Distributed Optimization of Event Dissemination Exploiting Interest Clustering

Max Lehn, Robert Rehner and Alejandro Buchmann
Databases and Distributed Systems, Technische Universität Darmstadt, Darmstadt, Germany
{mlehn, rehner, buchmann}@dvs.tu-darmstadt.de

***Abstract*—In distributed real-time applications, such as online games or interactive conferencing systems, timeliness of update event dissemination is a prevailing requirement. Direct connections are often the best solution. However, participants' connection bandwidths, especially uplink capacities of asymmetric end user Internet connections, impose a limit to this.**

We present and evaluate a lightweight local algorithm that optimizes many-to-many event dissemination in highly dynamic scenarios. The algorithm exploits the high clustering of interest networks, as observed in virtual environments based on local vision, but also in social networks of many kinds. Our approach allows reducing traffic by aggregating messages as well as balancing traffic among participants based on their capabilities.

***Index Terms*—Distributed applications, multicast, peer-to-peer, overlay, latency, optimization.**

I. Introduction

In massively multiplayer online games (MMOG), large numbers of players connect over the Internet to join a common virtual world. With these games running in real time, there is an imminent need to synchronize the virtual game worlds of all players as tightly as possible to maintain user satisfaction. Noticeable differences in the synchronization among users due to end-to-end latencies or bandwidth limitations will negatively impact user experience. In most games, the relevance of information about game events for a particular player is shaped by the virtual world vicinity. Like in the real world, close events are more important than distant ones. Therefore, the highest communication demand is within local groups.

Similar requirements can be imposed by other kinds of real-time group communication applications, e.g., text, audio, or video chat in social networks. The envisioned scenarios have in common that they need a many-to-many communication in potentially dynamic groups. We have identified the following challenges: (1) The targeted applications are very latency-sensitive. Since latency in the Internet is not predictable, however, overlays can only provide best-effort guarantees. (2) Every participant is an event producer and has an individual set of interested participants. (3) Virtual locations of participants, and therefore their interests, are highly dynamic. (4) Interest among participants often has gradations. While some are of high priority, delay or message loss from others might be tolerable. (5) Some participants may be of particularly high interest, causing heterogeneous fan-outs. (6) The participants' resources are heterogeneous. (7) Additional infrastructure (e.g., multicast servers) might or might not be available.

With simple server-based solutions, physical server placement is crucial for keeping latency low (1). This raises the cost of service deployment compared to a simple server setup. Traditional application layer multicast (ALM) solutions, e.g., [1], [2], [6], form multicast groups, which are not optimized with respect to (2) and (3). These are conceptually tackled by context-aware [3] or parametric [7] publish/subscribe. Such generic approaches, however, do not optimize topology for latency based on interest locality. Peer-to-peer approaches for networked virtual environments, such as VON [5] and pSense [12], focus on interest management, but do not employ sophisticated event dissemination strategies. Points (5)-(7) call for an adaptive solution.

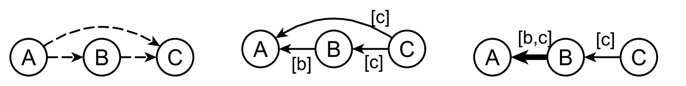
II. System Overview

We present an approach that addresses these challenges by employing neighboring peers for message multiplication only where necessary. We apply a *local optimization* algorithm to incrementally improve the forwarding configuration. A node always starts sending its messages to the receivers directly. When it exceeds its available bandwidth limit, it will request other nodes with spare bandwidth to serve as message forwarders and multipliers. More specifically, each node locally evaluates a *utility function* based on its local knowledge to decide where it is appropriate to forward its messages using a neighboring node. This is most efficient if the forwarding node itself is a receiver of the data. Then, no additional messages are required. Essential for this to work is a strong clustering of the interests among participants. In virtual environments, this is an immediate result of the players' local vision. But it is also true for social networks [10].

Eventually, we achieve two effects. First, we *shift load* among participants to balance the work based on the nodes' capabilities. Second, *aggregation* of forwarded messages saves total bandwidth. Of course, message indirection generally adds latency. Our approach therefore allows *trading latency for bandwidth* via a utility function: using direct connections usually induces the lowest possible latencies; investing some latency for indirection can save bandwidth.

A. Illustrative Example

For a simplified example, consider the interest cluster in Fig. 1a. Nodes A and B are both interested in the updates from C (in virtual environment terms, A and B can see C). Additionally, A is interested in B's updates. This interest graph shows a simple version of a cluster. The necessary update message path is thus directed in opposite direction of the interest edges (Fig. 1b). Intuitively, if C sends the same message to A and B, it can as well send the message



(a) Interest (b) Messaging (c) Aggregation

Fig. 1. A simple example of an interest cluster (a), the resulting messaging scheme (b) and its forwarding options (c).

