Enabling P2P Gaming with Network Coding

Balázs Lajtha, Gergely Biczók and Róbert Szabó

High Speed Networks Laboratory
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics
{lajtha.balazs,biczok,szabo}@tmit.bme.hu

Abstract. The popularity of online multiplayer games is ever-growing. Traditionally, networked games have relied on the client-server model for information sharing among players, putting a tremendous burden on the server and creates a single point of failure. Recently, there have been efforts to employ the peer-to-peer paradigm for gaming purposes, however, latency-sensitive action games still pose a formidable challenge. The main contribution of this paper is the design of a novel peer-to-peer gaming framework based on random linear network coding. We briefly evaluate the performance of the proposed framework; our initial results suggest a significant reduction in network latency that comes at the expense of a small data traffic overhead. Although further evaluation is clearly needed, we believe that our approach can be the foundation of a truly peer-to-peer communication architecture for networked games.

1 Introduction

Playing computer games is the favorite pastime of hundreds of millions of people. The population of players is very diverse, men and women, children and grandparents, construction workers and university professors all have a tendency to use their computer for entertainment purposes. In the past decade, online games gained popularity; specifically, multiplayer online games are the most successful, such as Call of Duty (a first-person shooter) or World of Warcraft (a role-playing game) [1]. Designing an architecture which meets the strict requirements of online gaming, offers a good quality of experience for the users, and proves to be efficient under the unpredictable conditions of the current Internet is a challenging task.

Traditionally, networked games have relied on the client-server architecture: players' home computers act as clients, with (one or) multiple servers situated in the higher tiers of the network. As an alternative, some games allow for hosting a server at sufficiently equipped home computers, making multiplayer gaming possible in a local network (LAN) setting. Either way, massively multiplayer games can put a tremendous burden on a single server, both from the traffic and computational load viewpoint. This can result in inefficient network resource utilization and also creates a single point of failure. The notion of a peer-to-peer gaming architecture emerges naturally, however, only a small number of role-playing games use this concept: action games are more sensitive to network delays, as lags may render the gameplay experience unsatisfactory.

On the other hand, the recently proposed network coding principle [2] could open new avenues for packet switched networks. While network coding has been proven to

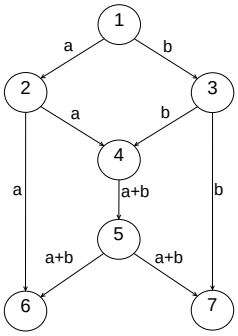


Fig. 1. Network coding in the butterfly topology

be effective in a wireless environment [3] and also in core networks, it's usefulness in traditional P2P applications like file-sharing and video streaming has been widely debated [4] [5] [6]. From a networking point of view, multiplayer gaming shares similarities with both content distribution applications, and as such, it could potentially benefit from network coding.

In this paper we present a practical network coding approach to multiplayer gaming over a peer-to-peer overlay. We replace the standard unicast forwarding mechanism with a random network coding mechanism. Since home computers, especially ones used for gaming, are powerful, coding and decoding packets on-the-fly is feasible. Our initial performance evaluation shows encouraging results: average latency of player state updates is reduced in a wide range of scenarios. Specifically, network coding outperforms unicast by more than 30% when participating peers are heterogeneous in terms of access bandwidth. The cost for this improvement is a potential overhead regarding generated data traffic; however, this extra traffic is proportionally marginal in a number of scenarios, and its absolute volume is always low. We argue that applying random network coding in this context is an extremely promising research direction, as it has the potential to be the foundation of a truly peer-to-peer architecture for networked games.

The remainder of this paper is structured as follows. Section 2 gives an overview on network coding techniques and online games. Our main idea, the application of network coding for peer-to-peer online gaming is presented in Section 3. We evaluate the performance of the proposed method in Section 4. Finally, Section 5 identifies open issues and concludes the paper.

2 Related Work

Network coding. Network coding was proposed to be used for improving packet-switched network throughput by Ahlswede et al. in [2]. In a classical routed approach packets are transmitted without change from data sources to destinations. Meanwhile, in a system implementing network coding, nodes are allowed to modify packets, encode and re-encode packets on the path, as long as the receivers are able to extract the content of the original packets. Network coding usually doesn't use compression, but achieves a larger throughput in multicast or broadcast scenarios with the optimal usage of network topology.

