Bandwidth Efficiency Improvement of Online Games by the use of Tunneling, Compressing and Multiplexing Techniques

Jose Saldana, Jenifer Murillo, Julián Fernández-Navajas, José Ruiz-Mas, José I. Aznar, Eduardo Viruete Navarro
Communication Technologies Group (GTC) – Aragon Inst. of Engineering Research (I3A)
Dpt. IEC. Ada Byron Building. CPS Univ. Zaragoza
50018 Zaragoza, Spain
{jsaldana, jenifer.murillo, navajas, jruiz, jiaznar, eviruete}@unizar.es

Abstract—**The enterprises that develop online games need hardware and bandwidth resources in order to give a good service to the users. These games produce a high rate of small UDP packets from the client to the server, so the overhead is significant. The actions of the players have to be propagated to the server and to the rest of the players in a very short time, so network delays are very critical. This work presents a method that saves bandwidth, by the addition of a local agent which queues packets, compresses headers and uses a tunnel to send a number of packets into a multiplexed one. The behaviour of the system has been studied for IPv4 and IPv6, showing that significant bandwidth savings can be achieved. For certain titles, up to 38% of the bandwidth can be saved for IPv4. This percentage grows up to 54% for IPv6, as this protocol has a bigger overhead. The cost of these bandwidth savings is the addition of a new delay, which has an upper bound that can be modified. So there is a tradeoff: the bigger the added delays, the bigger the bandwidth savings. On the other hand, the tests show that if the number of players is big enough, the added delays can be acceptable in terms of user's experience.**

Keywords-gaming; delay; multiplexing; compressing; measurement; network games; Quality of Experience; First Person Shooter

I. INTRODUCTION

Online gaming via Internet is a service that grows quickly worldwide. There are some titles which have millions of users, so the enterprises that develop the games have to face a difficult problem whenever a new title is released: they need hardware and bandwidth resources in order to avoid the saturation of their infrastructure. As the success of a new title is not very predictable, they may have to over-provision these resources, because the users that buy the game have to receive a good service. In [1] a study of the behavior of the *gamers* was presented, and the authors concluded that they are very difficult to satisfy: if they have connection problems, they usually leave and never return, and they do not tend to be loyal to a server.

Two of the most popular genres of online games are MMORPGs (Massive Multiplayer Online Role Playing Games) and FPS (First Person Shooters). Reference [2] studied the traffic of MMORPGs, concluding that they have some characteristics like periodicity, locality, and self-similarity. Another conclusion is that they have less bandwidth and real-time requirements than FPS.

In FPS the actions of the players have to be propagated to the server and to the rest of the players in a very short time, so network delays are very critical. These games produce a high rate of small UDP packets (typically some tens of bytes) from the client to the server, so the overhead caused by IP/UDP headers is significant. Server to client packets are typically bigger.

There are some scenarios where many players of the same game share the path from the access network to the game server: e.g. Internet Cafés, which are very popular in many countries, frequently have computers able to run online games, and gamers usually go there to play in groups (Fig. 1a and 1b). The traffic between servers of the same game is another scenario where the packets of many players share the same path (Fig. 1c).

The traffic of these groups of players could be compressed and multiplexed in order to save bandwidth, taking into account that the access network is usually the bottleneck. And, in case of DSL, the uplink has normally less bandwidth than the downlink. By the addition of a local agent which queues packets, compresses headers and sends multiple packets into a bigger one, some bandwidth saving could be achieved, at the cost of adding new delays, mainly caused by the retention time at the queue.

In order to avoid workload to the central game server, some proposals [3], [4] included network elements (proxies) next to the access network. In contrast, the local agent proposed in the current work could be distributed with the game application, in the same way as local servers are distributed with certain titles. The tunnel from the clients to the server could either be created by a dedicated machine (Fig. 1a), or by the computer of a player (Fig. 1b), thus avoiding the cost of a new machine.

At the other side, the server would have to implement the demultiplexer and decompressor, which would imply some processing capacity, and space to store the *context* of each flow, i.e. the information necessary to rebuild the compressed

This work has been partially financed by CPUFLIPI Project (MICINN TIN2010-17298), MBACToIP Project, of Aragon I+D Agency and Ibercaja Obra Social, and NDCIPI-QQoE Project of Catedra Telefonica, Univ. of Zaragoza.

headers, (some tens of bytes, as we will see [5]). This will not imply a scalability problem, as the server already stores the state of the game for each player. On the other hand, the savings in terms of bandwidth and packets per second may also be beneficial for the server.

If the number of players is big enough, it can be expected that, while adding small delays, a big number of packets could be multiplexed into a bigger one. Bandwidth saving not only affects the gaming traffic, but it can also be beneficial for the background traffic which shares the access with it. Furthermore, multiplexing has another advantage: the number of packets per second the router has to manage will be reduced.

There exist other applications and scenarios where many real-time flows share the same path, e.g. VoIP trunking; and the use of multiplexing and compressing techniques has been proposed and even standardized [6] for them. Many online games present similar traffic patterns, generating a high rate of tiny packets, and thus presenting a big overhead. The novelty of the present work is the use of similar multiplexing techniques for these games' traffic, in which significant gains can also be obtained, without losing quality.

