Enhanced DR-tree for low latency filtering in publish/subscribe systems

Luciana Arantes, Maria Gradinariu Potop-Butucaru, Pierre Sens, Mathieu Valero
LIP6 - University of Paris 6 - INRIA, 104, avenue du Président Kennedy 75016, Paris, France
[FirstName.LastName]@lip6.fr

***Abstract*—Distributed R-tree overlays emerged as an alternative for efficiently implementing DHT-free publish/subscribe communication primitives. Overlays using R-tree index structures offer logarithmic delivery garantis, guarantee zero false negatives and considerably reduce the number of false positives. In this paper we extend the distributed R-trees (DR-trees) in order to reduce event delivery latency. Our optimizations target both the structural organization of the DR-Trees and the publication policies. The contribution of the current work steams in an extensive evaluation of the novel structure along four parameters: latency, load, scalability and the rate of false positives. The enhanced structure performs better than the traditional distributed R-tree in terms of delivery latency. Additionally, it does not alter the performances related to the scalability, nor the load balancing of the tree, and neither the rate of false positives and negatives filtered by a node.**

***Index Terms*—Publish/subscribe, Distributed R-Trees, Performance evaluation, Distributed multiplayer games**

I. Introduction

Publish/Subscribe primitives are efficient communication abstractions very popular in large scale systems where the number of nodes participating to a particular application is limited to a strict subset of the network nodes. Recently, publish/subscribe primitives found an interesting application in massively distributed video games where the pertinent information has to be efficiently distributed to the interested parties only. In these systems the amount of information a node has to process is critical since nodes have to conserve their computational power and bandwidth in order to fully satisfy the users expectation. Therefore, communication primitives targeted to reduce noisy events (false positives or negatives) are highly requested. Publish/Subscribe implemented on top of distributed R-trees (DR-trees) overlays, first introduced in [1], are proven to be efficient communication primitives. They have been designed to offer zero false negatives and reduce the number of false positives. Interestingly, they also offer a logarithmic delivery complexity. These characteristics make them appealing for applications like P2P video games where nodes have to process only pertinent information. However, their main drawback is their unbalanced load. That is, nodes in charge of the top levels of the overlay have to deal with an important load due to the high traffic they have to process (new subscriptions and events are generally filtered using a top-down strategy). Therefore, in P2P video games where the maintenance of the overlay is performed by the players themselves[1], delivery latency of updates and nodes' load are import concerns. The aim of this paper is to improve DR-tree in order to offer low delivery latency while maintaining their original features related to reduced number of noisy events and load balancing.

[1]Note that in these systems players are mainly concerned with their bandwidth and fast reactivity.

Our contribution. In this paper we optimize the DR-tree overlays in order to meet the requirements of massively distributed video games such that pertinent information is quickly distributed to all the interested parties without degrading the load of nodes neither increasing the number of noisy events. Our optimizations are twofold. First we target structural optimization duplicating the virtual links between nodes in the distributed R-tree. Then we propose novel strategies for events dissemination that fully exploit the new added links. The real contribution of the paper steams in the evaluation of [1] according new criterias and in the extensive evaluation of the performances of our optimized publish/subscribe communication primitive targeting latency. The new structure performs better than the traditional distributed R-tree in terms of latency. Additionally, it does not alter the performances related to the structure scalability, the load balancing and the rate of false positives and negatives a node has to filter.

II. Related Work

Publish/subscribe systems have received much attention and have been extensively studied in the last few years [2], [3]. In such systems, consumers specify *subscriptions*, indicating the type of content that they are interested in, using some predicate language. For each incoming message (*event*), a *content-based router* matches the message contents against the set of subscriptions to identify and route the message to the set of interested consumers. Therefore, the consumers and the producers are unaware of each other and the destination is computed dynamically based on the message contents and the active set of subscriptions.

Traditional content routing systems are usually based on a fixed infrastructure of reliable *brokers* that filter and route messages on behalf of the producers and the consumers. This routing process is a complex and time-consuming operation, as it often requires the maintenance of large routing tables on each router and the execution of complex filtering algorithms (e.g., [4], [5], [6]) to match each incoming message against every known subscription. The use of summarization techniques (e.g., subscription aggregation [7], [8]) alleviates those issues, but at the cost of significant control message overhead or a loss of routing accuracy.

Another approach to content routing is to design it free of broker infrastructure, and organize publishers and consumers in a peer-to-peer overlay through which messages flow to interested parties. Several designs of DHT-based peer-to-peer publish/subscribe systems were proposed [9], [10], [11], [12], [13], [14], [15]. The main advantage of these approaches is their scalability, although most of them suffer from two problems: the loss of accuracy (apparition of false negatives or false positives) and poor latency in scenarios with high churn. In this paper we are interested in publish/subscribe communication primitives that meet the massively distributed games requirements: reduce number of noisy events, load balancing and low latency. Hence, for such approaches to be efficient, the overlay on top of which the primitive is implemented must: avoid *false negatives* (a registered consumer failing to receive a message it is interested in); minimize the occurrence of *false positives* (a consumer receiving a message that it is not interested in); *self-adapt* to the dynamic nature of the systems, with peers joining, leaving, and failing; *balance the load* of the subscribers in charge of the overlay maintenance and efficiently distribute events to the interested parties (provide a low latency). None of the previously mentioned systems meet all these criteria.

