP messages are enough to detect module failure. Third, even though the failure is not detected for a longer period, it will not have any impact on media plane execution and let users exchange media packets even during failures.

2) Failover Procedure: After detecting failure, we perform fail-over procedure when *Session Controller* module declares *S-Incoming Leg Control* module out-of-service and takes charge of non-responding module. When *S-Incoming Leg Control* does not respond and triggers Timer B expiration, we first deactivate the link between *Session Controller* and *S-Incoming Leg Control*. Then *Session Controller* module loads the failed module's executable through preloaded configurations and configures it by activating the link between itself and newly loaded module (acting as *S-Incoming Leg Control*). In other words, *Session Controller* connects with *S-Incoming Leg Control* through a local loop. 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, then *Session Controller* module replays the failed messages (i.e. INVITE SIP message in our example) at newly setup module and resumes the control-plane operation.

To proceed with message replay and resume the service, in-service module should have access to the session states of the failed module. However, such 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 INVITE SIP message which ultimately reaches at *S-Incoming Leg Control* module of S-CSCF which 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.; and modifies the INVITE SIP message and then sends the modified message 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 PROGRESSING). On this failure, the in-service module (which takes charge of failed module) replay the SIP messages starting from stored checkpoint. That is replaying all the SIP message upto the checkpoint over newly launched module (residing locally).

3) When a Timeout is Not a Failure: It is possible that proposed, but configurable, timeout value does not represent actual module failure. Such rare case occurs when the link between two module is severely congested or *S-Incoming Leg Control* module goes through random failure Â- not impacting the functionality of *S-Incoming Leg Control*. We still define such case as a failure that impacts user Quality of Service (QoS), and performs failover procedure. Indeed, such failover procedure helps reducing link congestion. This is because during failover procedure links (interfaces) between modules are reconfigured (changed) and congested links are removed from execution path.

4) Complexity: The algorithm 2 contains three parts: (1) check for failure, (2) if failure happens then do fast retry, (3) if failure persists then start failover. As a worst case scenario, the failover procedure kicks-in that re-configures the interfaces. To reconfigure all interfaces, the algorithm needs to iterate through all interfaces / links. Hence, the complexity of algorithm 2 is O(n), where n represents the number of interfaces to be re-configured.

VII. Implementation

We use open source IMS platform (OpenIMS [12]) and open source cloud operating system (OpenStack [13]) to implement the functionalities of IMS protocol and NFV, respectively. OpenIMS provides basic implementation of IMS NFs and is

deployed over OpenStack platform. OpenStack provides full flexibility on how IMS NFs are managed on cloud platform by providing abstraction of common hardware resources through virtualization and meets compute, networking and storage demands of different IMS applications. We spent significant efforts to modify source code in both platforms to suite our needs.

A. State-of-the-Art Implementation of IMS

OpenIMS has coupled all IMS NFs by implementing them over single virtual machine (e.g. VMware [39]) 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 [40], [41]. We use default timers as specified by IMS and OpenStack documents [38], [42]. Our system configurations are based on the guidelines and parameters provided by 3GPP [42], OpenStack [38], and CISCO's IMS [43]. We consider this implementation as state-of-the-art vIMS with which we compare our design.

B. Implementation of Proposed vIMS

We exploits OpenIMS modular structure and adopt its implementation to our needs. We describe our efforts as below:

1) Defining Modules: OpenIMS provides basic IMS implementation where it does not implement all of the modules as defined by 3GPP specification [24]. We modify OpenIMS source code to add many more modules (such as breaking incoming and outgoing connection through *Incoming/Outgoing Session Control Leg* modules). We achieve transition between these modules when one functional module calls other functional module in a chain of SIP messages.

2) 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.

Irrespective of packet processing job, *Media Update* module requests control instructions for a batch of predicted metadata over a fixed time interval (implemented as 2 seconds), and generates an event of new control instructions only when the predicted metadata causes different sets of control instruction. *Media Processor* receives new control instructions through a callback function.

