Flexible Protocol Reconfiguration for Emergency Communication Systems

Ji Zhang, Hai Li, and Shujuan Hou

The necessity for services of emergency communication systems to be uninterrupted and reliable has brought forward strict requirements for express software upgrades of base stations. Based on reconfigurability technology, this paper presents a protocol reconfiguration scheme for emergency communication base stations. By introducing the concept of 'local safe state,' the proposed scheme could make the updating and replacement of the protocol software flexible and maintain interactions during the updating procedure to achieve seamless system upgrades. Further, taking TETRA protocol stack and VxWorks operating system as an example, specific processes and realization methods of reconfiguration are proposed in this paper, along with the measurements of the factors impacting on the system performance. Simulation results show that, compared with traditional technology, the method proposed significantly improves the system efficiency and does not interrupt system services.

Keywords: Reconfiguration, emergency communication, safe state, reconfiguration management plane, component, VxWorks.

I. Introduction

The emergency mobile communication system not only provides all the features of common mobile communication systems, but also provides the function of command and dispatching, so it has been widely used in many fields, such as railways, airports, harbors, and gas fields. In order to respond to emergency events, the emergency mobile communication system has strict requirements for uninterrupted serving. However, when the maintenance of a base station (BS) of an emergency communication system is required, including updates or replacements of the protocol software, the BS is forced to stop the activities of all the components until the system is updated and rebooted. During this period, responses to new service requests are impossible, and the current traffic stream has to be interrupted. So, the reliability of the system is undoubtedly reduced.

At present, many researchers focus on reconfiguration technology in order to implement protocol software maintenance, such as changing the network of a device due to its mobility, changing routing algorithms of switches, and adding or changing the security module in protocol stacks [1]. One of the important aspects for protocol reconfiguration is the decomposition of protocol functionality into components able to be reassembled dynamically [2]. Several designs of protocol component framework have been proposed. In [3], a framework has been proposed for creating, removing, and replacing protocol modules at runtime. The framework preserves the module state as a data structure to manage the existing connections. When reaching a safe reconfiguration point, the whole system will be locked until the reconfiguration is finished. The DiPS framework [4] aims at the development of industrial protocol stacks from some

Manuscript received Jan. 26, 2010; revised Aug. 12, 2010; accepted Aug. 26, 2010.

Ji Zhang (phone: +86 10 68918326, email: izpursuer@gmail.com), Hai Li (corresponding author, email: haili@bit.edu.cn), and Shujuan Hou (email: shujuanhou@bit.edu.cn) are with the School of Electronics and Information, Beijing Institute of Technology, Beijing, China.

doi:10.4218/etrij.11.1510.0008

components which mainly concentrates on how to describe protocol building blocks and external requirements and how to build a stack from these descriptions. However, the performance issues are not mentioned. A UML profile for protocol components and protocol reconfiguration is proposed in [5], but only the structural design is provided. Niamanesh and Jalili [6] proposed the dynamic-reconfigurable architecture for a protocol stack which is a software framework for the dynamic reconfiguration of two communicating protocol stacks with a distributed algorithm, but during the updating procedure, no interactions between two communicating stacks are allowed. The adaptive multilevel code update protocol [7] enables energy-efficient code updates via support for multilevel protocols, such as full-image, module-level, function-level, and instruction-level. It adaptively selects a protocol which meets the deadline of applications and, based on a cost analysis of several protocols, consumes less energy. A reference model for a multimode protocol [8] is based on the idea of identification of commonalities between multiple envisaged modes and a separation of the common and the mode-specific parts of the protocol, incorporating the ideas of functional partitioning and reconfiguration of layers to achieve both horizontal and vertical convergence between multiple modes. Two more practical prototypes of switching between different radio access technologies using protocol reconfiguration are given in [9] and [10], but the technologies of these papers are more suitable for altering the whole software, not updating partial modules.

In all of the above methods, the system will be frozen in a global safe state in order to update the software. When the system enters the safe state, no interactions between mobile stations (MSs) and BSs are allowed. So, these methods will inevitably result in a long period of service interruption which is intolerable to emergency communication systems. In order to overcome the shortcomings of these methods, this paper presents a component-based dynamic reconfiguration method for uninterrupted protocol software maintenance. We recommend that each component has its own safe state, and when a component is updated, other unchanged components will maintain their interactions in their local safe state.

The rest of this paper is organized as follows. Section II describes the protocol stack model and the proposed protocol reconfiguration method. Section III introduces an implementation of a reconfigurable terrestrial trunked radio (TETRA) protocol stack based on VxWorks platform. Section IV analyses the impact of the reconfiguration process over the system performance through the experimental results. The last section presents our conclusion and intentions for future work.