VBI [16] is a framework to build several containement-based structures (such as R-tree, M-tree, X-tree etc. . .) over a virtual balanced binary tree. The virtuality of that tree is a very important point as it introduces a distinction between the overlay topology and peers organization. VBI [16] distingues two kind of logical nodes; data nodes -leaves, that stores objects- and routing nodes -internal nodes-. Each peer is responsible for one data node and one routing node. Each routing node maintains sideways routing table containing links to particular nodes at the same level and an upside table that contains links to its ancestors. Those extra routing tables are used to balance load amongst peers using as much as possible horizontal routing. The fixed degree of the logical tree and the way it is mapped on peers are the two of the main conceptual differences with our approach. Moreover VBI mainly targets fair load balance while our approach mainly target low delivery latency. Our structure is more specialized as it's dedicated to use spatial filters as a publish/subscribe underlayer.

In massively distributed video games the most popular publish/subscribe system is Mercury having a similar design with [15]. Mercury [17] is a peer-to-peer DHT supporting multi-attribute range-queries and explicit load balancing on top of which a First Person Shooter (FPS) dedicated publish/subscribe has been built and used in Caduceus [17] and Colyseus [18]. Subscriptions are mapped on range queries, publications on classic DHT $put()$ operation and each attribute to a dimension. Mercury creates one ring per dimension; each peer belongs to several rings. It doesn't scale with dimension number however it performs well in systems with moderated number of dimensions. Each peer knows for each ring its predecessor, its successor and has k long links obtained by lazy random-walk. k may vary from one peer to another, from one node to another and from one ring to another. On publication, an event is inserted in each ring where it is routed according to the corresponding attribute (resp. dimension). Under the assumption of uniform node's ranges on each ring, Mercury route any event in $O((log^2 n)/k)$. Due to the ring-overlay design nodes in Mercury have to process both false positives and negatives.

III. Publish/Subscribe Model

We consider a distributed dynamic system where publishers and subscribers are organized in a broker-free overlay. Every peer in the overlay may have three roles: publisher/subscriber and router. Also, the peers may participate in the event dissemination, i.e., the event matching and forwarding process is completely distributed among the peers in the system.

In the following we borrow the model proposed in [11], [14], [1]. We assume that an event contains a set of *attributes* with associated values. In this work we consider complex filters expressed as the conjunction of multiple range predicates. Geometrically, these complex filters define poly-space rectangles in an Euclidean space. This representation captures well the range filters expressed in most popular publish/subscribe systems (e.g., [2], [19], [7], [20]).

An event specifies a value for each attribute and corresponds geometrically to a point. Without restraining the generality, we illustrate our algorithms on two-dimensional filters corresponding to rectangles in a two-dimensional space. If one attribute is undefined, then the corresponding rectangle is unbounded in the associated dimension. If an attribute is composed of disjoint ranges, the subscription will be represented as multiple rectangles. In that case, we can split the original subscription into several new subscriptions, one per rectangle, or merge the multiple ranges of every attribute to produce a single subscription, at the price of degraded accuracy.

Many publish/subscribe systems are based on the property of *subscription containment*,[2] which is defined as follows: subscription S_i contains another subscription S_j (written $S_i \sqsupseteq S_j$) iff any event m that matches S_j also matches S_i. The containment relationship is transitive and defines a partial order. Geometrically, subscription containment corresponds to the enclosure relationships between the poly-space rectangles. When organizing the peers based on the containment relationship of their subscriptions, only the peers that are interested in an event will participate in the matching and forwarding procedure. In this way, events can be quickly disseminated without incurring significant filtering cost.

IV. R-Trees Overlays

In this section we recall the main caracteristics of the R-Tree index structure and its distributed version.

A. R-Trees index structures

R-trees were first introduced in [21]. An R-tree is a height-balanced tree handling objects whose representation can be circumscribed in a poly-space rectangle. Each leaf-node in the tree is an array of pointers to spatial objects. An R-tree is characterized by the following properties:

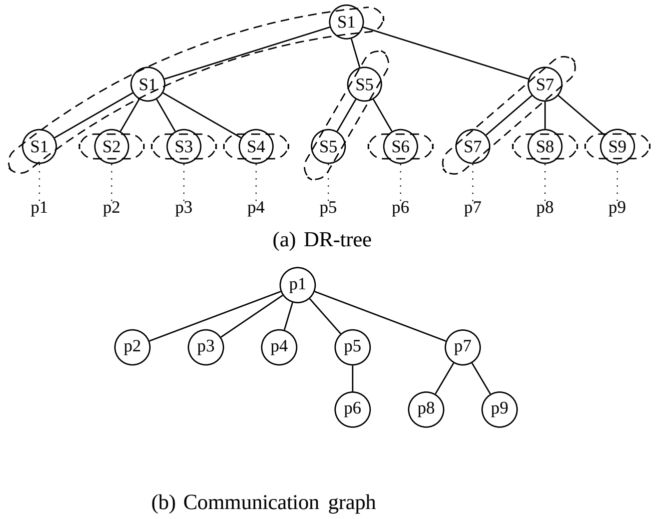
- Every non-leaf node has a maximum of M and at least m entries where $m \leq M/2$, except for the root.
- The minimum number of entries in the root node is two, unless it is a leaf node. In this case, it may contain zero or one entry.
- Each entry in a non-leaf node is represented by (mbr,p), where the mbr is the *minimum bounding rectangle* (MBR) that encloses the MBRs of its child node and p is the pointer to the child node. Each entry in a leaf node is represented by (mbr,oid), where the mbr is the MBR that spatially encloses the object and oid is the pointer to the object.
- All the leaf nodes are at the same level.
- The height of an R-tree containing N objects is $\lceil log_m(N) \rceil - 1$.
- The worst space utilization for each node except the root is m/M.