A typical example for the exploitation of network topology is the "butterfly" (see Fig. 2) . In this network a source (1) sends two flows at the rate of one packet per time slot to both node 6 and 7. Each edge has a throughput of one packet per time slot. With no network coding, four packets have to share the minimum cut of three edges ($2 \rightarrow 6$, $4 \rightarrow 5$ and $3 \rightarrow 7$). The problem can only be solved by increasing the capacity of link $4 \rightarrow 5$. On the other hand, application of network coding (e.g., using bitwise $a + b$) at node 4 and 5 allows bottleneck link $4 \rightarrow 5$ to carry the combined information of the two packets. With information decoded at nodes 6 and 7 (e.g., using bitwise $(a + b) - a$ and vice versa), network coding achieves a throughput of 4 packets per time slot.

In [2] it was proven that with network coding the information rate from a source to a set of nodes can reach the minimum of the individual max-flow bounds. In [7] a constructive proof was given that the theoretical maximum information rate can be achieved by linear network coding. In linear network codes flows are broken into vectors over a field. Each node in the network can linearly combine vectors to create a new packet. Giving a method for constructing optimal network codes using only linear transformations opened the way to the application of network coding when topology is available and changes are seldom.

In dynamically changing networks, advertising network changes and rebuilding the coding infrastructure creates a large overhead. Instead of a static coding scheme, random linear network coding (RLNC) has been proposed in [8]. Random codes differ from traditional linear codes in that linear combinations are generated by each node on the fly. Transmitted packets contain the combination of packets and the coefficients associated with each source vector present in the packet. Authors showed that the performance of RLNC exponentially approaches that of linear network coding. By doing so, RLNC can provide a solution as good as any network coding scheme making use of the network topology. RLNC makes the application of network coding possible in ad hoc networks and networks with a high churn rate, with no central authority or strong distributed computing required to design and maintain network coding schemes. Benefits of applying RLNC in P2P applications were studied in [5] for file transfer applications. An implementation of a P2P media system using RLNC was proposed in [6].

Chiu et al. [4] showed that applying network coding alone at peers in an overlay network will not result in improved throughput, which questions the effectiveness of network coding in a file-sharing scenario. However, our focus is on peer-to-peer gaming, where reducing network latency is a first order concern, while bandwidth consumption is not critical.

Networked games. Games can be seen as discrete interactive simulation: the game model is evaluated and changed periodically in a game loop. Users interact with the game model through avatars. The avatar control mechanism is not event based, user input is scanned at a specific point of the ever-running game loop and processed based on the state of the avatar. In present networked multiplayer games each player maintains a local copy of the game model. Input generated by all players has to be available to each player in order to keep these copies synchronized. This is achieved through periodical player state update messages.

The client-server architecture has the benefit of a central authority that maintains the game model and broadcasts both player state and game object state updates. Servers can implement security features, e.g., filtering out malicious players based on their in-

game or network activity. A central server can also optimize bandwidth consumption and eliminate some cheats by sending player state updates only from those players and objects that may be visible to a certain player. Servers can be operated by the game's publisher or a company dedicated to host online games, but this is not always necessary: in a number of games any player can act as a server. The disadvantages of both approaches are obvious: dedicating resources to host multiplayer games is expensive, as servers must be localized to maintain low response times, and game load is concentrated to the evening hours [9]. On the other hand, making one of the players take up the server's role will consume the player's local resources and reduce her and (due to limited bandwidth) other players' gaming experience. Moreover, this way a single point of failure exists, hence a connectivity problem at the server can prevent all the participants from playing.

