Contents lists available at Science-Gate

International Journal of Advanced and Applied Sciences

Journal homepage: http://www.science-gate.com/IJAAS.html

Adaptive byzantine fault tolerance support for agent oriented systems: The BDARX

Ayesha Batool Alvi [1], Muhammad Adnan Hashmi [1, *], Zishan Hussain Chuhan [1], Muhammad Atif [1], Ijaz Ahmed [2]

[1]*Department of Computer Science and Information Technology, The University of Lahore, Lahore, Pakistan*
[2]*Higher Colleges of Technology, Dubai, United Arab Emirates*

A R T I C L E I N F O

Article history:
Received 6 June 2018
Received in revised form
16 December 2018
Accepted 19 December 2018

Keywords:
Multi-agent systems
Fault tolerance
BDARX
DARX

A B S T R A C T

Multi-agent systems (MAS) with fault tolerance capabilities have got much attention during the recent years. Many fault-tolerance mechanisms have been proposed. Dynamic agent replication scheme (DARX) architecture is one of the most studied fault-tolerance architectures for multi-agent systems. It deals with adaptive dynamic replication schemes to make agent systems more fault tolerant, but it does not handle Byzantine faults in MAS environments. This paper proposes Byzantine DARX (BDARX) architecture which endows DARX with the ability to handle Byzantine faults, and thus it allows the building of complex software systems to deal with arbitrary faults and make the system more reliable and efficient. The efficiency of the proposed architecture is demonstrated in an application domain of vehicular ad-hoc networks (VANET).

1. Introduction

Over the years, multi-agent systems (MAS) have emerged as an appropriate paradigm for the development of complex intelligent systems. In MAS, different agents coordinate and communicate with each other to attain certain goals and solve problems which are beyond the capabilities of any individual agent. These systems are widely used in diverse kinds of applications like unmanned autonomous vehicles (Patron et al., 2008), air traffic control, smart grids and other areas (Khalili et al., 2018).

These systems need to run smoothly without any failure even in the presence of faulty behavior of different agents (Briot et al., 2007). In a large scale MAS, failures ratio grows with the number of agents, hosts and duration required for agents' task execution (Stanković et al., 2017). It is very difficult to recognize faulty agents in advance to avoid their crashes (Ductor et al., 2011), so there is strong need for fault tolerance schemes in such systems. Fault tolerance is one of the major challenges in multi-agent systems. It is crucial to design a fault tolerance structure for large scale multi agent systems which can detect and resolve failures and provide continuity of processing.

Adopting corrective and preventive measures are two key approaches for achieving fault tolerance. When a host crashes due to sudden disruption or disconnection from the network then this kind of situation can be handled by applying fixes at algorithmic, architectural or platform level. In these situations, multi-agent systems are usually equipped with a defense mechanism. An efficient and proved way to attain fault tolerance in such systems is the application of replication strategies. For large scale and open multi-agent systems, resource aware adaptive replication has been rarely addressed. This paper tries to fill this gap by proposing an adaptive replication strategy that provides fault tolerance under certain resource limits.

Some initial work, to make multi-agent systems more tolerant to faults, uses the approach of providing redundancy in the system at different levels. Tolerance is provided in two different patterns i.e. non-critical agent and soft response time. In the former, the system may still be operational even if one or more agents fail permanently, while in the latter, a time required for processing of global functions allows for some variability.

Dynamic agent replication extension (DARX) (Almeida et al., 2007) is architecture for achieving adaptive fault tolerance in MAS environments. Currently, it does not handle the occurrence of arbitrary byzantine faults. The main objective of this

* Corresponding Author.
Email Address: muhammad.adnan@cs.uol.edu.pk (M. A. Hashmi)
https://doi.org/10.21833/ijaas.2019.02.009
Corresponding author's ORCID profile:
https://orcid.org/0000-0003-4844-5853

research is to extend the design of current DARX replication architecture to support, detect and handle byzantine faults. Our proposed extension makes existing DARX framework more reliable and byzantine faults tolerant for complex distributed MAS environments.

Rest of the paper is structured as follows. In Section 2, existing approaches towards fault tolerance are discussed. Section 3 presents detailed overview of DARX architecture that this paper extends. Section 4 explains the proposed extension BDARX. Section 5 demonstrates the application of BDARX in the domain of VANET. Finally, section 6 concludes the paper.

2. Related works

Intelligent agents based software systems work in open environments with the ability to behave autonomously in dynamically changing contexts (Genza and Mighele, 2013). Due to the inherent distributed nature, these systems face many problems like faults due to synchronization issues, run time node failures, security threats and denial of services attacks. The understanding of such kind of issues in dynamic distributed environments and learning of how to manage them is a challenging task. Software engineering discipline provides structures and techniques to manage complexity and faults in these systems to make them more faults tolerant. In the following, a review of existing fault-tolerance techniques in multi-agent systems is presented.