II. Protocol Reconfiguration Method

1. Component-Based Protocol Stack Model

The component-based design is one of the key enabling technologies for designing reconfigurable software [5]. The reconfigurable protocol stack can be divided into a public framework and several functional components. The public framework provides the basic functions of the protocol stack, which cannot be configured at runtime. The functional components, which support dynamic reconfiguration, are extracted from the protocol stack to form a series of independent modules according to specific functions. All of the functional components are installed inside the public framework to realize the whole functionality of a specified protocol, guaranteeing the new component could be configured in a proper way to achieve dynamic reconfiguration.

After the definition of the protocol component, it is necessary to build the reconfiguration management plane (RMP) to manage the corresponding functional entities during reconfiguration. RMP could implement the uninstallation and reconfiguration of functional components as well as the extraction and recovery of the system's running status. RMP is the platform proposed to embody the functions and interfaces enabling reconfiguration, which provides layer abstractions to applications and services on one hand, and to terminal equipment and network devices on the other [11], [12].

Figure 1 shows the structure of the protocol stack with RMP. The reconfiguration procedure will be completed under the control of RMP.

2. RMP

The functional entities of RMP consist of the context management (CM), policy provision (PP), software download management (SDM), and reconfiguration management (RM). The function of each module is described as follows:

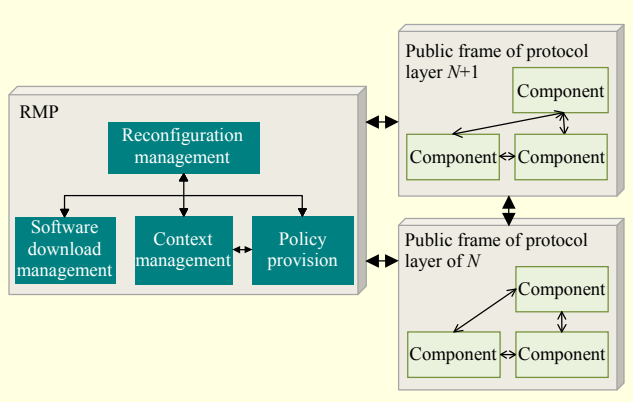

Fig. 1. Protocol stack model with RMP.

CM. This module monitors and updates the contextual information of a running system, including service status and other information related to resources which will be used for the establishment of reconfiguration strategy as well as the state restoration after reconfiguration. In addition, this entity will maintain the local component metadata, expressing and circulating reconfigurable device capabilities, and semantic properties of reconfigurable protocol stacks [13], [14], which is used to verify the validity of a component.

PP. This module verifies the validity of the downloaded component by using the information offered by the CM module and forms the reconfiguration policy. The reconfiguration policy determines which services should be suspended and which system status should be saved during reconfiguration and informs the RM module about the services and interfaces provided by the downloaded components. The system will finish the reconfiguration process in terms of the policy formed by this entity.

SDM. This module downloads and verifies the integrity of the component. After that, the component will be stored in the local file system.

RM. This module monitors the reconfiguration command from core network and manages the entire process of reconfiguration according to the policy made by the PP module as well as coordinates the normal signaling and traffic stream during reconfiguration.

RMP is outside of the functional entities of the protocol stack, but the reconfiguration support function exists in all protocol sublayers. By communicating with the functional components, RMP could obtain the current system running state, constitute the reconfiguration policy, and avoid system restart so as to realize the smooth transition of service after components upgrade.

3. Local Safe State with Interactions

Communicating components should be frozen in a safe state, or suspending state [15], before updating. A safe state for a reconfiguration has been defined as a state that has no interaction with the other components [6]. However, for an emergency communication system, the components in a BS cannot interact with an MS in the safe state so the MS cannot receive enough downlink PDUs in a term. As a result, the MS will think the BS cannot be reached and then initialize cell reselection so the link between the MS and the BS will be broken even when the signal is still strong enough. The BS therefore needs to transmit some PDUs to maintain the link in the safe state.

The concept of a local safe state is proposed in this paper. When a component is updated, RM will send a freeze message to all associated components with a parameter indicating which component will be updated. The components receiving the freeze message will enter their own local safe state. Each component can have different behavior in a local safe state.