As a response to the above-defined problems, some massively multiplayer online games (MMOG, mostly role-playing and adventure) are migrating to P2P networks. MMOGs are played by thousands of players contributing to the same huge and detailed virtual environment. This virtual world is permanent, with players joining and leaving. In these games player state updates can be less frequent and the emphasis is on dispatching game object state updates to all players. To make such games easier to scale, P2P overlay based games were proposed in [10], focusing on the partitioning of the game-space to optimize communication by separating users into groups. The proposed techniques are not fast enough for first person view games such as action games, shooters (FPS) and simulators. Authors of [11] proposed Colyseus, a framework which utilizes efficient object location and speculative prefetching besides game-state partitioning to deal with latency requirements. Another relevant piece of work is the Donnybrook system [12]: it uses a sophisticated method to estimate which objects and other players are important to a given player, thereby reducing the frequency of state updates. Additionally, updates are disseminated via overlay multicast. While the achievements of these works are valuable, real-world games requiring high responsiveness are still server-based.

In this paper we concentrate on the efficient communication of frequent player state updates (synchronization). We believe that reducing network traffic and latency through limiting recipient lists, clustering based on game state and speculative prefetching are important ingredients of a peer-to-peer gaming framework, however, an efficient dissemination scheme for frequent in-game updates is essential for a practical system. The nature of such kind of traffic is multicast, therefore we expect that overlay-based network coding could significantly help in such a scenario [13].

3 Overlay Network Coding for P2P Gaming

In a multiplayer gaming scenario, latency of player state updates comes from two factors, the one-way trip time of the packet and available bandwidth. Both are dependent of factors like home networking equipment, cross-traffic generated by the end user, overall network load, geographical distance and traffic shaping equipment used by network providers.

If a server-based solution is deployed, each node uploads only a single packet to the server, and downloads the packets of every other peer from the server. Bandwidth

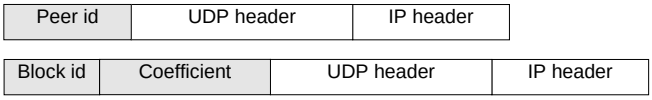


Fig. 2. Packet headers: unicast vs. RLNC

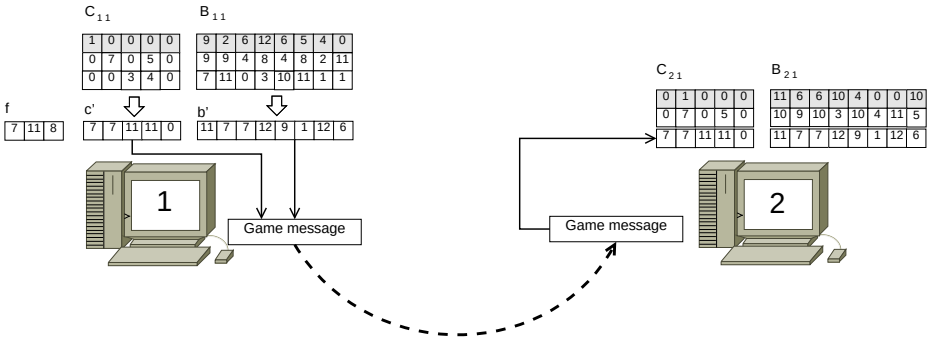


Fig. 3. Random network coding in action (modulo class: 13)

limitations are usually present only at the server, especially when the server connects to the Internet through an asymmetric home access link such as DSL or cable. In a server-based scenario, latency due to trip time depends only on Internet factors such as BGP policies, load balancing mechanisms, congestion and link availability.

Peer-to-peer gaming architectures use an overlay network to disseminate player state updates among peers. There are two basic characteristics of the overlay which determine overall latency: its topology and the forwarding mechanism used. From the topology standpoint, overlays can be fully or partially connected. We restrict our investigations to fully connected overlays for the sake of simplicity.

In a full mesh overlay the simplest packet forwarding strategy is unicast, when each peer sends its update packet directly to every other peer. We propose an alternative forwarding mechanism, network coding. The main difference brought by network coding is the possibility of combining available packets before forwarding them. While optimal bandwidth consumption and computational overhead can be achieved with fixed linear coding schemes, this approach is not suitable for network gaming over an unreliable medium. With a fixed coding scheme packets are expected to be present at given nodes at a given time. In peer-to-peer gaming packet generation is indeed synchronized, but packet transport times vary in a wide range. Hence buffers have to be introduced to ensure that all packets required for decoding are present as the coding scheme requires it. This is not desirable when the goal is to reduce overall latency. Random linear network coding can work with asynchronous packet flows, as the coding scheme doesn't specify which packets have to be combined, packets contributing to the outgoing packet are chosen from the available packets randomly. The price of this freedom is a larger packet header, containing the coefficients of the original packets that are combined to get the current one (see Fig. 2). Moreover, decoding is more computationally intensive, as a new linear equation system has to be solved for every block to be decoded.