In a classical R-tree structure, the actual objects are only stored in the leaves of the tree and the internal nodes only maintain MBRs.

[2]The term *covering* is also commonly used in the literature.

B. Distributed R-tree Overlay

Distributed R-trees (DR-trees) introduced in [1] extends the R-Tree index structures where subscribers are self-organized in a balanced virtual tree overlay based on the semantic relations between their subscriptions. We consider that each filter is a rectangle which can be represented by using coordinates in a two dimensional space. The overlay preserves the R-trees index structure features: bounded degree per node and search time logarithmic in the size of the network. Moreover, the proposed overlay copes with the dynamism of the system.



(a) DR-tree

(b) Communication graph

Fig. 1: Nodes distribution and resulting overlay

Each leave (subscriber) of the DR-Tree is assigned to a peer of the overlay. Furthermore, some peers can be responsible for both leaves and internal nodes of the DR-tree. Thus, since an internal node keeps information about the MBRs of its children, a peer filters for every internal node that it holds the events for the respective internal node's subtree. Some subscribers are responsible for both leaves and internal node of the DR-tree. The choice of which subscriber are promoted to be responsible to internal nodes are discussed in [1]. Figure 1a shows an example of a DR-Tree and a possible assignment of subscribers to peers: subscription $S_1 \dots S_9$ are distributed among peers $p1 \dots p9$. Each peer holds exactly one leaf of the DR-Tree and a dotted cloud emphasizes that the same subscriber can be both a leave and an internal node. For instance, $p1$ is responsible for subscription S_1, which appears in different levels of the tree including the root while $p6$ is responsible for S_6 which is just a leave. How nodes of a DR-tree are distributed amongst peers of the overlay depends on node join/leave and split algorithms which are described in [1].

The communication graph related to the DR-tree of Figure 1a, which expresses the communication links between the subscribers/peers, is shown in Figure 1b. Two peers are neighbors in the communication graph if and only if they hold nodes that are neighbors in the DR-tree, i.e., a node is neighbor of its children and father nodes. For instance, $p1$ and $p3$ are neighbors in the communication graph because $p1$ is responsible for S_1 and $p3$ is responsible for S_3 and the former is the father of the latter.

It is worth pointing out that in our approach, discussed in the following, links are added to the communication graph but the logical structure of the DR-tree is kept unchanged.

V. Optimized Distributed R-Tree

In this section we detail the optimizations we propose for the classical DR-trees described in Section IV. We address both the topological extensions and publishing strategies.

A. Topological extensions

In order to reduce the delivery time of published events and hence the latency of events distribution we have added links to the communication graph improving thus the connectivity of the corresponding DR-Tree. We propose tree extensions by adding links between peers based on the different relations between the nodes they hold, which are the following:

Brothers Connections: If two peers hold nodes which are brothers, i.e., have the same father node, a link between them is added to the communication graph. The brother relation is symmetric, transitive and non-reflexive. We could have linked all brothers in a ring or multi-ring structure. However, for the sake of efficiency, we have chosen a crossbar to keep brothers relation. This structure offers the maximal performance in terms of latency since messages within the brother set are routed in one hop.

Root Link Connections: In this extension, all peers of the communication graph have a link to the peer that holds the root node of the DR-Tree.

Ancestors Connections: A node is the ancestor of another node if and only if the former is the father of the latter or the father of an ancestor of the latter. In this extension, we consider thus that every subscriber is aware of all of its ancestors. Links are added to the communication graph to this end.

B. Publishing strategies

We denote that a local event is an event that has been published by the node itself, that an upward event is an event that a node has received from one of its children, and that a downward event is an event that a node has received from its father. A publishing strategy thus defines the traffic rules, i.e., the routes that local, upward and downward events should take.

The containment relation between MBRs entails the filtering in both directions. Therefore, in the classical DR-trees [1], the publishing strategy, denoted in the sequel **Double Wave strategy**, consists in forwarding the local event produced by a node both to its interested children and its father. Further, every internal node which receives an upward event from one of its children adopts the same strategy: it forwards the event to its interested children and its father. Note that the **Double Wave** publishing strategy was only described in [1]. The simulations were conducted based on a simplified strategy: publication via the root node.

In the following we introduce three new publishing strategies which exploit the addition links that characterize the topological extensions described above.

Brothers wave strategy: brother links are used to exploit "tree-locality" of a publication. The idea as follows: events that interest a node might also interest its brothers as well. Therefore, the publication strategy is the following: local events are forwarded to publisher's interested brothers, children, and father; upward events are also forwarded to the receiver's interested brothers and father (if the receiver is not the root); downward events are forwarded to the interested children.

Root link wave strategy: Local events are forwarded to the root peer, then they are sent downward to interested children.

Ancestors wave strategy: ancestor links are used to maximize messages diffusion parallelization. Local events are forwarded to interested children and every ancestors of the publisher; upward and downward events are forwarded to interested children.

VI. Performance Evaluation

This section presents a set of results aimed at evaluating the performance of the three new publishing strategies (Brothers wave, Root link wave and Ancestors wave) compared to the original one (Double wave).

