Terracast

Live Multicast on Wide Area IP Networks

Erik van Zijst

Spring 2005

Vrije Universiteit,

Faculty of Sciences, Department of Mathematics and Computer Science,

Amsterdam, The Netherlands

erik@marketxs.com

On two occasions I have been asked [by members of Parliament!], "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

-- Charles Babbage

CONTENTS

1. **Introduction and Problem Description**.....1
1.1 Multicast Routing.....2
1.2 Adequate Flow Control and Timely Delivery.....3
1.3 Guaranteed Delivery.....3

2. **Terracast Software Architecture**.....7
2.1 High-Level Overview.....7
2.2 Terracast Network Addresses.....8
2.3 Protocol Endpoints.....10
2.4 Router Packet Switching Core or Kernel.....11
2.5 Interface Modules.....12
2.6 Interceptor Pipelines.....13
2.7 Client Programming Library.....17

3. **Routing on an Overlay Network**.....19
3.1 Overlay Network Topologies.....19
3.1.1 Robustness.....19
3.1.2 Dynamic Routing Algorithm.....20
3.1.3 Shortest Path Routing.....20
3.1.4 Efficient Multicast Distribution.....21
3.1.5 Autonomous Overlay Topology Construction.....22
3.1.6 Conclusion.....23
3.2 Dynamic Routing.....24
3.2.1 Link-State Routing.....24
3.2.2 Distance-Vector Routing.....25
3.2.3 Distance-Vector Versus Link-State.....26
3.2.4 ExBF - The Extended Bellman-Ford Protocol.....26
3.3 Scalable ExBF Routing in Large Networks.....28
3.3.1 Terracast Address Summarization Protocol.....29
3.3.2 Summarization Opacity and Border Gateways.....33

4. **Multicast Routing**.....36
4.1 Introduction.....36
4.2 Algorithms for Multipoint Packet Distribution.....37
4.2.1 Reverse Path Flooding.....38
4.2.2 Reverse Path Broadcasting.....38
4.2.3 Reverse Path Multicasting.....39
4.2.4 Dense-Mode Vs Sparse-Mode.....40
4.2.5 Single-Source Vs Multi-Source Multicast.....40
4.2.6 Terracast's Multicast Algorithm.....42
4.2.7 Asymmetrical Link Characteristics.....46
4.3 Reliable Multicast.....46
4.3.1 XtremeCast.....47
4.3.2 PGM.....48
4.3.3 MTCP.....49
4.3.4 Digital Fountain.....49
4.3.5 Terracast's Reliable Multicast Algorithm.....50
4.4 Layered Multicast.....52
4.4.1 Previous Work in Layered Multicast.....54
4.4.2 Terracast's Layered Multicast Algorithm.....55
4.4.3 Restoring Global Packet Order.....58
4.4.4 Ignoring Packet Reordering.....60
4.4.5 Buffer Size Considerations.....61
4.4.6 Packet Prioritization and Multicast Congestion Control.....62

5. **Layering Real-Time Market Data**.....69
5.1 Encoding The Base Layer.....69
5.2 Encoding The Enhancement Layers.....71

6. **Related Work**.....75
6.1 Scattercast.....75
6.2 End System Multicast.....77

7. **Evaluation**.....79

8. **Conclusions**.....81

LIST OF FIGURES

1 A Teracast overlay network....7
2 Schematic view of a Terracast router daemon....11
3 State transition diagram for the interface module....13
4 Packet interceptors pipelines at interface level....14
5 Sequence diagrams for packet interceptors....16
6 Terracast's RPC-based client library....18
7 Finding the best overlay network topology....22
8 Backtracking paths in ExBF routing tables....27
9 Example network with hierarchical node addresses....30
10 Summarization opacity and stretch-factor....34
11 Applications of reverse path forwarding....39
12 The process of joining an inactive multicast group....43
13 Time complexity for multicast link failure....44
14 NAK-based reliable multicast....50
15 The process of selective packet recovery and reordering....58
16 Terracast congestion control algorithm....64
17 Prioritized packet queue....67
18 Data stream encoder for layered market data....72

LIST OF TABLES

1 Different communication protocols offered by Terracast.....10
2 Algorithm used by Terracast to summarize addresses.....33
3 Examples of summarized addresses.....34

ACKNOWLEDGMENTS

Terracast has been a big project. Both in terms of duration, as well as the amount of ground it covers. It was originally started as a small initiative at MarketXS to investigate new techniques that could further strengthen the company's ability to offer cheap, yet reliable data distribution channels. However, while I was learning more about multicast techniques, I became more enthusiastic and let the project expand in complexity.