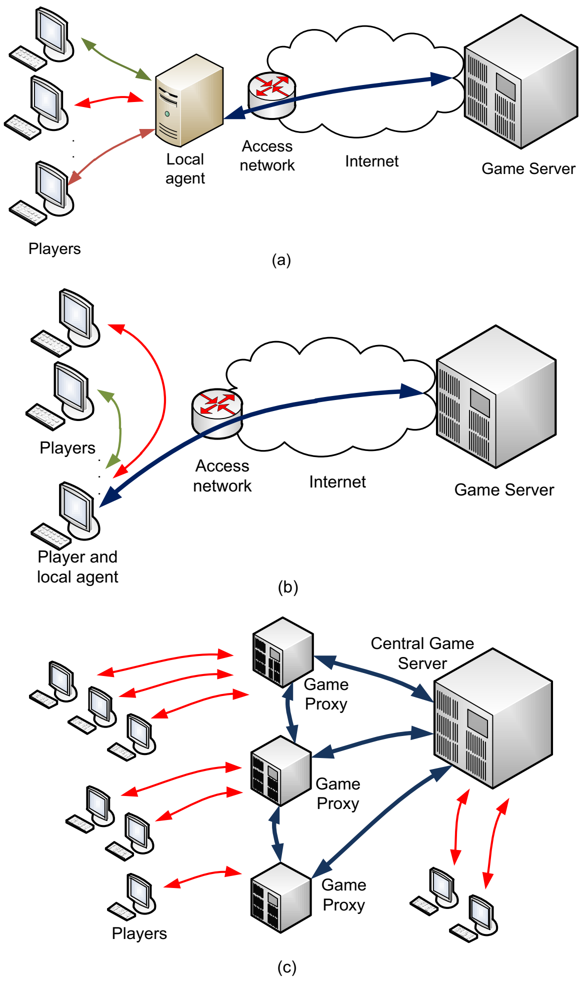


Figure 1. Scenarios where many players share the same path: a) players sharing an access network, using a local agent. b) players using the computer of another one in order to create the tunnel c) Traffic between proxy servers of the same game. Thick lines represent the traffic of a number of players.

We will mainly study the scenarios presented in Fig. 1a and 1b, but most of the conclusions can also be applied to the third one (Fig. 1c). Only client to server traffic is studied, as bigger savings can be obtained and in many scenarios (e.g. DSL) it is transmitted via the most restrictive link. We have to measure the impairments on the parameters that determine the quality experienced by users, in order to properly tune the parameters that define the tradeoff between bandwidth saving and quality.

Although this technique could be applied to other game genres, we will use FPS traffic in this work, as these games have very stringent real-time requirements. The subjective quality mainly depends on delay and packet loss [7]. The System Response Time (SRT: the time the system needs to detect a user event, to process it and to send the updated game state to the local output device), has to be maintained under a certain value.

The rest of the work is organized as follows: Next section presents the related works. Section III explains the proposed tunneling, compressing and multiplexing method. Section IV details the results. The paper ends with the conclusions.

II. Related Works

A. *Online Gaming Traffic*

There is a significant amount of literature regarding to the traffic of online games. We will only consider active traffic, i.e. the one generated once the game has started. This traffic presents two different behaviors: first, the client application is in charge of communicating the actions of the players to the server, using small packets with a small period. Second, the server calculates the new state of the game and broadcasts it to all the players, using bigger packets, whose size depends on the number of players. Ref. [8] presented a method to extrapolate server to client traffic, obtained from empirical measurements. They obtained size distributions for an N-player game, from measured traffic of 2 and 3 players. In that work it is also said that the client to server size distribution is independent of the number of players.

In [9] a 500 million packet trace of Counter-Strike was analyzed, and a conclusion was that the design of the game targets the saturation of the bottleneck, which is the last-mile link. Also in [10] the characteristics of many online games were analyzed, in terms of packet size and inter-packet time. In [11] a survey of different traffic models for 17 popular games can be found. These studies show that these games generate a high rate of small packets. This behaviour produces a big overhead, so bandwidth savings can be achieved by means of header compressing and multiplexing.

In [9] it is also said that the main bottleneck is frequently the number of packets per second that a router can manage, and not the bandwidth of the access line. Routers are usually designed for big packets, and can experience problems when managing bursts of small packets.

B. *Infrastructure for Supporting Online Games*

The problem of the infrastructure that supports the games has also been studied in some works. From the point of view of the user, Ref. [12] presented an algorithm to allow the client to adaptively select the best server for a certain online game. This could allow a group of users to play in the same server, and use multiplexing techniques.

From the point of view of the server, there are two architectures to support this service: centralized and distributed. In the first one, there is a server that maintains the state of the game and distributes it to the players. The problem is that the server represents a bottleneck. In distributed architectures [13] there is no need for a central server, as the players exchange the information. But this architecture is generally not used in commercial games.

The problem of the scalability of the required infrastructure for these games was also studied by Mauve et al [3], and they proposed the use of proxies in order to provide congestion control, robustness, reduce delays and avoid cheating. Some proxies could be distributed next to the players, avoiding some workload to the central server. Ref. [4] also proposed the use of *booster-boxes*, which would be next to the router and could know the state of the network, being able to give network support to the applications. As said in the introduction, the solution proposed in the current work could even run in the machine of a player.