The basics of random network coding as implemented in our framework are shown in Fig. 3. Equations and variables correspond to a single game step, hence no indexes will be introduced for the time. At each game step, an update message is generated by each peer. Each message corresponding to a single network coding packet. Assume that a virtual array A is composed of these n messages. This array is split into k blocks of rows, denoted by A_i. Having smaller number of rows in a block reduces the computation needs of the decoding process. Packets received from neighbors contain the block number i of the message it was generated from, the coefficient c used and the encoded message b. Received messages for each block are stored at peer j along with the local message in the working arrays B_{ji} (local message is stored in B_{ji} if message of peer j is part of block i). Received c coefficients for these messages are stored in the arrays C_{ji} with the local message having a coefficient of 1. At the beginning of the game step, arrays B_{ji} and C_{ji} are empty except the local message of each peer stored in one of the B_{ji} arrays and the associated coefficient in C_{ji}. Random linear network coding guaranties that the equation

$$B_{ji} = C_{ji} \times A_i \tag{1}$$

(obviously true at the beginning) will always hold. This is ensured through the coding mechanism.

A given outgoing message of peer j is created from a chosen block i. The message to be sent is generated with the help of a random vector f of the length of the number of rows in B_{ji}. The message consisting of c' and b' is constructed as follows:

$$c' = f \times C_{ji} \quad \text{and} \quad b' = f \times B_{ji}. \tag{2}$$

Note, that

$$b' = f \times B_{ji} = f \times C_{ji} \times A_i = c' \times A_i. \tag{3}$$

When peer l receives the message, c' and b' will be appended to the arrays B_{l_i} and C_{l_i}:

$$(B_{l_i})' = B_{l_i}|b' \quad \text{and} \quad (C_{l_i})' = C_{l_i}|c'. \tag{4}$$

Hence if

$$B_{l_i} = C_{l_i} \times A_i \tag{5}$$

then

$$(B_{l_i})' = (C_{l_i})' \times A_i. \tag{6}$$

In order to retrieve the original messages (rows of A_i), the above equation has to be solved for all k blocks at each peer j. Inverting C_{ji} requires at least $\lceil \frac{n}{k} \rceil$ linearly independent messages received from each block. It has been shown in [14] that the probability of selecting linearly dependent combinations becomes negligible even for a small code field size. This result was achieved in a streaming application where network coding was applied at the source where an entire block of data was available.

In peer-to-peer gaming, data are available distributed between all peers, and network coding is only applied during message forwarding. In the beginning of a given game round, each peer has access only to a limited number of messages and peers in small proximity will usually have knowledge of the same subset of messages. This in turn may lead to sending messages that are not independent from those the destina-

tion peer has already received. These irrelevant messages may constitute a considerable traffic load, but may not affect overall delivery latency. Our first network coding method, *RLNC*, is the basic implementation of a linear network coding system. RLNC operates on the full mesh, every peer sends a message periodically to each neighbor. The packet is a random linear combination of already received messages. The second method, *RLNC+*, implements an additional mechanism for reducing unnecessary transmissions: each peer maintains a message array which contains messages sent to and received from its neighbors. Subsequent outgoing messages are sent only to neighbors that could be interested in the content of the packet, this way, unnecessary packet sending could be spared.

