A model and architecture for conducting hierarchically structured auctions

P.D. Ezhilchelvan, S.K. Shrivastava and M.C. Little

Department of Computing Science, University of Newcastle

Newcastle upon Tyne NE1 7RU, UK

Abstract

The paper develops a distributed systems architecture for dependable Internet based online auctions, meeting the requirements of data integrity, responsiveness, fairness and scalability. Current auction services essentially rely on a centralised auction server. Such an approach is fundamentally restrictive with respect to scalability. It is well-known that a tree-based, recursive design approach caters well for scalability requirements. With this observation in mind, the paper develops an approach that permits an auction service to be mapped on to globally distributed auction servers. The paper selects a suitable auction model that treats sellers and buyers symmetrically. This symmetry enables a computational node to play at one level of the tree the role of a seller by dealing with a group of potential buyers as well as to play the role of a potential buyer at the next higher level. Such a symmetric auction (also known as a double auction) is used for supporting a standard auction to be carried out in a hierarchic manner. An architecture is developed and basic algorithms and protocols are presented, together with correctness reasoning.

Keywords and phrases: Double auctions, federated auction house, Bid servers, the Internet, synchronous/asynchronous communication, deadlines, auction fairness.

1. Introduction

Current auction services essentially rely on a centralised auction server. Such an approach is fundamentally restrictive as too many users can overload the server, making the whole auction process unresponsive. We require an auction service to be *scalable*, i.e., capable of providing its end users with "satisfactory" Quality of Service (QoS), regardless of the number of those users and their geographical distance. We therefore investigate ways of enabling widely distributed, arbitrarily large number of auction servers to cooperate in conducting an auction. Allowing a user to bid at any one of the servers is our principal way of achieving scalability and responsiveness, as the total load is shared amongst many servers. However, the fundamental *fairness* property of an auction must be preserved: all participant bidders in the auction must have an equally fair chance for submitting a successful bid, and that all participant sellers must have an equally fair chance for selling their items. For simplicity, we consider that there is one seller, one item to be auctioned and many bidders, so fairness will be concerned only with the bidding process (generalisation is clearly possible, but not considered here). Achieving fairness of auctions conducted over the Internet using a single auction server is a challenging problem as it is [1,2], since differing message transmission delays experienced by bidders can clearly compromise an auction's fairness. Achieving fairness of an auction conducted over a group of auction servers makes the problem even harder, but this problem must be solved in order to obtain scalability without sacrificing responsiveness. This is the problem addressed in this paper. The hierarchic auction model developed in the paper together with the set of protocols to realise such an auction over arbitrarily distributed auction servers are the main contributions of the paper.

In an earlier paper we described an approach that permits an auction service to be mapped on to globally distributed auction servers [3]. We were assuming a model of auction service, presently used by most auction sites, that does not involve the service broadcasting bid messages to all the participants in an attempt to mimic the conventional (non-electronic) English open-cry or Dutch auctions. Here we attempt to close this 'gap' and consider a faithful (as faithful as practicable) realisation of English, Dutch auctions over the Internet that does require that a bid placed be promptly made known to all the bidders. However, we believe that the framework presented here is sufficiently flexible and can be adapted to other types of auctions discussed in the literature [e.g., 4,5,6].

2. Model and Architecture

2.1 Distributed Auction House Architecture

It is well-known that a tree-based, recursive design approach caters well for scalability requirements. We adapt this approach to develop the *distributed auction house architecture* that can scale over the Internet. An *auction house* is where buyers and sellers must go in order to conduct an auction. In addition to running the auction, it is also responsible for setting up and guaranteeing various contracts that are used to create and manage the auction. For example, certifying that the seller is authorised to sell (and also has) the item, ensuring that bidders have sufficient credit limits (and have not been previously barred from bidding), and guaranteeing specific quality of service contracts. The auction house paradigm transfers relatively easily from the physical to the electronic world, and represents a "concrete" entity that users can reason about. From the outside, the auction house essentially represents a "black box"; internally, however, the contracts it enforces, such as security, authentication, and bidder/seller anonymity, help to provide the assurances traders (buyers and sellers) expect from their real-world equivalents. If the auction house allows agents to participate in auctions on behalf of bidders then it will be necessary to ensure that a security sand-box exists for them to reside within.

An auction house may actually be composed of many physically remote *auction rooms* that co-operate to provide the abstraction of a single centralised action house; an auction room itself may be (recursively) composed of several auction rooms and so on; auction rooms will be taken here as indivisible atomic units within an auction house. The auction rooms may be owned by different organisations, who have agreed to work together towards the sale of a particular item. Each auction room has an *auctioneer* who collects bids submitted in that room and determines whether the bidding should continue or be terminated. The auctioneer of one of the rooms is designated as the *head or root auctioneer* and will determine, possibly in consultation with the seller, the auction rules and disseminate them at the start of an auction to the entire house, e.g., whether the auction type is English or Dutch, the minimum selling price (the *ask*), etc. It also ensures that the seller has the item and the right to sell it. The agreed auction rules permitting, the seller can actively participate in the auction and modify the ask depending on the demand perceived. For brevity, we assume that the seller and the root auctioneer are in the same auction room.

The *Bidders* in an auction room place their bids with the auctioneer. How bid placement happens will depend upon the type of auction (e.g., sealed-big auction versus open-cry). A bidder may be required by the auction room to provide proof that he has the required credit limit. Note, as in a physical-world auction, a bidder need not directly take part in the auction, but send an agent to bid on his behalf. How intelligent the agent is, will depend upon the bidder's requirements and potential limitations on the bidder's connectivity to the auction house; for example, if the bidder

cannot maintain a reliable connection to the auction house then it may make sense to send an agent to the auction house to participate in the auction and for it to simply report back at the end of the auction. If the auctioneer of a given room is not the root, then he forwards bids he receives to the root auctioneer, essentially acting as a bidder-of-bids. The root auctioneer decides the winner as per the auction rules disseminated at the start.

Each auction room is free to impose its own constraints on the buyers and sellers who use it. So, for example, one auction house may require all bidders to be known, whereas another may allow certain (or all) bidders to remain anonymous. Likewise, the quality of service guarantees provided by one auction house may differ from those provided by another. Such flexibility in the auction contracts that are imposed by auction rooms may be the deciding factor in how a bidder (and seller) chooses an auction room for conducting his trade. The distributed or federated auction house improves scalability and fault-tolerance. However, the fact that each auction room may have different contracts with its bidders may pose some interesting problems for federation.

Types of contract include:

- quality of service for connections between bidder and auction house.
- bidder/seller anonymity.
- specifying whether or not the history of bids placed by individual bidders can be seen by other bidders.
- specifying whether or not bid histories will be seen by bidders.
- setting up a third-party escrow service for items which are being sold.
- the winning bid or the identity of the final winner to be /not to be made public.

2.2. Auction model: single auction room

As stated earlier, for the sake of simplicity, we assume that there is just one seller and a single indivisible item is being sold. Assume initially that auction house has only one auction room with whom the seller and the bidders are registered. We select an auction model that treats sellers and buyers symmetrically. This symmetry enables a computational node to play at one level of the tree the role of a seller by dealing with a group of potential buyers *as well as* to play the role of a potential buyer at the next higher level (this aspect will be discussed in a subsequent section).