C. *Compressing Algorithms*

Some IETF standards that compress headers were developed many years ago: first, VJHC [14] presented a method to compress IP/TCP headers. Some years later, IPHC [5] was also able to compress UDP and IPv6 headers. At the same moment, CRTP was presented in order to compress IP/UDP/RTP headers, and some years later it was enhanced and named ECRTP. But these two protocols are not suitable for compressing gaming traffic, as it is not RTP. There was an interesting proposal [15] of using a protocol similar to RTP for online gaming. An advantage of this proposal is the possibility of reusing generic services, avoiding the need of implementing them for every game. But nowadays commercial games mainly use IP/UDP packets.

These algorithms compress headers in a hop-by-hop way, using the high redundancy of IP, TCP and UDP header fields in order to avoid sending some of them. A *context* is defined, which is first transmitted from the sender to de destination with the first headers. The different header fields are classified into *non-change, random, delta* and *inferred.* The first ones are only sent in full headers. *Random* headers are sent without compression, while *delta* ones are codified using less bytes than the original size of the field. Finally, *inferred* headers can be obtained from the fields of other layers, e.g. the length of the packet can be obtained from the level 2 corresponding field.

ROHCv2 [16] is a more recent standard, which is also able to compress IP/UDP/RTP headers and IP/UDP ones. It reduces the impact of context desynchronization, by providing a feedback mechanism from the decompressor to the compressor. It uses three different compression levels, which correspond to operation modes: *Initialization and Refresh, First Order* and *Second Order*. In the last level, the header can be compressed to just one byte [17]. The use of these advanced techniques makes the implementation more difficult [18], and may add bigger processing delays.

III. Compressing and Multiplexing Method

In this section we will explain the proposed compressing and multiplexing method. A study of the bandwidth efficiency which can be achieved is also presented.

A. *Tunneling, Compressing and Multiplexing Algorithm*

In RFC 4170 [6], the IETF approved Tunneled Compressed RTP (TCRTP) in order to compress and multiplex RTP flows. First, ECRTP compressing is applied, and many packets are included into the same one using PPPMux. Finally, a L2TP tunnel is used in order to send the whole multiplexed packet end to end.

We have used a similar scheme, but in our case the traffic is not RTP, so we can only compress IP/UDP headers using IPHC or ROHCv2. We will call this method *Tunnel-Compress-Multiplex* (from now TCM). Fig. 2 shows the protocol stack and the structure of a TCM packet. It can be divided into the next parts:

- **Common Header (CH):** Corresponds to the IP, L2TP and the PPP headers.
- **PPPMux header (MH):** It is included at the beginning of each compressed packet.
- **Reduced header (RH):** It corresponds to the IP/UDP compressed header of each original packet.
- **Payload (P):** It is the UDP payload of the original packets generated by the application.

B. *Theoretical Analysis of the Proposed Method*

We will next make an analysis of the bandwidth saving which can be achieved by the use of this technique. We assume that the multiplexed packet size will never be above 1500 bytes, which is certain for the traffics of this work.

As said in the introduction, packet delay has a very big importance for this service. So we have used a multiplexing policy that maintains packet delay under an upper bound. In the multiplexer it is defined a period, named *T*, and a multiplexed

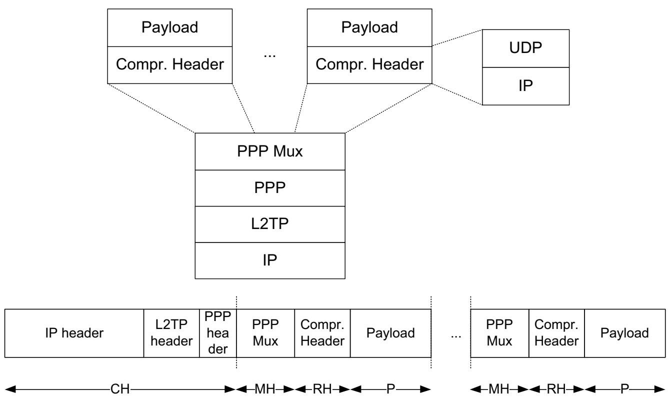


Figure 2. TCM protocol stack and scheme of a multiplexed packet.

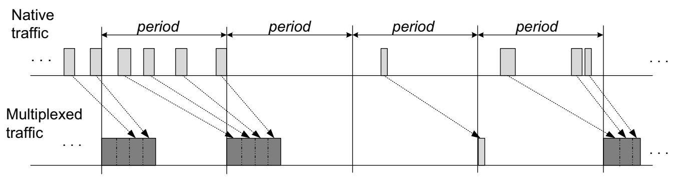


Figure 3. Behavior of the multiplexing policy.

packet is sent at the end of each period, including all the arrived ones (Fig. 3). There are two exceptions: if there is no packet to multiplex, nothing will be sent; and if there is only one packet, it will be sent in its native form, as the use of a tunnel would make it be bigger.

We will refer to the packets generated by the application as *native*, in contrast to multiplexed (*mux*) ones. With this policy, the average retention delay will be *T/2*, and its upper bound will be *T*.