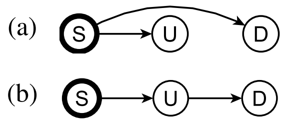


Fig. 2. Redirect operation initiated by node S

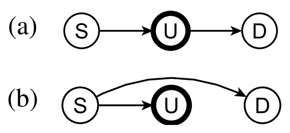


Fig. 3. Shortcut operation initiated by node U

only to B and let B forward it to A (Fig. 1c), halving its bandwidth usage. B now has to additionally send C's updates to A, but B's overhead in this situation is usually lower, because it can piggyback C's messages to its own, assuming there is some timing synchronization of B's and C's updates. Especially when the message payload is small, saving packets, and thus packet headers, reduces the total traffic. Compression and application-specific message aggregation techniques can bring further savings. But even if the total traffic reduction is marginal, this option allows shifting traffic between nodes. If B has more spare bandwidth than C, this may still be a good choice. In any case, the rerouting has to be paid with an increased latency on the path from C to A.[1] The decision whether to use the indirection depends on the available bandwidths and latencies between the nodes and finally on the application requirements.

III. System Model

Each node V represents one participant in the system and generates events with updates of its own state. Other nodes may be more or less interested in the updates from a particular node:

Definition 1: The interest function $I : V \times V \to [0,1]$ quantifies the interest level of one node in another. $I(v,u)$ is the interest of node v in the updates of node u. Generally, the interest level is continuous, but in simple cases, it can be binary. Then, the interest function defines as $I : V \times V \to \{0,1\}$.

Definition 2: $G = (V,E)$ is the subscription graph, where

$$E := \{(v,u) \mid v \in V \wedge I(v,u) \geq \tau\ \}.$$

$\tau \in (0,1]$ is the interest threshold, i.e, the minimum interest level required for receiving any updates from the corresponding node. Therefore, v is subscribed to u's updates iff $(v,u) \in E$. Although subscriptions are binary, the continuous level of interest serves as an indicator for the delivery priority.

For each subscription (v,u), there will be a corresponding *event flow* from u to v. Events may be sent directly from u to v or via any number of intermediate *forwarder* nodes. In this paper, we assume that events are sent on a regular basis and do not exceed a predefined maximum throughput per flow. The sequence of (forwarding) steps that messages of a particular flow take is defined as their *path*.

IV. Algorithm

In this section, we present the optimization algorithm. Each node maintains a forwarding table containing an entry for every outgoing event flow. Such a flow may either originate from this node, or the node is a forwarder. A forwarding table entry consists of a pointer to the next hop node as well as subscription metadata like origin, destination, interest level, and the current path length and total delay.

[1] If the network has triangle inequality violations (TIVs), the latency might even be lower. Zheng et al. [13] have shown that TIVs are common on the Internet. These can be exploited to improve latencies in overlay routing.

For each new subscription, a direct connection between origin and subscriber is established. All nodes continuously run iterations of their local path optimization algorithm. There are two basic operations for optimization:

Redirect (REDIR) For an existing outgoing event flow, a node selects a forwarder from its local neighborhood (i.e., its interest set), that will forward the events for the next hop (Fig. 2). This operation increases the path length by one.

Shortcut (SHORT) If a node is neither origin nor destination, it can take itself out of the path (Fig. 3). This operation decreases the path length by one.

The optimization algorithm works incrementally. In fixed time intervals, each node repeatedly performs one iteration:

0) Let r_v be the current forwarding configuration from the point of view of the local node v. Let
$$N_v := \{u \in V \mid (v,u) \in E \vee (u,v) \in E\}$$
be its open neighborhood in the subscription graph G.
1) Enumerate all possible rerouting operations (redirect and shortcut). Let $\mathcal{O}$ be the set of all options:
$\mathcal{O} := \{\text{SHORT}(s,d) \mid (s,d) \in r_v \wedge v \notin \{s,d\}\} \cup$
$\{\text{REDIR}(s,d,u) \mid (s,d) \in r_v \wedge u \in N_v \wedge (u,s) \in E\}$.
$(s,d) \in V \times V$ is a subscription of node d for s's events in v's forwarding configuration r_v. The operation $\text{REDIR}(s,d,u)$ will create a redirection of the path (s,d) via u (Fig. 2). The operation $\text{SHORT}(s,d)$ will remove v from the path (s,d) (Fig. 3). Applying an operation transforms the current configuration r into r'.
2) Find the best option according to the utility function $u : R \to \mathbb{R}$, with R being the configuration space: $r_{max} := \arg\max_{r' \in \{o(r) \mid o \in \mathcal{O}\}} u(r')$.
3) If $u(r_{max}) - u(r) \geq \epsilon$, then activate r_{max}, otherwise do nothing. The threshold $\epsilon \geq 0$ accounts for the transition cost. Greater values of ϵ reduce the risk of oscillations.

