A NEW APPROACH OF CONCURRENT CALL HANDLING PROCEDURE IN MOBILE NETWORKS

P. K. Guha Thakurta\textsuperscript{1} Misha hungyo\textsuperscript{2} Jahnavi Katikitala\textsuperscript{3} and Darakshan Anwar\textsuperscript{4}

\textsuperscript{1,2,3,4}Department of Computer Science and Engineering, National Institute of Technology, Durgapur, West Bengal
\textsuperscript{1}parag.nitdgp@gmail.com, \textsuperscript{2}mkonghar@gmail.com, \textsuperscript{3}jahnavi6392@gmail.com, \textsuperscript{4}dara.singh09876@gmail.com

ABSTRACT

A new approach for handling concurrent call requests by a number of senders in mobile cellular network is proposed in this paper. The concurrent access is resolved with the introduction of semaphore concept. The several factors are identified to establish a priority factor (PF) for the sender node. Based on this PF value, the right sender is selected by the receiver in case of concurrent requests. This selection algorithm executes in linear time. The effectiveness of the proposed model is analyzed with the introduction of progress graph.

KEYWORDS

Concurrent call requests, semaphore, mobile networks, progress graph

1. INTRODUCTION

In mobile cellular networks, mobile nodes communicate with each other using multi-hop links. This structure is stationary because there are base stations (nodes) in every cell. Each node in the network has call forwarding capability to other nodes [5]. Till date, various routing strategies have been designed to address the problem of finding routing path with efficient congestion control technique. Simultaneously, it needs a more efficient methodology to increase throughput and reduce network latency at the same time. To provide the efficient routing strategy, the nodes are grouped into manageable clusters based on different parameters and the Quality of Service (QoS) availability of each node. Here, the cluster heads (CHs) play the role of local coordinators and they maintain the QoS values of all cluster members. Using this information, a CH can forward the calls to the corresponding destinations [4]. Thus, the network design problem associated with this is to find a least cost or a maximum revenue network, reducing the redundant admitted calls.

Once the clusters are formed and maintained, these can be used to handle incoming call requests. When a mobile node sends a call request, the call is forwarded through the path as: CS\textsuperscript{\textarrow{\textrightarrow}}BSS\textsuperscript{\textarrow{\textrightarrow}}LeaderS\textsuperscript{\textarrow{\textrightarrow}}LeaderR\textsuperscript{\textarrow{\textrightarrow}}BSR\textsuperscript{\textarrow{\textrightarrow}}CR [1], where CS, BSS and LeaderS denote the sender’s node, its corresponding BS and the CH of that cluster respectively. Similar notations are used for...
In this paper, a new approach for handling concurrent call requests by the multiple numbers of sender nodes in mobile cellular networks is proposed. The concurrent access is resolved with the introduction of semaphore [2] concept. The priority factor (PF) is calculated by the receiver in case of multiple numbers of senders. Here, PF is dependent on the available bandwidth (BW) to handle call request, timestamp (t), and repeated request factor (RRF) of the sender node. With the help of this RRF value, the number of unsuccessful attempts by a sender node is defined. Thus the node with the highest PF is selected as the right one. In order to avoid indefinite waiting time for the selection by receiver node, t, and RRF values of the specific sender whose requests had been processed already, are set to zero. With the introduction of PF, the proposed model in this work executes in linear time. In addition, the effectiveness of the proposed model is analyzed using progress graph.

The remainder of the paper is organized as follows. In section II, the proposed model is described. Next, the advantage of this work is concluded in section III.

2. PROPOSED MODEL

The model proposed in this work obeys the following system model.

2.1. System Model and assumptions

Suppose, there are three clusters of nodes shown in Fig. 1. Two of them represent the sender’s clusters among three, whereas the remaining one denotes the receiver’s cluster. The sender nodes and are trying concurrently to connect with the receiver through their respective CHs.

![Fig. 1: Concurrent call requests from multiple sender’s to a receiver](image)

Now, the receiver node has a responsibility to select one of them having the highest priority factor (PF). This PF is dependent on three following factors.

(a) Bandwidth (BW): It is the amount of available BW to handle the call request. Generally, it depends on the underlying hardware architecture and the network operating system used by the sender node.