3) Failure Detection Procedure: We detect failure by implementing finite state machine (FSM) (as shown in Fig. 8). In FSM implementation an operation must start from an initial state and transit to another accepted state. To achieve this, we create FSM transition table that transits from a given state 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). By doing so, the proposed FSM only executes on necessary functional module.

4) Fail-Over Procedure: To successfully execute fail-over procedure, we are required to immediately resume IMS operation by (1) isolating faulty module, and (2) performing failover by reconfiguring interfaces. To achieve these goals, we keep track of on-going device session before fault using a hash table to store/retrieve user's session information. 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 hash table. Then in-service module updates the network configurations at incoming and outgoing interfaces and contacts neighboring modules of failed module using on-going request identities. Thereafter, the in-service module takes charge of failed module's operations and resumes the IMS service.

VIII. Evaluation

We evaluate how our proposed design reduce latencies and improve vIMS fault tolerance. The state-of-the art vIMS described in Section VII-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 Open IMS Core. Each IMS NF is implemented over a separate server to make cluser of VMs in OpenStack.

We now present experimental results showing how our proposed vIMS reduce latencies and improve fault tolerance.

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 the first second, 2 calls are dialed (2 calls per second), then 3 calls per second and so on, upto 300 calls per second. This experiment is inaccordance to the experiment we show in Empirical Study section (Section V-A).

In state-of-the-art design, as the call rate increases the media-plane latencies start increasing, as shown in Fig. 9. 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. 9 shows that proposed design significantly reduces media-plane latencies. These latencies remain under 50 msec even when MRF is processing around 40,000 simultaneous

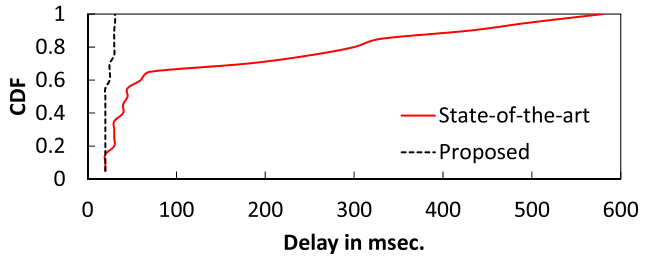
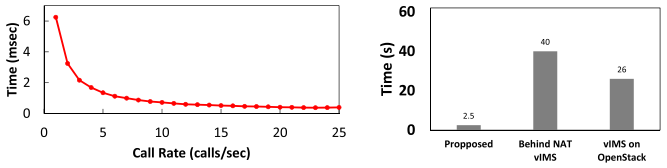


Fig. 9. Comparing state-of-the-art vIMS media plane latencies with proposed vIMS design under increasing call rate. Proposed design reduces latencies by pipelining media plane execution with control instructions. Loops between control and media plane incurs latencies in current state-of-the-art vIMS implementation.



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

Fig. 10. (a) Increasing call rate means there are always more number of SIP messages flowing through IMS NFs modules. This helps to quickly start failure detection procedure in case next hop does not respond to SIP message request; (b) Proposed design takes a couple of seconds to recover from the failure, i.e. time taken in detecting failure and reconfiguring the modules by isolating failed module. Behind NAT vIMS NFs and OpenStack use keep-alive mechanism to detect the failure. IMS without OpenStack implementatio requires human intervention to hangup media plane.

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* over a separate interface, without meddling media packets execution. *Media Processor* module keeps only one copy of control instructions and updates it as soon as it receives new control instructions (calculated based on packet metadata prediction).

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.

1) 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 after the failure has triggered. 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. Therefore, we are interested to know how quickly failure is detected under control-plane operation. Fig. 10a 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.

2) Failure Detection: Failure detection procedure starts as soon as failure detection mechanism triggers, i.e. SIP message is received at IMS NF module that transitions to *SIP Received* state (refer Fig. 8). Failure detection in our proposed IMS, Behind NAT IMS (that uses Session Traversal Utilities for NAT (STUN) protocol [44]), 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 [45]. Note that, OpenStack can only detect the failure of those modules that connect two NFs and cannot detect internal modules failure. This is because OpenStack monitors the working of different NFs through heart-beat mechanism and does not monitor every individual module.

