# Shooting Around the Corner: The Problem of Real-time Services

**By Jose Saldana, Dan Wing, Julián Fernández-Navajas, José Ruiz-Mas, Muthu A.M. Perumal, Gonzalo Camarillo, Michael Ramalho**

A new draft has been submitted to the Transport Area working group (WG): draft-saldana-tsvwg-tcmtf. The primary goal of this draft is optimization of real-time flows independently from the use of real-time protocol (RTP) (RFC 3550: A Transport Protocol for Real-Time Applications, RFC 3551: RTP Profile for Audio and Video Conferences with Minimal Control). This draft proposes that a number of small packets be compressed, multiplexed, and bundled into one packet, with the resulting packet being forwarded using a tunnelling scheme. This proposal has demonstrated its ability to save bandwidth while concurrently reducing the packets' per-second rate. Preliminary tests have shown that significant bandwidth savings can be achieved while not introducing delays that could degrade the real-time user experience for gamers and the like.

It's 22:37 in San Jose, California. After a hard day of work, Jack is at home playing a game he has just bought: a first-person shooter game. He is exploring the scenarios and shooting virtually everything that appears in his screen. He spots a running enemy named "Wang" and he shoots. His screen shows Wang falling down.

It's 14:37 in Tokyo, Japan. Wang is spending his lunch break at school playing the same game. He is running through the same scenario and notices an enemy named "Jack" nearby. He quickly goes around a corner in order to avoid getting shot. But shortly after he has already turned the corner, a message that reads "you are dead" appears in the screen. He gets angry and tells his girlfriend: "Have you seen this? This player is cheating! He's shot me around the corner!"

## The latency problem

What's the problem? What Jack, Wang, and Wang's girlfriend are probably ignoring is the real cause of the problem: network latency. The server where they are playing the game is in California, only 80 km from Jack's house, but 8,200 km from Wang's school. So the network latency for Jack is very small and mainly due to equipment, whereas Wang's packets must travel through a submarine cable along the bottom of the Pacific Ocean. Figure 1 illustrates how when Jack shoots, that information quickly arrives at the server, which calculates the result and decides that Wang is dead. But the packet telling Wang's computer this information takes 150 milliseconds to arrive, due to a number of routers, the access network, and even the speed of light based delay. By the time this information arrives at Wang's application, he has already hidden around the corner.

This is only one example of the Internet's limitations when dealing with real-time services. The same problem appears in VoIP (voice over IP), remote desktop solutions, video conferencing, and database access, among others.

The Internet was designed as a *best-effort* network, which does not warrant a maximum delay. This was enough for traditional delay-insensitive applications, such as web browsing, e-mail, or FTP. However, new real-time

New real-time services are raising the question: Is the Internet adequate for them?

services are raising the question: Is the Internet adequate for them? Although many quality-of-service (QoS) mechanisms have been specified within the IETF, and used in many network scenarios (such as enterprise intranets), the Internet remains mostly *best-effort*-(non QoS)-based transport.

## The efficiency problem

Another interesting question is related to packet size, which usually ranges between 40—1,500 bytes. Since every packet must include the IP and the TCP or UDP (User Datagram Protocol) headers, we have a simple rule: the bigger the packet, the better the efficiency. Traditional services tend to