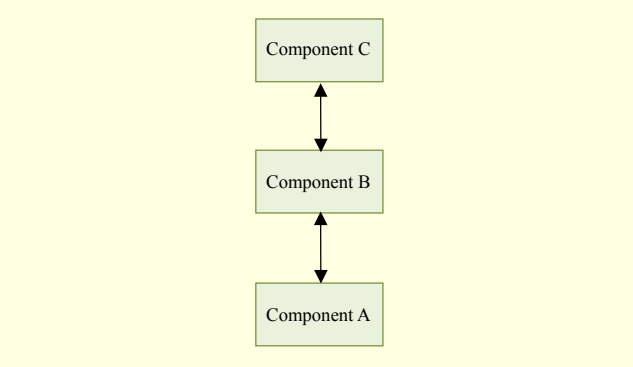

Fig. 2. Layered components.

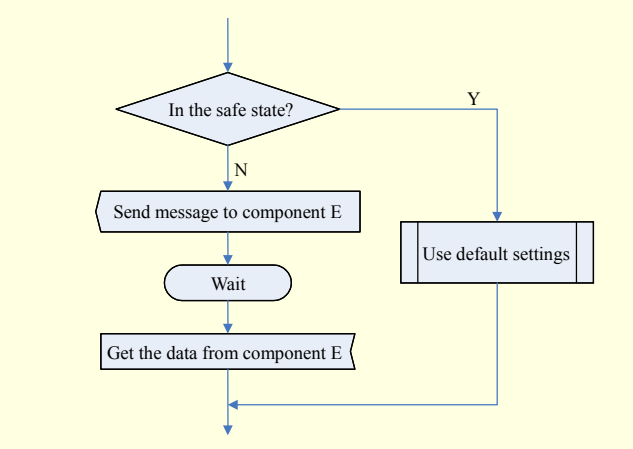

Fig. 3. Data source checking.

Next, two common scenarios are used to explain the behavior design of local safe state.

Scenario 1. Three components are layered as in Fig. 2. If component C is updating, the interactions between components A and B will still be executed just like normal, and all the messages sent to component C should be buffered until the update of the component C is done.

Scenario 2. Component D will retrieve data from component E, but if component E is missing, default settings can also be used by component D. In this scenario, component D can choose the proper data source according to the current state, as depicted in Fig. 3.

When reconfiguration ability is added to a protocol stack, the code should be changed to maintain the minimum function of a module when other modules are updating. From this point of view, it is better to have loosely coupled components when designing a component-based protocol stack model.

4. Process of Protocol Components Reconfiguration

Now the protocol stack can be reconfigured based on the defined protocol components and RMP. The process consists of a downloading procedure and updating procedure.

A. Downloading Procedure of a Component

The downloading procedure of a component is initiated by the network management (NM). Figure 4 shows the detailed steps of this procedure. The downloading procedure includes the following steps:

Step 1. The NM sends a reconfiguration request.

Step 2. Once receiving the reconfiguration request, the RM returns a reconfiguration response.

Step 3. The NM then sends the metadata of the component being replaced.

Step 4. After receiving the whole metadata, the RM sends a policy provision request to the PP.

Step 5. The PP compares the local metadata in the CM with the downloaded metadata. If the component being replaced is not in the list of the local metadata, or the version is older than the current component, the RM should report the NM that the reconfiguration is invalid.

Step 6. If the validity of the component is confirmed by the PP, it should formulate the reconfiguration policy in terms of the metadata and system operation state.

Step 7. The PP provides the RM with reconfiguration policy.

Step 8. The RM informs the SDM to initiate the downloading of component software.

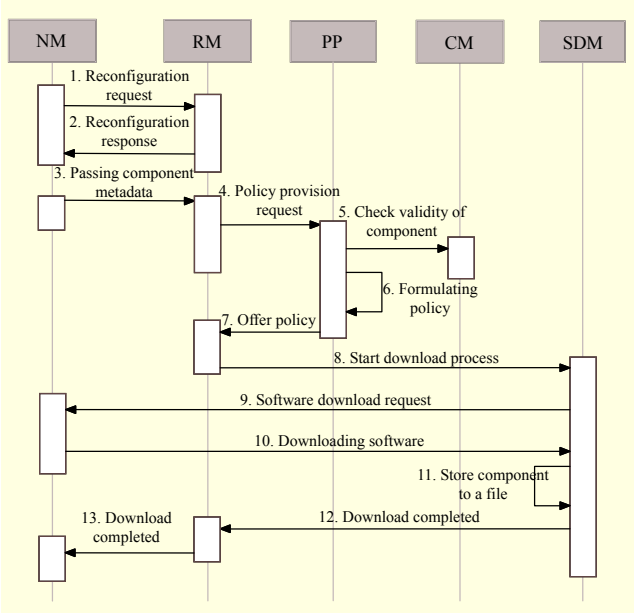

Fig. 4. Sequence diagram illustrating downloading process of component.

Step 9. The SDM sends a download request to the NM.

Step 10. After the download request is confirmed, the NM begins to transmit the component software package to the BS.