A. Utility Function

We identify two main factors for the utility function:

Bandwidth utilization Obviously, no node can exceed its bandwidth limit. Since many end user Internet connections provide only asymmetric bandwidths, we ignore the downstream limitation in this analysis. We will model the nodes' available bandwidths as penalty terms to allow exceeding bandwidth limits temporarily[2].

Path latency Targeting latency-critical applications, the latency of events from the originating node to the subscriber is the second factor. Depending on the application, there

[2] In real systems, available bandwidth cannot be exceeded, even temporarily. Such occurrences will lead to message loss, which can be controlled by prioritizing high-interest neighbors and reducing update frequency.

might be a an upper bound, a target value, and/or a lower bound for latency (e.g., to ensure fairness). The values may also depend on the interest level. Again, latency bounds are not modeled as constraints, but using penalties.

Other possible components of the utility function include a prediction on the reliability of forwarding nodes and the projected remaining duration of a subscription. Furthermore, queuing delays could be modeled more explicitly than only by the bandwidth utilization [9]. We assume that for each factor there is a cost function c_f. Optional weights w_f allow trading cost factors for each other. The total cost for a given configuration r is calculated as $c(r) := \sum_{f \in F} w_f * c_f(r)$. The utility is simply defined as the negative cost function; a negative utility value has no effect on the above algorithm.

For a concrete gaming scenario, we have derived the following cost functions, which will be used in the evaluation: $c(r) := w_{\text{bw,S}} * c_{\text{bw,S}}(r) + w_{\text{bw,U}} * c_{\text{bw,U}}(r) + w_{\text{lat}} * c_{\text{lat}}(r)$, with *bw,S* representing the link utilization of the sender (cf. Fig. 2 and 3, node S), *bw,U* the link utilization of the forwarder (node U), and *lat* the latency of the respective subscription. The link utilization cost function is $c_{\text{bw,v}}(r) := \max\left(U_v(r) - 0.4, 0\right)^6$, where $U_v(r)$ is the link utilization of node v with the configuration r. Latency cost is calculated as

$$c_{\text{lat}}(r) := \sum_{e \in E} \max\left(l_e(r) - l_e^{\text{target}}, 0\right)^2,$$
$$l_e^{\text{target}} := 0.5 * 2^{-2*I(e)}.$$

l_e^{target} is the target latency (in seconds) of the subscription $e \in E$, which in this case depends on the interest level. $l_e(r)$ is the path latency (in seconds) of subscription e in configuration r. Finally, $w_{\text{bw,S}}$ and $w_{\text{bw,U}}$ are set to 64, and w_{lat} is set to 1.

B. Implementation Notes and Discussion

To make an implementation of the presented algorithm efficient, it is necessary to maintain the required local knowledge of each node without too much overhead. For enumerating all possible rerouting options (Step 1 of the algorithm), a node needs to know its two-hop neighborhood. Generally, this would require transmitting neighbor lists, which causes significant overhead. In our case, however, a node only needs to know common neighbors of itself and a potential forwarder. This can be efficiently communicated via Bloom filters, which can be piggybacked on top of packets that are transmitted between neighbors anyway. For the local utility function, a node needs to know the spare bandwidth of its neighbors, which can be transmitted along with the neighbor filters. Path length and latency are metrics that cannot be computed based on local knowledge. The forward hops and latency can be piggybacked along the path. To inform all nodes along the path, a message has to be transmitted in reverse direction.

When nodes that serve as multipliers fail, the affected routes need to be adapted in a timely fashion. Therefore, nodes should have a good awareness of the liveliness of their forwarders. If the interest between the nodes is symmetric, the sender will receive updates of the forwarder, which can be used to detect inactivity. Otherwise, control data packets can be used.

V. Evaluation

For the evaluation, we use a round-based simulation. In each round, each node performs one optimization step of the algorithm. The round-based model abstracts from the frequency of iterations, which can be adjusted to the application needs. We used a geo-location host model adapted from GNP [11] and a link model with node bandwidths from 100 to 500 kbit/s, at a stream throughput of 512 bytes/s. We assume messages have an overhead from packet headers in the same order of magnitude as the payload. For small messages, this assumption is realistic: using UDP transport, IPv4 and UDP headers have a total size of $20 + 8 = 28$ bytes. With TCP, this grows to $20 + 20 = 40$ bytes, and IPv6 makes it even worse.