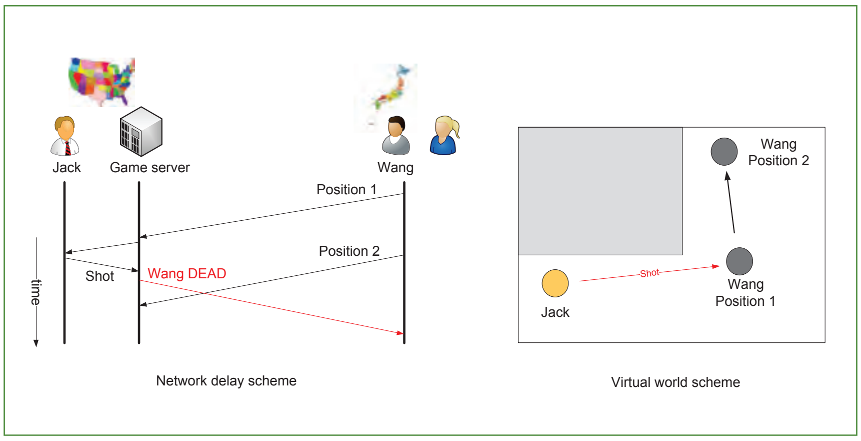

Figure 1. Schema of the latency problem for Wang

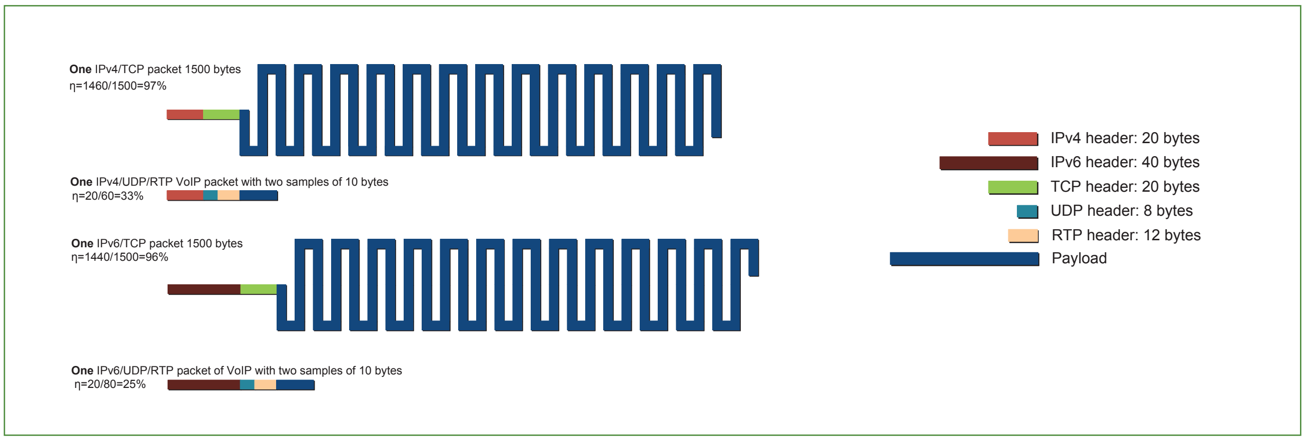
One IPv4/TCP packet 1500 bytes
η=1460/1500=97%
One IPv4/UDP/RTP VoIP packet with two samples of 10 bytes
η=20/60=33%
One IPv6/TCP packet 1500 bytes
η=1440/1500=96%
One IPv6/UDP/RTP packet of VoIP with two samples of 10 bytes
η=20/80=25%
IPv4 header: 20 bytes
IPv6 header: 40 bytes
TCP header: 20 bytes
UDP header: 8 bytes
RTP header: 12 bytes
Payload

Figure 2. The efficiency problem. *Note: Packet sizes are to scale.*

maximize packet size; since delay is not critical, they can wait until they have almost 1,500 bytes to send. But the problem with real-time services is that the information has to arrive quickly, sometimes with many applications, and sometimes with a fixed cadence. As a consequence, tiny information chunks are sent using a high frequency, which implies very small packets. Figure 2 illustrates how a packet of 1,500 bytes has an efficiency of 97 percent, whereas a VoIP packet carrying two voice samples of 10 bytes each has its efficiency reduced to 33 percent; in other words, only one out of three bytes is voice information while the others are headers. If IPv6 is used, the problem becomes even worse—20 more bytes are included in the IPv6 header and VoIP small packets' efficiency is reduced to 25 percent. Logically, the problem is almost negligible for big packets.

There is another issue that real-time services have highlighted: The packets per-second limit. Traditional network-equipment-processing capacity was designed to manage a mix of big and small packets (referred to as Internet Mix or IMIX). If the average packet size diminishes dramatically, and the number of packets per second is high, the route lookup function may become a bottleneck in addition to link speed.

## Description of some real-time services

In this section we will focus primarily on describing VoIP and online games, as they are the services for which tests have been deployed. Other services, such as remote desktop or video conferencing, could also be considered.

### VoIP

VoIP was one of the first widely deployed real-time services. It uses Real-time Transport Protocol (RTP), the IETF protocol designed to deliver real-time content, which uses 12 bytes for protocol header and works over UDP, so the total overhead is 40 bytes for IPv4 and 60 bytes for IPv6. One RTP packet includes a small number of voice samples, which use different voice codecs, some of them having a high robustness to packet loss. Since interactivity is high, retransmission of lost packets is not deployed; the destination application has to deal with packet loss, trying to conceal it. The maximum recommended round-trip delay is around 150 milliseconds.

### Online games and virtual environments

Online games are a real-time service in which avoiding delay is critical. Some research has shown that the maximum round-trip delay tolerated by gamers is about 150 to 200 milliseconds. In addition, gamers are extremely fickle:

Gamers are extremely fickle: if a game does not work properly, they may leave the server and never return.

if a game does not work properly, they may leave the server and never return. Games use client-server architectures for many reasons: the convenience of maintaining the consistency of the game, synchronization, cheating prevention, and recording high scores. But the main reasons are commercial: The providers can charge for use or sell the server software.