2.1. Chameleon

Chameleon (Kalbarczyk et al., 1999) is an adaptive fault tolerance system that functions using reliable mobile agents.

It works by using embodied techniques based on a special set of agents that are sustained by a fault tolerance manager and hosts. These hosts help in handshaking mechanism with the fault tolerance manager through agents.

A flexible fault tolerance implies the capability to dynamically meet the progressing fault tolerance needs of a software application. Chameleon attains this purpose by providing a re-configurable structure. The dynamic re-configuration of Chameleon allows the component functionalities to extend or change at run time. Components can be added or deleted from the system without disturbing other components in operation. But unfortunately, due to its centralized fault tolerance manager, this architecture has many problems as the other preceding methodologies in literature.

2.2. DimaX

DimaX (Faci et al., 2006) is a multi-agent fault tolerant architecture, which deals with failures occurred because of bugs and the ones due to machine crashes. But it does not handle uncontrolled byzantine failures. It provides fault-tolerance for multi-agent applications by means of replication techniques where replication framework is based on DARX architecture (Almeida et al., 2007). Several services like naming, fault detection and fault recovery are provided by DimaX. Its fault tolerance mechanism is based on DARX and DIMA (Guessoum and Briot, 1999). DIMA offers a set of libraries to form MAS. Its kernel is an organization of proactive mechanisms which embodies proactive and autonomous entities. At the same time DARX works as a middle-ware in the DIMAX platform. DARX and DIMA both provide components for communication, execution-control and naming but at diverse levels, that's why integration of these architectures requires a set of extra components.

At the application level, the DIMA messages are communicated by those agents who are transferred through the DARX middle-ware. Additionally, DARX offers a fault detection tool. It handles server machines crash failures in three steps inside every replication group. First step reveals ultimate failure in the group. Next step evaluates the context of failure and its criticality level. In the third step, recovery is done if the missing replica is a group leader. For this purpose, a new agent is elected as a monitor and is automatically activated. This relies on the assessment of a new follower or backup which may or may not be instantiated. However, if some leader is failing, which do not have any backup or follower then it cannot be recovered.

Throughout a distributed computation, the criticality of an agent changes during the process. A very high criticality level is given to that single agent who is responsible for combining the results at the start of an application. To validate the results of DimaX, multiple series of experiments are made. In the initial series, experiment evaluation is based on the performance of the proposed multi-agent architecture and the intended adaptation algorithms (Guessoum et al., 2005).

2.3. DARX: An inspiring model

DARX (Almeida et al., 2007) is an architecture which is intended to develop reliable, adaptive and scalable faults tolerant MAS. It employs dynamic adaptive replication methodologies for this purpose. It provides different active and passive strategies (Zubair and Manzoor, 2016).

DARX works as a stand-alone middle-ware so the majority of its fundamental ideology can be reused independently in other architectures too (Overeinder et al., 2003). The provision of decision making support for a comprehensive fault tolerance support for each agent as per its context and evolution is the main aspiration of DARX. It includes several generic classes, as it is based on Java framework, which helps developers during the procedure of implementing reliable multi-agent applications. Furthermore, JVM induces features of machine independence and portability to this architecture which is attractive for distributed systems. Additionally, the provision of remote method invocation (RMI) support enables suitable high level abstractions for allowing scalability of distributed solutions (Marin et al., 2001).

In order to provide long term fault tolerance support in open and distributed systems, DARX has multiple dimensions including nodes selection, level of fault tolerance support, and scalability of overall system. This model does not handle byzantine faults through its built in architecture.

2.4. Twin BFT: Virtual technology based fault tolerant system

Twin BFT (Dettoni et al., 2013) is a solution to provide byzantine fault tolerance. In this approach replication schemes are implemented to allow byzantine faults in omission tasks. This approach uses 2f+1 replicas to handle f faults at maximum. Two virtual machines are installed on a host machine which keep synchronizing with each other and act as a failure detector of each other. This architecture is also based on client-server model and implements fault tolerance at server level only.

The architecture of this model is given in Fig. 1. In this architecture, the communication among different hosts in the same node is performed through shared memory called postbox. Virtual machines (VM) at different nodes communicate through LAN. A different platform on each virtual machine within same node decreases the chances of simultaneous faults in both VMs. This model handles byzantine faults but in virtual domains and at server end only.