An interesting parameter is the bandwidth relationship *BWR*, which is the division of *mux* and *native* bandwidths. We will denote the number of arrived packets in a period as *k*. *NH* denotes the size of an IP/UDP header. In order to obtain *BWR*, we will first calculate the number of bytes sent in a period when multiplexing is applied, so we have to distinguish the case of having one packet:

$$size_{mux} = \Pr(k=1)(NH + E[P]) +$$

$$+ \Pr(k>1)\,[CH + E[k|k>1](MH + E[RH] + E[P])] \quad (1)$$

Next, the expected value of the size of the *native* packets arrived is:

$$size_{native} = E[k]\,(NH + E[P]) \quad (2)$$

And dividing (1) and (2) we obtain *BWR*:

$$BWR = \frac{\Pr(k=1)}{E[k]} + \Pr(k>1)\,\frac{CH}{E[k](NH+E[P])} +$$

$$+ \Pr(k>1)\,\frac{E[k \mid k>1]}{E[k]}\,\frac{MH+E[RH]+E[P]}{NH+E[P]} \quad (3)$$

The first term is caused by the decision of not multiplexing when having only one packet. The second one expresses how the common header is shared by the whole packet. It becomes smaller as the number of multiplexed packets increases. The third term depends on the compressing algorithm, and in the average packet size generated by the application.

So, if we have a big number of users, or a big period, the number of packets will be big, and the first and second terms will become negligible. Regarding to the third one, Pr*(k>1)* will almost be 1, and the same will happen to *E[k|k>1]/E[k]*. So we can obtain the expression for an asymptote for *BWR:*

$$BWR_a = \frac{MH+E[RH]+E[P]}{NH+E[P]} \quad (4)$$

We observe that the smaller the value of *E[P],* the smaller the value of the asymptote. So the presented technique should have a good behavior in applications that generate a high rate of small packets, as FPS games do. Logically, it is expected that the bigger the number of players, the better the behavior, as the same number of packets can be multiplexed with less added delays. And the increase of *T* will also be beneficial for *BWR,* but we can not increase it indefinitely, as players are very sensitive to delay.

On behalf of clarity, we will not include here the calculation of *E[k|k>1]*, Pr*(k=1)* and Pr*(k>1)*. These calculations are included in the Appendix. As shown there, they may vary depending on the behavior of each game.

In order to get numerical and graphical results, we will now use the real parameters of some commercial games, and the ones used in the proposed protocols:

- *NH*: 28 bytes for IPv4/UDP and 48 bytes for IPv6/UDP.
- *CH*: 25 bytes for IPv4: 20 correspond to IP, 4 to L2TP and 1 to PPP header. For IPv6, *CH*=45 bytes.
- *MH*: 2 bytes, corresponding to PPPMux.
- *E[P]:* The value of the UDP payload depends on the application used.
- *E[k]:* The number of packets per second generated by the *N* players of the game.
- *E[RH]:* In this example, to be in the worst case, we have considered IPHC compressing UDP headers to 2 bytes, by using only 8 bits for the CID field, and avoiding the optional checksum. IPv4 and IPv6 headers can also be compressed to 2 bytes. So we will consider 4 bytes average for compressed headers and 28 or 48 bytes for full headers, which are sent every 5 seconds (the default F_MAX_TIME parameter of IPHC).

Using these values, we obtain the values shown in Table 1. They are the values of the asymptote, i.e. the best *BWR* that can be achieved if the number of users and the period are big

TABLE I. *BWR* ASYMPTOTE VALUES FOR DIFFERENT GAMES

Game	*Engine*	*E[P]*	λ	BWR_a *IPv4*	BWR_a *IPv6*
Unreal T 2003	Unreal 2.0	29.5	25	62%	46%
Quake III	Id Tech 3	36.15	93	65%	50%
Quake II	Id Tech 2	37	26.38	66%	51%
Counter Strike	GoldSrc	41.09	24.65	68%	53%
Halo 2	Halo2	43.2	25	69%	54%

enough. They are obtained for IPv4 and IPv6. We have selected some popular games, and the concrete values have been obtained from [10] and [11]. The values for Halo 2 refer to a console with only one user [19]. The value of λ for Quake 3 is the one obtained with the fastest graphic cards.

The obtained values are significant. All the games allow bandwidth savings above 30% for IPv4, and this saving can grow up to 54% if IPv6 is used in some titles.

In order to have a better idea of the system benefits, some graphs will next be presented, to illustrate the behavior of the *BWR* not only in the asymptotic value, but also for different users and periods. As we have to depict the graphs for a concrete game, we have selected Half Life Counter Strike 1, due to its popularity and the availability of many studies of its behavior [9], [20]. In Fig. 4a we have depicted *BWR* for IPv4 as a function of the number of players and the period. If we fix the number of players, we obtain Fig. 4b and if we fix the value of the period, we obtain Fig 4c. The asymptotic behavior can be observed for both parameters, so the most interesting zone is when the bandwidth relationship between 0.70 and 0.75. As an example, if we look at the *20 players* graph of Fig. 4b, once the value of 0.75 is reached, the increase of the delays in order to improve the bandwidth saving will only achieve a little benefit.

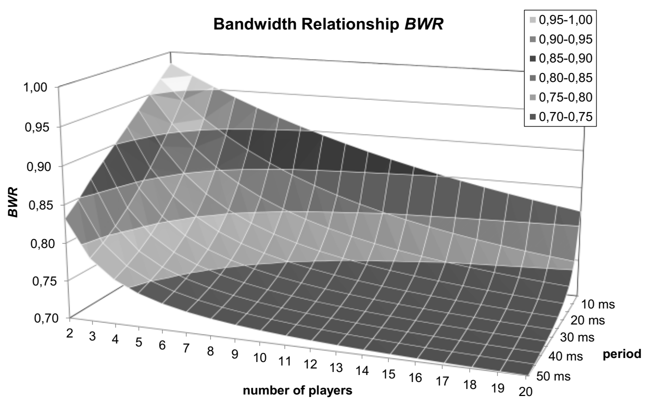
It may also be noticed that the increase of the number of players has also an influence. Logically, if there are more players, the same value of *E[k]* can be achieved with smaller values of *T*. So we confirm that the increase of the number of players is always beneficial. In fact, if there are only 5 players, perhaps it would be better to maintain the value of *BWR* around 0.80. Logically, the value of the network delay will have an influence on the decision of the value of *T*. If the network is fast, we can add a bigger delay, thus increasing bandwidth saving.