Different game genres can be distinguished:

- *First Person Shooters (FPS):* A virtual scenario shared by some tens of players is created. Every player controls an avatar that has to accomplish a mission or kill all the enemies. The weapon can be improved according to the score. The action is fast, and the aim of the player is crucial. Although the sessions may last up to two hours, the game is divided into short rounds, which may be of some minutes. The client application generates high rates of tiny UDP (non-

*Continued on next page*

Real-time Transport Protocol) packets, which go to the server. When the new game state has been calculated, it is sent to every player. Depending on the number of surviving players, these packets can be bigger.

- *Massively Multiplayer Online Role-Playing Games (MMORPGs):* This genre has become very popular, especially in Asia. Some titles have achieved more than 10 million subscribers (e.g., *World of Warcraft*). They are normally set in magical or historical environments and the player controls an avatar with a long-term life. He obtains better abilities, weapons, and curses by means of missions. Trading is also permitted. The player can connect to different servers, called *shards,* which are shared by some thousands of gamers. The action is less interactive than in FPSs. For example, in the case of MMORPGs, for purposes of fighting, first you select the enemy with your mouse, and then you may choose the weapon or curse to use against them. Nevertheless, these games can still be considered a real-time service: The speed and the ability of the player are decisive in determining who wins and who loses. The vast majority of these games use small TCP packets, which is surprising, since they are "real-time services" using TCP.
- *Real Time Strategy (RTS):* In these games, the player creates a city or a civilization, manages the resources, and amasses armies to attack other players and dominate the virtual world. The real-time requirements are looser.
- *Sports games:* There are a great variety of these types of games. The differences between their network behaviours are big, making it very difficult to establish general rules.
- *Some other virtual environments,* such as *Second Life,* also have become popular in recent years. A single server is usually shared by a large number of users.

Although games are real-time services, they do not use RTP to deliver information to the server. Instead, they use bare UDP or sometimes TCP. Also, unlike VoIP, they do not present a fixed-packet cadence; in other words, inter-packet time may vary depending on a player's actions and changes to the game's environment.

## Traffic optimization

Now that we have recognized the two problems, and the kind of traffic generated by real-time services, the question becomes, *can we optimize the traffic in order to improve packet efficiency as well as to reduce the packets per second?*

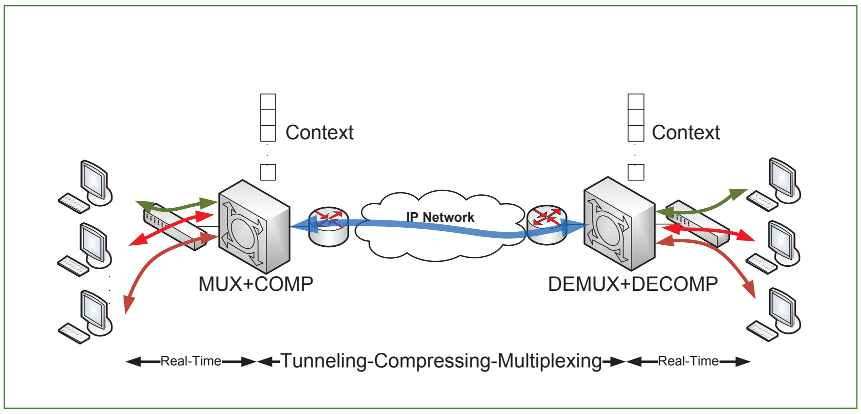

Figure 3. Tunnelling, compressing, and multiplexing traffic flows schema

### The idea of Tunnelling Compressed Multiplexed Traffic Flows (TCMTF)

Multiplexing a number of payloads into a single packet can be seen as a solution for improving network efficiency. If only one flow is present, the number of samples included in a packet can be increased, but at the cost of adding new packetization delays. However, if a number of flows share the same path from an origin to a common destination, then a multiplexer can build a packet in which a number of payloads share a common header. A demultiplexer will then be necessary at the end of the common path in order to rebuild the packets as they were originally sent, thus making multiplexing a transparent process for the two hosts that are exchanging data.

The headers of the original packets can also be compressed in order to save more bandwidth, using one of the header compression schemes defined by the IETF. The headers rely on the fact that many header fields are the same for every packet in a flow. Additionally, they use *delta* compression in order to reduce the number of bits required by a field; they only transmit the difference between the value of a field in a packet and in the previous one. Bandwidth is saved, but at the cost of the need for storing a series of values, the *context,* in the compressor and decompressor These store the header fields that are the same for every packet of the flow, and can thus be avoided. There is another counterpart: The possibility of context desynchronization, which would imply bursts of corrupted packets. Nevertheless, modern header compression schemes, such as Robust Header Compression (ROHC), are indeed robust against this.