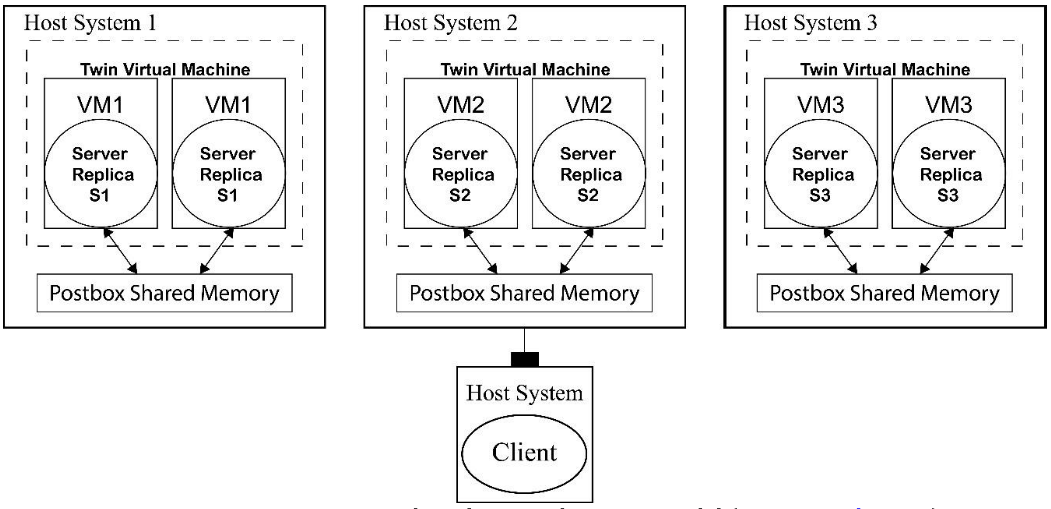

Fig. 1: Twin BFT: Twin virtual machines architecture model (Dettoni et al., 2013)

2.5. Vbam: Byzantine atomic multicast in LAN

Vbam protocol (Silva et al., 2013) is designed for fault tolerant communication among different client and server nodes in a virtualized environment. This protocol handles byzantine faults in communication, among servers and between clients and servers, using common virtualization and data sharing technologies as abstractions.

Fig. 2 shows the architecture of this protocol. It provides fault tolerant architecture for client-server nodes in virtualized environments. Every physical machine has a single virtual machine configured. A shared memory is configured among server nodes, which is called Distributed Shared Register (DSR). It maintains the sequence of messages between nodes and clients with the help of sequencer. This architecture does not handle autonomous entities for fault tolerant systems.

2.6. Reliable communication in dynamic network with byzantine faults

In this work, a byzantine problem is discussed where two network nodes want to communicate reliably with each other but some nodes, byzantine, in between have been compromised or are showing abnormal behavior.

With the help of constructive proof approach, optimal, sufficient and necessary conditions for reliable communication are proved. This work supports byzantine faults handling for dynamic network environments but lacks support for distributed multi-agent environments.

Another approach has been presented in (Araragi, 2006). This work is based on Castro and Liskov's (1999) byzantine fault tolerance method. Replication strategies are implemented without only being maintained at server level. Further, this model resolves synchronization issues of receiving messages among replicas.

2.7. Byzantine fault tolerance (BFT) for agent systems

3. The architecture of DARX

DARX consists of generic Java classes which provide support for implementing agents with fault tolerance capabilities. It consists of different components for failures detection, replication, fault handling and observation as shown in Fig. 3.

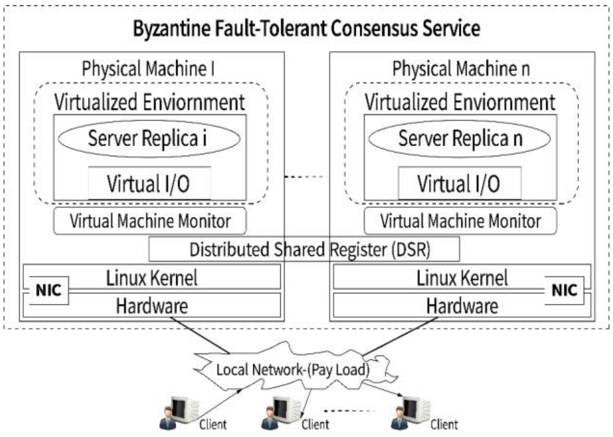

Fig. 2: Vbam architecture (Silva et al., 2013)

3.1. Failure detection and naming service

The primary role of naming and failure detection layer is to maintain a list of valid sites, agents and their replicas which participate in the application. Failure detectors and name servers are the components which play central role in fault detection. A separate autonomous thread exists in each DARX server which acts as a fault detector and name server.

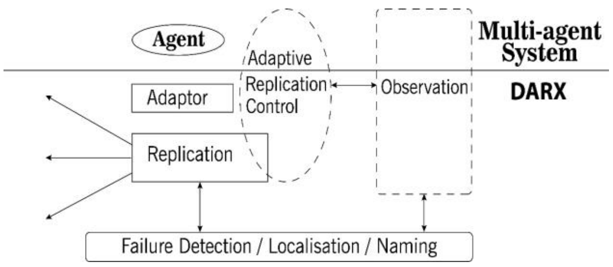

Fig. 3: Application architecture of DARX (Marin et al., 2003)

Physical faults in servers are detected by the periodic exchange of heart beat messages, while software faults are monitored through software processes. The agents are divided into two types of groups i.e. local and global groups. The name servers communicate with each other by piggybacks on heartbeats used for failure detection. In case of failure of a DARX server, all agents hosted by it are released and are replaced by replicas from other servers. New leaders from these replicas are elected as the failure notification is initiated from the naming server.