We can make a final observation: as said in previous sections, another limitation of commercial routers is the number of packets per second they can manage. This method also reduces this number by a factor of *E[k]*.

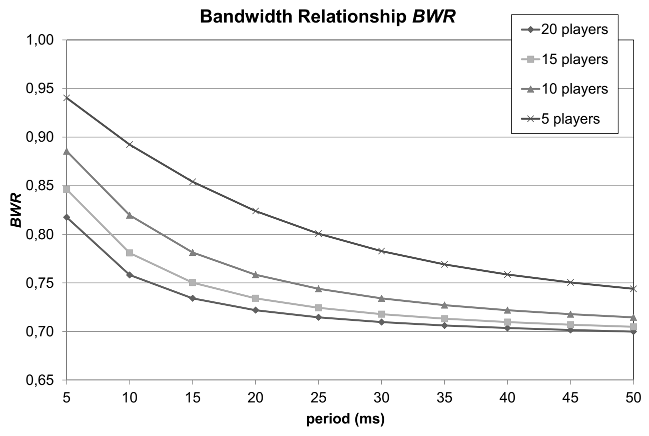
C. System delays

In this subsection we study the impact of the proposed method in the SRT. In Fig. 5 we show a scheme of the system with the delays that are added in order to obtain SRT.

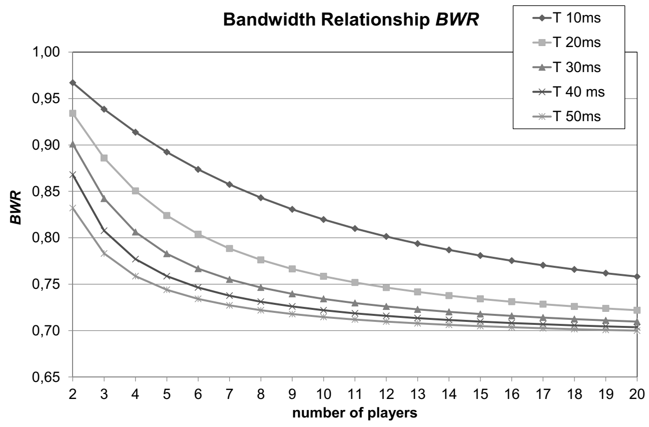
- $T_{retention}$ is the time a packet is retained at the queue of the multiplexer.
- $T_{process}$ represents the time spent in both the multiplexer and demultiplexer. In [21] a multiplexer for RTP traffic was implemented, and the processing time was less than 1 ms.
- T_{queue} is the time spent at the queue of the access router. The method presented in this paper does not directly modify this time.
- $T_{network}$ is the network delay, which is neither affected.



(a)



(b)



(c)

Figure 4. a) *BWR* as a function of the number of players and the period. b) *BWR* as a function of the period. c) *BWR* as a function of the number of players.

So the only significant delay which is added is $T_{retention}$, which is *T/2* average. In [7] it is said that latency tolerance is between 150 and 180 ms for Quake III, and above 200 ms for Counter Strike, so this retention delay can easily be assumed.

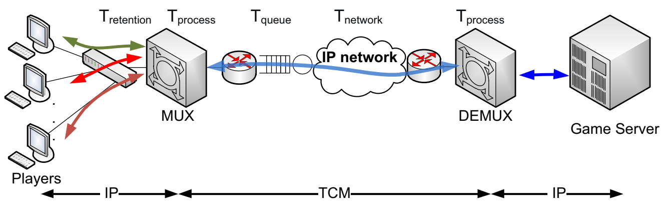


Figure 5. Delays of the system

In this paper we have not considered the possibility of modifying the application but, if it could be done, a first synchronization phase could be implemented in order to make all the computers in the same game generate the packets in the same moment, so this delay could be significantly reduced for the games that use a fixed inter-packet time.

IV. Tests and Results

In this section we will show some simulation results obtained with real game traces. First, we will describe the method used in order to generate the traffic for the tests.

In order to compare the simulation results with the theoretical ones, we have used the same game in this section: Half Life Counter Strike 1. The traffic traces have been obtained from CAIA project (e.g. the trace for 5 players is in [22]). There are available traces from 2 to 9 players. There is an offset of the first 10,000 packets, and only the next 5,000*number_of_players packets are included, to ensure that all the packets correspond to active game traffic, which is the one we are studying.

In order to obtain traces for more players, we have added some of them, e.g. we have obtained a trace of 20 players in *dedust* scenario, by the addition of the traces of 9, 6 and 5 players in the same scenario. This can be done due to a property of the client to server traffic, whose distribution is independent of the number of players [8], [20]: the client to server traffic of a 20-player game will be similar to the addition of three games of 9, 6 and 5 players. Logically, we have cut the time of the traces to the shortest one, obtaining traces of 110 seconds.