Logically, a tunnelling scheme will be necessary in order to send the bundle via the public Internet. So we have a global scheme, including tunnelling, header Compressing and Multiplexing Traffic Flows (TCMTF), as shown in figure 3.

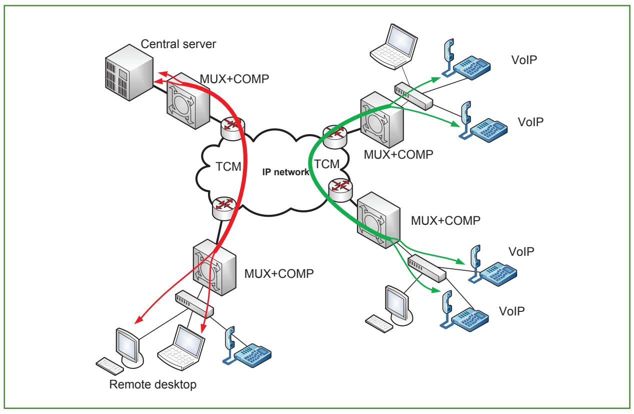

Figure 4a. Scenario in which optimization can be deployed.

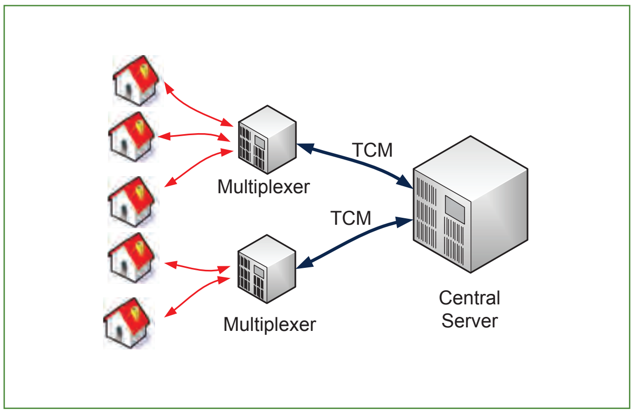

Figure 4b. Scenario in which optimization can be deployed.

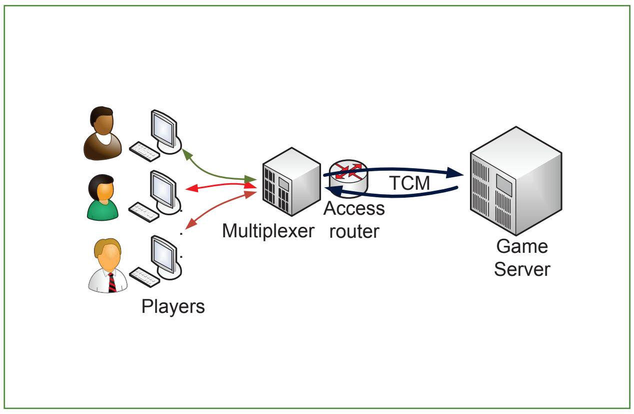

Figure 4c. Scenario in which optimization can be deployed.

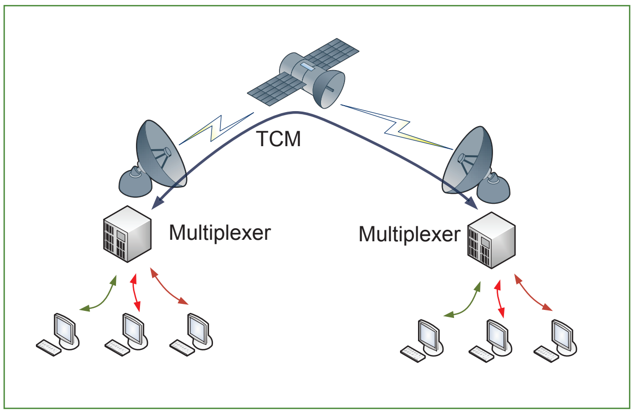

Figure 4d. Scenario in which optimization can be deployed.

Internet cafés, a very popular way to use the Internet in developing countries, are also places where people may simultaneously play the same game using the same server.