A symmetric auction (also known as a *double auction*) admits aspects of *bargaining* and we derive our basic model as a simple variation of the seminal, *k*-double auction model of Chatterjee and Samuelson [5,7]. This model involves a single buyer and a single seller who respectively submit a *bid* β and an *ask* α; if β exceeds α, then a trade is consummated at the price $k\beta + (1-k)\alpha$, where $0 \leq k \leq 1$. We extend this model in various ways as described below.

The seller initiates a bargain *round* by quoting an ask price and bidders are invited to place bids that exceed the quoted ask. A bid, once placed, cannot be withdrawn. A bidder's offer is made known to all other bidders who are encouraged to out-bid that offer before the expiry of a publicly-announced deadline which is computed to be the maximum of two deadlines: *ask-based deadline*: a time interval of at least D_a units must elapse since the ask was quoted; *bid-based deadline*: a time interval of at least D_b units must elapse since the highest bid was last placed; we take this to mean that following the placement of a new bid that is *greater than or equal to* the current highest bid, the bid-based deadline is extended by at least D_b time. The ask-based

deadline ensures that the round terminates when no bid is placed (which can happen if the ask quoted was too high); i.e., ensures the *liveness* or *termination* of the trading process. Using the ask-based deadline alone can tempt the bidders to place their bids in the 'last minute' so that their bids are less likely to be known to others before the deadline and therefore less likely to be out-bid. Such last-minute bidding can lead to *winner's curse* or *seller's disappointment*. In the former, the winner regrets that he placed a far higher bid only because he had no sure way of guessing the bidding intentions of his competitors due to the scope for last-minute placement of bids. The latter is caused when all bids placed are submitted at the last minute and are above the ask price only by a minuscule amount. The seller may feel that he has gained little by choosing electronic auction as against the traditional trading means such as placing an advertisement in a newspaper. Having a bid-based deadline provides time for a bidder to place a higher offer and thus introduces *liveliness* into the bidding process. Using both the deadlines also maintains symmetry between placements of asks and bids: both the ask and the (current) highest bid are guaranteed to be publicly known for some minimum time.

Just as a bidder is allowed to place a bid that is larger than the current highest bid, the seller is also allowed to increase his ask at any time during a bargain round. (The seller is aware of every bid placed and hence of the bidding pattern.) We restrict that the seller cannot decrease his ask during a round. (Decreasing the ask would mean the current round being abandoned and a fresh round initiated.) For every new ask quoted, a new deadline is computed and enforced. A bargain round *terminates* once the seller has quoted his *final* ask *and* after every bidder has been given sufficient time to outbid the highest bid. The highest bid at the end of the round is called the *final bid*. If it is less than the final ask, the seller can either initiate a new bargain round probably quoting a smaller initial ask or *give up* the trade. If the final bid is larger than the final ask, the trade is *consummated*; if two or more bidders had placed the same final bid, then a single bidder among these finalists is selected through a draw that is statistically fair. The winner takes the item, paying the price according to the value agreed for *k*.

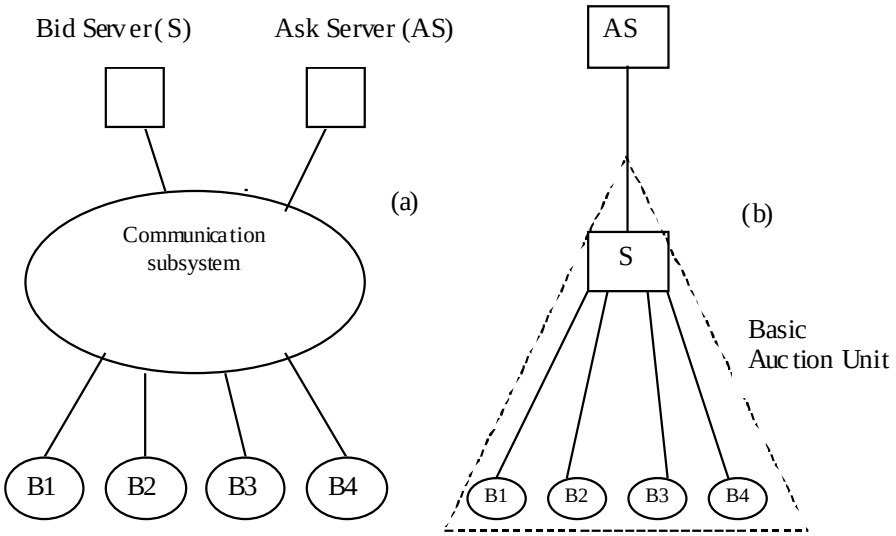

Figure 1. Basic auction unit: (a) Physical view (b) A Logical View.

The above single room auction model can be realised with the help of three types of nodes. An *ask-server* (AS) node playing the role of the seller; a *bidder* (B) node is operated by a potential buyer; and a *bid server* node (S) representing an auction room (see fig. 1). Before taking part in the trade, each bidder B must register with the server, and this process deals with issues relating to bidder/server authentication, exchange of cryptography keys, authorisation on the bidder's spending limits etc. The group comprising a server *S* and the bidders registered with *S*, will be called *a basic auction unit.*

2.3. Hierarchic auction model

We now extend the above model to a hierarchic auction model to support a finitely large number of bidders, geographically wide apart, to take part in an auction. The extended system, shown in Figure 2(a), consists of a number of basic auction units, shown as triangles, which are rooted at, and connected to each other through, their respective bid servers (the synchronous and asynchronous communication assumptions will be discussed in the next section). Bid servers are organised into a tree structure. The auction room of the root auctioneer is at the root of the tree and the seller interacts only with the root auctioneer to quote his initial ask and to convey his desire to increase or not to increase his latest ask. A bidder is required to register with only one of the auction rooms whose auctioneer is responsible for taking the registered bidders' bids and for providing up-to-date state information about the auction process (such as the highest placed so far, latest ask price, deadline for receiving bids, etc.). Each bid server must periodically diffuse its state information to other servers to ensure that each bidder (respectively the seller) has global information about the bidding process to enable him to place a higher bid (respectively higher ask) should he so desire. A scalable architecture requires that an auctioneer (bid server) not know the address of every other auctioneer in the house; otherwise, adding/removing a room would mean informing each auctioneer of the change. We describe below how the features of the tree structure are used to meet this scalability requirement while achieving the house-wide information dissemination.

A server at one level acts as the ask-server to a set of servers shown at the next lower level; the latter are called the *child servers* of the former and the tree shows them to be directly connected to their *parent* server; for example, S_4 and S_5 are child servers of S_8, i.e., the parent S_8 acts as the ask-server for S_4 and S_5. Observe that a given server has exactly one other server *acting* as the ask-server, while the root server alone interacts directly with the *actual* ask-server *AS*. The servers that have no child servers attached to them, are called *leaf servers*.