3) Failover: Once the failure is detected, the failover procedure starts. Fig. 10b shows how much time different systems take to get back to service. Our proposed design simply reconfigures modules and loads the faild module's executable, and takes only 500 msec to recover from failure after failure detection (that takes 2 seconds). 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).

4) System Load: We also tested failure detection and recovery during IMS SIP signalling load in our design, as shown in Fig. 11. Increasing call rate generates increasing number of SIP messages, loading IMS towards its capacity. However, the failure detection and recovery delays upto few seconds on peak call rate which is acceptable because the system is going to reach its capacity anyway. This delay mainly comes from thread contention for system resources where triggering of failure detection timer (Timer B) is slightly delayed, and overloaded nighboring modules also respond late during failover procedure.

C. Discussion

In operational network, both IMS NFs and LTE NFs report total number of bytes they have processed to Policy and Charging (PCRF) NF. Then PCRF applies operator specific algorithm that takes total number of bytes processed at LTE and IMS as inputs in order to calculate subscriber's billing record. We discover that discussed failure triggers discrepancies under two different scenarios at charging system (i.e. PCRF). In first scenario, originating device can successfully transmit its UL media packets towards terminating

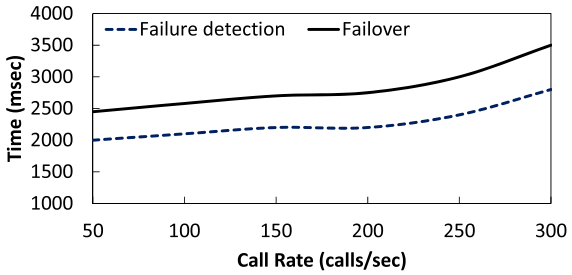
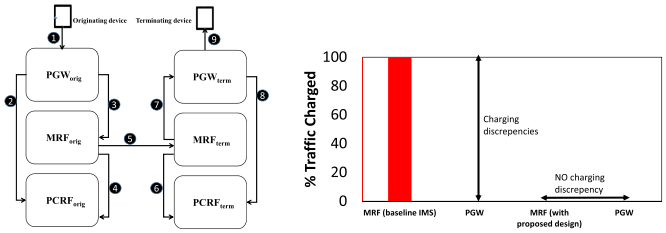


Fig. 11. Failure detection and failover time during system load. As the call rate increases, more number of SIP messages traverse through IMS NFs modules. This reflects a scenario of SIP signalling processing load on different modules. However, failure detection and failover time slightly increases during SIP signalling load.



(a) NFs traversal **(b)** Charging result discrepancies

Fig. 12. (a) Step by step procedure describing NFs traversal when originating device initiates call to terminating device. In case of discussed failure, originating device will be charged even if it does not receive any packet from terminating device (speech mute issue). (b) IMS failure causes charging report discrepancies at PCRF NF. IMS reports number of bytes it processed, whereas PGW does not report any charging info because it has dropped all the packets. Our design eliminates such charging discrepancies.

device's MRF (from steps 1 to 5 in Fig. 12a). For these UL data packets the subscriber is charged by the PCRF when both PGW (step 2) and MRF (step 4) correctly report charging records to PCRF. However, originating device never receives any DL media packets and is billed even during speech mute issue.

In the second scenario, MRF at terminating device network processes media packets and reports charging record to its PCRF (step 6). It forwards media packets to PGW that drops the packets because the user has de-registered from the network after failure. Therefore the PGW does not initiate any charging report to PCRF NF. This issue can potentially results into user overcharging because PCRF has processed charging record for those media packets that terminating device have never received. We show this in Fig. 12b where MRF in-state-of-the-art IMS implementation (i.e. in baseline IMS) charges all DL packets but PGW does not charge any packet. This results into potential charging discrepancies at PCRF. In contrast, our design does not cause any discrepancy as it quickly recovers from failure.

IX. Related Work

We compare our work against recent efforts on NFV, vIMS and middle boxes' fault tolerance space.