## Scenarios in which optimization can be deployed

We can find different network scenarios in which the traffic of a number of flows shares the same path. Figure 4a illustrates an enterprise with a number of offices, in which tunnels can be established for merging VoIP flows (green lines), and also for multiplexing the traffic of a remote desktop application (red lines). For example, the traffic of the users of a game in a town or a district can be multiplexed by the ISP and sent to the central game server (figure 4b). Internet cafés, a very popular way to use the Internet in developing countries, are also places where people may simultaneously play the same game using the same server. So a tunnel could also be established from the router, or even from the computer of one of the players, to the game server (figure 4c). Another scenario is a satellite link (figure 4d), which may manage the bandwidth by limiting the transmission rate, measured in packets per second to and from the satellite. If small packets are used, this will result in poor utilization of the bandwidth, establishing an upper bound for the number of calls that can utilize the link simultaneously. Multiplexing small packets into a bigger one would improve the efficiency. This may be especially interesting for compressing TCP acknowledgements.

## Current status and new proposal

The IETF has already tackled this problem for RTP, first with cRTP in 1999 (RFC 2508: Compressing IP/UDP/RTP Headers for Low-Speed Serial Links), which compressed headers across links; then enhanced to work over networks with loss or reordering with ECRTP in 2003 (RFC 3545: Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Re-ordering); and finally with RFC 4170: Tunnelling Multiplexed Compressed RTP (TCRTP) in 2005, which had the main aim of improving the efficiency of multiple RTP streams across a network. This is useful in the scenarios in which

Continued on next page

*Shooting Around the Corner: The Problem with Real-time Services, continued*

many VoIP conversations share the same path: we can do "voice trunking" between two offices of an enterprise, or group a number of conversations of a network provider. TCRTP, approved as a "best current practice," merged three layers (figure 5): First, RTP/UDP/IP headers were compressed using ECRTP; next, a number of header-compressed packets were multiplexed with PPP Multiplexing (PPPMux). Finally, the bundle was sent using an L2TP tunnel (RFC 2661: Layer Two Tunnelling Protocol).

But many things have happened since 2005:

- The outbreak of wireless access networks, which enable people to access the Internet from almost anywhere. These wireless scenarios are prone to packet loss, and may add bigger delays than the ones we can find in wired environments.
- The approval of ROHC (RObust Header Compression) in 2007 as RFC 4995: The RObust Header Compression (ROHC) Framework, updated by RFC 5795: The RObust Header Compression (ROHC) Framework in 2010. This header-compression standard was specifically designed for links with high loss and high round-trip times. It not only compresses RTP, but also IP and UDP. In addition, ROHC for TCP was defined by RFC 4996: RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)
- The approval of RFC 5856: The Integration of Robust Header Compression over IPsec Security Associations in 2010 as a framework for integrating ROHC over IPsec (ROHCoIPsec), which targets the application of ROHC to tunnel mode Security Associations (SAs). It reduces the protocol overhead associated with packets traversing between IPsec SA endpoints. This is achieved by compressing the transport layer header and inner IP

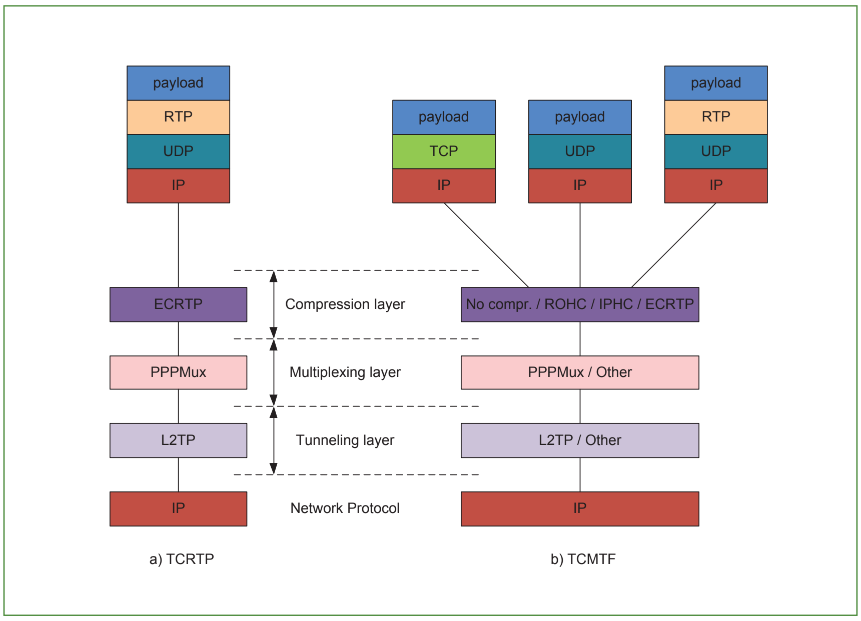

Figure 5. Protocol stack of TCRTP (left) and protocol stack of TCMTF (right).