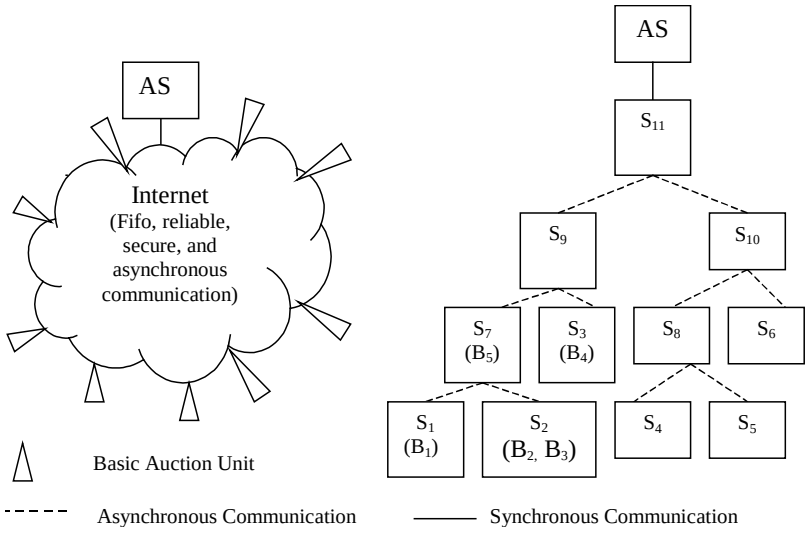

Figure 2. (a) Global architecture. (b) Logical Tree of Servers.

Fig. 2(b) shows eleven servers being arranged in a tree, with the root server S_{11} interacting with ask server AS. (some specific bidders of a few servers are shown in the figure for an example to be discussed later). Recall that each *S* has its own bidders registered directly with it. Each server (say S_9) periodically reports the latest bids (if any) it has received from its bidders, to those nodes (S_{11}, S_7 and S_3) that are *directly* connected to it in the tree structure. The collection of latest bids reported by a server is called an *episode,* and the union of all episodes generated by every server gives the *history* of the bidding process. A server (say S_{11}) diffuses an episode that it receives to all nodes it is directly connected to, excluding the one from whom it received the episode (in this case the diffusion will be to S_{10} and AS), and displays to its bidders all the bids contained in the received episode. Further, if it finds the received episode contains the highest bid, it sets a new deadline of D_b for accepting fresh bids so that its bidders can place new higher bids if they so wish. For example, say, S_{10} has one of its bidders having placed the highest bid of $1000; if it receives S_9's episode that indicates that a bidder there has also placed the highest bid of $1000 (or more), then S_{10} displays this information and sets a new deadline for placing bids so that its bidder has the opportunity to out-bid the highest bidder with S_9. Since any two servers in the tree are connected, a bid placed with one server is displayed (together with new deadlines where appropriate) by every server in the system to their respective bidders. Thus, a bidder not only gets to know a bid placed with another server in the system, he also gets some time to place a new higher bid if the remote bid is larger than or equal to its current bid.

An increase in ask is also diffused in the same way as an episode, except that it travels only downwards along the tree. Every time a server is informed of an ask increase, it sets a new deadline of D_a. The auction process terminates after (i) the seller has irreversibly declared the final ask, and (ii) every server has received and displayed (together with new deadlines where appropriate) the *last* episode of every other server. The terminating condition (ii) ensures that every server composes and displays an identical and global bid history, giving sufficient time to its bidders to outbid the highest bid in the displayed history. This condition will be met once the

bidders stop placing new bids. The details on determining the termination of the auctioning process are presented in the paper.

Though our basic model is developed primarily to implement standard auctions in a scalable manner, we remark that implementing this model can help conduct Internet-based double auctions as well. Since a single seller quoting multiple asks during a given round, is functionally same as multiple sellers quoting a fixed ask price, it is straightforward to extend an implementation of this model for the general double auction model involving multiple bidders and multiple sellers. Though double auctions on Internet are not yet popular, they are known to be efficient in terms of incentives they offer to participants. The analysis of Wilson [8] indicates that a double auction is efficient in the sense that with a sufficiently large number of sellers and bidders (possible in the Internet set-up), there is no other trade rule for which it is common knowledge that all participants are better off in expectation.

3. Protocol

In this section we describe how the servers can be programmed to implement the hierarchic auction model and how standard English and Dutch auctions can be supported. We present the server protocol in two parts, the second part executed only by non-leaf servers. We begin by stating the main assumptions.

3.1. Communication and failure assumptions

We make the following assumptions concerning a basic auction unit:

(i) S is built to be a failure-free node. In practice, this would be achieved by a fault tolerant cluster of nodes that strive to maintain service availability despite internal component failures.

(ii) The communication between a B node and its S is fifo ordered, reliable, secure and synchronous. In fifo ordered communication, if message m_1 is sent before m_2 by a correctly functioning node, then at every correct and common destination of both m_1 and m_2, m_1 is never received after m_2. Communication is reliable and secure in the sense that a correct node receives exactly once what was sent to it, without any accidental or intentional corruption; and it is synchronous in the sense that a message sent by a correct node is received by a correct destination within some delay that is bounded by a known constant δ.

The bound δ must be large enough to account for the maximum transmission, authentication and queuing delays a message possibly can experience; where a denial-of-service attacks on the network are possible, δ should also include the maximum time it might take to mask the effects of successful attacks.

Observe that a known δ enables *S* to provide timeliness guarantees on the services it offers to bidders. For example, a bidder can be guaranteed of a time delay within which he can expect the bid he placed to be acknowledged by *S*. Further, for bid-based deadlines to be meaningful, D_b must be more than 2δ: with at most δ time to inform the bidders of the deadline extension and at most δ for a placed bid to reach the server, the bidder has only D_b-2δ to plan and act. Thus, there are many reasons which make it essential that δ exist within the basic auction unit and be known.

The synchronous communication assumption within the basic auction unit can be met realistically over the Internet - at least for nodes geographically close to each other – once the planned QoS enhancements to the Internet [9] are realised.

(iii) All functioning nodes within an auction unit have their clocks synchronised within a small known bound ε. We assume the use of efficient clock synchronisation algorithms such as [10,11] which can yield small ε even when nodes are subject to malicious attacks and arbitrary failures.

(iv) Every message *m* delivered by the communication system to the server *S* gets a timestamp *m.ts* within π time, where π is a small constant (known to *S*) that bounds the message reception delay at *S*.

(v) The ask-server, like the auction server, is assumed to be failure-free. The communication between the ask-server and the root bid server is also fifo-ordered, reliable, secure and synchronous. For simplicity, we will assume that the communication delays between these two servers are also bounded by δ.

(vi) We assume that the bid servers are scattered over the Internet, and a bidder is required to register with the 'nearest' one so that the requirements of a basic auction unit described earlier (particularly, the δ-bounded communication requirement) are met.

(vii) The communication between bid servers is asynchronous: that is, provided that servers S_i and S_j remain connected, a message sent by one to the other is *eventually* delivered.

We will assume that communication between any two servers, if broken, is eventually restored. This assumption permits the server communication be fifo ordered, reliable but *not* synchronous: the known bound δ assumed within the basic auction unit no longer exists between servers. It should be noted that an eventual restoration of inter-server communication is essential for ensuring both liveness *and* fairness of the auction process. Consider, for example, a bidder B_1 who is registered with server S_1 that is permanently disconnected from the rest of the system. Other servers cannot conclude the round without knowing the maximum bid from S_1; otherwise, it is possible that the item is unfairly sold to a bidder who had placed a smaller bid than what B_1 placed (with S_1).