The goal of our optimized DR-trees is to offer a publish/subscriber system that exploits locality of subscribers' interests and efficiently disseminates events by adding extra links between peers. They are thus very suitable for maintaining the state of distributed multiplayer games since optimized DR-Trees provide both filtering of information and low event delivery latency, which are essential features for these kind of application. It is worth remembering that in distributed multiplayer game, each participating node (player) only needs information relevant to his/her associated player. In this way, the optimized DR-tree can be used by the game application in order to update fast the view that each player (node) has of the game. Based on these arguments, our evaluation performance tests characterize the behavior of different distributed applications in terms of publication patterns and users interests, i.e., subscription distributions.

A. Simulation environment and parameters

Experiments were conducted on top of PeerSim[22], a Java-based discrete event simulator. They last 600 cycles where a cycle is a discrete unit of time. Publication frequency was 0.5 event per cycle for each peer. Network latency between two peers was 1 cycle with a jitter of $\pm$ 0.1 cycle.

We have considered a 2D virtual area of $[\![0, 1024]\!] \times [\![0, 1024]\!]$ and a network with 1024 peers with one subscriber per peer. The communication latency distribution between peers is homogeneous; the impact of heterogeneity and nodes placement will be investigated in future works. Each peer (subscriber) has just one zone of interest, whose height and width are uniformly randomly distributed between $[\![5, 50]\!]$, and one zone of publication. We denote the *covering zone* of a peer the MBR of the uppermost level that it holds.

Every non-leaf node of the DR-Tree has a maximum of M=8 and a minimum of m=4 entries, except the root which has 2 entries. For the sake of evaluation, nodes can be grouped by level: 0 is the root level, 1 is root's children level, and so on. The level of the leaves is equal to the RTree height which is equal to 4 in our experiments.

Subscription distribution: Most of the massively distributed video games present hotspot zones, i.e. "popular" regions in which a group of peers have similar interests. Thus, based on population distributed of existing games, we have considered in our experiments four hotspot distribution configurations for the 1024 peer subscriptions of the system:

- **Cold** (no hotspot): subscriptions are uniformly randomly distributed.
- **Warm** (not very "popular" hotspots): the number of hotspots is $1024/8 = 128$.
- **Hot** ("popular" hotspots): the number of hotspots is $\sqrt{1024} = 32$.
- **Burning** (very "popular" hotspots): the number of hotspots is equal to $log(1024) = 10$ hotspots.

The *Cold* and *Warm* hotspot distributions respectively model the population distribution of deserted zones of DVE [3][23] and interested zones of FPS [4] [24] games. The *Hot* distribution represents the population distribution of dense zones of DVE like towns in MMO-RPG [5] (*World Of Warcraft*, [25], *Dofus* [26]) or popular islands of *Second Life*) [27]) while the *Burning* one maps the population distribution of massive battlefields in MMO-RPG or wide events (concert, meeting).

Publication pattern: Peers (players) subscribe to the geographic area where they are located and publish events related to their positions/movements/actions. However, in video games, players are usually interested in a small part of the game map (zone) and they only interact with entities that are in that zone. Such a behavior thus implies that the publication zone of a peer corresponds to its zone of interest, i.e., a peer publishes just in its own zone of interest. To our experiments, we have then considered that publications are uniformly randomly distributed in publishers' subscription zones.

Metrics: As previously explained, our goal in proposing new publishing strategies is to provide low publication delivery latency without unbalancing load, increasing noisy events such as false positives (DR-Tree does not present false negatives) or limiting scalability of the system. Hence, the metrics we have used to evaluate the four strategies are:

- **Latency**: the average time (in cycles) elapsed between the moment an event is published and its delivery to all subscribers which are interested in it.
- **Message load**: this metric concerns both the *fan in*, the average number of received messages per peer, and *fan out*, the average number of sent messages per peer.
- **False positive**: the average number of false positives per level of the DR-Tree.
- **Scalability**: this metric concerns the delivery latency when the number of peers increases.

B. Latency

Latency measures the elapsed time between the moment an event takes place and the moment all interested subscribers are aware of it (e.g. the time elapsed between a bomb's explosion and the moment every near player is warned of it; the elapsed time between a player kills another one and the moment every witness "sees" this action, etc.).