3.2. Observation service

A good level of knowledge regarding dynamic characteristics of applications and environments is gathered for making DARX fault tolerance process more efficient. An observation service is designed to gather this kind of knowledge. This service piggybacks on the flow of heartbeats emissions by failure detection service. Each local DARX server contains an integrated Observation Service module.

3.3. Replication management

Fault tolerance in DARX is attained through agent dependent software replication strategies. Different fault tolerance strategies exist including optimistic, pessimistic and the one where agents have no fault tolerance needs. The selection of these depends upon the evolution of criticality of agents over time. For this purpose, replication groups (RG) are maintained by DARX.

Each RG may have active or passive replication policies. In active replication policy, all replicas remain synchronized and participate in calculation processes, while in passive replication strategy only one replica remains active and keeps sending state information to other standby agents. Several replication policies may co-exist in the same RG.

A replication group can be described on the basis of following data:

- Criticality of related agents in a replication group.
- The degree of replication that how many replicas it contains.
- The list of replicas arranged according to the ordered capability of leadership.
- The list of all replication policies implemented within a replication group.
- The relationship mapping between replicas and the policies.

DARX allows for dynamic change of replication strategy due to which replicas and the replication strategies acting within a replication group can be added and removed. As a case for recovering some of the missing active replicas, the decision may be taken by activating the backup replica which is most suitable within the replication group. In case the backup replica is missed or crashed, then to sustain the reliability of replication group some new replica starts working. If the criticality value of some agent decreases then there is a possibility of either to suppress a replica or to change the policy of that replica from active to passive one. Fig. 4 depicts the architecture of a replica in DARX framework. It shows DARX message communication with RemoteTask. A TaskShell consists of a replication manager having a replication policy and communication with agent. Every agent inherits DARX task object to enable the functionalities of DARX. It is also wrapped by task shell to handle interaction of agents through inputs and outputs. An independent thread exists for every task shell and task manager (Marin et al., 2003).

4. The BDARX: extending DARX to handle byzantine faults

The proposed model is developed by making modifications into the DARX fault tolerance architecture for multi agent systems and by using the protocol of byzantine fault tolerance (BFT) method for agent systems. The BDARX architecture as shown in Fig. 5 helps in making failure model more reliable to deal with arbitrary faults like byzantine faults. However dealing with such faults is most challenging.

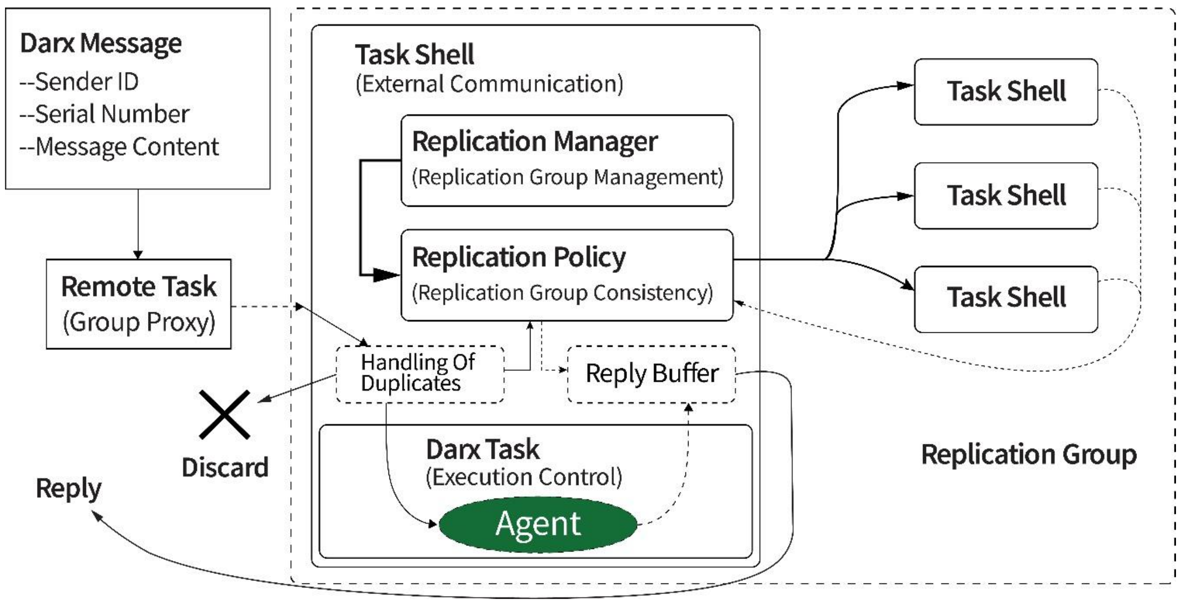

Fig. 4: Structure of a replica in DARX (Marin et al., 2003)

The basic assumption is *n = 3f+1 (f>=0)* is the least number of replicas that can suffer arbitrary byzantine attacks without failure of the whole system. It is also assumed that out of *3f+1* hosts, not more than *f* (as well as their replicas) are under attack and the remaining *2f+1* replicas uphold their original behavior. If, even out of *2f+1* replicas, *f* replicas are not communicating then the behavior of remaining *f+1* will be similar to the behavior of all the *2f+1* replicas. Because, under control agents might not indefinitely give results, so we can expect only *2f+1* results all the time and *f+1* in case the agents who are not faulty are not communicating.