## Newcomer Experience: Arranging a Last-Minute Online Gaming Tutorial

*By Jose Saldana, University of Zaragoza*

IETF 83 in Paris, France, was my first IETF meeting. I enjoyed the Newcomers' Orientation, where I met many interesting people, and the meeting mailing list, where people asked all sorts of questions from running routes to directions to the Louvre. The variety of these discussions and the wide range of people I met gave me an idea: I offered to host an informal tutorial to discuss the kinds of network traffic generated by online gaming.

I was amazed by the positive response. I worked with the IETF Secretariat to book a room for my tutorial on Tuesday morning just after my presentation to the Transport Area working group (WG). About 25 people attended the 50-minute tutorial that I arranged via email list and coordinated with the Secretariat in only about three hours.

I demonstrated three games and presented some of the traffic optimizations that we've studied and are now trying to standardize, including tunnelling, header compression, and multiplexing. We used Wireshark to capture the traffic and saw the different options that game developers use for each genre: UDP for first-person shooter and real-time strategy games, and TCP for massive-multiplayer, online, role-playing games.

I learned from the audience's questions and comments, and several more requests for tutorials rolled in over the rest of the week. All in all, I was surprised by the speed with which things happen at IETF meetings, and I left the meeting very glad that I decided to stay the whole week.

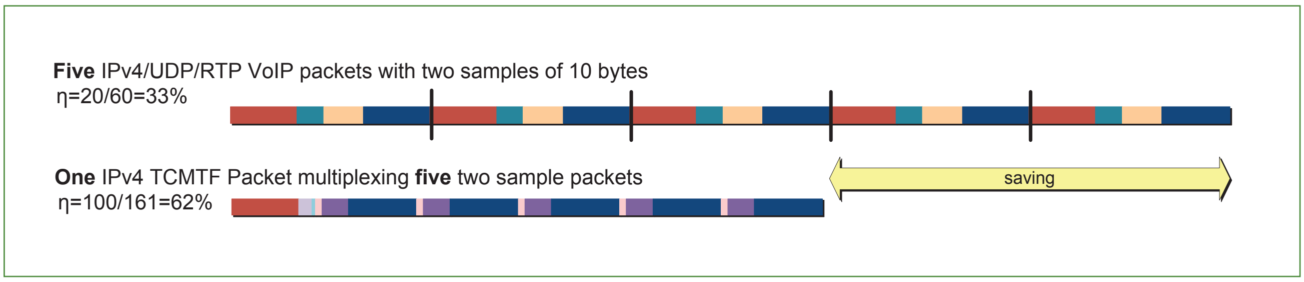
**Five** IPv4/UDP/RTP VoIP packets with two samples of 10 bytes
η=20/60=33%
**One** IPv4 TCMTF Packet multiplexing **five** two sample packets
η=100/161=62%
saving

Figure 6a. Header compression results: VoIP

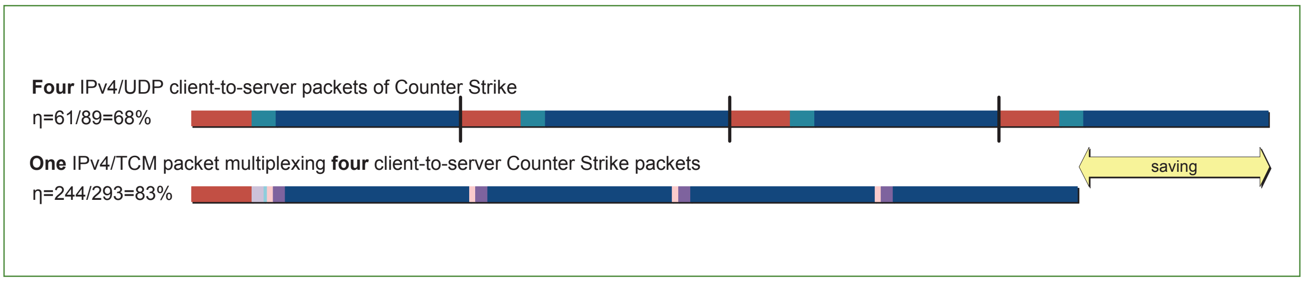
**Four** IPv4/UDP client-to-server packets of Counter Strike
η=61/89=68%
**One** IPv4/TCM packet multiplexing **four** client-to-server Counter Strike packets
η=244/293=83%
saving