Step 11. After receiving the whole package, the SDM checks the correctness and save it to the local file system.

Step 12. The SDM informs the RM of the download completion after storage.

Step 13. The RM informs the NM that software downloading is completed and then starts the component updating process.

B. Updating Procedure of a Component

When receiving the indication of download completion from the SDM module, the RM module will initiate the updating procedure. Figure 5 shows the detailed steps of this procedure. The updating procedure includes the following steps:

Step 1. The RM sends a freeze request to all components and then all components will enter the local safe state. In the local safe state, each component will maintain the minimum function.

Step 2. After all components enter their own safe state, the RM uninstalls the old component.

Step 3. After uninstallation, the RM saves all of the system state information to the CM.

Step 4. The RM loads a new component from the file system. If any error occurs during loading, the RM recovers the original component and informs the NM of the reconfiguration failure.

Step 5. If loading is successful, the RM recovers the system state from the CM. If the data representations of the old and new components differ, the state transfer must comprise a translation between old and new components' data representations.

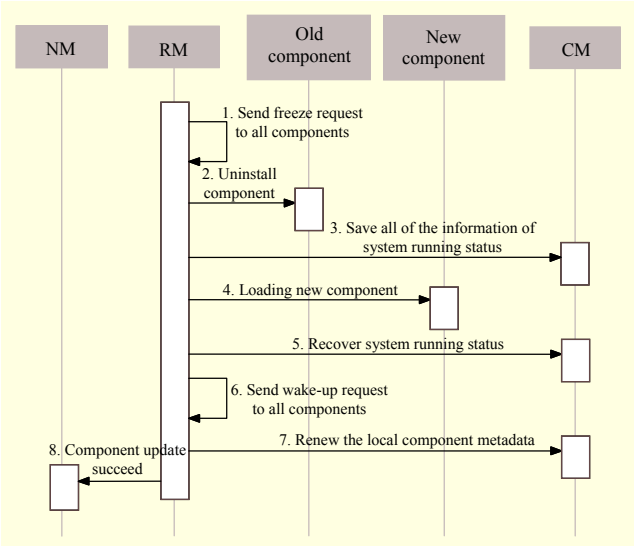

Fig. 5. Sequence diagram illustrating updating process of component.

Step 6. After recovery of the system operation state, the RM sends a wake-up request to all components so that they can exit the local safe state to support the full features of the system.

Step 7. The RM then informs the CM to renew the local component metadata.

Step 8. Finally, the RM informs the NM of the completion of reconfiguration.

III. Implementation of Protocol Component Reconfiguration

For clarity, we focus on the TETRA system [16] to introduce the proposed method. It should be noted that the reconfiguration scheme proposed here is not limited to the TETRA system and can also be used for other wireless communication systems, such as UMTS and WiMax. One reason that we choose the TETRA system is that it has a stricter requirement of continual services than other systems. The TETRA protocol stack is implemented on VxWorks. As a platform of protocol processing, VxWorks meets the stability and real-time requirements of digital trunked BS perfectly. The method discussed here is not confined to VxWorks but applicable to all operating systems supporting dynamic module loading.

1. Reconfigurable Protocol Stack of TETRA

The protocol architecture of TETRA air interface (AI) follows the generic layered structure which includes three layers: the physical layer, the data link layer, and the network layer.

Here we use layer 2 (L2) of TETRA AI protocol stack as an example to illustrate the component-based protocol stack model. Layer 2 may be further subdivided into the medium access control (MAC) and logical link control (LLC). The MAC sublayer should handle the problem of sharing the medium by a number of users. The LLC sublayer resides above the MAC and is responsible for controlling the logical link between an MS and a BS over a single radio hop.

The public framework of MAC sublayer includes the general MAC function, channel state module, and layer management. The main function of each part is defined as follows:

- The general MAC function is responsible for interaction with the hardware platform, handling the encoding and decoding of PDUs, synchronization of frames, and multiplexing of logic channel.
- The channel state module maintains the allocation status and access status of physical channels. As a core module of MAC public framework, there are interfaces between channel state module and all functional components.
- Layer management uses TMC-SAP for the transfer of management information inside each layer, which is not transferred over the AI.

There are six functional components of the MAC sublayer. The name and function of each one are as below:

Random access control component. This component takes charge of the procedure of random access and determines the access permission for each slot. The component could modify the access parameters dynamically and inform the broadcast control component of these modifications.

Reserved access control component. When an MS has further signaling to send after the initial access, the BS may reserve the slots for the MS. This component takes charge of the procedure of reserved access and the reconstruction of fragmented service data unit from LLC. It shall inform the channel state module to update the channel access status after slot reservation.