A simulation has been conducted using Matlab, in order to obtain the compressed and multiplexed traffic traces, as shown in Fig. 6. First, the trace is separated into the individual traces of the different players, and the server to client traffic is eliminated. For generating the traffic for the tests, we extract the generation time, the user and the size of the packet from the real traces. Next, IP/UDP compressing is applied to each flow. Finally, using the period *T*, the sizes and times of the multiplexed packets are calculated.

The studied game has three different behaviors, depending on the graphic rendering method [11], [20]. In our case, the traces have been obtained with OpenGL rendering, which is the most common one, and has inter-packet times of 33 ms and 50 ms with a 50% of probability each. This makes the value of λ be 24 packets per second. The analysis of the probabilities for this traffic has been included in the Appendix, subsection B.

Fig. 7 compares the theoretical values of *BWR* and the ones obtained in the simulations. It can be observed that the obtained values are similar to the theoretical ones, except for small values of the period and the number of players. The cause of this is that inter-packet times are not exactly the expected ones, as they have a small variation around 33 and 50 ms, as it can be seen in the histogram (Fig. 8).

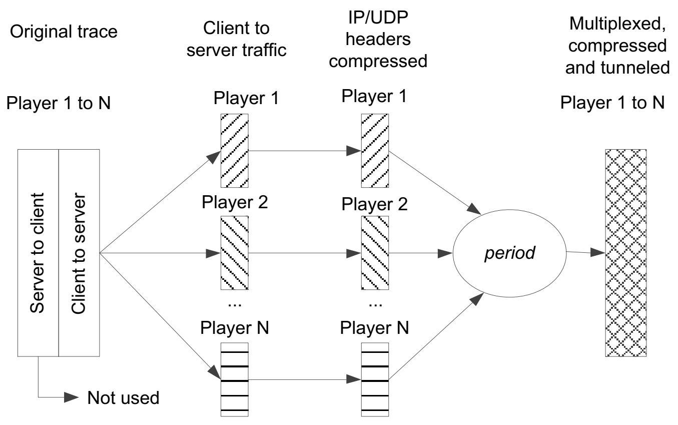


Figure 6. Method used to build the traces.

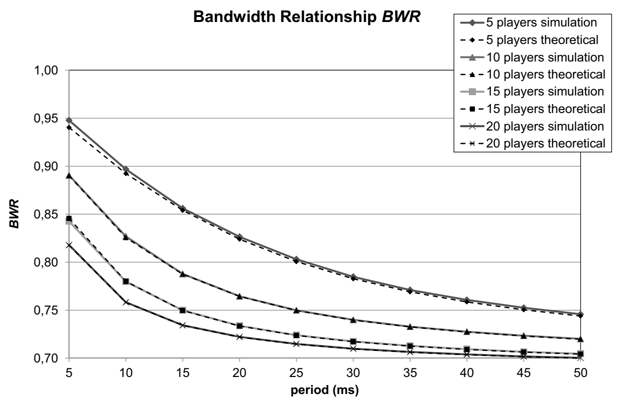


Figure 7. Comparative of the theoretical and the simulation results of *BWR*

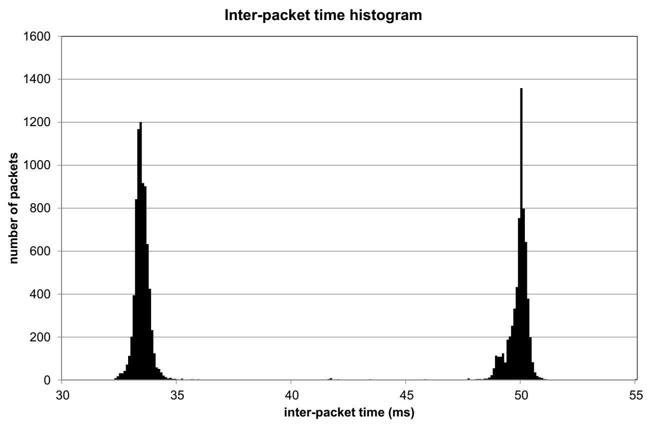


Figure 8. Histogram of inter-packet time in ms

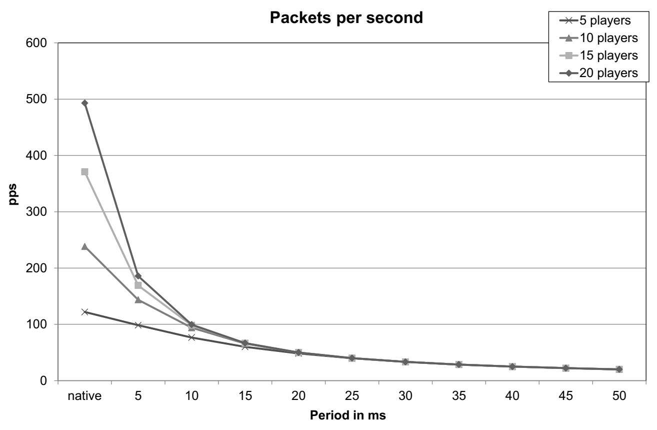


Figure 9. Packets per second managed by the access router.