Figure 6b. Header compression results: FPS

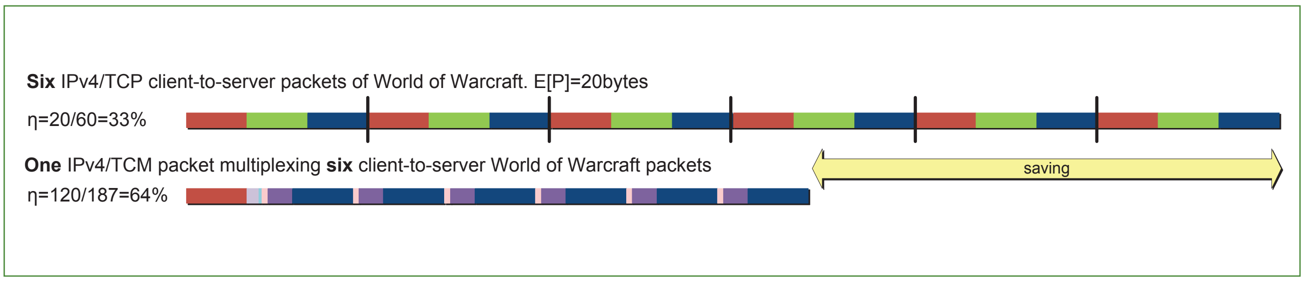
**Six** IPv4/TCP client-to-server packets of World of Warcraft. E[P]=20bytes
η=20/60=33%
**One** IPv4/TCM packet multiplexing **six** client-to-server World of Warcraft packets
η=120/187=64%
saving

Figure 6c. Header compression results: MMORPG

header of packets at the ingress of the IPsec tunnel, and decompressing these headers at the egress.

- The popularity of many real-time services—online games have become very popular applications. As we have seen, they do not use RTP protocol.

As a consequence, we have considered doing the same thing as TCRTP, but also in cases when real-time flows do not use RTP. We can use ROHC to compress their headers, and do something similar in order to include a number of packets into a PPP Multiplexing (PPPMux) bundle.

Figure 5 illustrates this new proposal, which includes the three layers, but also considers more options: Different traffic types can be used and compressed—TCP, UDP, and also RTP—in the same way it was compressed by TCRTP. The compressing protocol will have to be selected depending on many factors: the scenario, the availability of processing and memory resources, etc. In addition, a null header compression is considered, taking into account that in some cases there may be many context synchronization problems. With respect to multiplexing and tunnelling, other options different from PPPMux and L2TP may also be considered.

Finally, as mentioned previously, in some services, interpacket time is not fixed. So we must define a policy in order to determine which packets are multiplexed in each bundle. We can do that either by defining a fixed number of packets or defining a maximum packet size. Another option is to define a period or a timeout, which may be more adequate for setting an upper bound for the added delay.

## Preliminary tests

Figures 6a-c illustrate the savings that could be achieved by TCMTF. *(Note: The colors of the headers correspond to the layers in figure 5. Headers and payloads are to scale.)* Figures 7a-c show the bandwidth savings that have been obtained for the same services, by means of simulations based on ECRTP (Enhanced Compressed RTP) or IP header compression (IPHC) over PPP, Layer 2 Tunnelling Protocol, Version 3 (L2TPv3) and PPPMux.

*Continued on next page*

In Figures 7a-c we can see the bandwidth saving, measured as the quotient of TCMTF bandwidth divided by the native one. In figure 7a it can be seen that significant savings can be achieved when multiplexing different numbers of G.729a voice flows, depending on the number of flows and on the number of samples per packet (1, 2, or 3 samples, which means 10, 20, or 30 bytes of payload). The savings present an asymptote, which implies that when the number of multiplexed flows is high, the difference will be small in terms of bandwidth.

Tests have also been deployed for an FPS (*Counter Strike*), which sends UDP packets (figure 7b). The graph shows the bandwidth savings depending on the period and the number of players. If the period is small, the added delays can be kept in the order of 10 or 20 milliseconds, in order not to annoy the players. It must be taken into account that the average added delay is half the period.

Finally, figure 7c shows an example of the gains achieved for an MMORPG—the bandwidth savings are higher than the ones obtained for the FPS, however the number of players and the multiplexing period must be higher. This is not a problem, since the interactivity of these games is not as critical as in FPSs.

## Conclusion

Summing up, the TCMTF proposal is able to mitigate the efficiency problem by sharing a common header across multiple payloads. Additional delays will be incurred, but they are small enough that they will not harm subjective quality. As we have seen, being able to both optimize RTP flows and bare UDP or even TCP can save bandwidth and reduce the number of packets per second generated—compelling advantages in the scenarios we've illustrated here.