A. NFV

Reference [46] provides general purpose NFV platform. References [47] and [48] make use of software and hardware choices to meet specific service demands. References [49] and [50] 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. References [51]–[53] discuss algorithmic approaches to solve network slicing problem. Authors propose solutions for choosing optimal number of virtual instances to meet the requirements of mobile traffic. References [54] and [55] solve NF placement problem for 5G networks. In contrast to NF slicing and NF placement works, this paper addresses the reliability issues in NFV.

B. vIMS

Our work [56] and [57] discuss fault tolerance issues in vIMS. Reference [56] shows that highly modular 3GPP standardized vIMS network functions incur latencies and are susceptible to failures. To address these issues, we let IMS modules to reconfigure in real time. Reference [57] proposes modular redundancy design to address fail stop failure. In contrast, this paper addresses fault tolerance and low latency issues in NFV by providing vIMS as a use case. Reference [58] provides dynamic resource allocation algorithm for vIMS, [58] discusses merits of deployment strategies of vIMS. Reference [59] enhances vIMS features for M2M. But these efforts do not discuss latency and fault tolerance aspects in vIMS.

C. Fault Tolerance

References [17] and [60] propose logging NF states during normal operations and reconstructing them after a failure. Their approaches cannot address real-time and transitory NF sessions recovery. References [46], [61], and [62] discuss fault tolerance in non-IMS (SIP based) voice over IP applications. Reference [63] discusses general load balancing strategies in vIMS and does not discuss vIMS working during faults.

D. Latency

References [64] and [65] imply dynamically scheduling schemes to meet changing traffic demands in NFV. References [66] and [67] propose mobile edge computing designs to reduce latencies. Reference [68] proposes trading off latency with other performance metrics. All of above approaches fail to reduce system latencies which come from virtualizing system and its components interactions.

X. Conclusion and Future Work

Through IMS case study, we show that highly modular virtualized system 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.

From this work, we gain deep system insights and identify other research issues – to be discussed in future work. Few of them include (1) multiple modules failure, as well as complete NF failure cases, (2) improving prediction with high variance of media metadata (i.e. device mobility and its presence in low radio signal strength areas), and (3) studying security issues when some modules may skip important security modules during failure.

REFERENCES

[1] *What is NFV*. Accessed: Aug. 2018. [Online]. Available: https://www.etsi.org/images/files/ETSITechnologyLeaflets/NetworkFunctions Virtualization.pdf

[2] *Virtual Solutions for Your NFV Environment*. Accessed: Aug. 2018. [Online]. Available: https://www.f5.com/pdf/solution-center/network-functions-virtualization-nfv-solution-overview.pdf

[3] (2013). *Survey: NFV Expected to Deliver OPEX, CAPEX Benefits*. Accessed: Aug. 2018. [Online]. Available: https://www.fiercewireless.com/tech/survey-nfv-expected-to-deliver-opex-capex-benefits

[4] (2015). *NFV, Initiative to Reduce Network Expenses (CAPEX & OPEX Both)*. Accessed: Aug. 2018. [Online]. Available: https://www.linkedin.com/pulse/nfv-initiative-reduce-expenses-capex-opex-both-apoorvgupta/

[5] (2012). *Ericsson Blade System (EBS) for IMS*. Accessed: Aug. 2018. [Online]. Available: http://archive.ericsson.net/service/internet/picov/get?DocNo=266/03819-FAP130506&Lang=AE&HighestFree=Y

[6] (2017). *HP Blade System*. Accessed: Aug. 2018. [Online]. Available: https://h20195.www2.hpe.com/v2/getpdf.aspx/4aa4-8125enw.pdf

[7] (2018). *Build Massively Scalable Soft Real-Time Systems*. Accessed: Aug. 2018. [Online]. Available: http://www.erlang.org/

[8] (2011). *N-Version Programming*. Accessed: Aug. 2018. [Online]. Available: http://www.inf.pucrs.br/zorzo/cs/n-versionprogramming.pdf

[9] (2016). *VoLTE Subscribers to Total 310 Million Come Year-End*. Accessed: Aug. 2018. [Online]. Available: https://www.telecompetitor.com/report-volte-subscribers-to-total-310-million-come-yearend/

[10] *Policy and Charging Control Architecture*, document TS 23.203, 3GPP, 2013.