3.2. Protocol: Part I

A server *S* essentially performs the role of an intermediary between bidders, and between bidders and its (acting or actual) ask-server. It receives individual bids; whenever a higher bid is placed or the number of bidders with the same highest bid changes, it announces to the bidders of the change and the new round-closing time. It periodically reports the history of the bids received to its ask-server AS_S which is either the parent server of *S* or *AS* if S is the root server. When AS_S quotes a new ask, *S* announces this change and extends the round-closing time.

```
Initialise(double initial_ask) {
   double cur_max := 0.0; /* current maximum bid placed in the sub-tree rooted at S*/
   Set-of-identifiers  finalists := {};
      /* identifiers of bidders registered with S and placed cur_max */
   int max_bidders := 0;  /* cardinality of finalists */
   Set-of-identifiers premier_list := {};
      /* identifiers of bidders in the sub-tree rooted at S and placed cur_max */
   int premier_bidders := 0; /* cardinality of premier_list */
   double cur_ask := initial_ask; /* the current ask is initialised to the quoted initial*/
   Bid_History H := <⊥, Φ>;
      /* H set to an empty list ⊥ and a timestamp Φ of zeros */
   Episode E := <⊥, my_id, 0>;
   Time Da:= ..; Time Db := ..; /* ask- and bid-based intervals are initialised*/
   Time round_close_time := clock_time + Da;
   Timeout status_report_timeout := clock_time + τ;
   Boolean TC0 := TC1 := TC2 := TC3 := false;
      /* round terminating conditions set to false /
   Boolean IsFBSent := false; /*  Is Final Bid (FB) sent to ASs? */
}  /* end initialise */
```

Figure 3. Variables of server *S* and their Initial values.

The variables maintained by *S* are shown in figure 3, the algorithm for *initialise(..)*. The variable *cur_max* holds the currently known maximum of all the bids placed with the servers of the sub-tree rooted at *S*, and the *cur_ask* the current ask. The set variable *finalists* and the integer *max_bidders* respectively hold the id's and the number of those bidders who are registered with *S* and had placed a bid of *cur_max*.

The set variable *premier_list* of *S* contains the id's of all those bidders who are registered with a server of the sub-tree rooted at *S* and had placed a bid of *cur_max*, and will be the same as the *finalists* if *S* is a leaf server. It can also be defined as follows: the set of all elements in the *finalists* of *S* and in the *premier_list* or *finalists* of every child server C of *S* depending on whether C is a non-leaf or a leaf server respectively; the integer *premier_bidders* has the cardinality of *premier_list*. When bidders wish to remain anonymous, a bidder's identifier should not be revealed to any server other than the one he is registered with. We accommodate bidder anonymity by letting *premier_list* be a multiset (which can have the same element appearing more than once), containing all elements of *finalists* of *S*, and every child server C appearing as many times as the cardinality of C's *premier_list* or *finalists* depending on whether C is a non-leaf or a leaf server respectively. For example, referring to fig 1(b), let us suppose that *cur_max* of S_9 be β, and that the bidders B_1, B_2, B_3, B_4, and B_5 have placed a bid for β (at their respective servers as shown in the figure). The *finalists* of S_1, S_2, S_3, S_7, and S_9 will be $\{B_1\}$, $\{B_2, B_3\}$, $\{B_4\}$, $\{B_5\}$, and $\{\ \}$respectively. The *premier_list* of non-leaf servers S_7 and S_9 are $\{B_5, S_1, S_2, S_2\}$ and $\{S_3, S_7, S_1, S_2, S_2\}$ respectively.

The `Bid_History` variable *H* contains (i) a bid frequency list *H.freqList* which is a list of 2-tuples <*amt*, *n*> and indicates for every bid amount *amt* placed the number *n* of bidders who had placed *amt*, and (ii) a vector timestamp *H.ts* which is an array of natural numbers and is indexed by node-id's; *H.ts*[*S*'] represents the latest sequence number of the episode received from *S*' (see below). $\perp$ and Φ respectively denote an empty list and an integer array of zeros. We use the + operator to combine two lists L_1 and L_2: for every bid amount *amt* in L_1 and L_2, $L_1 + L_2$ will have an entry <*amt*, *n*> where *n* is the total number of bidders who had placed a bid for *amt* according to both L_1 and L_2. *S* builds its bid history as an amalgamation of *episodes* where an episode could have been built locally or remotely at a different server. The `Episode` variable *E* is made up of bid frequency list *E.freqList*, *E.author* holding the node-id of the server that built *E* and a sequence number *E.seq#*.

We use the $\oplus$ operator for *sequenced* merging of *E* onto *H* (and return a new Bid_History value), if *H* does not already contain *E*: if (*H.ts*[*E.author*] = *E.seq#* - 1) then {*H'.freqList*:= *H.freqList* + *E.freqList*; *H'.ts*[*E.author*] := *E.seq#*; *return H'*;} else {*return H*;}. We define a boolean operator $=_l$ between Bid_History variables H_1 and H_2, which return true if the *freqLists* of variables H_1 and H_2 have the same largest bid placed by the same number of bidders. Say H_1.*freqList* indicates the largest bid to be max_bid_1 placed by n_1 bidders, and in H_2.*freqList* the largest bid is max_bid_2 placed by n_2 bidders; if $max\_bid_1 = max\_bid_2$ and $n_1 = n_2$ then $H_1 =_l H_2$; otherwise, $H_1 \neq_l H_2$. When $H \oplus E \neq_l H$, we will say *H extends* due to (merging of) episode *E*.

The *status_report_timeout* indicates the time when the latest *episode* built by *S* should be sent to AS_S. Among the variables maintained by *S*, the following are publicly displayed and can be accessed (e.g., by a remote procedure call) by a bidder: *cur_ask*, *round_close_time*, and

($H \oplus E$).*freqList*. They are collectively called *Display*. The variable *round_close_time* indicates the latest time by which a bid should be received for it to be accepted by the server.