Broadcast control component. The BS sends broadcast information including cell synchronization and network information.

Traffic transmission control component. It forwards the voice or circuit mode data traffic to the destination MSs according to the traffic routing table and can also process the stealing signaling carried in the voice payload.

Channel assignment component. This component handles channel allocation and release by communicating with the channel state module to update the current channel usage.

Power control component. This component involves itself in the closed-loop power control with the signal strength measurement of L1, which is the basis of cell reselection.

Using the same method, the public framework of LLC is defined as follows:

- Logic link management shall handle the establishment, maintenance, and release of the basic link and advanced link and forward the signaling from the L3.
- Formatter management deals with the encoding and decoding of PDUs and finds the right route for the primitives from the MAC sublayer.
- Layer management is responsible for the management of the LLC layer.

The functional components of LLC are defined as follows:

Basic link control component. This component, composed of the link control module, basic link acknowledged module, and basic link unacknowledged module, is used for transmission of acknowledged data and unacknowledged data.

Acknowledged advanced link control component. This component is composed of a link control module, receiver, and

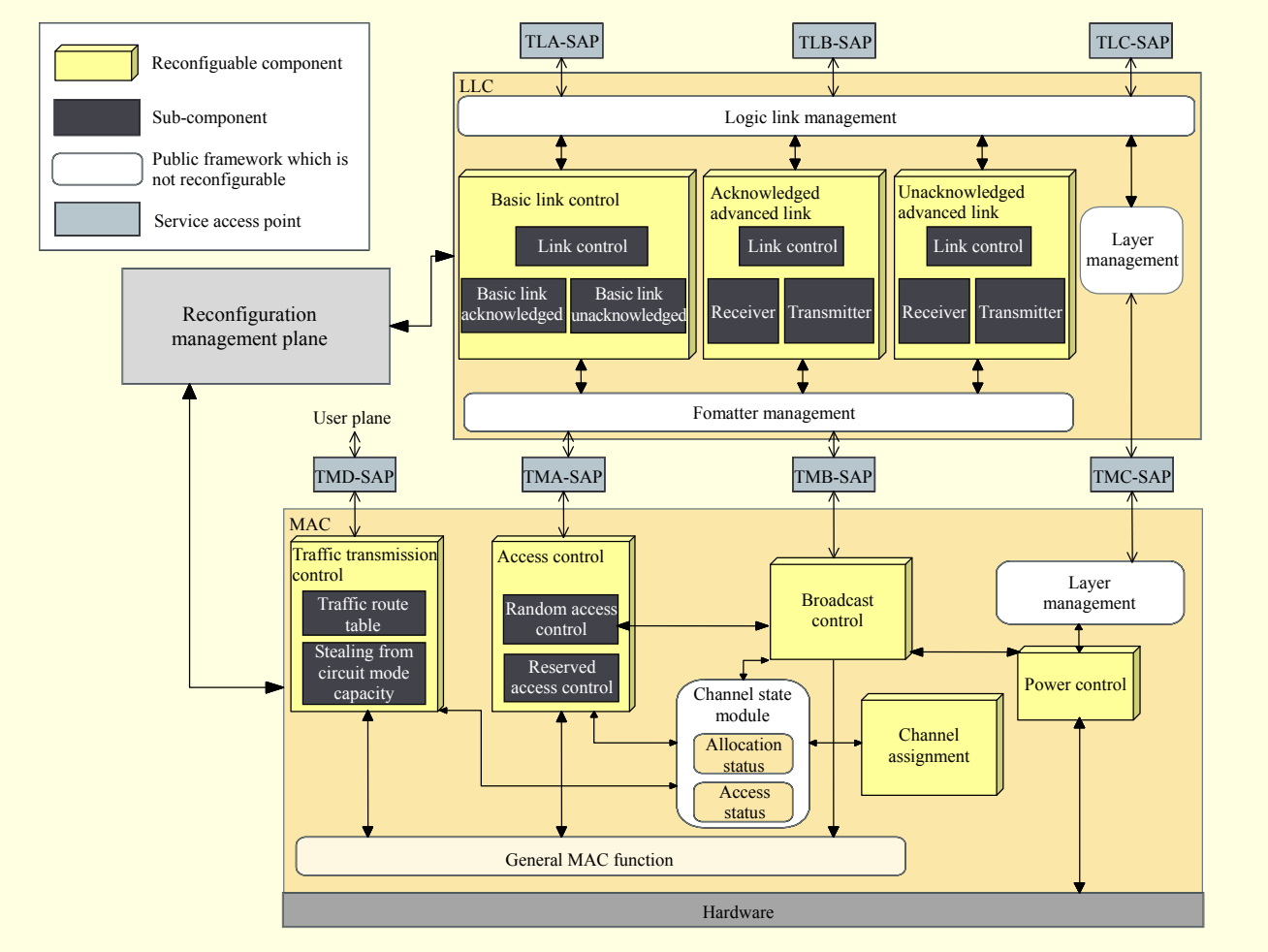

Fig. 6. Reconfigurable L2 protocol stack of TETRA.

transmitter, providing better quality of service than the basic link for the transmission of packet data.