First, I examined protocols such as plain IP multicast, which let to specialized protocols such as PGM that could offer more functionality. After I convinced myself that the company needed these features, my attention was drawn to networks with high tolerance to link failures. As the company makes extensive use of the public Internet for data dissemination, resilience is an important feature. This lead me to studying overlay networks that come with custom routing protocols. When I realized that multicasting real-time streaming data is difficult to combine with heterogeneous networks such as the Internet, I added layered multicast protocols and multicast flow control to the project's scope.

This complexity increased the project's risk, nevertheless CEO Floris Alkemade kept supporting and gave me time and resources to continue. I want to thank Floris for this, as Terracast could not have reached its current status without his support.

I would also like to thank Marc Visser who managed Terracast from a business perspective and looked for additional ways to use Terracast, in particular in the IPTV area.

Many people have directly or indirectly assisted the Terracast project. Specifically, I would like to thank Gill Waters and John Crawford of the University of Kent at Canterbury, Arno Bakker and Berry van Halderen of the Vrije Universiteit, Bart van Praag of the Universiteit van Amsterdam, and Paul Johnstone, Paul Hobkirk, Gordon Lai and Loh Wing Leong of Telerate, Paul Faas of HollandCentraal, as well as a large number of collegues at MarketXS, for providing shell accounts on geographically dispersed machines to test the routing algorithms in a realistic environment.

In particular I would like to thank Maarten van Steen for having supervised the project, despite his awfully busy agenda. During our occasional sessions at the university Maarten often put things back in perspective by always understanding immediately what had taken me weeks or months to figure out.

SUMMARY

This thesis describes an overlay network system called Terracast that offers an end-to-end solution for multicasting live data streams over the Internet. In contrast to most other research in the area of live multicast, Terracast does not focus on the dissemination of audio or video streams, but on real-time market data. Real-time market data and live video are both isochronous data streams that share many characteristics, but market data typically requires a more reliable network service. While a video stream is fairly tolerant to a limited amount of data loss, it can corrupt market data, rendering it obsolete or even dangerous to use. Terracast anticipates these needs for a high level of reliability by offering a resilient overlay network and a reliable multicast protocol. Like most other overlay networks that are intended to be deployed on the public Internet, Terracast's multicast distributions suffer from the heterogeneous nature and fluctuating capacity of the Internet. To be able to serve large groups of receivers with different Internet connections, while sustaining real-time throughput, Terracast employs layered multicast techniques.

Although Terracast was built from scratch, lots of protocols and algorithms it uses were taken from existing designs. Others were modified versions of existing protocols. Yet, Terracast also contains some unique features. In particular the way it combines layered data streams with reliable multicast was not seen in other work. Another unique component is a special encoder that prepares live market data for transmission as a layered multicast stream.

PROJECT BACKGROUND

The Terracast project was started almost two years ago as a research effort at MarketXS to see if it was possible to use the ever expanding, public Internet as an efficient, scalable backbone for distributing live market data across the globe. At the time MarketXS operated a centralized financial *ticker-plant* that was used by customers to obtain market data for use in their own backend systems or websites. A financial ticker-plant is a centralized system operated by a market data provider and used to process and distribute market data to clients. Data was accessed using a special API and exchanged over direct TCP connections. As the focus moved towards providing large volumes of live streaming market data to an increasing number of customers, MarketXS found itself distributing largely the same non-interactive content over the Internet to individual client applications using standard unicast TCP connections. The cost of both the required processing power at the distribution part of the ticker plant, as well as the necessary Internet bandwidth, increased linearly with the number of connected clients.

This is different to how larger competitors handle data distribution. Coming from a less globally wired past, the major competitors have traditionally used satellite broadcast systems to get non-interactive live content to their customers. Although this comes with fairly high initial costs, it scales practically infinitely with the number of clients.

Being founded at the end of the Internet hype, MarketXS has relied on the Internet as a flexible and cheap communications medium which appeared to be suitable for all but the most demanding institutions. However, as the amount of data and the number of clients grew, the need for a more scalable distribution mechanism emerged. Satellite was not favored, as that would invalidate the service's selling point that no client side hardware was required. Ideally, market data would be published to the Internet just once with a fixed cost, where it would be multicast to all paying customers. Unfortunately, since the Internet does not allow for such distribution natively, a virtual network was designed to reduce the amount of required bandwidth at the ticker plant without abandoning the Internet.

Because the Internet offers a best-effort delivery service of which the quality and throughput can greatly vary, the new virtual distribution network should be very robust with no single points of failure and should be able to constantly deliver its data with minimal delay, regardless of network congestion.