The four boolean variables TC0 to TC3 help *S* to determine whether a bargain round is over. TC0 becomes true once *AS* has quoted its final ask and is used only if *S* is the root server. TC1 becomes true for the pair <*cur_ask* = α_S, *History* = H_S> once *S* has displayed α_S and H_S.*freqList* until the *round_close_time* (displayed along with <α_S, H_S>) is gone past and *S* has dealt with all bids placed. If ever *S* is to increase *round_close_time* in future (due to an increase in α_S or an extension to H_S), it resets TC1 to false. TC2 becomes true for <α_S, H_S> if (1) TC1 becomes true in *S* for <α_S, H_S>, and (2) TC1 is known to have become true in every child C of S for <α_S, H_S>. A leaf *S* has no child; hence TC1 $\Leftrightarrow$TC2. TC3 becomes true for <α_S, H_S> once TC1 and TC2 have become true for <α_S, H_S> and *S* has computed the *premier_list*. Recall that for a leaf *S* that has no child, *premier_list* $\equiv$ *finalists*; so, TC1 $\Leftrightarrow$TC3. For *S*, the terminating condition TC = TC1 $\wedge$ TC2 $\wedge$ TC3; additionally, if S is root then TC = TC0 $\wedge$ TC. As soon as TC becomes true for <α_S, H_S>, *S* sends a *FinalBid* message to AS_S containing {α_S, H_S, *cur_max*, *premier_bidders*} and sets *IsFBSent* to true.

```
cycle    /* task T1 */
   recv_msg b from bidder;
   if (b.amt ≥ cur_ask and authentic(b) and
      b.ts ≤ round_close_time + π + ε) then
   {store b in Bid_DB; Update(b); mark b processed;}
endcycle  /* task T1 /

cycle    / task T2
   if (clock_time > round_end_time) and no_unprocessed_bids in Bid_DB and (not TC1) then
   {TC1 := true;} / no more bids to be accepted unless ask changes or History extends
/endcycle / task T2 /

cycle    / task T3 /
   Boolean TC := TC1;
   if (non-leaf server) then {TC := TC and TC2 and TC3;}
   if (root server) then {TC := TC and TC0;}
   if ( (status_report_timeout or TC) and E.freqList ≠⊥) then
         /* it's time to report to AS_S or TC becomes true, and local E is not empty
      {if (H ≠_l H ⊕ E) /* due to E, H extends to include new/more highest bid(s), so...
         then { IsFBSent := false;} /* ... flag that new Final_Bid msg needs to be sent
      H := H ⊕ E; /* merge E into H */
      send_msg Status(cur_max, E) to AS_S;
      status_report_timeout:= clock_time + τ;
      E.seq# ++; E.freqList := ⊥; // next Episode with empty list
      } /* end if ( (status_report_timeout ....
   if (TC and not IsFBsent) then
      {send_msg Final_Bid(cur_ask, H, cur_max, premier_bidders) to AS_S;
       IsFBSent := true;}
endcycle  /* task T3 */

cycle    /* task T4 */
   recv_msg a from AS_S;
   switch a {
      case NewRound(ask):{
         initialise(ask);// variables initialised with cur_ask set to the quoted ask
         start_all_other_tasks;
         mcast_msg Display to bidders;} // multicast Display variables
         mcast_msg a to child servers;} // null operation for a leaf S
      case NewAsk(α, E, isFinal):{
         mcast_msg a to child servers;
         TC0 := isFinal; /* isFinal indicates if α is the final ask */
         double old_ask := cur_ask; cur_ask:= max{ α, old_ask };
         Bid_History old_H := H; H:= H ⊕ E;
         if old_ask ≠ cur_ask and old_H ≠_l H then
               /* somechange in cur_ask and H; so, */
```

```
          { IsFBSent := false; round_close_time := clock_time + max{Da, Db};
            TC1 := TC2 := TC3 := false; mcast_msg Display to bidders;}
          else if old_ask ≠ cur_ask then
            { IsFBSent := false; round_close_time := clock_time + Da;
              TC1 := TC2 := TC3 := false; mcast_msg Display to bidders;}
          else if old_H ≠l H then
            { IsFBSent := false; round_close_time := clock_time + Db;
              TC1 := TC2 := TC3:= false; mcast_msg Display to bidders;}
        }

       case Agreed(max_bid, Server_Id): {
          Winner_Id := Server_Id;
                /* Check if Agreed message indicates me as a winner?
          if (Server_Id = My_Id and max_bid = cur_max) then {
              if (leaf-server) then {
                  int w = random_draw (max_bidders);
                  // a number w from 1.. premium_bidders is randomly chosen
                   Winner_Id = wth element in finalist;}
            else {          // if not a leaf server
                   int w = random_draw (premium_bidders);
                    Winner_Id = wth element in premier_list;}
                     /* end if ..then ..else ..
                 // Is the winner a bidder registered locally?
              If (Winner_Id ∈ finalists) then /* always true for leaf
                  {send_msg Consummate_Trade() to Winner_Id;
                   finalists := finalists - {Winner_Id};
                  }       /* end if (Winner_Id..*/
             }        /* end if (Server_Id ...*/

                // inform the losers of the draw
          send_msg Not_A_Winner() to finalists;
          mcast_msg Agreed(cur_max, Winner_Id) to
              premier_list - finalists; /* a leaf mcasts to none
          kill_all_other_tasks;}
        } /* endcase */

       case Aborted(max_bid, Server_Id): {
          kill_all_other_tasks;}

    } // end switch
endcycle    /* task T4 */
```

Figure 4. Part I of the Server Protocol**.**

Part I of the protocol has four concurrent tasks as shown in fig 4. Task T1 deposits a received bid *b* in a database *Bid-DB* if *b* is timely, authentic and is for an amount that is not smaller than *cur_ask*; it processes *b* in *Bid-DB* by calling the procedure *Update(b)* which essentially updates the variables of *S* according to the contents of *b* (see figure 5). Task T2 sets TC1 to true if *Bid-DB* contains no *b* to be processed and the *round_close_time* is past. Task T3 is responsible for sending the periodic *Status* message and the *Final_Bid* message. If *E.freqList* $\neq\perp$, a *Status* message containing *cur_max* and *E* is sent to AS_S, which is then followed by *E* being merged with *H*, and *E.freqList* and *E.seq#* being set to $\perp$ and incremented (by 1) respectively. If TC is true, a *Final_Bid(..)* message is sent to AS_S. Task T4 responds to a message *a* received from *AS* which can be one of four types: *NewRound(..)* message indicating the start of a new round, *NewAsk(..)* providing a new ask for the on-going round, *Agreed(…)* informing that the trade is consummated for the winning bid *max_bid* and indicating whether the winner is registered with one of the servers in the sub-tree rooted at *S* or *Aborted(..)* informing that the trade is given up.

```
Update(bid b){
   E.freqList := E.freqList + <b.amt, 1>;
   if b.amt < cur_max then { exit;}  // not the highest bid, so exit.
   if b.amt > cur_max then {  // new highest bid
      cur_max := b.amt; max_bidders := 1; finalists = {b.sender};}/* end if b.amt > ..
   if b.amt = cur_max then { // one more bid with the existing highest amt
      max_bidders :=+ 1; finalists := finalists ∪ {b.sender};} /* end if b.amt = ..
   IsFBSent := false; round_close_time:= clock_time + Db;
```

```
   TC1 := TC2 := TC3 := false;
      // announce changes in Display values to all bidders;
   mcast_msg Display to all registered bidders;}
     } //end Update
```

Figure 5. Update Procedure.