Unacknowledged advanced link control component. This component has the same structure as the acknowledged advanced link control component.

The model of component-based L2 protocol stack of TETRA AI is illustrated in Fig. 6.

2. The Configuration of VxWorks

The process of protocol reconfiguration involves software downloading, storage, and installation. VxWorks permits a user to add code to a target system at runtime. This operation, allowing users to install target applications or to extend the operating system itself, provides the underlying support for reconfiguration, which makes it possible for the BS to achieve expansion and upgrading without hardware modifications.

VxWorks should be configured to support the process:

i) Add network components to the VxWorks kernel. TCP/IP provides a reliable guarantee for the downloading of AI protocol components.

ii) Configure the VxWorks kernel with the TrueFFS file system by including the core component in BSP, INCLUDE_TFFS. After checking the integrality, the SDM will save the downloaded component to TrueFFS which is reentrant, thread-safe, and supported on all CPU architectures that host VxWorks. Call the sysTffsFormat() routine to format the FLASH at the first time of system startup and use the usrTffsConfig() routine to mount the VxWorks DOS file system on a TrueFFS flash drive before the initialization of the AI protocol stack when system restarts.

iii) Configure the VxWorks kernel with a target-resident system symbol table by including INCLUDE_STANDALONE_SYM_TBL to dynamically identify the new modules. When a new component is installed in the OS kernel, the name and address information of variables and functions will be added into the symbol table for VxWorks to search for the service entrance of the component.

iv) Configure VxWorks with the INCLUDE_LOADER component and the INCLUDE_UNLOADER component to include loader and unloader for the dynamic installation and removal of protocol components. In the process of reconfiguration, the RM calls the unldByModuleId() routine to unload the old protocol component and calls loadModule() to install the new one from TrueFFS. Finally, the RM calls symFindByName() to search in the symbol table for the entrance function address of the new component.

3. Behavior of Local Safe State

When a component is updating, the access status component in the public frame of the MAC layer will prevent all of the random access requests on the main control channel (MCCH) to avoid channel allocation caused by the new call setup. As explained in section II, the BS therefore needs to transmit some PDUs to maintain the link between the BS and MS in the safe state. The BS will continue to send NULL PDU in the MCCH and set proper ACCESS-ASSIGN PDU in the access assignment channel to abandon all MSs' random access attempts. Although the MS cannot make a random access attempt when the serving BS is in a safe state, the MS will not leave the BS.

If the traffic transmission control component is not being updated, we can even keep all existing voice calls uninterrupted. Because the traffic transmission control component is seldom altered, this safe state strategy will be very useful for the emergency communication system.

Furthermore, we can design a similar strategy to provide more interactions in the safe state to minimize the effect on the system features during the procedure of protocol components reconfiguration.

IV. Performance Assessment

1. Factors Affecting Performance

The downloading procedure of the components is executed by an independent thread running in the background of the OS which does not affect the features of the original system, whereas the updating procedure will freeze the new service requests until the component is installed successfully and the system state is restored. Although this process could maintain the continuity of the original service, it has to cause some delay in the establishment of new services. There are three main factors that affect the reconfiguration performance:

i) The performance of the hardware platform, including the CPU processing power, network transmission speed, and reading and writing speed of FLASH.

ii) The task scheduling ability of OS.

iii) The granularity of the protocol component, which is a problem in accordance with the features of the protocol stack for module division. The granularity determines the code size of the components as well as the complexity of realization and the coupling degree with the external environment.

2. Performance Testing

The reconfiguration process time is measured on the hardware platform composed of Intel PXA270 processor at 416 MHz running VxWorks 5.5.

Service delay caused by a component upgrade consists of three parts: unloading of old component (t_U), loading of new component from the FLASH (t_L), and dynamic linking with the OS kernel (t_K).

Table 1 shows the mean value of service delay with different component sizes and different complexity. One hundred experiments were done for each component size. A larger size code takes more time to load from the FLASH, while the complexity of the component affects the unloading process as well as the dynamic linking process with the OS kernel. Because there is no reasonable and common standard to measure the software complexity, this item is not set in the table. The relationship between service delay and component complexity will be studied further in next step.