Our approach modifies the DARX architecture by using above proposed replication strategy instead of choosing simple active and passive strategies with any number of replicas. Our bound of *3f+1* replicas ensure byzantine faults handling in BDARX through its integrated adaptive replication based architecture.

In BDARX, a failure detection service is used to save the valid software elements which contribute to the supported application and the dynamic lists of all the running hosts. It also notifies about the supposed failure occurrences. There are two features of agent platforms. Firstly, the agents do not have fixed global organization; they acquire and provide services to other agents. Secondly, agents are created and removed dynamically. Therefore these agents dynamically choose other agents that coordinate with each other to perform a task properly. Consequently agents show autonomous behaviour, they change their behavior with time and may also exit the system.

The BDARX model is an extension of DARX to deal with byzantine faults in which replicas are created on both sides of the communication parties. For enabling fault tolerance abilities, each agent inherits the properties of a DARX task object, which enables BDARX to control the agent's execution. DARX task is a Java object that handles and supervises agent execution like start, end, suspension and resumption. Every DARX task is itself enclosed in a task shell, which deals with inputs and outputs of agents. Consequently for agents BDARX can operate as a mediator which is responsible for creating an accurate decision for which messages should be released or received. Primary replicas are further wrapped in enhanced shells that consist of an additional replication manager which acts as an independent thread. It communicates information with observation module and performs time by time re-examination of current replication strategy. If any strategy modification is being done, replication manager sends replication policy updates to the other replicas, so it helps to maintain reliability of the replication group. Each replica uses replication policy to find out how the communication should be handled regarding internal as well as incoming and outgoing data related to a replication group.

In Fig. 5, BDARX architecture is presented which is an extension of DARX. It consists of DARX messages with three elements: sequence no, current primary (leader replica), and the request. These are passed to action module. The main module TaskShell consists of Replication Manager, Replication Policy and Agreement Protocol. This information is forwarded to action module. The action represents different states of message/request which is one of the following forms: null, send, add and read (request). As agents behave in a dynamic way, they sometimes act as a server and sometime as a client. So, it means that all agents have equal position. In BDARX, for handling byzantine faults, it is necessary to create replicas on both sending and receiving agents. After that the replication manger has to choose a replication strategy. Byzantine fault tolerance strategy consists of multiple states including three phase agreement protocol (pre-prepare state, prepare state and commit state), checkpoints and view-change.

Consider there are two agents α and β; as α sends message towards current primary/ leader replica of β, it starts sender timer clock, and if before expiry of sender timer clock of α, it receives *f+1* acknowledgements, the replica of α accepts those acknowledgements. It is necessary that every dedicated replica of receiving agent β must reply to replicas of sending agent α. By sending an acknowledgment message, in pre-prepare state, there will be no replica having multiple sequence numbers for the same request, therefore duplication cannot exist in sequence numbers.

As some replica receives *2f+1* valid pre-prepare messages from different replicas, it enters into prepare state and checks sequence number and request. Afterwards replica broadcasts a commit message following the contents of the prepare message. When some replica receives 2f*+1* valid commit messages from multiple replicas, it enters into commit state, and then it checks sequence number and request. Afterwards replica executes the request and returns outcomes to the user.

Checkpoints are useful for making sure that the request is being processed by fault free replicas. After receiving *2f+1* checkpoint messages, replicas maintain the state at which request is being executed and remove the information kept before request execution.

After receiving a message/request directly or indirectly from sender agents, every replica at receiving agent starts a timer, if that timer expires before request completion, the replicas try to change current primary or leader replica of the agent, this process is called view-change.

During communication between two agents, a replica cannot send a new message to any agent until the agent sends back an acknowledgement message for previously sent message, it works in FIFO order. View-change mechanism works as follows:

- Replica of α multicasts message to all the available replicas of β if it does not receive the same *f+1* acknowledgment from the replica of β prior to the expiry of its sender timer.
- When some replica of agent β gets the same *f+1* messages from various replicas of α prior to receiving message from the current primary, then its receiver timer starts and transmits the message to the current primary.
- In case receiver timer of replica of agent β expires, view-change procedure gets started.

Each message through BFT protocol is encrypted with a secret key from sender replica and is shared by the use of public keys.