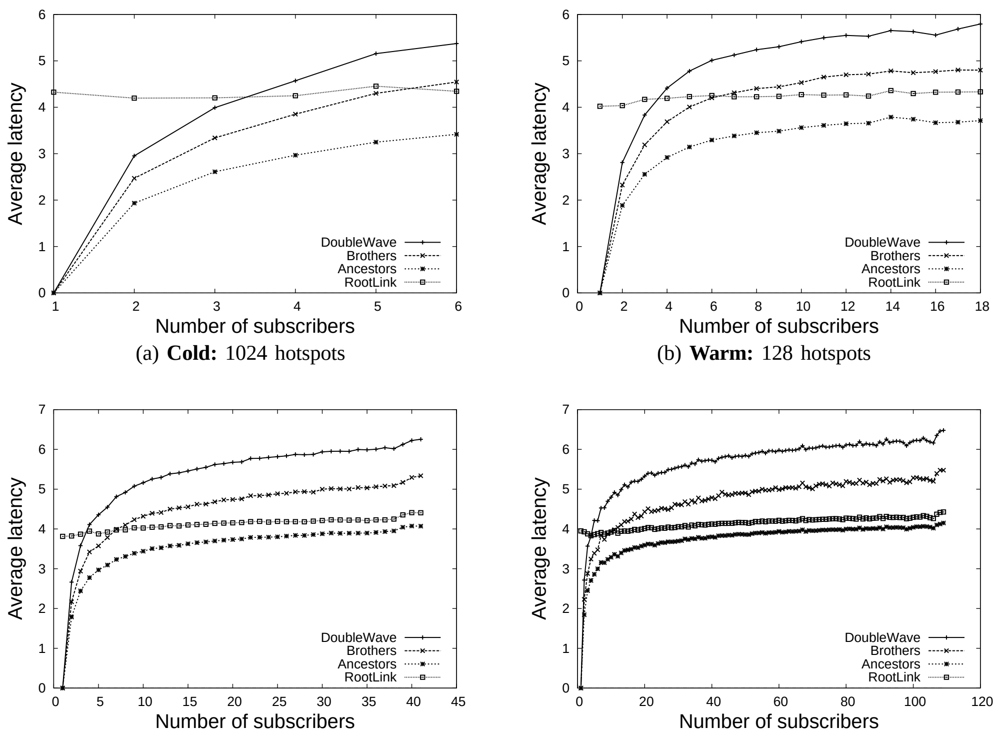
Figure 2 shows the latency evaluation results for the four publication strategies defined in Section V. X-axis corresponds to the number of subscribers concerned by a publication. Notice that the colder hotspots are, the lower the number of interested subscribers is. Y-axis corresponds to the average total publication time.

Since a peer publishes in its respective zone of interest, a publication is delivered at least to it. Thus, except for the *Root Link* strategy where every publish event must be first send to the root, whenever an event is delivered to exactly one peer (the event publisher), the average global publication time is equal to zero independently of the hotspots distribution. The *Root Link* strategy exhibits a linear behavior regardless of

[3]Distributed Virtual Environments

[4]First Person Shooter

[5]Massively Multiplayer Online Role Playing Game



(a) **Cold:** 1024 hotspots (b) **Warm:** 128 hotspots

(c) **Hot:** 32 hotspots (d) **Burning:** 10 hotspots

Fig. 2: Average total propagation time versus event popularity

the publication popularity. In fact, the latency of this strategy depends only on the height of the tree since publications are systematically sent to the root and then propagated to the leaves. However, for the other three strategies, the higher the number of peers concerned by an event, the higher the chances that the event will have to be forwarded far from the the publisher and hence routed through more peers. Therefore, such an event will take more time to be delivered to all interested subscribers.

Ancestors, *Brothers* and *Root Link* latency gains are quite significant for all hotspot distributions compared to the *Double Wave*. In particular, *Ancestors* strategy is around 35% better than *Double wave* strategy for the *Burning* distribution, and around 45% better for the *Cold* distribution. *Ancestors* strategy is always better than the other strategies because it parallelizes the diffusion of publications. We also observe that below 4 subscribers the *Root link* strategy is less efficient than the other ones since the latter benefit from locality of subscription which avoids the propagation of publications to the root. On the other hand, when such a locality decreases, i.e., the number of subscribers interested in the event increases, publications need to be forwarded to the root, which explains why both the *Root Link* and the *Ancestors* strategies present lower delivery latency compared to the other two since the extra link to the root allow publications to bypass inner levels.

It is worth pointing out that the curves of Figure 2 could be roughly interpreted as “the number of hops versus the number of reachable nodes in the communication graph” except for the *Root Link* strategy. Thus, since the communication graph is a tree, whose height is majored by the height of the DR-tree, the curves have a logarithmic behavior. The inflexion point of curves corresponds to the tree’s height. Figure 2 shows also that hotspots distribution has small impact on overall latency; in any case, each strategy’s curve stabilizes around the same value, which is an interesting result for video games since latency is always a matter of concern for them. Furthermore, the zone of interest of a player is very likely to change during the game but, due to the mentioned stabilization, such a change will probably not affect the game’s reactivity.

C. Message load

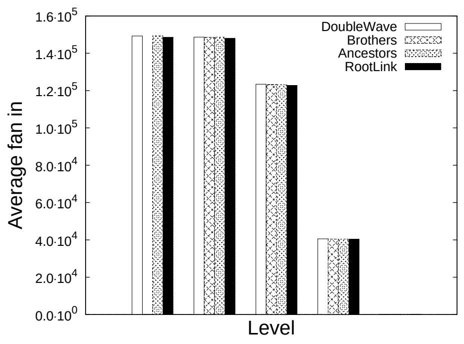
In the previous section we have shown that *Ancestors*, *Brothers* and *Root Link* strategies provide significant latency gains when compared to *Double wave*. In the following we investigate two metrics *fan in* and *fan out* which are related to the node load. For a given peer, both the *fan in* and the *fan out* are dependent of the following three factors:

- peer’s zone of interest
- peer’s routing upward activity
- peer’s routing downward activity

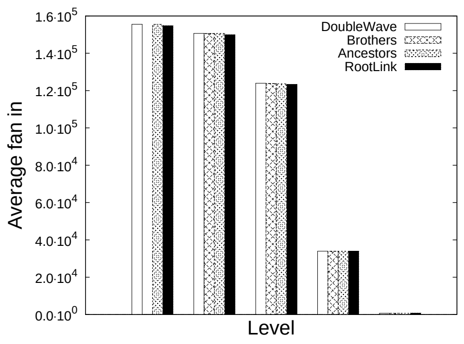
Note that these factors may have very different order of magnitude according to the publication strategy and the level of the peer in the overlay.

a) **Fan in evaluation:** Figure 3 shows some results related to the *fan in* metric. The X-axis corresponds to the R-Tree levels. The leftmost level is the root, the rightmost level corresponds to the leaves, and the in-between levels correspond to internal nodes. For the Y-axis, each bar represents the average *fan in* of peers at a given level. The standard variation of *fan in* for peers at each level is very low.