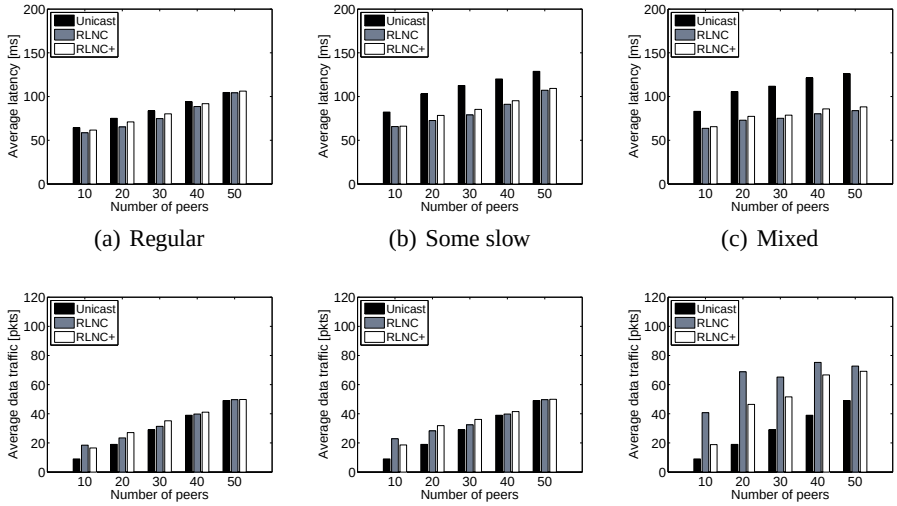
4 Performance Evaluation

To provide an initial performance assessment we developed a JAVA-based simulation tool. The environment consists of the configuration generator, responsible for creating, saving and loading network topology, the simulator and the log processing pipeline. Our network core simulator provides delayed packet delivery on point to point links. For the proof-of-concept evaluation, peers were connected directly to this core, and no computation overhead (decoding) was taken into account.

Network model. Our simulator was created to handle network traffic generated by player state updates, and didn't considered game object changes. We also presumed that addresses of the players are available, and players do not leave or join during a match as this is the case in most match-based multiplayer games. We placed our players in the Internet, as in LAN games connections are more reliable, bandwidth is almost always sufficient and the impact of network latency on gameplay can be neglected.

Simulations were run over networks consisting of 10, 20, 30, 40 and 50 peers. The traffic generated by the user interactions was uniform in time as player updates are not event but state based. We assumed UDP as the transport protocol of choice, since real-time traffic does not tolerate the added latency of the feedback loop in TCP. A typical player state update messages size ranges from 10 to 100 bytes, and is sent to each other player 10-60 times per second [15]. In our experiments we used a 10 Hz update frequency and player state update message of 50 bytes, totaling 78 bytes with IP and UDP headers. When using random linear network coding, the message size increases with the coefficient. This depends on the block size, and ranges from 5 bytes for 10 peers to 25 bytes for 50 peers, as we used a single block for all peers. For coding we used a fixed field size of 13 (modulo class).

We measured a round-trip time of 15-87 ms towards different European servers [16]. Based on these results we used one-way peer-to-peer latency values of 5 to 40 ms uniformly distributed with a jitter of 5 ms. Since typical home broadband connections are asymmetric, we assumed that upload bandwidth would determine the effectiveness of different forwarding mechanisms (downlink is assumed to be unlimited). Based on bandwidth characteristics three peer classes were used: slow peers had an upload bandwidth just enough to send out a single packet to each other peer (62.4 kbit/s to 312 kbit/s depending on the number of participating peers), the regular peer had 624 kbit/s and fast peers were granted a 8 Mbit/s uplink. We used four different scenarios in terms of participating peer distribution: regular (regular peers only), some slow (90%regular



(a) Regular (b) Some slow (c) Mixed

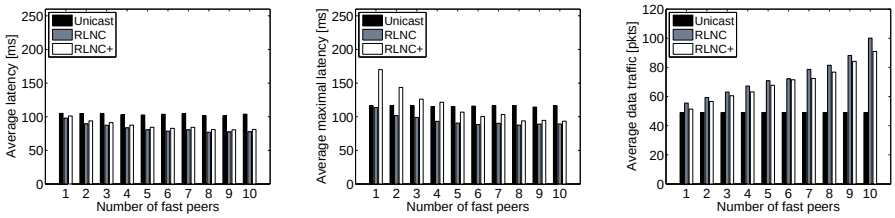
(d) Regular (e) Some slow (f) Mixed

Fig. 4. Average latency (upper) and data traffic generated (lower) for different peer distributions

and 10% slow), some fast (90% regular and 10% fast), and mixed (80% regular, 10% slow, 10%fast).