Figure 7 shows the time consumption of each part of service delay with different component sizes and almost the same complexity by adding lots of useless and duplicate codes. We can conclude that the time of linking with the OS kernel contributes the most time to the total delay. The unloading time is stable when simply increasing the code size without changing the complexity, but the loading time of new component is determined by the code size only and has nothing to do with the complexity. The dynamic linking time with the OS kernel is influenced by both of the factors. Regarding a module of 1 MB (protocol component or the whole protocol

Table 1. Service delay caused by component update process.

Size of protocol component (B)	Service delay (ms)			
	Unloading of old component (t_U)	Copy component from TFFS to SDRAM (t_L)	Link to the OS kernel (t_K)	Total time of component upgrade
24k	4.8	0.6	7.7	13.1
66k	5.0	2.3	9.7	17.0
256k	7.5	11.4	54.0	72.9
1,410k	80.3	62.9	862.7	1005.9

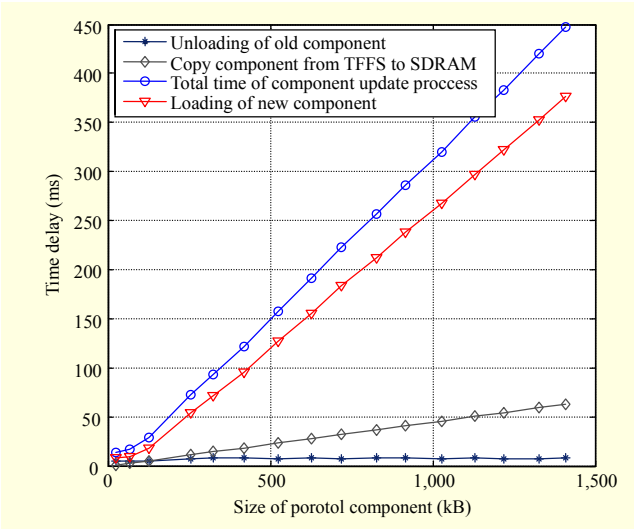

Fig. 7. Service delay caused by component update process.

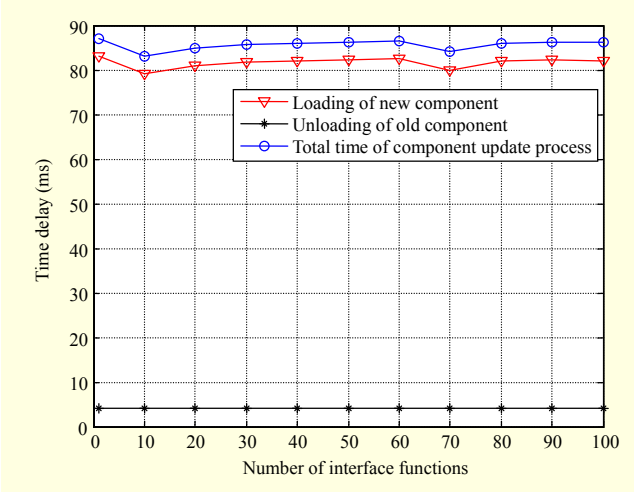

Fig. 8. Service delay with the number of interface functions.

stack), the reconfiguration time will generally not exceed 1 s. However, if the dynamic reconfiguration technology is not used, the total time of system restarting, including the establishment of the multitask environment of VxWorks kernel, loading of AI protocol stack, initialization of protocol software, and sending of a broadcast message, is approximately 26 s. So, the reconfiguration method proposed in this paper greatly improves the efficiency of the system. Unlike the method introduced in [3] and [6], the system can still provide minimum function services, such as broadcast and voice forward, when the components are updating. Even when the broadcast control component needs to be updated, the service delay is less than the value of the timer RADIO_DOWNLINK_TIMEOUT defined in [11] so that the MS will not leave the current serving cell.

We also tested the effect of the number of interface functions of a component on the services delay. The experiment result is shown in Fig. 8. From this result, the number of interface functions will not significantly affect the performance, so we can ignore this factor during the interface design of a component.

V. Conclusion

Protocol reconfiguration technology provides a comprehensive reconfigurable capability to a communication system related to all aspects of network architecture and all levels of protocol standard. This paper presented a solution for base station protocol reconfiguration of an emergency communication system by creating a component-based protocol framework and introducing the reconfiguration management plane into the original protocol stack. The solution makes full use of the characteristics of supporting dynamic loading of applications to reduce the maintenance costs of the base station and achieve remote upgrade protocol stack flexibility while ensuring smooth service transition. The main framework and process of protocol components reconfiguration can also be applied to other communication stacks. However, the behavior of each module in the safe state should be carefully designed according to respective protocols.