[11] (2018). *3GPP Specification Series*. Accessed: Aug. 2018. [Online]. Available: http://www.3gpp.org/dynareport/24-series.htm

[12] (2015). *OpenIMS—The Open Source IMS Core Project*. Accessed: Aug. 2018. [Online]. Available: http://www.openimscore.org/

[13] *OpenStack Open Source Cloud Computing Software*. Accessed: Aug. 2018. [Online]. Available: https://www.openstack.org/software/

[14] (2004). *Quality of Service Design Overview*. Accessed: Aug. 2018. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=357102

[15] (2018). *Forecasting With Single Exponential Smoothing*. Accessed: Aug. 2018. [Online]. Available: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc432.htm

[16] *NFV and the Fear of Failure*. Accessed: Aug. 2018. [Online]. Available: http://virtualizedworld.com/2016/09/nfv-and-the-fear-of-failure

[17] J. Sherry *et al.*, "Rollback-recovery for middleboxes," in *Proc. ACM SIGCOMM*, 2015, pp. 227–240.

[18] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2015, pp. 255–260.

[19] M.-A. Kourtis *et al.*, "Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 74–78.

[20] *AT&T Raises SDN Network Transformation Goal to 55% for 2017*. Accessed: Aug. 2018. [Online]. Available: https://www.fiercetelecom.com/telecom/at-t-raises-sdn-network-transformation-goalto-55-for-2017

[21] *DoCoMo to Offer NFV-Based LTE in 2016*. Accessed: Aug. 2018. [Online]. Available: https://www.lightreading.com/carrier-sdn/nfv-(network-functions-virtualization)/docomoto-offer-nfv-based-lte-in-2016-/d/d-id/709223

[22] (2014). *High Availability is More Than Five Nines*. Accessed: Aug. 2018. [Online]. Available: https://archive.ericsson.net/service/internet/picov/get?DocNo=10/28701-FGB101256&Lang=EN&HighestFree=Y

[23] *IMS Release 10 Tutorial*. Accessed: Aug. 2018. [Online]. Available: http://disi.unitn.it/locigno/didattica/AdNet/10-11/IMS_Tutorial_Scalisi.pdf

[24] *List of IMS-Related Specifications*. Accessed: Aug. 2018. [Online]. Available: https://www.onlinelibrary.wiley.com/doi/pdf/10.1002/9780470695135.app1

[25] (2016). *VoLTE Global Status—46 VoLTE Systems Launched*. Accessed: Aug. 2018. [Online]. Available: https://gsacom.com/paper/volte-global-status-30-volte-systems-launched-2/

[26] *Call Volume Statistics*. Accessed: Aug. 2018. [Online]. Available: http://www.arcep.fr/index.php?id=10519&L=1

[27] *LTE QoS Baiscs*. Accessed: Aug. 2018. [Online]. Available: http://www.simpletechpost.com/2013/01/quality-of-service-qos-in-lte.html

[28] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*. Hoboken, NJ, USA: Wiley, 2012.

[29] *Media Server Control Using the IP Multimedia (IM); Core Network (CN) Subsystem*, document TR 24.880, 3GPP, 2008.

[30] (2011). *Call Control Extensible Markup Language*. Accessed: Aug. 2018. [Online]. Available: https://www.w3.org/TR/ccxml/

[31] (2004). *Voice Extensible Markup Language*. Accessed: Aug. 2018. [Online]. Available: https://www.w3.org/TR/voicexml20/

[32] *Internet Protocol (IP) Multimedia Call Control Protocol Based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP)*, document TS 124.229, 3GPP, 2014.

[33] C.-Y. Li *et al.*, "Insecurity of voice solution VoLTE in LTE mobile networks," in *Proc. ACM CCS*, 2015, pp. 316–327.