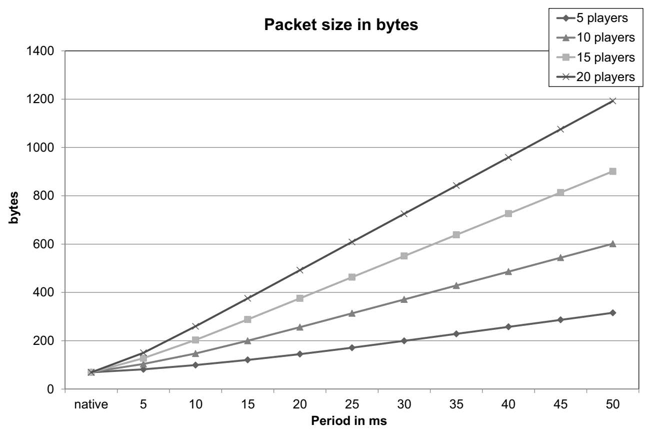


Figure 10. Size of *mux* packets.

Fig. 9 presents the number of packets per second managed by the access router. As explained in section II.A, a reduction of this parameter can also be interesting. We observe that the bigger the period, the smaller the amount of packets per second, which tends to be the inverse of the period, despite the number of players. Fig. 10 presents the packet size, which grows linearly with the period.

V. CONCLUSIONS

This work has presented a tunneling, compressing and multiplexing method, which can be used to achieve bandwidth savings by compressing headers and grouping packets into bigger ones. It can be useful in order to reduce the overhead of online gaming traffic, as these applications usually generate a high rate of small packets. The method uses an IP/UDP compressing protocol, multiplexing with PPPMux and a L2TP tunnel in order to work end-to-end.

The enterprises which develop games could be interested on reducing the bandwidth, and also the number of packets per second they have to manage. The bandwidth savings can also be interesting in order to obtain a better behavior in access networks with limited bandwidth.

The method has been tested with the traffic of FPS games, because these applications have very stringent temporal constraints, as players demand a high interactivity. Simulations have been conducted in order to study the bandwidth savings, and the results show that the bandwidth can be reduced up to 38% for IPv4 and over 50% for IPv6. The added delays can remain small if the number of players sharing the same path is big enough. As a future line, we consider the deployment of an algorithm that dynamically adjusts the multiplexing parameters depending on the number of players and the network statistics, as delay and packet loss.

APPENDIX

Here we present the calculations necessary to obtain an analytical expression of $E[k|k>1]$, which is required to build the graphs of *BWR* presented in section III. First, we obtain $E[k]$ as:

$$E[k] = \Pr(k=0)\, E[k|k=0] + \Pr(k=1)\, E[k|k=1] +$$

$$+ \Pr(k>1)\, E[k|k>1] \tag{5}$$

And taking into account that $E[k|k=0]=0$ and $E[k|k=1]=1$, we obtain:

$$E[k|k>1] = \frac{E[k] - \Pr(k=1)}{\Pr(k>1)} \tag{6}$$

So we need to obtain the expressions for $\Pr(k=0)$, $\Pr(k=1)$ and $\Pr(k>1)$. In the previous analysis, we have defined k as the total number of packets arrived to the multiplexer, i.e. the sum of the packets from each player. Now, we define l as the number of packets of a single player. We consider that the N different players' packet arrivals are independent, so $E[k]=N\,E[l] = N\,\lambda\,T$.

A. Constant packet rate

The different probabilities depend on the statistical distribution of the inter-packet delay of the studied game. We will consider a constant packet rate, as it occurs in many games [11]. Let t be the inter-packet time. We will consider $T < 2t$, in order to avoid big added delays, so the maximum value of l is 2, and consequently:

$$E[l] = \lambda\, T = \Pr(l=1) + 2\Pr(l=2) \tag{7}$$

If we have $T \le t$, then $\Pr(l=2)=0$, so using (7):

$$\Pr(l=1) = E[l] = \lambda\, T \tag{8}$$

$$\Pr(l=0) = 1 - \Pr(l=1) = 1 - \lambda\, T \tag{9}$$

And if we have $T > t$, then $\Pr(l=0)=0$, so knowing that the sum of the probabilities is 1, and using (7), we obtain:

$$\Pr(l=1) = 2 - E[k] = 2 - \lambda\, T \tag{10}$$

$$\Pr(l=2) = 1 - \Pr(l=1) = E[k] - 1 = \lambda T - 1 \qquad (11)$$

And now we can find Pr $(k=0)$:

$$\Pr(k=0) = [\Pr(l=0)]^N \qquad (12)$$

But Pr$(k=1)$ is null if there is more than one player and $T>t$, as every player will have sent at least one packet during the period. So the expression for Pr$(k=1)$ for $T \le t$ is:

$$\Pr(k=1)|_{T \le t} = \binom{N}{1} \Pr(l=1)[\Pr(l=0)]^{N-1} \qquad (13)$$

B. Two possible inter-packet times

In this subsection we consider the case of a game that generates packets using two different inter-packet times. Let t_1 be the smallest one and t_2 the biggest one, and p_1 and p_2 the respective probabilities of having t_1 and t_2. We will consider, as happens in the game considered in this work, that $T < 2t_1$, and $t_2 < 2t_1$. In this case, we have the next value for λ:

$$\lambda = \frac{1}{p_1 t_1 + p_2 t_2} \qquad (14)$$

We will now distinguish two cases. First, if we have $T < t_1$, then we have the same case as in (8) and (9), as Pr$(l=2)=0$.

