

The Buddycast Protocol

Version: 0.51 5/31/2005

Authors: Mel Beckman, Denny Bollay, and Brian Fox

Abstract

The Internet does not currently support RFC-1112 multicast transmission, forcing streaming content distributors to use unicast streams that consume large amounts of distributor-purchased bandwidth.

BuddyCast is a peer-to-peer protocol for distributing streaming content to an unlimited number of users by turning broadcast receivers into stream relays, or “buddies.” The protocol maintains resilience through the use of standby redundant buddies to replace buddies that leave the network. Broadcast distribution is controlled by a Buddycast Multicast Controller (MC), which hands off inbound requests for streams to available buddies and monitors the state of relayed streams. To this end, the MC maintains a network topology map, which the MC can exploit to improve performance and minimize the number redundant inter-network streams.

Because the number of available relays grows along with the number of receivers, Buddycast can scale to serve an audience of any size. This lets even the smallest broadband Internet user distribute streaming content to a large audience. Because content is sent only to willing receivers, the total bandwidth consumed on the Internet will always be less than an equivalent unicast broadcast – typically much less, as the MC only handles connection requests and buddy management.

Principle of Operation

The essential idea of Buddycast is a bucket brigade of UDP receivers, called a Buddynet, each passing on a stream packet it receives on to downstream receivers. A simple linear brigade would suffer from a serious weakness: if a buddy drops out of the brigade, all downstream receivers no longer receive the stream.

To repair such failures, the MC maintains a map of the buddy network and assigns each receiver a “backup buddy” when the receiver first joins the Buddynet. A receiver can then immediately switch to its backup without the intervention of the MC, resulting in fast recovery from simple topology changes.

The primary penalty of the Buddynet scheme is the program delay that increases the farther a receiver is down the buddy brigade. This usually is not a problem with streaming content, as long as the delay is not more than a minute or two. And because streaming content is highly tolerant of occasional missed packets, there is no need for Buddycast to maintain strict data integrity, making Buddycast an extremely lightweight protocol.

A secondary penalty is the accumulation of packet loss as a stream progresses down a buddy brigade. Receivers periodically report performance statistics to the MC so that the MC can reassign buddies when packet loss gets too high on a particular path. Because the MC knows the exact amount of loss for each node in the brigade, it can choose an appropriate point at which to break the chain and attach it to a better-performing receiver in the network.

Buddycast works even for two buddies screened behind separate firewalls, using the UDP glare technique of peering pass-through. The MC communicates a unique UDP port number pairing to both buddies, which can then establish two-way communication through each respective firewall. The buddies send an initial UDP burst at each other simultaneously, which opens up mutual ingress pores in the firewalls. The content stream can then be pushed from the upstream buddy to the downstream one; as long as bidirectional traffic continues between the buddies, the firewall pores remain open.

The MC maintains continuous control over a broadcast in progress, monitoring Buddycast receivers to measure network performance, and issuing topology change commands to receivers when network failures or performance problems require them. The MC assigns a unique token to each buddy to ensure network integrity and help prevent denial of service attacks. A content distributor can enhance a Buddynet's resilience by spawning load-sharing MCs, so that the death of an MC does not necessarily mean the end of the broadcast.

The number of downstream buddies for a relaying receiver is not specified by the protocol. The MC can use information it obtains during initial receiver negotiation – for example, the receiver's stated upstream bandwidth capacity -- to estimate a maximum number of downstream buddies. The MC can also dynamically change the number of downstream buddies based on the observed performance of a relaying receiver. These mechanisms are not specified in the Buddycast protocol, although a number of receiver and MC attribute variables are specified, which an MC can use to drive buddy selection and topology maintenance algorithms.

State Mechanics

Buddycast receivers go through a number of state transitions in the process of connecting to a Buddynet. They pass through other state transitions when recovering from a network topology change, either as the result of a buddy dropping out of the network, or in response to topology commands from the MC. The attached diagrams illustrate the state mechanics for each major network transition.

New receiver joins a Buddynet

Figure 1 illustrates the process of a new receiver joining a Buddynet. The receiver (R4) queries the MC to join the broadcast. The MC chooses a primary and a backup buddy (R3 and R2), and passes the assignment back to the new receiver. Existing receivers periodically transmit performance statistics to the MC. When the assigned upstream buddies next poll the MC, the MC notifies them of the new receiver. The MC then

notifies all three parties to start the UDP glare process, which results in the opening of firewall pores if necessary. Once two-way UDP communications is established, the primary buddy begins transmitting the stream, which the secondary buddy maintains a keep-alive conversation with the new receiver, pending a failure of the primary.

The time for a new receiver to join the network is the poll interval (p) plus the UDP glare delay (g), plus the elapsed time of the handshake (h). UDP glare takes a few seconds to synch up, receivers poll every ten seconds, and handshakes consume two seconds, then the maximum connection time would be $p+g+h$, or about 12 seconds.

Stream from a primary buddy times out

Figure 2 shows the process of recovering from loss of communications with a relay receiver. Assume the connection between R3 and R4 times out. R4 immediately asks its backup buddy, R2, to continue the stream. R4 can do this because it has maintained a keep-alive connection with R2. R4 then informs the MC that the backup buddy is now its primary buddy. The MC chooses a new backup buddy, and then executes a shortened version of the initialization process to join the receiver with its new backup.

In the simplest Buddycast implementation, each receiver has only one backup buddy. This is not a limitation, however; Buddycast supports an arbitrary number of backup buddies, and on less reliable networks (e.g. wireless) multiple backups may be desirable.