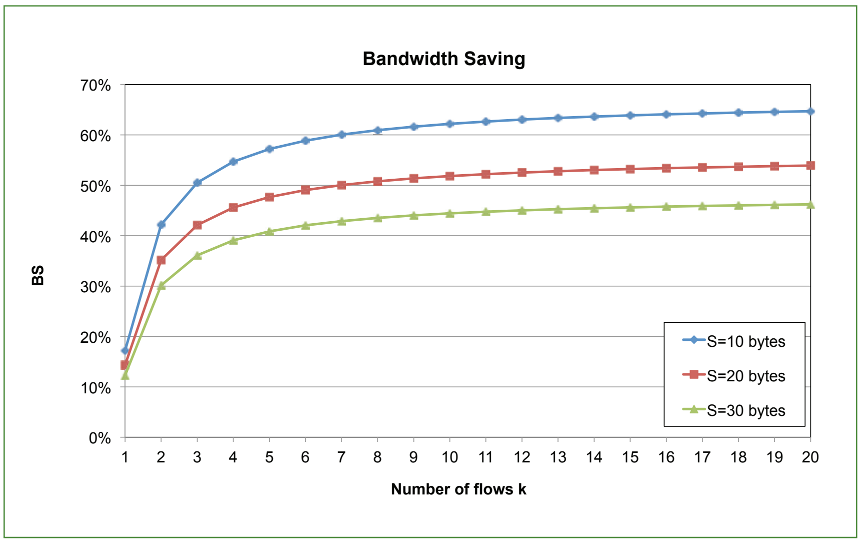

Figure 7a. Bandwidth savings for VoIP: G.729a codec with two samples per packet.

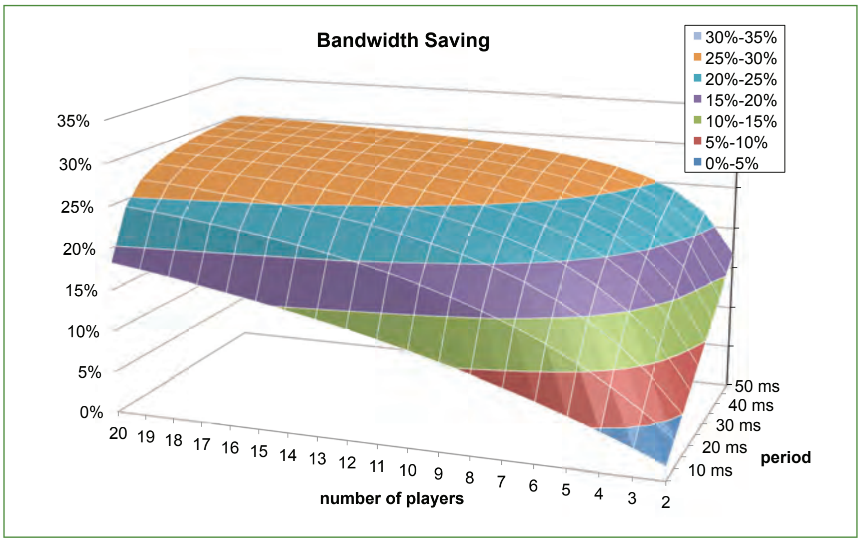

Figure 7b. Bandwidth savings for FPS: *Counter Strike*

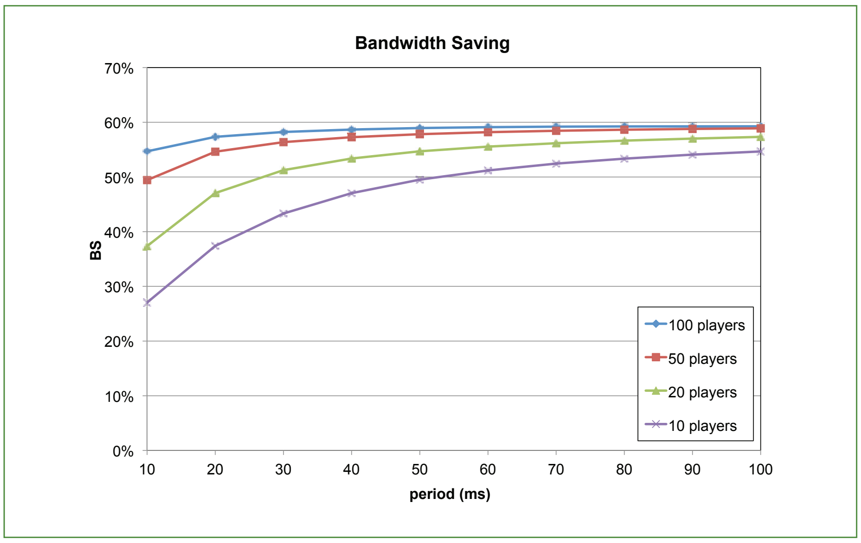

Figure 7c. Bandwidth savings for MMORPG: *World of Warcraft*