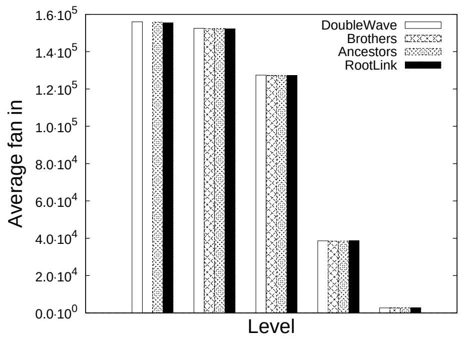
We can observe in Figure 3 that all strategies are roughly equivalent in terms of *fan in*, regardless of the hotspot distribution. Since DR-tree routing avoids false negatives, a peer receives an event either if it is interested in the event or if some of its children are. In other words, a peer receives an event only if the latter is in its zone of interest or in



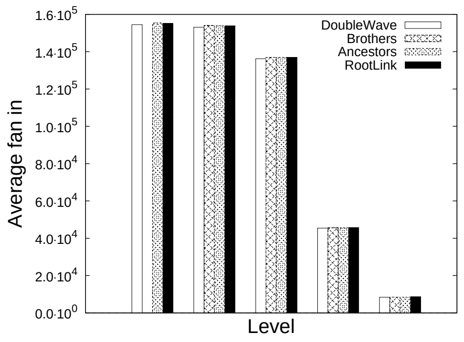
(a) **Cold:** 1024 hotspots



(b) **Warm:** 128 hotspots



(c) **Hot:** 32 hotspots



(d) **Burning:** 10 hotspots

Fig. 3: Fan in versus level

its covering zone. The closer to the root a peer is, the large its covering zone is and thus the higher the number of upward events it receives which explains why the bars of Figure 3 decreases when the level increases, independently of the hotspots distribution.

An interesting remark is that in the case of the *Brothers* strategy, the root peer *fan in* is equal to 0 and strictly equivalent to a leaf peer for all hotspot distributions. As its children know each other, the root peer is not involved in routing events and thus it receives only those events in which it is interested.

b) **Fan out evaluation:** Figure 4 presents some evaluation results of the *fan out* metric. Like to the *fan in* figure, the X-axis represents the DR-tree levels. In the Y-axis, each bar corresponds to the average *fan out* of peers at each level. The standard variation of *fan out* for peers of a given level is very low.

Similarly to the *fan in*, the closer to the root a peer is, the higher the number of upward events it has to forward to both its father and its children which have interest in them.

Two points are worth remarking with regard to the *Brothers* strategy. Firstly, the internal nodes are slightly more loaded than with other strategies. The explanation comes from the "horizontal routing" of such strategy which mostly involves leaves and internal peers in order to reduce the cost of event's upward propagation. Secondly, as already mentioned in the *fan in* evaluation, the root peer is not engaged in the routing of events. Hence, its *fan out* is equivalent to a leaf peer for any hotspot distribution.

Finally, we should point out that we observe an increase of load on the root node in the *Root Link* strategy since the root always forwards the publications to all its children which are interested in them. Contrarily, in the other strategies, the root does not forward the publications to those children from which it receives them.

D. False positive

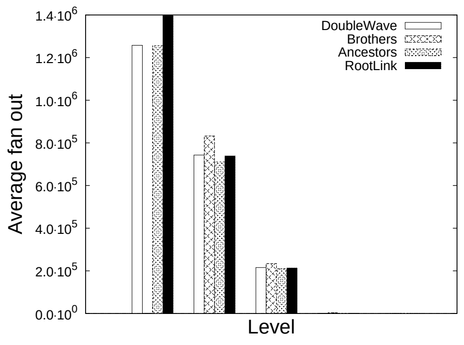
An event is considered as a false positive by a peer if the latter is not interested in it, i.e., if the event is in the peer's covering zone but not in the peer's zone of interest.

Figure 5 presents our evaluation results related to false positives. X-axis is the levels of the DR-Tree similarly to the *fan in* and *fan out* figures. In the Y-axis, each bar corresponds to the average percentage of false positives for peers of each level. As this metric is highly related to the *fan in*, the standard variation is also very low.

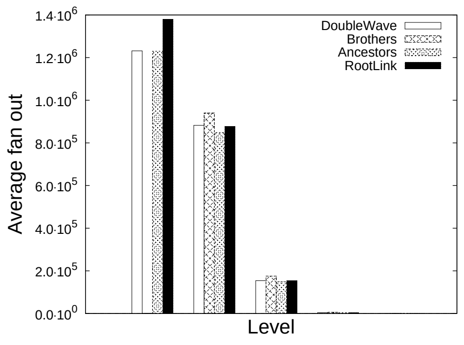
All strategies are equivalent in terms of false positive rate independently of the hotspots distribution. For a leaf peer, the zone of interest and covering zone are equals. Hence, it receives only events in which it is interested, i.e., no false positive occurs. However, the closer to the root a peer is, the wider its covering zone is and thus the higher the chances that it receives events that are in its covering zone but not in its zone of interest which leads to higher false positive rates.