(b) Timestamp: The timestamp of any call request is determined by the call submission time in the system and it is measured as the time recorded by the system clock. The request with least timestamp would become as the oldest call.
Repeated Request Factor (RRF): It is defined as the number of unsuccessful connection establishment by the sender node. Initially, it is set to 0 and is incremented by 1 after each unsuccessful attempt. Whenever the receiver selects the specific sender node for the connection establishment, then RRF is again set as 0 for that sender. The node having greater RRF value is considered as higher priority node. As available bandwidth limits the number of call requests that can be processed, so PF is directly proportional to bandwidth BW. With the help of RRF factor, a preference is given to a sender node which is calling repeatedly. So, a node with greater RRF value has a greater PF value. Also, preference must be given to an older request. With the help of timestamp, we select the oldest request. So, PF is inversely proportional to timestamp. Thus the terminology PF is defined as the ratio of the product of BW and RRF to \( t_s \). It is expressed as 
\[
PF = \frac{(BW \times RRF)}{t_s}.
\]

2.2. Proposed Functional Model

The model proposed in this paper considers the situation described in Fig. 1. The sender \( S_i \) sets its semaphores to busy state just before sending the requests. Then it checks the receiver’s status whether it is busy or idle. If the receiver is busy, the leader of \( S_i \) records \( t_{si} \) and corresponding RRF of \( S_i \) of the sender node and subsequently store them into a generic linked list.

The linked lists are represented with following fields – (a) for the leader of the receiver node: sender’s id, the PF value and address to the next node, (b) for the leader of the sender node: sender’s id, RRF of \( S_i \), \( t_s \) value and address to the next node. The linked representations are shown in Fig. 2 for Fig. 1.

![Fig. 2: Initial Linked Representation](image)

Fig. 2: Initial Linked Representation: (a) Linked List for \( L_R \)
(b) Linked List for \( L_{S_1} \)
(c) Linked List for \( L_{S_2} \)

the semaphore of \( R \) indicates idle, the \( L_R \) calculates the PF of the senders \( S_i \) that has been sending call requests and stores them into its own linked list. The receiver node then selects the sender that has maximum PF from the linked list and grants its requests. After the call has been granted, the \( t_{si} \) and the RRF of \( S_i \) are set again to initial values. This goes on dynamically for every node and the procedure is described by the following algorithm.

2.3. Algorithm:

**Input:** \( BW, \) initial values: \( t_{si} = 0, RRF_i = 0, \) flag = FALSE.

**Output:** \( \max [PF(S_i)] \)
**Declaration:**

\( R = \text{Receiver} ; \quad S_i = i^{\text{th}} \text{ Sender} ; \quad \text{Avail}_i = \text{Semaphore of } S_i ; \quad \text{Avail}_R = \text{Semaphore of } R ; \quad T_{S_{\text{tot}}} = \text{total timestamp} ; \quad T_{S_{\text{new}}} = \text{new Timestamp} ; \quad T_{S_{\text{pre}}} = \text{previous timestamp} ; \quad L_R = \text{Leader of the receiver} ; \quad \text{data storage} = \text{generic linked list} ;

\[
\begin{aligned}
\text{Concurrent\_Call\_Request}() &= \\
& \{ \\
& \quad \text{Call}(); \\
& \quad \text{Receive}(); \\
& \quad \text{RRF} = 0; \quad /^* \text{setting the values to zero after the call has been granted} */ \\
& \} \\
\text{Call}() &= \\
& \{ \\
& \quad \text{Avail}_R = \text{busy}; \quad /^* \text{setting its own semaphore to "busy" state before trying to connect to the receiver} */ \\
& \quad \text{While} (\text{Avail}_R = \text{busy}) \quad /^* \text{while the semaphore of the 'R' is in "busy" state the leader of Si increements the ts and RRF values accordingly} */ \\
& \quad \{ \\
& \quad \quad T_{S_{\text{new}}} = T_{S_{\text{pre}}} + T_{S_{\text{new}}}; \\
& \quad \quad \text{RRF}_{i-1} + 1; \\
& \quad \quad \text{Record these values into the respective leader's data storage;} \\
& \quad \} \\
\} \\
\text{Receive}() &= \\
& \{ \\
& \quad \text{if(count}(S) > 1) \quad /^* \text{for multiple number of senders} */ \\
& \quad \{ \\
& \quad \quad \text{While} (i <= n) \quad /^* \text{receive () executes until the computation of all PF for 'n' senders} */ \\
& \quad \quad \{ \\
& \quad \quad \quad L_n \text{ calculates the respective PF of the } S ; \\
& \quad \quad \quad PF = (3W_i \times \text{RRF} / T_{S_{\text{tot}}}) ; \\
& \quad \quad \quad \text{Each PF is then stored into the data storage of } L_n ; \\
& \quad \quad \} \quad \text{calculate } \max [PF(S)] \text{ from data storage;} \\
& \quad \quad \text{flag=} \text{TRUE;} \\
& \quad \} \\
\]
2.4. Example

Initially the random available BW for call handling by $S_1$ and $S_2$ are assumed as 12 mbps and 10 mbps respectively. Similarly, the initial timestamp values for these nodes are randomly considered as 10 and 20 respectively.