[34] *Virtual Router Redundancy Protocol (VRRP) Using Keepalived to Provide Quick Failover of Layer-3 Services*. Accessed: Aug. 2018. [Online]. Available: https://docs.openstack.org/liberty/networking-guide/scenario-l3ha-ovs.html

[35] *Alcatel-Lucent End-to-End IMS Solution*. Accessed: Mar. 2016. [Online]. Available: https://www.alcatel-lucent.com/solutions/communications-collaboration

[36] *High Reliability Using ATCA Platform for IMS*. Accessed: Mar. 2016. [Online]. Available: http://www1.huawei.com/en/products/core-network/singlecore/ims-core/index.htm

[37] *Fusion Programming*. Accessed: Aug. 2018. [Online]. Available: http://www.huawei.com/ilink/en/download/HW_327323

[38] *Default Openstack Timers*. [Online]. Available: http://docs.openstack.org/kilo/config-reference/content/cinder-conf-changes-kilo.html

[39] *VMware Virtualization for Desktops & Servers*. Accessed: Aug. 2018. [Online]. Available: http://www.vmware.com/

[40] A. Bernstein, "Availability for network function virtualization," Cisco, San Jose, CA, USA, Tech. Rep. 43, 2015.

[41] *Wireless Network Virtualization: Ensuring Carrier Grade Availability*. Accessed: Aug. 2018. [Online]. Available: https://www.sdxcentral.com/wp-content/uploads/2014/10/Wind-River_CRAN-white-paper.pdf

[42] *IMS Network Testing: IMS Configurations and Benchmarks*, document TS186.008–2, 3GPP, 2007.

[43] *Cisco ASR 5000 Series Command Line Interface Reference*. Accessed: Aug. 2018. [Online]. Available: http://www.cisco.com/c/dam/en/us/td/docs/wireless/asr_5000/10_0/OL-22948-01_CLI_Reference.pdf/

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

[45] (2017). *Apache ZooKeeper*. Accessed: Aug. 2018. [Online]. Available: https://zookeeper.apache.org/

[46] S. Palkar *et al.*, "E2: A framework for NFV applications," in *Proc. ACM SOSP*, 2015, pp. 121–136.

[47] R. Mahindra, H. Viswanathan, K. Sundaresan, M. Y. Arslan, and S. Rangarajan, "A practical traffic management system for integrated LTE-WiFi networks," in *Proc. ACM Mobicom*, 2014, pp. 189–200.

[48] S. Palkar *et al.*, "E2: A framework for NFV applications," in *Proc. ACM SOSP*, 2015, pp. 121–136.

[49] M. T. Raza, D. Kim, K.-H. Kim, S. Lu, and M. Gerla, "Rethinking LTE network functions virtualization," in *Proc. IEEE ICNP*, Oct. 2017, pp. 1–10.

[50] I. F. Akyildiz, S.-C. Lin, and P. Wang, "Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: An overview and qualitative evaluation," *Comput. Netw.*, vol. 93, pp. 66–79, Dec. 2015.

[51] M. Bagaa, T. Taleb, A. Laghrissi, A. Ksentini, and H. Flinck, "Coalitional game for the creation of efficient virtual core network slices in 5G mobile systems," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 469–484, Mar. 2018.