In the case $t_1 \le T < t_2$, we have to find Pr$(l=0)$, i.e. the probability of having no packets in a period T, which will be the probability of the *period* beginning during the first t_2-T seconds of an inter-packet time of duration t_2, so, considering that the consecutive inter-packet times are independent:

$$\Pr(l=0) = p_2 \frac{t_2 - T}{p_1 t_1 + p_2 t_2} = p_2 \lambda (t_2 - T) \qquad (15)$$

And now, using (7) and knowing that the sum of the probabilities has to be 1, we will be able to obtain:

$$\Pr(l=1) = \lambda[T - 2p_1(T - t_1)] \qquad (16)$$

$$\Pr(l=2) = p_1 \lambda (T - t_1) \qquad (17)$$

Finally, (12) and (13) can be used in order to obtain the probabilities of the different values of k.

REFERENCES

[1] C. Chambers, W. Feng, S. Sahu, D. Saha: Measurement-based Characterization of a Collection of On-line Games. In Proceedings of the 5th ACM SIGCOM conference on Internet Measurement (IMC'05). USENIX Association, Berkeley (2005)

[2] K. Chen, P. Huang, C. Lei: Game traffic analysis: An MMORPG perspective. In Proceedings of the international workshop on Network and operating systems support for digital audio and video (NOSSDAV'05), pp. 19-24. ACM, New York (2005)

[3] M. Mauve, S. Fischer, J. Widmer: A Generic Proxy System for Networked Computer Games. In Proceedings of the 1st workshop on Network and system support for games (NetGames'02), pp. 25--28. ACM, New York (2002)

[4] D. Bauer, S. Rooney, P. Scotton: Network Infrastructure for Massively Distributed Games. In Proceedings of the 1st workshop on Network and system support for games (NetGames'02), pp. 36-43. ACM, New York (2002)

[5] M. Degermark, B. Nordgren, D. Pink: RFC 2507: IP Header Compression (1999)

[6] B. Thompson, T. Koren, D. Wing. RFC 4170: Tunneling Multiplexed Compressed RTP (TCRTP), November (2005)

[7] S. Zander, G. Armitage: Empirically Measuring the QoS Sensitivity of Interactive Online Game Players. Australian Telecommunications Networks & Applications Conference (ATNAC2004), Sydney, Australia, December (2004)

[8] P. Branch, G. Armitage: Extrapolating Server To Client IP traffic From Empirical Measurements of First Person Shooter games. In Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames '06). ACM, NY, USA (2006)

[9] W. Feng, F. Chang, W. Feng, J. Walpole: Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. SIGCOMM Comput. Commun. Rev. 32, p. 18 (2002)

[10] W. Feng, F. Chang, W. Feng, J. Walpole: A Traffic Characterization of Popular On-Line Games. IEEE/ACM Trans. Netw., pp. 488-500 (2005)

[11] S. Ratti, B. Hariri, S. Shirmohammadi, A Survey of First-Person Shooter Gaming Traffic on the Internet, IEEE Internet Computing, pp. 60-69, September/October (2010)

[12] K. Lee, B. Ko, S. Calo: Adaptive Server Selection for Large Scale Interactive Online Games. In Proc. 14th International Workshop on Network and operating systems support for digital audio and video (NOSSDAV'04), pp. 152--157. ACM, New York (2004)

[13] L. Gautier, C. Diot: Design and Evaluation of MiMaze, a Multi-player Game on the Internet. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMS'98), IEEE Computer Society, pp. 233. Washington (1998)

[14] V. Jacobson: RFC 1144: Compressing TCP/IP Headers for Low-Speed Serial Links (1990)

[15] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg: RTP/I-Toward a Common Application Level Protocol for Distributed Interactive Media. In Proceedings of IEEE Transactions on Multimedia, pp. 152-161 (2001)

[16] G. Pelletier, K. Sandlund, RFC 5225: RObust Header Compression Version 2 (ROHCv2) (2008)

[17] A. Couvreur, L. M. Le-Ny, A. Minaburo, G. Rubino, B. Sericola, L. Toutain, Performance analysis of a header compression protocol: The ROHC unidirectional mode, Telecommunication Systems, vol. 31, no. 6, pp. 85–98 (2006)

[18] E. Ertekin, C. Christou: Internet protocol header compression, robust header compression, and their applicability in the global information grid. IEEE Communications Magazine, vol. 42, pp. 106-116 (2004)

[19] S. Zander, G. Armitage, A traffic model for the Xbox game Halo 2. In Proc. International Workshop on Network and operating systems support for digital audio and video (NOSSDAV '05). ACM, New York, NY, USA, pp 13-18 (2005)

[20] T. Lang, G. Armitage, P. Branch, H. Choo: A Synthetic Traffic Model for Half-Life. In Australian Telecom, Networks and Applications Conference (ATNAC) Melbourne (2003)

[21] H. Sze, C. Liew, J. Lee, D. Yip: A Multiplexing Scheme for H.323 Voice-Over-IP Applications. IEEE J. Select. Areas Commun. Vol. 20, pp. 1360-1368 (2002)

[22] L. Stewart, P. Branch: HLCS, Map: dedust, 5 players, 13Jan2006. Centre for Advanced Internet Architectures SONG Database, http://caia.swin.edu.au/sitcrc/hlcs_130106_1_ dedust_5_fragment.tar.gz