3.3. Protocol Part II

The second part of the protocol presented in figure 6 has two concurrent tasks, executed only by non-leaf servers. Task T5 deals with a message *c* sent by a child server C. *c* can be a *Status* message or a *Final_Bid* message. Within a *Status(max_bid, E_c)* message sent by C's task T3, *max_bid = cur_max* of C and E_c is the latest bid history episode composed by C. *max_bid* and E_c of the received *c* are used to update *H* and *cur_max* of S; then they are diffused upward as a *Status* message to AS_S if AS_S is not *AS*, and downwards to *S*'s child servers as a *NewAsk* message. This diffusion ensures that E_c (1) can reach every server in the tree, and (2) does not permanently circulate in the system. T5 simply stores a *Final_Bid* message received.

```
cycle   /* task T5 */
   recv_msg c from a child server C;
   switch c {
      case Status(max_bid, E_c):{
             // keep track of changes in values of cur_max and History
         double old_max := cur_max; cur_max:= max{max_bid, old_max};
         Bid_History old_H := H; H:= H ⊕ E_c;

             //react to any changes
         if cur_max > old_max then
         // increase in cur_max is not due to a bidder registered with myself, so..
           {finalists := { }; max_bidders := 0;}
         if old_H ≠_l H then
           {IsFBSent := false;  round_close_time := clock_time + D_b;
            TC1 := TC2 := TC3 := false; mcast_msg Display to bidders;}

              // propagate the received episode E_c, both up and down
         if (S not root) then send_msg Status(max_bid, E_c) to AS_S; /* upward diffusion */
         mcast_msg NewAsk(max_bid, E_c, true) to all child servers;
                /* downward diffusion /

}  /* end case Status(...

      case Final_Bid(ask_c, H_c, max_bid, max_bids):{
          deposit c;}} //
   end switch
endcycle / task T5 /
cycle    / task T6 */
   if (TC1) then {
      if (for all child C: Final_Bid(ask_c, H_c, cur_max_c, cur_max_bids_c) received and
         ask_c = max {cur_ask, cur_max} and H_c =_l H)
         then {TC2 := true;} else {TC2 := false;}  /*end if.. then .. else/
      if (TC2 and not TC3) then {
         premier_list := finalists;
         premier_bidders = max_bidders;
         / deal next with each child's bidders with bid = cur_max
         for every child-server C  do
            if Final_Bid(ask_c, H_c, cur_max_c, cur_max_bids_c) received and
                        ask_c = max {cur_ask, cur_max} and H_c =_l H and cur_max_c = cur_max)
            then {
                   int i := cur_max_bids_c;
                   premier_bidders = premier_bidders + i;
                   enter C into premier_list i times;
                   }/*end if */
         od
         TC3 := true;
      } /*end if(TC2 and not TC3)
```

```
   }  /*end if(TC1)
endcycle  / task T6 */
```

Figure 6. Part II of the Server Protocol.

Task T6 sets TC2 once TC1 becomes true, and TC3 when TC1 and TC2 are true. Recall that for a leaf server TC2 and TC3 automatically become true when TC1 becomes true; further, *premier_list = finalists*.

3.4. English and Dutch Auctions

With the server programmed to perform double auctions, standard auctions can be obtained by appropriately programming the auction server, *AS*. At the end of a given round, *AS* (i) consummates the trade (with *k* set to 1) if the final bid reported by *S* is not smaller than the ask; (ii) aborts the trade if the final bid reported by *S* is smaller than the reserve price set at the beginning of the trade; or, (iii) initialises a new round with the ask reduced by an amount set at the beginning.

```
/* ENGLISH/DUTCH AUCTION INITIALIZATION */
double reserve_price := .. ;  /* set the reserve price */
double ask_price := .. ;    /* set initial ask > reserve_price */
   /* set the amount by which ask to be reduced if round is unsuccessful
double reduce_by := ..; node_id S := ../* set S to server-id;
cycle
   send_msg NewRound(ask_price) to S;
   recv_msg m from S;
      if m = Final_Bid(ask_price, Hs, max_bid, max_bidders) then
      {
         if (max_bid ≥ ask_price and max_bidders > 0) then
         /* acceptable bids have been placed; consummate trade
            send_msg Agreed(max_bid , S) to S; exit;
         if (max_bid < ask_price and ask_price = reserve_price) then
         /* no acceptable bid even for the reserve price! abort trade;
             send_msg Aborted(max_bid, S) to S; exit;
         ask_price := max{ask_price-reduce_by, reserve_price};
      }
endcycle
```

Figure 7. Ask-Server Skeleton Code for Standard Auctions.

Dutch auctions do not permit explicit or open out-bidding between bidders. To accommodate this, the task T1 of *S* must check for bids *b* with *b.amt = cur_ask* (instead of *b.amt* ≥ *cur_ask* as shown in figure 4); and, the *round_close_time* must be decided solely by the ask-based deadline which is accomplished by initialising $D_b = 0$. For space reasons, we will not deal with multi-round/single-round sealed bid auctions except to note that the former can be implemented quite similar to Dutch auctions (see [3]).

3.5. Properties

Recall that in our basic auction model, *AS* initiates a bargain round with an initial ask price. A round so initiated is said to *terminate* once the root server *RS* sends a *Final_Bid ()* message to *AS*. It is guaranteed that a round does terminate if *AS* quotes its final ask and bidders stop bidding, and terminates announcing a unique <β, n> (n bidders having placed the final bid of β) if *AS* does not increase on the ask which it has declared as final. More precisely, the following properties are guaranteed:.

Termination: Once a given bargain round is initiated, *RS* eventually sends *AS* a *Final_Bid ()* message sometime after *AS* quotes its final ask and all bidders stop placing new, timely bids.

Unique final bid: For a given bargain round, *RS* never sends to *AS* more than one *Final_Bid ()* message, provided *AS* does not further increase on its final ask.

Fairness: Say, a server *S* accepts a bid *b* from a bidder B for an amount *b.amt*. If a bid *b'*, *b'.amt* ≥ *b.amt*, is accepted by (any server in) the system, *S* informs B of *b'* and gives B at least D_b-2δ time to place another bid.

Accuracy: Say, a given round terminates with final ask = α and final bid = β. The root server *RS* sends *Final_Bid*(α, *, β, *n*) message (to *AS*) if and only if *n* bids with amount β have been placed.

The proofs in appendix make use of the following assumptions:

A1: A server processes another server's messages in the received order.

A2: Before evaluating TC, task T3 puts a write lock on variables, *cur_ask*, H, *cur_max*, *max_bidders*, and *premium_bidders*, if the boolean *IsFBSent* is false; these locks are released if TC is evaluated to be false or if *IsFBSent* becomes true.

If *IsFBSent* is false at the start of task T3, *Final_bid(..)* message will be sent provided TC evaluates to be true; such a message will contain the values of some of the write-locked variables. A2 is necessary to ensure that these values do not change between the time TC is evaluated to be true and the time until the *Final_bid(..)* message is sent.

4. Concluding Remarks

In this paper we have developed a novel hierarchic auction model and its architecture to enable an auction to be conducted over a set of arbitrarily distributed auction servers. Allowing a user to bid at any one of the servers is our principal way of achieving scalability and responsiveness, as the total load is shared amongst many servers. From the point of view of responsiveness, we chose a particularly demanding auction process that requires that a bid placed be promptly made known to all the bidders, and have shown how this can be achieved over the Internet by employing hierarchic composition of auction servers.

Auction bidding is but one aspect of the complete bidding process that includes initial buyer and seller registration, scheduling and advertising of the event, actual bidding and trade settlement [6]. In this respect the architecture presented here is very attractive for conducting auctions on a global scale, as it enables a federation of *auction rooms* to co-operate. A framework for implementing the architecture for English auctions is described in [12].

Acknowledgement

Work reported here has been supported in part by UK Engineering and Physical Sciences Research Council, grant number GR/M94168 and ESPRIT project C3DS (project no. 24962).

References

[1] M.P. Wellman and P.R. Wurman, "Real time issues for Internet auctions", IEEE Workshop on dependable and real time e-commerce systems (DARE-98), Denver, June 1998.

[2] C.S. Peng et al, "The design of an Internet based real time auction system", IEEE Workshop on dependable and real time e-commerce systems (DARE-98), Denver, June 1998.