We can also observe in the same figure that the overall false positive rate decreases with the popularity of the hotspots since the number of zones of interest that overlap increases as well. The more they overlap, the higher the chances for a peer to receive events that are in its zone of interest which thus leads to slightly lower false positive rate.

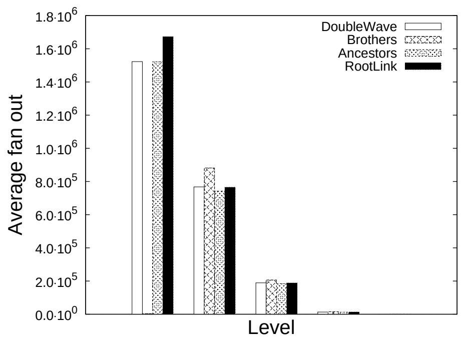
A third remark is that in the case of the *Brothers* strategy, root peer behaves like a leaf peer. As explained in the description of this strategy, the root peer only receives events in which it is interested in. Therefore, no false positive occurs.



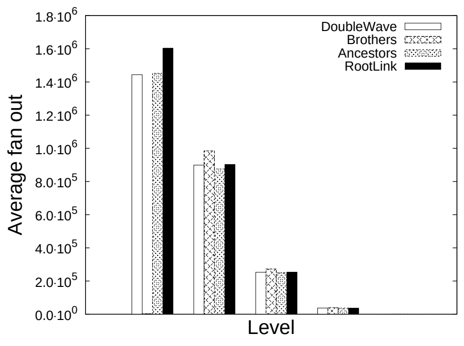
(a) **Cold:** 1024 hotspots



(b) **Warm:** 128 hotspots

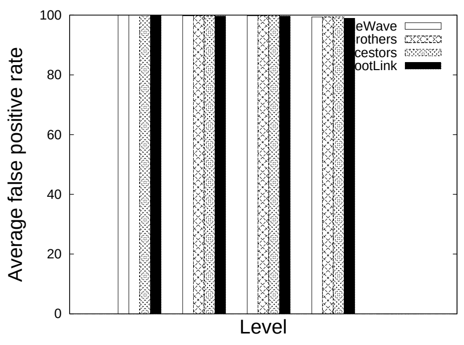


(c) **Hot:** 32 hotspots

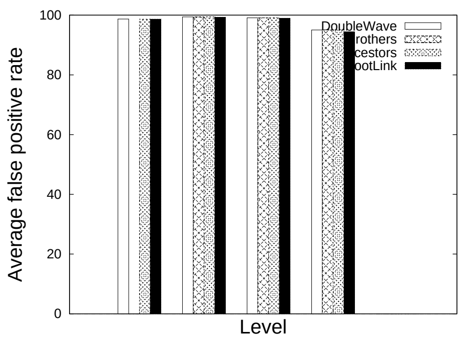


(d) **Burning:** 10 hotspots

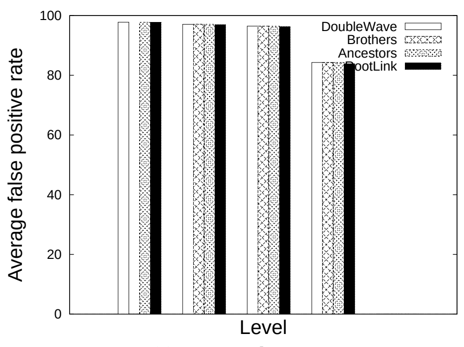
Fig. 4: Fan out versus level



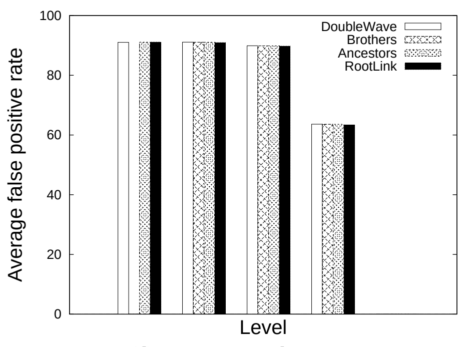
(a) **Cold:** 1024 hotspots



(b) **Warm:** 128 hotspots



(c) **Hot:** 32 hotspots

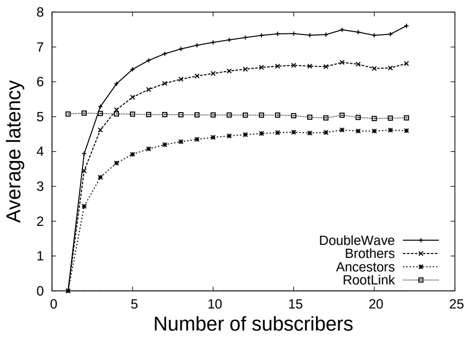


(d) **Burning:** 10 hotspots

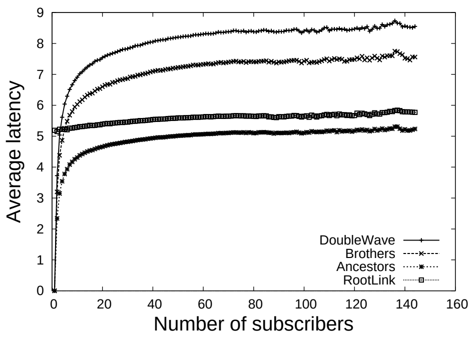
Fig. 5: False positive rate versus level

E. Scalability

We have conducted the same set of experiments as the ones shown in Figure 2, but with 10,000 peers instead of 1024. Our results are shown in Figure 6.