Further research will include improving the public framework of the protocol and deciding the granularity of components as well as the mechanism for verifying component's legitimacy and integrity.

References

[1] M. Niamanesh and R. Jalili, "A Dynamic-Reconfigurable Architecture for Protocol Stacks of Networked Systems," *Proc. 31st Annual Int. Computer Software Appl. Conf.*, vol. 1, 2007, pp. 609-612.

[2] E. Patouni, N. Alonistioti, and P. Magdalinos, "A Framework for Protocol Reconfiguration," *Proc. Seventh IFIP Int. Conf. Mobile Wireless Commun. Netw.*, 2005. http://www.ctr.kcl.ac.uk/MWCN2005/Paper/C200543.pdf

[3] Y. Lee and R. Chang, "Developing Dynamic-Reconfigurable Communication Protocol Stacks Using Java," *Software Practice Experience*, vol. 6, 2005, pp. 601-620.

[4] S. Michiels et al. "DiPS: A Unifying Approach for Developing System Software," *Proc. 8th Workshop Hot Topics Operating Syst.*, 2001, pp. 175-179.

[5] F. Foukalas et al., "Protocol Reconfiguration Using Component-Based Design," *Proc. IFIP Int. Federation Inf. Process*, 2005, pp. 148-156.

[6] M. Niamanesh and R. Jalili, "DRAPS: A Framework for Dynamic Reconfigurable Protocol Stacks," *J. Inf. Sci. Eng.*, vol. 25, 2009, pp. 827-841.

[7] S. Yi et al., "Adaptive Multilevel Code Update Protocol for Real-Time Sensor Operating Systems," *IEEE Trans. Ind. Informatics*, vol. 4, no. 4, 2008, pp. 250-260.

[8] L. Berlemannet al., "Reconfigurable Multi-Mode Protocol Reference Model for Optimized Mode Convergence," *Proc. European Wireless Conf.*, 2005, vol. 1, 2005, pp. 280-286.

[9] E.S. Cho et al., "SCA-Based Reconfigurable Base Station System," *Proc. IEEE Global Telecommun. Conf.*, 2006, pp. 1-4.

[10] W. König et al., "Reconfigurable Base Station Processing and Resource Allocation," *Proc. 16th IST Mobile Wireless Summit*, 2007, pp. 1-5.

[11] Z. Boufidis, N. Alonistioti, and M. Dillinger, "Adaptive Network Control and Management for Beyond 3G End-to-End Reconfiguration," *Proc. 14th IST Mobile Wireless Commun. Summit*, 2005. http://www.eurasip.org/Proceedings/Ext/IST05/papers/416.pdf

[12] N. Olaziregi, Z. Boufidis, and E. Mohyeldin, White Paper, *Management and Control Architecture of Reconfigurable Systems*, Wireless World Research Forum Work Group 6 White Paper, 2005. http://www.wireless-world-research.org/fileadmin/sites/default/files/about_the_forum/WG/WG6/White%20Paper/WG6_WP6.pdf

[13] V. Gazis, N. Alonistioti, and L. Merakos, "A Generic Model for Reconfigurable Protocol Stacks in Beyond 3G," *IEEE Wireless Commun.*, vol. 13, no. 3, 2006, pp. 70-78.

[14] N. Alonistioti, E. Patouni, and V. Gazis, "Generic Architecture and Mechanisms for Protocol Reconfiguration," *Mobile Netw. Appl.*, vol. 11, No. 6, 2006, pp. 917-934.

[15] M. Castaldi et al., "A Lightweight Infrastructure for Reconfiguring Applications," *LNCS 2649*, 2003, pp. 231-244.

[16] ETSI EN 300 392-2 V2.4.2, *Terrestrial Trunked Radio (TETRA); Voice plus Data (V+D); Part 2: Air Interface(AI)*, Feb. 2004.

Ji Zhang received his BS and MS in information and communication engineering from Beijing Institute of Technology in 2008 and 2010, respectively. His current research interests include embedded system design, wireless protocol, mobile backhaul, and ALL-IP Networks.

Hai Li received his BS and PhD in information and communication engineering from Beijing Institute of Technology in 1997 and 2002, respectively. He has authored or co-authored over 30 papers published in scientific journals or conference proceedings. His current research interests include embedded system design and communication protocol engineering. He is currently working at Beijing Institute of Technology as an associate professor.

Shujuan Hou received her PhD in signal and information processing from Beijing Institute of Technology in 2005. Currently, she is an associate professor of signal and information processing at Beijing Institute of Technology. Her main research interest is signal processing of mobile communication systems.