A non-relay receiver leaves a Buddynet

Many receivers in any given Buddynet are “leaf” nodes – at the end of the network with no assigned downstream buddies. When one of these receivers wishes to leave the network, it communicates its intent immediately to both the MC and all its buddies. Alternatively, a receiver may leave the network without notification (e.g., by crashing), in which case its buddies will learn of its departure when two-way communication times out.

Regardless of how a receiver leaves the network, once the departure is detected the cleanup process in Figure 3 occurs. In this example, R4 has left the network, an event first noticed by its primary buddy R2. R2 informs the MC (which would have eventually learned of the loss after missing a number of polls from R4). The MC then informs the backup buddy R1 to end its services for R4.

It’s possible, however, that the reason for R2’s loss of communication is the failure of the path between it and R4, not that R4 has died. In that event, R4 fails over to its backup, R1. Upon receiving the disconnect command from the MC, R1 responds that it is already primary for R4. Although the MC would have eventually learned this from R4 itself, R1’s announcement prevents a race condition where the MC unassigns R1 before R4 has notified the MC of the failover.

If R4 is truly dead, then the MC purges it from its tables and sends a reset notification to R4. This reset is sent blind purely as a cleanup measure – the MC will entertain no further

communications from R4 concerning this session, and R4 will have to start over from scratch to rejoin the network.

Optimizations

The Buddycast protocol formalizes only the state mechanics between the MC and receivers; it does not specify the internal algorithms employed by the MC to manage the Buddynet topology. However, several privately-implemented optimizations seem useful.

The first optimization is reconfiguring a Buddynet's topology in response to changing performance. If some paths in the network experience too much packet loss, the MC can reconfigure the network to reduce the loss. The Buddycast protocol doesn't specify algorithms for making these decisions, but we expect that all MCs will incorporate at least this enhancement, since it is very easy to implement.

A second optimization is choosing buddies based on their proximity to the downstream receiver. For example, a new receiver on a particular RFC-1918 private network (e.g., 10.0.0.0) could be assigned a buddy from the same physical network, based on a common public IP address, to avoid sending more than one stream over that private network's Internet connection. Similarly, a new receiver originating in a particular BGP Autonomous System (AS) could be assigned buddies from the same AS to avoid crossing a provider boundary, since intra-AS networks tend to be less congested than inter-AS ones. A yet higher order optimization could occur between geographic regions, such as continents.

Another optimization is measuring the actual outgoing available bandwidth of a relaying receiver rather than depending on a static estimate provided by the receiver during its initial negotiation. This could be done by prepending a bandwidth test stream before delivering the actual content stream, and evaluating the resulting statistics returned by on the relaying buddy's next poll. Knowing the actual available outgoing capacity at the start of a stream would give early warning of saturated bandwidth, letting the MC prune the buddy as a future relay.

A third optimization is for MCs to operate their own network of backup "buddy" MCs, so that a failing or saturated MC can share its load with, or fail over to, an affiliated MC. The Buddycast protocol does not specify this mechanism, since the best interoperability would likely be MCs running identical internal topology algorithms.

Specification Overview

The Buddycast protocol consists of several specifications:

- The .buddyc document. The .buddyc document is an xml-encoded summary of a stream program, including a textual abstract of the content, pointers to associated graphics and URLs, start and stop times, minimum bandwidth and receiver requirements, and meters indicating the current delivery state, such as elapsed time, byte, and packet counters.

- The content-selection string, passed as an argument to an MC in an http or https request. This string specifies the name of a content program and optional arguments, such as stream bandwidth selection, language, etc, that may be used to select between versions of a broadcast.

- The receiver/MC dialog, an XML description of the possible messages that can be exchanged between a receiver and the MC. The XML encodes both mandatory and optional attributes for each transaction; unimplemented optional attributes can be ignored by either party.

- The buddy/buddy dialog, an XML description of the possible messages that can be exchanged between two buddies. The XML encodes both mandatory and optional attributes for each transaction; unimplemented optional attributes can be ignored by either party.

- The UDP stream format, an encapsulation of an arbitrary byte stream that is initially encoded in the MC, is passed between receivers unchanged, and is decoded by each receiver to extract the content stream. This stream can be captured from off-the-shelf stream server software on the MC via localhost, and rerouted on each receiver to a corresponding stream player using the receiver's localhost. This approach may not work for all streaming media formats, but it will work for many.

By limiting the Buddycast specification to the minimum interactions required to propagate streams, the protocol makes it easy to invent new MC optimizations and features. XML encoding makes the protocol readily expandable while maintaining backwards compatibility. All Buddycast IP address elements support both IPv4 and IPv6, letting the protocol survive the transition to IPv6 whenever it occurs.

Sections to add:

Detailed Specifications. The XML schema, and packet formats, for each subspecification.

Internal Data Structures. An abstract description of recommended data structures for both receiver and MC. These are optional, but will help to achieve consistency in implementations.

State Mechanics Pseudocode. A pro-forma procedural description of all state mechanics.

Abuse Prevention. Anticipate the possible ways the protocol can be abused, both for DoS attacks and to hijack or alter broadcasts.

Reference Implementation. A working open-source implementation of Buddycast in Python is planned. There may be a reason to build it in another language, but the goal here is proof-of-concept. We leave clever and efficient implementation to the coding geek masses.

Performance Modeling. We expect to conduct formal simulations of Buddycast performance, to tweak the specs, see how well Buddynets scale, the effect Buddycast has on provider networks, and the value of various optimizations. Modeling results will be reported in other documents, but the highlights can be recapped here.

The RFC. At some point we have to distill this working document down to a text-based RFC and submit it for feedback. We should have a working reference implementation before doing this.