(a) **Cold:** 1024 hotspots



(b) **Hot:** 128 hotspots

Fig. 6: Total propagation time for 10,000 peers

The shape of the curves is quite similar to those of Figure 2 (i.e., similar inflexion points, linear behavior for *Root Link* strategy and asymptotic behavior for the others). The 10-times multiplication of peers number results in an increase of the average latency by 25% for all strategies. Such an overhead can be explained since latency is closely related to the communication graph's height which is majored by DR-tree's one which grows logarithmically with peers number. The DR-tree we have considered in our experiments has a degree of (m=4;M=8) which implies that the height of the tree (and therefore the height of the communication graph) increases when the number of peers grows from 1024 to 10,000.

VII. Conclusion

In this article we proposed structural modifications of DR-Trees by adding shortcut links in their communication graph in order to efficiently disseminates events. Based on the results of extensive evaluation experiments, our paper shows that, compared to the traditional DR-Tree, our optimized DR-Trees reduce event delivery latency, do not degrade load balancing or scalability of the system, and do not entail more false positives. Furthermore, the same evaluation performance tests, which characterize both publication pattern and subscription distributions of different types of multiplayer games, confirme that our optimized DR-tree meet the requirements of distributed video games, i.e., scalability, low publication latency, reduction of noisy events, and load balancing. Note that these are essential concerns in distributed games in order to maintain fairness between players and conserve their computational power and bandwidth.

References

[1] S. Bianchi, A. K. Datta, P. Felber, and M. Gradinariu, “Stabilizing peer-to-peer spatial filters,” in *ICDCS*, 2007, p. 27.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra, “Matching events in a content-based subscription system,” in *PODC*, 1999, pp. 53–61.

[3] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, “The many faces of publish/subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[4] M. Altinel and M. Franklin, “Efficient filtering of XML documents for selective dissemination of information,” in *VLDB*, 2000, pp. 53–64.

[5] C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi, “Efficient filtering of XML documents with XPath expressions,” *VLDB Journal, Special Issue on XML*, vol. 1, no. 4, pp. 354–379, 2002.

[6] Y. Diao, P. Fischer, M. Franklin, and R. To, “YFilter: Efficient and scalable filtering of XML documents,” in *ICDE*, 2002.

[7] A. Carzaniga, D. Rosenblum, and A. Wolf, “Design and evaluation of a wide-area event notification service,” *ACM TOCS*, vol. 19, no. 3, pp. 332–383, 2001.

[8] C.-Y. Chan, W. Fan, P. Felber, M. Garofalakis, and R. Rastogi, “Tree pattern aggregation for scalable XML data dissemination,” in *VLDB*, 2002.

[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowston, “Scribe: A large-scale and decentralized application-level multicast infrastructure,” *IEEE JSAC*, vol. 20, no. 8, pp. 1489–1499, 2002.

[10] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, “Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination,” in *Proceedings of the International Workshop on Network and OS Support for Digital Audio and Video*, 2001.

[11] A. Gupta, O. Sahin, D. Agrawal, and A. E. Abbadi, “Meghdoot: Content-based publish:subscribe over P2P networks,” in *Middleware*, 2004.

[12] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, “Epidemic algorithms for reliable content-based publish/subscribe: An evaluation,” in *ICDCS*, 2004.

[13] R. Chand and P. Felber, “Semantic peer-to-peer overlays for publish/subscribe networks,” in *Euro-Par*, 2005.

[14] E. Anceaume, A. Datta, M. Gradinariu, G. Simon, and A. Virgillito, “A semantic overlay for self-* peer-to-peer publish subscribe,” in *ICDCS*, 2006.

[15] S. Voulgaris, E. Rivire, A. Kermarrec, and M. van Steen, “Sub-2-Sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks,” in *IPTPS*, 2006.

[16] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, “Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes,” in *ICDE ’06: Proceedings of the 22nd International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2006, p. 34.

[17] A. R. Bharambe, M. Agrawal, and S. Seshan, “Mercury: supporting scalable multi-attribute range queries,” *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 353–366, 2004.

[18] A. Bharambe, J. Pang, and S. Seshan, “Colyseus: a distributed architecture for online multiplayer games,” in *NSDI*. Berkeley, CA, USA: USENIX Association, 2006, pp. 12–12.

[19] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, “Content based routing with Elvin4,” in *AUUG2K*, 2000.

[20] G. Cugola, E. D. Nitto, and A. Fugetta, “The JEDI event-based infrastructure and its application to the development of the OPSS WFMS,” *IEEE Transactions on Software Engineering*, vol. 27, no. 9, pp. 827–850, 2001.

[21] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1984, pp. 47–57.

[22] “Peersim: A peer-to-peer simulator.” [Online]. Available: http://peersim.sourceforge.net/

[23] Solipsis opensource decentralized metaverse. [Online]. Available: http://www.solipsis.org/

[24] Unreal tournament 3. [Online]. Available: http://www.unrealtournament.com/

[25] World of warcraft. [Online]. Available: http://www.worldofwarcraft.com/

[26] Dofus. [Online]. Available: http://www.dofus.com/

[27] Second life. [Online]. Available: http://secondlife.com/