Hence replicas cannot change messages received from others.

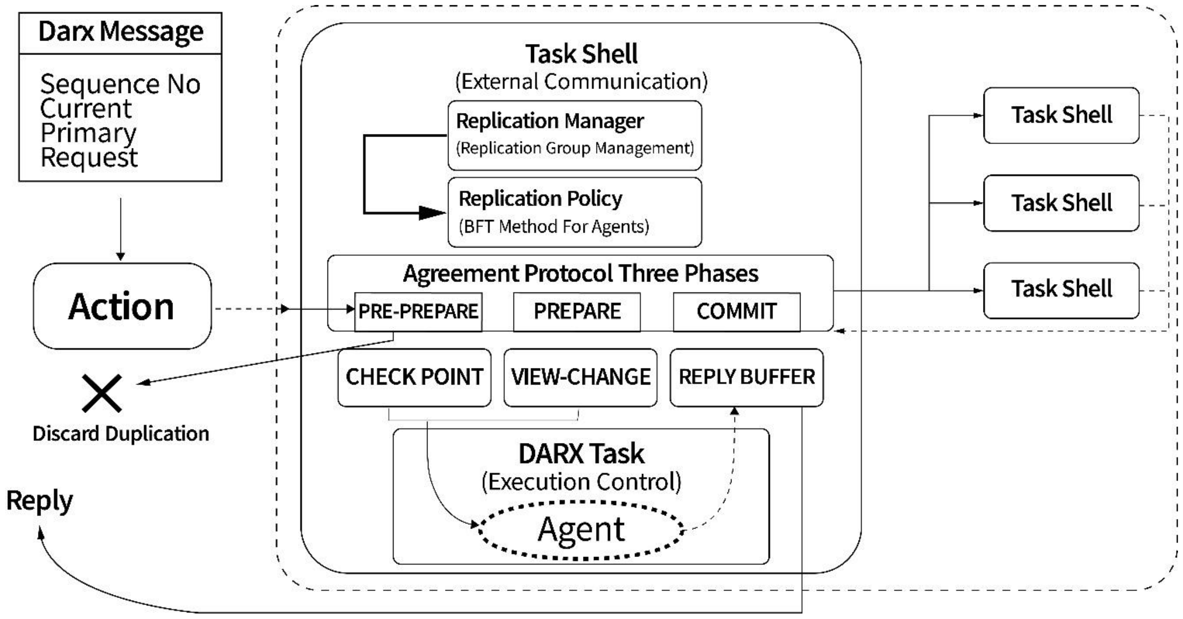

Fig. 5: BDARX replication architecture

5. Case study: VANET (Vehicular ad hoc network)

The vehicular adhoc network (VANET) is a wireless network of vehicles that communicate with each other (Samara and Al-Raba'nah, 2017). Such network helps vehicles in the specific range of 100 to 300 meters to share information on any ongoing activity. VANETs are based on high bandwidth wireless communication mechanism. They are used in different applications like lifeguard services, rescue services, infotainment and multi-media sharing services (Chahal et al., 2017). The better sharing system between rescue services can help to reduce the time to reach at some location and provide help at the earliest.

In VANETs, communication is done between road side unit to vehicle (R2V) and vehicle to vehicle (V2V) as shown in Fig. 6. This communication is being done as BDARX architecture. A short range communication system for safety and infotainment applications works for both R2V and V2V environments. VANETs are designed to have a high data transfer rate and minimize the latency rate in communication. Agent oriented systems may help these networks to work in the best way and provide fault tolerance in such situations so that things remain as much uninterrupted as possible. BDARX protocol can be applied in the area of VANET for handling different fault tolerance related issues.

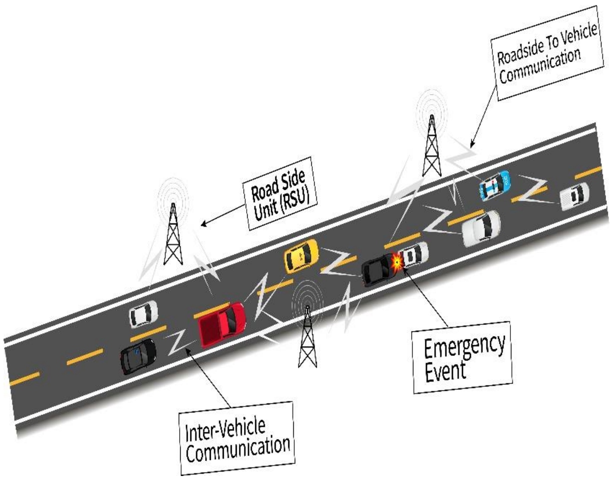
Roadside To Vehicle Communication
Road Side Unit (RSU)
Emergency Event
Inter-Vehicle Communication

Fig. 6: Communication patterns in VANET

Some of these issues are discussed below: Security is one of the major concerns in such open networks where nodes join and leave on the fly. In VANETs security becomes a prime issue because it deals with lifeguard services and any attack on the system can lead to life threatening vulnerable situations. BDARX adaptive replication system of nodes based on their criticality level ensures the level of security and replication for the nodes. In the following we have discussed a scenario of highest criticality where continuous status updates about the vehicles are required even if some byzantine fault occurs in the system.