Experimental results. When processing the results, our main concern was the average latency. We defined latency as the time elapsed from the beginning of the game round until the respective peer becomes aware of all other players' states. In the unicast scenario this happens when a packet from every peer is received. In the network coding scenario received packets were scanned for a solution to the linear equation created from the received coefficients after the reception of each packet. If the equation becomes fully determined all player update packets can be decoded. Other key performance indicators were the maximum latency in a single round and the number of packets received before decoding could be achieved. Average and maximum latency measures quality of experience, while the generated data traffic measures network load. Note, that all latency, maximum latency and traffic figures are averaged over 100 game rounds.

Average latency values for different participating peer populations can be seen in Fig. 4(a)-4(c). In a network with regular peers, network coding (RLNC) performs slightly better then traditional unicast mechanism, even with its larger packet size which reduces the packet sending rate. This is due to it's flooding-like behavior: packets originating from one source will find the shortest paths, while in the unicast solution the direct link between two peers may have a larger delay. When some slower peers are also present in the overlay, network coding shows greater improvement over the traditional forwarding method. On the top of that, the heterogeneity of the mixed scenario (slow, regular and fast peers are also present in the system) induces the highest gain for RLNC among all scenarios. Improvement in this setting can be more than 30%.



(a) Average latency (b) Average maximum latency (c) Average data traffic

Fig. 5. Impact of fast peers (50 peers, some fast)

Note, that network coding with redundancy protection (RLNC+) performs comparably to RLNC in terms of average latency. However, its added benefit becomes visible in Fig. 4(e)-4(f), where generated data traffic is shown. In the first two scenarios traffic generated by network coding mechanisms is only slightly more than that of unicast. On the other hand, RLNC produces much more traffic in the mixed peer setting, with RLNC+ reducing this overhead significantly.

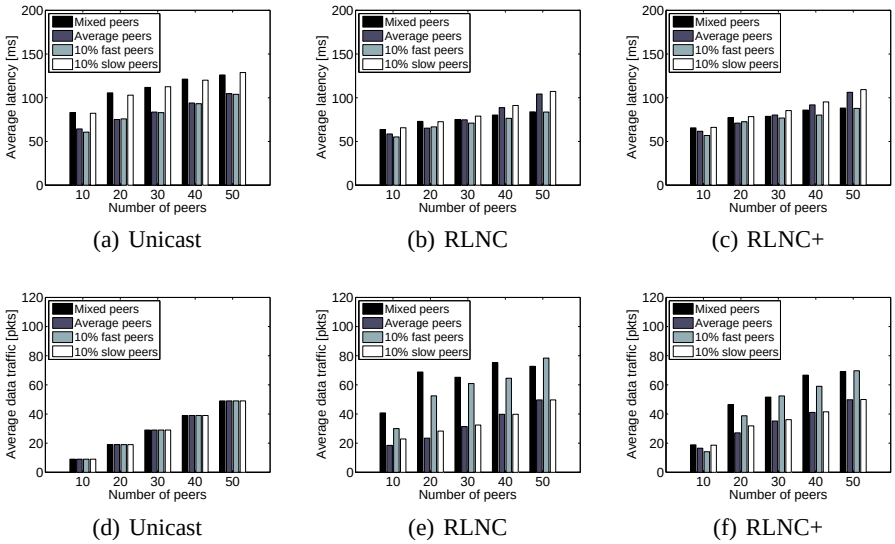
The next series of experiments studied the impact of fast peers on the system. We used a network of 50 peers, with 1 to 10 fast peers beside the regular ones. Fig. 5 shows how the latency benefit of network coding increases with the ratio of fast peers. The amount of generated data traffic gives an explanation for this behavior: the more peers with high bandwidth connections are present, the higher redundancy can be achieved in the system. This redundancy provides improved latency figures. Note, that RLNC+ produces considerably high maximum latency values, when only a few fast peers are present. This shows that restricting the scope of recipients is more beneficial when peer heterogeneity is higher.