A. Virtual Reality Scenario

In the virtual reality scenario, a variable number of nodes is placed randomly in a fixed-size region. The interest function is a Gaussian of the euclidean virtual-world distance. The number of nodes is varied from 50 to 200, resulting in an average fan-out between 7 and 30. Since the algorithm works on each node's neighbor set, the total number of nodes is less significant than the fan-out, i.e, the interest set size. The clustering coefficient is around 0.64, independent of density.

The iteration process in a static scenario is illustrated in Fig. 4, showing the distribution of uplink utilizations after each iteration. The average bandwidth utilization is reduced from almost 100% to less than 75%. Initially, more than 35% of the nodes have a utilization of greater than 100%, i.e., are overloaded. After 17 iterations, the ratio is reduced to less than 3%. Overloaded nodes are relieved (top right) at the expense of sparsely utilized nodes (bottom left). Eventually, the majority has a utilization between 60 and 80%.

Fig. 5 shows the initial and optimized uplink utilizations as well as path length and latency stretch depending on the number of nodes. Both mean and 90th percentile of the initial link utilization grow linearly with the number of nodes, because the number of outgoing message streams equals the fan-out. With 50 nodes, link utilization is below 100%, hence, there is no need for further reduction. But the higher the density and therefore link utilization gets, the greater becomes the improvement in the optimized case. Especially the most utilized nodes, represented by the 90th percentiles, profit the most. In return, reduction of link utilization in the denser cases is paid with an increased path length and latency stretch.

To demonstrate the trade-off between latency and bandwidth utilization, Fig. 6 shows the results for 11 weight settings between "free" bandwidth (0.0 on the x-axis) and "free" latency (1.0). The values at 0.0 represent the purely direct delivery configuration. At the other end, when latency does not have any costs associated, the last bit of bandwidth saving is bought with significantly added latency. Between the extreme cases, the weights can be used to adjust the preference between link utilization and indirection latency.

B. Clustering Scenario

To examine the optimization potential depending on the clustering, we use a synthetic graph generator to build con-

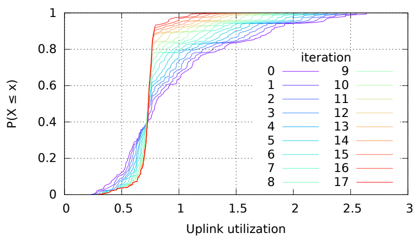


Fig. 4. CDF of uplink utilization for the iterations of a typical optimization process with 150 nodes.

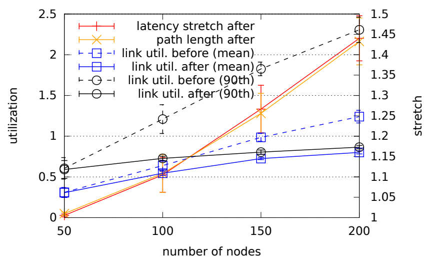


Fig. 5. Uplink utilization by the number of nodes.

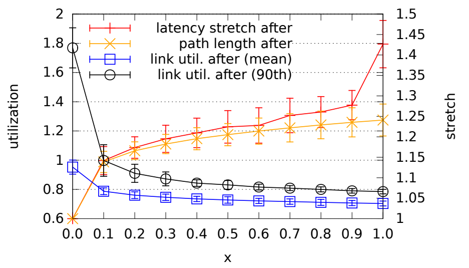


Fig. 6. Link utilization and path length depending on weight factors of utility function. $w_{\mathrm{bw,S}} = w_{\mathrm{bw,U}} = 64 * x$, $w_{\mathrm{lat}} = 1 - x$.

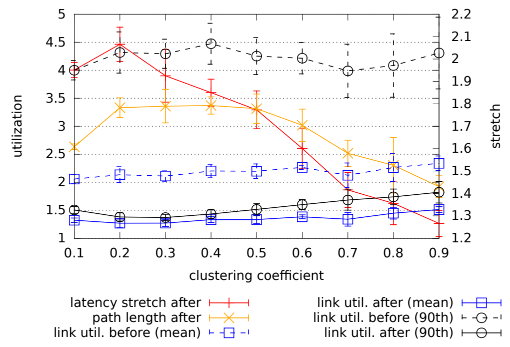


Fig. 7. Optimization potential depending on clustering coefficient.

nected subscription graphs. The variant of the random intersection graph algorithm by Deijfen and Kets [4] allows us to generate random graphs with a predetermined degree distribution and clustering coefficient. The algorithm's parameters are tuned to generate graphs with a gaussian distributed node degree with an average of 25 and clustering coefficients of $0.1, 0.2, ..., 0.9$. The nodes' link capacities are reduced so that the initial mean link utilization is above 2 and the 90th percentile is above 4.