**STEP 1:** Leaders are recording the respective values into its data storage as shown in Fig. 3.

![Fig. 3: After step 1: (a) Data Structure of $L_{S_1}$ after entering the values; (b) Data structure of $L_{S_2}$ after entering the values](image)

**STEP 2:** $L_R$ calculates PF of each $S_i$ and stores them into its corresponding data storage as shown in Fig. 4.

![Fig. 4: After step 2: (a) Linked List for $L_{S_1}$; (b) Linked List for $L_{S_1}$; (c) Linked List for $L_{S_2}$](image)

**STEP 3:** Leader of Receiver $L_R$ calculates the maximum of the PF from its data storage and then forwards the call to the respective receiver. Here, $PF(S_1) > PF(S_2)$. Hence $S_1$’s request is granted first.

**STEP 4:** Set the values of $RRF_1$ of $S_1$ to initial value as 0 and it is shown in Fig. 5.

![Fig. 5: After step 4: (a) Linked List for $L_R$; (b) Linked List for $L_{S_1}$ after its call has been accepted; (c) Linked List for $L_{S_2}$](image)
2.5. Correctness

To prove the correctness of the proposed algorithm, a concept of progress graph [3] is used. The Progress graphs have intrinsic properties that are formalized by following postulates as given below.

P1: The concurrency state of a system defines a unique point in a progress graph.
P2: A transition from a state represented by a point p1 to a state represented by a point p2 is a ray rooted at p1 with direction p1 → p2.
P3: A point is feasible if and only if it is not within a forbidden region. The forbidden region is the region that violates the constraints on the relative progress of the processes imposed by the signal events in progress graph.
P4: The time between two synchronizing events within each process is greater than zero.

Now, in our work, we summarize the events of the senders (S1 and S2) as follows in table 1:

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(Avail_i) = waiting for Avail_i;</td>
<td>P(Avail_i) = waiting for Avail_i;</td>
</tr>
<tr>
<td>V(S_i) = Signal S_i;</td>
<td>V(S_i) = Signal S_i;</td>
</tr>
<tr>
<td>V(Avail_i) = Signal Avail_i;</td>
<td>V(Avail_i) = Signal Avail_i;</td>
</tr>
<tr>
<td>V(S_i) = Signal S_i;</td>
<td>V(S_i) = Signal S_i;</td>
</tr>
</tbody>
</table>

Following these events for S1 and S2, the corresponding progress graph is shown in Fig. 6. Therefore, it is clearly seen that there is neither forbidden state, nor unsafe state as there is no starvation and deadlock. Thus the concurrency among the call requests is preserved.

Fig. 6: Progress graphs
2.6. Discussion

Presently, while handling concurrent calls no preference is given to a sender node that is repeatedly trying to connect to a particular node. So, this may lead to indefinite waiting time for such node. For illustration, considering a sender node A sending a call request to a receiver node R repeatedly for (n-1) times and its request has not been processed yet. Now, sender node A sends the request for the nth time and concurrently a different node B sends a call request to same receiver node R for the first time. In such scenario, the system may usually process either of the requests without giving any preference to sender node A.

In our proposed approach, this indefinite waiting time can be handled by assigning a RPF factor to each node of the network. This RPF factor denotes the unsuccessful attempt of a sender node. Initially, it is set to 0 and incremented by 1 after each unsuccessful attempt. So, in the above scenario, RPF factor for A is n and RPF factor for B is 1. At the receiver node R, sender A is selected because of its greater RPF value.

3. CONCLUSIONS

An efficient approach for concurrent call handling procedure is described in case of cluster based call scheduling for mobile networks. The semaphore concept is introduced here to resolve this concurrent issue. To determine the right sender among the alternatives, a priority factor (PF) is established. The several factors are determined to provide PF for a sender node. Furthermore, the proposed model executes in linear time. The performance of the model is analyzed with the help of progress graph. Moreover, we are extending our work with the help of logical clock to provide a compact and more efficient model for handling such concurrent events.

REFERENCES


AUTHORS

P. K. Guha Thakurta is an Assistant Professor at National Institute of Technology, Durgapur for the department Computer Science and Engineering with an experience greater than eight years. He published 5 International Journal and 11 International Conference papers. His area of interest include DBMS, Network, Algorithm Analysis and Design, Formal language and automata.

Jahnavi Katikitala is working as a software engineer at Samsung Research India, Bangalore. She has pursued her Bachelors of Technology from National Institute of Technology, Durgapur. Her area of interest include Network, Algorithm Analysis and Design & Operating systems.

Misha Hungyo is pursuing her Masters in Computer Science and Engineering in Motilal Nehru National Institute of Technology, Allahabad. She did her Bachelors from National Institute of Technology, Durgapur. Her area of interests are Computer Networking, Operating Systems, Data Structures and Algorithms.

Darakanwar is pursuing her Masters in Computer Science and Engineering at C-DOT Bangalore. She has pursued her Bachelors of Technology degree from National Institute of Technology, Durgapur. Her area of interest include Network, Algorithm Analysis and Design.