Latency and data traffic results grouped by forwarding mechanisms are shown in Fig. 6. It can be observed that network coding improves the average latency in every scenario. It is important to emphasize that 150 ms is barely tolerable [17], while 100 ms is acceptable for online shooter and action games; our results suggest that this limit can be met even in case of a large number of participants. Moreover, the overhead of network coding compared to unicast is almost non-existent if only regular or some slow peers are playing; in more heterogeneous scenarios, there is some reasonable overhead (with RLNC+ generating slightly less traffic). Note, that this overhead might be recognizable percentage-wise, but the absolute volume of extra data traffic remains very low (20 extra packets per peer in a 50 peer scenario).

5 Conclusion and Future Work

The recently proposed technique of network coding has been shown to boost network capacity compared to the traditional store-and-forward mechanism in a variety of scenarios. Here, we have presented a very different use-case for the same method: reducing network latency. Our promising results in this area shows a clear need for further understanding of the possible benefits and side-effects of network coding.

In this work, we have introduced a practical framework for peer-to-peer networked gaming based on random linear network coding. We have shown that our proposed



(a) Unicast (b) RLNC (c) RLNC+

(d) Unicast (e) RLNC (f) RLNC+

Fig. 6. Average latency (upper) and data traffic generated (lower) for different forwarding mechanisms

method outperforms traditional unicast in terms of average network latency in a wide range of scenarios. Furthermore, the absolute traffic overhead has been shown to be low for all settings analyzed.

Our initial performance evaluation indicates that the proposed method is worthy of further research. One important area is the specific codes used. Also, more simulations are needed to investigate the impact of more complex topologies, packet loss and computational overhead. Moreover, a prototype implementation and testbed measurements with a real game are essential to fully understand the behavior of our system.

References

1. D. C. Jones. “The monster that is – World of Warcraft”, `http://www.bme.eu.com/news/world-of-warcraft`, accessed April 15th, 2010.
2. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network Information Flow”, IEEE Transactions on Information Theory, 2000.
3. S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, “XORs in the air: practical wireless network coding”, In: Proc. of ACM SIGCOMM 2006.
4. D. M. Chiu, R. W. Yeung, J. Huang, and B. Fant, “Can Network Coding Help in P2P Networks?”, In: Proc. of WiOpt 2006.
5. Y. Wu, Y. C. Hut, J. Li, and P. A. Chou, “The Delay Region for P2P File Transfer”, In: Proc. of ISIT 2009.
6. C. Wu, B. Li, and Z. Li, “Dynamic Bandwidth Auctions in Multioverlay P2P Streaming with Network Coding”, IEEE Transactions on Parallel and Distributed Systems, 2009.
7. S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear Network Coding”, IEEE Transactions on Information Theory, 2003.
8. T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, “Random Linear Network Coding Approach to Multicast”, IEEE Transactions on Information Theory, 2006.

9. Online Gamers Research. http://www.video-games-survey.com/online_gamers.htm, accessed April 15th, 2010.
10. B. Knutsson, H. Lu, W. Xu, and B. Hopkins, “Peer-to-Peer Support for Massively Multiplayer Games”, In: Proc. of INFOCOM 2004.
11. A. Bharambe, J. Pang, and S. Seshan. “Colyseus: A Distributed Architecture for Online Multiplayer Games”, In: Proc. of NSDI 2006.
12. A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. “Donnybrook: Enabling large-scale, high-speed, peer-to-peer games”, In: Proc. of ACM SIGCOMM 2008.
13. Y. Zhu, B. Li, and J. Guo. “Multicast with Network Coding in Application-Layer Overlay Networks”, IEEE JSAC, 2004.
14. Y. Wu, P. A. Chou, and K. Jain. “A Comparison of Network Coding and Tree Packing”, In: Proc. of IEEE ISIT, 2004.
15. W. Feng, F. Chang, W. Feng, and J. Walpole. “Provisioning Online Games: A Traffic Analysis of a Busy Counter-Strike Server”, In: Proc. of IMW 2002.
16. Speedtest. http://www.speedtest.net, accessed April 15th, 2010.
17. G. Armitage. “An experimental estimation of latency sensitivity in multiplayer Quake 3”, In: Proc. of ICON 2003.