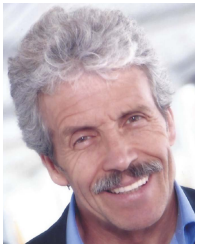
[52] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, "Optimal VNFs placement in CDN slicing over multi-cloud environment," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 616–627, Mar. 2018.
[53] A. Laghrissi, T. Taleb, and M. Bagaa, "Conformal mapping for optimal network slice planning based on canonical domains," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 519–528, Mar. 2018.
[54] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: The controller placement problem," in *Proc. IEEE GLOBECOMM*, Dec. 2016, pp. 1–6.
[55] T. Taleb, M. Bagaa, and A. Ksentini, "User mobility-aware virtual network function placement for virtual 5G network infrastructure," in *Proc. IEEE ICC*, Jun. 2015, pp. 3879–3884.
[56] M. T. Raza and S. Lu, "Enabling low latency and high reliability for IMS-NFV," in *Proc. ACM/IEEE/IFIP CNSM*, Nov. 2017, pp. 1–9.
[57] M. T. Raza, H.-Y. Tseng, C. Li, and S. Lu, "Modular redundancy for cloud based IMS robustness," in *Proc. ACM MobiWac*, 2017, pp. 75–82.
[58] G. Carella *et al.*, "Cloudified IP multimedia subsystem (IMS) for network function virtualization (NFV)-based architectures," in *Proc. IEEE ISCC*, Jun. 2014, pp. 1–6.
[59] M. Abu-Lebdeh, J. Sahoo, R. Glitho, and C. W. Tchouati, "Cloudifying the 3GPP IP multimedia subsystem for 4G and beyond: A survey," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 91–97, Jan. 2016.
[60] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico replication: A high availability framework for middleboxes," in *Proc. ACM Symp. Cloud Comput.*, 2013, Art. no. 1.
[61] M. Bozinovski, L. Gavrilovska, and R. Prasad, "Fault-tolerant SIP-based call control system," *Electron. Lett.*, vol. 39, no. 2, pp. 254–256, Jan. 2003.
[62] H. Pant, A. R. McGee, U. Chandrashekhar, and S. H. Richman, "Optimal availability and security for IMS-based VoIP networks," *Bell Labs Tech. J.*, vol. 11, no. 3, pp. 211–223, 2006.
[63] F. Lu, H. Pan, X. Lei, X. Liao, and H. Jin, "A virtualization-based cloud infrastructure for IMS core network," in *Proc. IEEE CloudCom*, Dec. 2013, pp. 25–32.
[64] P. Duan, Q. Li, Y. Jiang, and S.-T. Xia, "Toward latency-aware dynamic middlebox scheduling," in *Proc. IEEE ICCCN*, Aug. 2015, pp. 1–8.
[65] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Proc. IEEE NetSoft*, Apr. 2015, pp. 1–9.
[66] B. Yang, W. K. Chai, G. Pavlou, and K. V. Katsaros, "Seamless support of low latency mobile applications with NFV-enabled mobile edge-cloud," in *Proc. IEEE CloudNet*, Oct. 2016, pp. 136–141.
[67] D. Lake, "Enhancing mobile services with edge computing capabilities," *IEEE Softwarization*, Mar. 2016.
[68] R. A. Hamada, H. S. Ali, and M. I. Abdalla, "An IMS-based LTE-WiMAX-WLAN architecture with efficient mobility management," in *Proc. IEEE MELECON*, Apr. 2016, pp. 1–6.

Muhammad Taqi Raza received the master's degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 2017, where he is currently pursuing the Ph.D. degree with the Computer Science Department.

His research interests include network security, network function virtualization, and networked systems and their applications with an emphasis on cellular security.

Songwu Lu is currently a Professor with the Computer Science Department, University of California at Los Angeles, Los Angeles, CA, USA. His research interests cover wireless networking, mobile systems, sensor networks, and data center networking.

Mr. Lu was on the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING, and was on the Boards of the IEEE TRANSACTIONS ON MOBILE COMPUTING, *ACM Wireless Networks*, and the *IEEE Wireless Communications Magazine*.



Mario Gerla received the Engineering degree from the Politecnico di Milano, Milan, Italy, in 1966, and the M.S. and Ph.D. degrees in engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 1970 and 1973, respectively.

From 1973 to 1976, he was with Network Analysis Corporation, New York, NY, USA. He is currently a Professor with the Faculty of the Department of Computer Science, UCLA. He has designed and implemented various network protocols (channel access, clustering, routing, and transport) under the Defense Advanced Research Projects Agency and the National Science Foundation grants. His research interests include distributed computer communication systems and wireless networks. His work is cited over 21 000 with an h-index of 118.

Xi Li is currently an Associate Professor and the Vice Dean with the School of Software Engineering, University of Science and Technology of China. He has been directing research programs in the Embedded System Lab, examining various aspects of embedded system with the focus on performance, availability, flexibility, and energy efficiency.

He has led several national key projects, several national 863 projects, and NSFC projects. He is a member of the ACM and a Senior Member of CCF.