The results (Fig. 7) show that a higher clustering coefficient increases the potential for optimization. Surprisingly, this is not expressed by a decreasing link utilization after the optimization. Instead, the necessary latency stretch is lower for higher clustering coefficients. The behavior of the algorithm is a result of the utility function used. The utility function can therefore also be tuned according to application needs in such case. Overall, the results show that a high clustering improves the performance of our algorithm.

VI. Conclusion

We have presented a dynamic many-to-many message dissemination approach that uses direct connections between overlay nodes for minimum latency and applies message forwarding and aggregation where this is the better option. Since the optimization algorithm runs locally on each node using knowledge only about the local neighborhood, it is applicable in scenarios with highly dynamic subscriptions. In the evaluation we made three main observations: (1) The optimization process effectively shifts the load to less utilized nodes, leading to a distinctively more uniform load distribution and significantly fewer overloaded nodes. (2) The utility function allows controlling the trade-off between link utilization (fairness) and induced latency. (3) A high clustering coefficient of subscriptions improves the performance of the algorithm.

This work has focused on the high-level algorithm definition and an evaluation on a high level of abstraction. The next steps will include a specification and implementation of the concrete protocol in a real application. A suitable candidate application is the multiplayer game Planet PI4 [8]. Furthermore, we want to adapt this scheme for event dissemination in mobile ad-hoc networks (MANET) and mixed P2P/MANET scenarios.

Acknowledgment

This work has been funded by the German Research Foundation (DFG) in CRC 1053 "MAKI" and GRK 1343 "Topology of Technology".

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, p. 205, Oct. 2002.
[2] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "OMNI: An efficient overlay multicast infrastructure for real-time applications," *Computer Networks*, vol. 50, no. 6, pp. 826–841, Apr. 2006.
[3] G. Cugola, A. Margara, and M. Migliavacca, "Context-aware publish-subscribe: Model, implementation, and evaluation," in *IEEE Symp. on Computers and Communications (ISSC'09)*, Jul. 2009, pp. 875–881.
[4] M. Deijfen and W. Kets, "Random intersection graphs with tunable degree distribution and clustering," *Probability in the Engineering and Informational Sciences*, vol. 23, no. 4, pp. 661–674, Oct. 2009.
[5] S.-Y. Hu, J. Chen, and T. Chen, "VON: a scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.
[6] J. Jannotti and D. Gifford, "Overcast: reliable multicasting with on overlay network," in *4th Symposium on Operating System Design & Implementation (OSDI'00)*. USENIX Association, 2000, pp. 197–212.
[7] K. R. Jayaram, P. Eugster, and C. Jayalath, "Parametric Content-Based Publish/Subscribe," *ACM Transactions on Computer Systems*, vol. 31, no. 2, pp. 1–52, May 2013.
[8] M. Lehn, C. Leng, R. Rehner, T. Triebel, and A. Buchmann, "An online gaming testbed for peer-to-peer architectures," in *ACM SIGCOMM Comp. Commun. Rev.*, vol. 41, no. 4. ACM, Aug. 2011, pp. 474–475.
[9] K. Mokhtarian and H.-A. Jacobsen, "Minimum-Delay Overlay Multicast," in *32nd IEEE International Conference on Computer Communications (INFOCOM'13)*, 2013.
[10] A. Nazir, S. Raza, and C. C.-N. Chuah, "Unveiling facebook: a measurement study of social network based applications," in *8th ACM SIGCOMM conference on Internet measurement conference (IMC'08)*. ACM, Oct. 2008, p. 43.
[11] T. Ng and H. Zhang, "Towards global network positioning," in *1st ACM SIGCOMM Workshop on Internet Measurement (IMW'01)*. ACM, 2001, pp. 25–29.
[12] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. P. Buchmann, "pSense - maintaining a dynamic localized peer-to-peer structure for position based multicast in games," in *Eighth International Conference on Peer-to-Peer Computing*, 2008, pp. 247–256.
[13] H. Zheng, E. Lua, M. Pias, and T. Griffin, "Internet routing policies and round-trip-times," in *Passive and Active Network Measurement*, ser. LNCS. Springer Berlin Heidelberg, 2005, vol. 3431, pp. 236–250.