[3] F Panzieri and S K Shrivastava, 'On The Provision of Replicated Internet Auction Services', IEEE Intl. Workshop on Electronic Commerce, WELCOM'99, Proc. of 18th IEEE Symp. on Reliable Distributed Systems, Lausanne, 19 October, 1999, pp. 390-395.

[4] P.R. Wurman, W.P. Walsh and M.P. Wellman, "Flexible double auctions for electronic commerce: theory and implementation", Decision Support Systems, 24, 1998, pp. 17-27.

[5] P. Klemperer, "Auction theory: a guide to the literature", Journal of Economic Surveys, 13(3), July 1999, pp. 227-286.

[6] M. Kumar, S.J. Feldman, "Internet Auctions", Proc. 3rd USENIX Workshop on Electronic Commerce, Boston (MA), Aug. 31 - Sept. 3, 1998, pp. 49-59.

[7] K. Chatterjee and W Samuelson, 'Bargaining under Incomplete Information', *Operations Research*, Vol 31, 1983, pp. 835-51.

[8] R Wilson, 'Incentive Efficiency of Double Auctions', *Econometrica*, Vol. 53, 1985, pp. 1101-15.

[9] L.L. Peterson and B.S. Davie, "Computer Networks: A systems approach", Morgan Kaufmann, (2nd edition), 2000.

[10] N. Vasanthavada and P. N. Marinos, "Synchronisation of Fault-Tolerant Clocks in the Presence of Malicious Failures," IEEE Transactions on Computers, Vol. C-37(4), pp.440-448, April 1988.

[11] P. Verissimo, L. Rodrigues, and A. Casimoro, "Cesium Spray: A Precise and Accurate Global Clock Service of Large Scale Systems," Journal of Real Time Systems, Vol. 12(3), 1997.

[12] P Ezhilchelvan and G Morgan, 'A Dependable Distributed Auction System: Architecture and an Implementation Framework', To appear in the proceedings of the Fifth IEEE International Symposium on Autonomous De-centralised Systems, Dallas, Texas, April 2001.

Appendix: Correctness Reasoning

Assumptions

A1: A server processes another server's messages in the received order.

A2: Before evaluating TC, task T3 puts a write lock on variables, cur_ask, H, cur_max, max_bidders, and premium_bidders, if the boolean IsFBSent is false; these locks are released if TC is evaluated to be false or if IsFBSent becomes true.

If IsFBSent is false at the start of task T3, Final_bid(..) message will be sent provided TC evaluates to be true; such a message will contain the values of some of the write-locked variables. A2 is necessary to ensure that these values do not change between the time TC is evaluated to be true and the time until the Final_bid(..) message is sent.

Definition: A history H is said to extend to H' if H' contains more bidding information than H. Formally, there exists a non-empty and finite sequence of E_1, E_2, .. E_k, such that H' = (.. ((H⊕E_1)⊕E_2) .... ⊕E_k) and H' $\neq_l$ H. When H extends to H', H' is said to be an extension of H.

Lemma 1: Let E_1 and E_2 be two successive episodes generated by some server in a given round: E_1.author = E_2.author and E_2.seq# = E_1.seq#+1. A server S, S ≠ E_1.author, must receive E_1 at least once before it receives E_2 for the first time.

Proof by contradiction. Assume that S receives E_2 for the first time without ever having received E_1 before. Let E_2 reach S for the first time after being transmitted through a sequence Σ of servers, Σ = E_1.author, S_{x1}, S_{x2}, .., S_{xk}. Note that E_1.author generates and sends E_1 before E_2, communication between servers is reliable and fifo ordered and that messages from a given server are processed in the received order (A1). So, for S not to have received E_1 through Σ, some server in Σ, say S_{xi} , 1≤i≤k, must have ignored (and hence not diffused) the message containing E_1 which it received. This is possible only if S_{xi} has halted its task T5 which can happen only if it has received Aborted() or Agreed() message. In that case, S_{xi} would also ignore the message containing E_2 which it would later receive, and the E_2 that S is considered to have received could not have been transmitted through the sequence of servers Σ. A contradiction.

The above validates the definition of the opeartor ⊕ given in section 3.1: when S merges the received episodes with its H in the received order, it builds a bid history without any omission or duplication of the contents of E.freqList of the received episodes.

Observation 1: Server S increases its round_close_time only if its cur_ask increases and/or its H extends. Thus, the potential postponing events (ppe's) that have the potential for postponing the deadline for accepting bids, are when S receives

- ppe1: a bid b with b.amt ≥ cur_max (see algorithm for Update(..)),
- ppe2: NewAsk(α_p, E_p) such that α_p > cur_ask or H⊕E_p $\neq_l$ H (task T4 for case NewAsk), and
- ppe3: Status(max_bid, E_c) message such that H⊕E_c $\neq_l$ H (in task T5).

Note that an occurrence of ppe1 is time-dependent: ppe1 can happen only if the current time of S has not gone past the round_close_time of S; whereas, ppe2 and ppe3 can occur irrespective of whether S's round_close_time is gone past or not.

Lemma 2: Say a non-leaf S sends a Final_Bid(cur_ask_s, H_s, cur_max_s, *) message at time T_{FB}. Let S evaluate TC1, TC2 and TC to be true at T_1, T_2 and T_{TC} respectively for the last time before T_{FB}. At all time T, $\min\{T_1, T_2\} \le T \le T_{FB}$, cur_ask of S = cur_ask_s, cur_max of S = cur_max_s and H of S = $_l$ H_s.

Proof : Between T_{TC} and T_{FB}, these variables are write-locked. Say, the value of any one of the three variables changes in the interval $\min\{T_1, T_2\} \le T \le T_{TC}$. This must set TC1 and TC2 to false, as per the code for update(..), T4 for case NewAsk(..) and T5. But at T_{TC}, TC is evaluated to be true, i.e., TC1 and TC2 must have become true again. This means that T_1 and T_2 cannot be the latest times when TC1 and TC2 respectively became true for the last time before T_{FB}. Hence the lemma.

Remark: The lemma is true for a leaf S as well, when references to TC2, T_2 and task T5 are removed.

Corollary 2.1: At T_{FB}, T_{FB} > round_close_time.

Proof: At T_1, T_1 > round_close_time. It will remain true until cur_ask increases or H extends. By lemma 2, neither happens until T_{FB}.

Definition: server level l: The level l_s for server S indicates the position of S in the logical tree, counted from the bottom such that the root-server has the highest level. Formally, if S is a leaf server, $l_s = 0$; otherwise, l_s = max { l_c of every child server C of S} + 1.

Lemma 3: Say, S sends a Final_Bid(cur_ask_s, H_s, cur_max_s, *) message at T_{FB}. If, after T_{FB}, S does not have to process a NewAsk(α_p, E_p) message such that $\alpha_p > cur\_ask_s$ or $H_s \oplus E_p \neq_l H_s$, then

(i) it cannot receive a Status(*, E_c) message such that $H_s \oplus E_c \neq_l H_s$, and

(ii) it cannot accept bids from its bidders.

Proof: By induction. Suppose that $l_s = 0$. Since no leaf server ever receives a Status(..) message, (i) is obviously true. By corollary 2.1, S cannot be accepting bids at T_{FB}. Only an occurrence of ppe2 can therefore cause S to increase its round_close_time; by given, ppe2 cannot occur. So, the lemma is true for a leaf S.