We have taken the scenario of a cash delivery van on the road with security vans around it. The company monitors real time coordinates and status information of security van for any quick response in case of any emergency. In this extreme critical situation, BDARX provides solution for efficient handling of VANET. The software agent installed on cash van creates replicas on the security vehicles. In case of any attack on the van or hijacking of the network, where *f (>=0)* nodes are under control of attacker and the other *f* are not communicating, even then if company receives *f+1* messages about actual status, the company will respond accordingly. Such type of implementation ensures a strong fault tolerance mechanism for the network. The sequence diagram in Fig. 7, describes the discussed environment using BDARX architecture for better communication and fault handling. If main node 'Cash Van' faces some issues, its replicas play role for fault tolerance. Moreover, encrypted messages flowing among nodes ensure privacy of information on the open network. Checkpoints are established at different time intervals so in case of any link failure, communication data from last check points are restored and the whole system becomes alive in a short time.

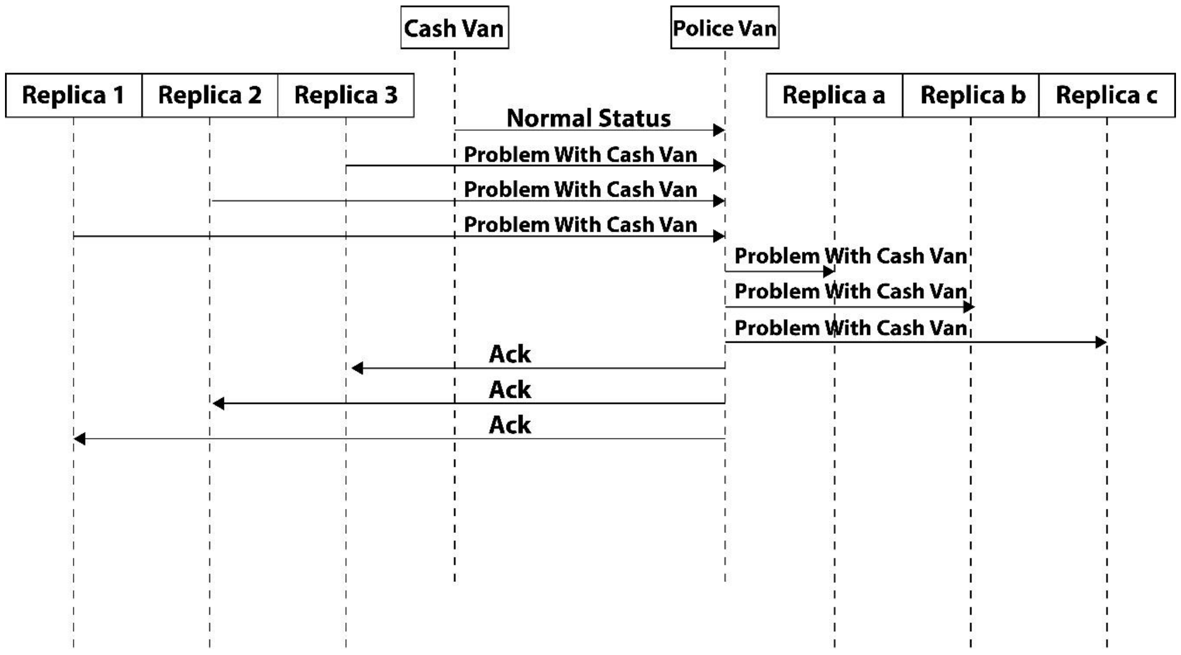

Fig. 7: Sequence diagram of VANET scenario

6. Conclusion

In this paper, we have proposed BDARX an extension of DARX architecture for handling byzantine faults in multi-agent systems. Existing DARX architecture deals with crash related faults in networks but does not deal with byzantine faults. We have induced byzantine support in DARX protocol. The proposed architecture deals with byzantine faults in more efficient manner. The effectiveness of proposed framework is demonstrated in the application domain of VANETs.

Currently, this work lacks support for complex distributed MAS architectures on different platforms. In future, we want to test our architecture on different environments with extended support for distributed multi-agent systems.

Compliance with ethical standards

Conflict of interest

The authors declare that they have no conflict of interest.

References

Almeida A, Briot JP, Aknine S, Guessoum Z, and Marin O (2007). Towards autonomic fault-tolerant multi-agent systems. In The 2nd Latin American Autonomic Computing Symposium, Petropolis, Brazil.