Assume that $l_s > 0$ and the lemma is true for all $l < l_s$ (induction hypothesis). By T_2 (the time when S evaluates TC2 to be true for the last time before T_{FB}), S must have received a Final_Bid(cur_ask_s, H_s, *, *) message received from each of its child (see task T6). Assume, contrary to the lemma, that S receives a Status(*, E_c) message for the first time after T_{FB}, at time T_{st}. Let C be the child server that sent that Status(..) message, and let t_{FB} and t_{St} be the times when C sent its Final_Bid(cur_ask_s, H_s, *, *) and Status(*, E_c) messages respectively. By the code for Task T3 and assumption A2, cur_ask of C = cur_ask_s and H of C = $_l$ H_s, at t_{FB}. Clearly, $t_{FB} \ge T_{FB}$ and $t_{St} \ge T_{St}$ (we assume here a message can be received in zero time). Since $T_{FB} < T_{St}$, $t_{FB} < t_{St}$. C could send a Status( ) message at t_{St}, only by accepting new bids or receiving a Status() message from its child server (if any) after t_{FB}. By induction hypothesis, the lemma applies to C; this means that C must have received from S a NewAsk(α_p, E_p) message such that $\alpha_p > cur\_ask_s$ or $H_s \oplus E_p \neq_l H_s$, during the interval ($t_{FB}$, t_{St}). That is, S must have sent NewAsk(α_p, E_p) during the interval (T_{FB}, T_{St}). S can send such a NewAsk() message only in two contexts:

(a) In T5, S received Status(max_bid, E_c) from a child server with max_bid = α_p and $E_c = E_p$, and

(b) In task T4, S received NewAsk(α_p, E_p) from its parent.

(a) is not possible as C's Status(..) message is the first Status(..) message S receives after T_{FB}; so, only (b) is possible and this contradicts what is given: S receives no such NewAsk(α_p, E_p) from its parent after T_{FB}. Thus, we prove (i) of the lemma. With (i) proven, the arguments used for the case $l_s = 0$ prove that (ii) is also true for a non-leaf S. Hence the lemma.

Corollary 3.1: Say, S sends a Final_Bid(cur_ask_s, H_s, cur_max_s, *) message at T_{FB}. If, after T_{FB}, S does not have to process a NewAsk(α_p, E_p) message such that $\alpha_p > cur\_ask_s$ or $H_s \oplus E_p \neq_l H_s$, then, for all T, $T \geq T_{FB}$,

(i) cur_ask of S = cur_ask_s, cur_max of S = cur_max_s and H of S $=_l$ H_s, and

(ii) S does not send a Final_Bid(..) message at T.

Proof: For cur_ask or cur_max to increase, or for H to extend, at least one of the ppe's must happen. By given, ppe2 does not happen, and by the lemma, ppe1 and ppe3 cannot occur. So, (i) is true.

At T_{FB}, IsSentFB is true. So long as it remains true, T3 does not send a Final_Bid(..) message. It can become false, only when the value of cur_ask or cur_max increases or H extends. By (i), it is not possible.

Lemma 4: Say, S is any server in the sub-tree rooted at R and R ≠ S. Let cur_max_r, cur_ask_r and H_r be the values of R's cur_max, cur_ask and H respectively at time T. If S is accepting bids at T, then R cannot evaluate TC2 to be true at T.

Proof: Let us define $\Delta l = l_r - l_s$. Proof is by induction on Δl. Suppose that $\Delta l = 1$. That is, S is a child server of R. Assume (to the contrary) that R evaluates TC2 to be true at T. This means that R must have received from S a Final_Bid(cur_ask_r, H_r, *, *) message before T. Say, S sent that message at time t_{FB}. Note that at t_{FB} cur_ask of S = cur_ask_r and H of S $=_l$ H_r. Since T is accepting bids at T, by lemma 3, S must have received during (t_{FB}, T) a NewAsk(α_r, E_r) message such that $\alpha_r > cur\_ask_r = \max\{cur\_max_r, cur\_ask_r\}$ or $H_r \oplus E_r \neq_l H_r$, from R. Note that R sends a NewAsk(..) message in Task T5, or in Task T4 after receiving a NewAsk(..) message. In both contexts, R updates its variables cur_max, cur_ask and H with the contents of the received message. Given that $\alpha_r > cur\_ask_r = \max\{cur\_max_r, cur\_ask_r\}$ or $H_r \oplus E_r \neq_l H_r$, it cannot therefore be true that cur_max_r, cur_ask_r and H_r are the values of R's cur_max, cur_ask and H respectively at T. This is a contradiction.

Suppose that $\Delta l > 1$ and the lemma is true for $\Delta l - 1$ (induction hypothesis). Let P be a child server of R such that S is also a server in the sub-tree rooted at P. (Such a P must exist as $\Delta l > 1$.) Note that $l_p - l_s = \Delta l - 1$. Let cur_max_p, cur_ask_p and H_p be the values of P's cur_max, cur_ask and H respectively at time T. We need to consider two cases: the condition ($cur\_ask_p = cur\_ask_r$ and $cur\_max_p = cur\_max_r$ and $H_p =_l H_r$) is (a) true and (b) false. For case (a), the lemma is true by induction hypothesis: P has not evaluated TC2 to be true at T, hence it could not have sent Final_Bid(cur_ask_r, H_r, *, *) without which R cannot evaluate TC2 to be true. For case (b), two

sub-cases arise: P has sent (b1), and has not sent (b2), a Final_Bid(cur_ask_r, H_r, *, *) before T. The lemma is done for (b1) as R cannot evaluate TC2 to be true without having received P's Final_bid(..) message. For (b2), the lemma is proved based on the same reasoning used for the case $\Delta l = 1$: assume that R evaluates TC2 to be true at T; since the variables of P change after P sent the Final_bid(..) message, P must have received a NewAsk(..) message from R before T (by corollary 3.1); therefore cur_max_r, cur_ask_r and H_r cannot be the values of R's cur_max, cur_ask and H respectively at T.

Remark: The lemma implies that the root server RS cannot have its TC2 being true and therefore cannot terminate the round, while a server in the system is taking bids. That is, before RS terminates the round it must receive every episode that a server constructs with non-empty *freqList* during that round.

Termination with Unique Final Bid

Say, *AS* quotes its final ask and bidders stop bidding by real-time T_0. Sometime after T_0, no server will generate an *E* with *E.freqList* $\neq \perp$. RS eventually constructs, and therefore every server also constructs (due to downward diffusion of episodes), the global round history, RH, which contains all bids accepted by every server in the system. Every server, starting from the leaf ones, sends *Final_Bid*(*, RH, *, *) message. So, RS will have its TC met and sends to AS a *Final_Bid()* message, say FB, thus terminating the round. Since AS does not increase on its final ask, RS or any server will not receive a *NewAsk* message that can force it to extend *its round_close_time.* By corollary 3.1, RS will never send another *Final_Bid*.

Fairness

Say a server generates in a round an Episode E with non-empty E.*freqList*. This means that RS could not have terminated that round. Given that a path exists between any two servers and that E is diffused both downwards and upwards along the tree, every other server will receive E and if E extends the local H, the registered bidders are informed, and are given at least $D_b - 2\delta$ time, to place new bids.

Accuracy follows from the fact RS has constructed the global round history RH when it terminates the round.