Araragi T (2006). Byzantine fault tolerance for agent systems. In the International Conference on Dependability of Computer Systems, IEEE, Szklarska Poreba, Poland: 232-239. https://doi.org/10.1109/DEPCOS-RELCOMEX.2006.11

Briot JP, Aknine S, Guessoum Z, Malenfant J, Marin O, Perrot JF, and Sens P (2007). Multi-agent systems and fault-tolerance: State of the art elements. Deliverable, FTCAT Project, EuroControl INO CARE III Programme, Brétigny-sur-Orge, France.

Castro M and Liskov B (1999). Practical byzantine fault tolerance. In the 3rd Symposium on Operating Systems Design and Implementation, New Orleans, USA, 99: 173-186.

Chahal M, Harit S, Mishra KK, Sangaiah AK, and Zheng Z (2017). A survey on software-defined networking in vehicular ad hoc networks: Challenges, applications and use cases. Sustainable Cities and Society, 35: 830-840. https://doi.org/10.1016/j.scs.2017.07.007

Dettoni F, Lung LC, and Luiz AF (2013). Using virtualization technology for fault-tolerant replication in LAN. In the New Results in Dependability and Computer Systems, Springer, Heidelberg, Berlin, Germany: 131-140.

Ductor S, Guessoum Z, and Ziane M (2011). Adaptive replication in fault-tolerant multi-agent systems. In the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, IEEE Computer Society, Washington, D.C., USA, 02: 304-307. https://doi.org/10.1109/WI-IAT.2011.206

Faci N, Guessoum Z, and Marin O (2006). DimaX: A fault-tolerant multi-agent platform. In the International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, ACM, Shanghai, China: 13-20. https://doi.org/10.1145/1138063.1138067

Genza N and Mighele E (2013). Review on multi-agent oriented software engineering implementation. International Journal of Computer and Information Technology, 2(3): 511-520.

Guessoum Z and Briot JP (1999). From active objects to autonomous agents. IEEE Concurrency, 7(3): 68-76. https://doi.org/10.1109/4434.788781

Guessoum Z, Faci N, and Briot JP (2005). Adaptive replication of large-scale multi-agent systems–towards a fault-tolerant multi-agent platform. In the International Workshop on Software Engineering for Large-Scale Multi-agent Systems, Springer, Berlin, Heidelberg, Germany: 238-253.

Kalbarczyk ZT, Iyer RK, Bagchi S, and Whisnant K (1999). Chameleon: A software infrastructure for adaptive fault tolerance. IEEE Transactions on Parallel and Distributed Systems, 10(6): 560-579. https://doi.org/10.1109/71.774907

Khalili M, Zhang X, Polycarpou MM, Parisini T, and Cao Y (2018). Distributed adaptive fault-tolerant control of uncertain multi-agent systems. Automatica, 87: 142-151. https://doi.org/10.1016/j.automatica.2017.09.002

Marin O, Bertier M, and Sens P (2003). DARX-a framework for the fault-tolerant support of agent software. In the 14th International Symposium on Software Reliability Engineering, IEEE, Denver, USA: 406-416.

Marin O, Sens P, Brio JP, and Guessoum Z (2001). Towards adaptive fault-tolerance for distributed multi-agent systems. In the 3rd European Research Seminar on Advanced Distributed Systems, Spring School and Workshop, Madeira Island, Portugal: 195-201.

Maurer A, Tixeuil S, and Défago X (2014). Reliable communication in a dynamic network in the presence of Byzantine faults. arXiv preprint. Available online at: https://arxiv.org/abs/1402.0121

Overeinder B, Brazier F, and Marin O (2003). Fault tolerance in scalable agent support systems: Integrating DARX in the agentscape framework. In the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE, Tokyo, Japan: 688-695.

Patron P, Miguelanez E, Petillot YR, and Lane DM (2008). Fault tolerant adaptive mission planning with semantic knowledge representation for autonomous underwater vehicles. In the IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, Nice, France: 2593-2598. https://doi.org/10.1109/IROS.2008.4650836

Samara G and Al-Raba'nah Y (2017). Security issues in vehicular ad hoc networks (VANET): A survey. arXiv preprint. Available online at: https://arxiv.org/abs/1712.04263

Silva MRX, Lung LC, Magnabosco LQ, and de Oliveira Rech L (2013). Vbam–byzantine atomic multicast in LAN Based on virtualization technology. In the New Results in Dependability and Computer Systems, Springer, Heidelberg, Berlin, Germany: 365-374. **PMCid:PMC3794961**

Stanković R, Štula M, and Maras J (2017). Evaluating fault tolerance approaches in multi-agent systems. Autonomous Agents and Multi-Agent Systems, 31(1): 151-177. https://doi.org/10.1007/s10458-015-9320-6

Zubair M and Manzoor U (2016). Mobile agent based network management applications and fault-tolerance mechanisms. In the 6th International Conference on Innovative Computing Technology, IEEE, Dublin, Ireland: 441-446. https://doi.org/10.1109/INTECH.2016.7845118