1. INTRODUCTION AND PROBLEM DESCRIPTION

The Internet has been tremendously popular and has seen phenomenal growth during its few decades of existence. Especially since the birth of the World Wide Web the amount of online content has been spectacular and steadily growing. However, despite its versatility in digital communication, interoperability and internationally accepted communication protocols, its fundamental design has not changed much since its conception and it is easy to forget that the system does not excel in everything. Watching live TV, for example, is something which has still not happened over the Internet, even though television has been around almost twice as long as the Internet Protocol and represents a huge market. The technical reasons for this are fairly fundamental. This thesis will study the Internet's shortcomings regarding live content and in particular content that is to be delivered to thousands or more subscribers simultaneously. Also, having identified the problematic areas, it will introduce a software-based system that can run on top of the existing, unmodified Internet as a virtual network and alleviates some of the issues, making the network more suitable for live data distribution to large audiences.

The Internet is a packet-switched network where data is exchanged in small units or packets that are independently transported over the network. An important strength of packet-switching is that it allows for very flexible use of the physical network wires. When two communicating parties have no data to exchange for a certain period of time, no packets are sent and the wires can carry packets from other parties. More conventional systems such as the traditional telephone system, most cable television networks and radio are circuit-switched. They reserve a fixed amount of capacity or bandwidth for each communication session, regardless of the channel's use. This fundamental difference has important consequences for the type of applications they support best. The cable television network for example is perfect for delivering a live video stream of constant quality and without noticeable delay because the necessary bandwidth is reserved on the network and can never be used by other streams. This puts a hard limit on the total number of concurrent streams. On the Internet bandwidth is not reserved, but available to, and shared by, everyone. The consequence is that it cannot guarantee a minimum amount of end-to-end bandwidth, often causing live video streams to appear jerky and frames to be skipped.

Distributing real-time market data over the Internet to many concurrent receivers requires the network to consistently offer a substantial amount of bandwidth. Unfortunately the

properties of the shared communications medium cannot guarantee such sustained performance and when the medium fails to meet the throughput requirements either temporarily or permanently, the service will suffer from delayed delivery or miss packets all together.

The most popular communication protocol over the Internet is the reliable TCP protocol. However, since this protocol only supports point-to-point connections, a unique connection needs to be set-up from the source to each individual receiver. Not only does this waste bandwidth as each connection carries the same market data stream, the parallel unicast connections also have to compete for bandwidth, which can lead to permanent congestion and impact the timely delivery for all receivers.

The use of an Internet multicast protocol can offer substantial improvements over the use of parallel unicast TCP connections. Multicast subscription and routing protocols such as DVMRP [WAI88] and PIM [ADAM04], [EST98] allow for data packets to be sent to the network once while the network itself clones the packets as they are routed to each subscribed receiver, making sure each participating network link carries only a single copy of each packet. Although this minimizes the bandwidth requirements, IP multicast, in contrast to TCP, does not offer guaranteed, nor in-order delivery. Hence, the use of IP multicast services does not necessarily make the Internet or any other wide-area, shared IP network suitable for distribution of real-time market data.

Following from these requirements are the features a packet-switched network will need to support in order to be a useable transport medium for dissemination of real-time, non-interactive content to large audiences.

1.1 Multicast Routing

At the very least, the network needs to support one-to-many communication so that it can send data packets from a data source to more than one receiver, ideally without putting extra stress on the network or source when the number of receivers increases. Multicast routing can be offered in various ways. The most common way is to let the receivers tell the network, but not necessarily the source, which streams they want to receive and let the network compute data distribution paths that deliver the right packets to each receiver. Multicasting can also be done by letting the source encode the list of receivers in each data packet, thereby freeing the network from the potentially computationally intensive task of maintaining multicast distribution paths. Although this approach usually does not scale to large audiences, applications such as Mobile

Ad Hoc Networks or MANETs [JI03] frequently use this and one approach has even been patented [MAR04]. A third approach places all logic at the receivers by letting the network apply a broadcast mechanism whereby each packet is delivered to every connected node and the receivers filter out only those packets that are interesting. Although this can be wasteful, its simplicity can still make it attractive especially in situations where bandwidth is abundant. Even though they do not use packet-switching, conventional radio and cable television do exactly this.

1.2 Adequate Flow Control and Timely Delivery

Because non interactive live data streams do not actively anticipate network congestion the way TCP does, the network will need to manage the available bandwidth in a way that allows for fair or equal division among the streams. Without this, high volume streams can consume a large capacity percentage of overloaded links and depending on the router scheduling policies, cause less active streams to suffer delayed delivery or packet loss. An alternative to letting the network components handle the flow control and congestion is to put the responsibility at the source and receivers. If this makes the streams well-behaved, network resources will be implicitly divided fairly, similar to when only TCP connections are used.

Shifting responsibility for flow control management from the network components to the communicating peers usually requires a form of feedback information from the network or the receivers. In this case it is important that the amount of feedback does not grow linearly with the size of the audience, as that would reduce the scalability of the multicast. Even when a scalable form of feedback information can be realised and the data stream adapts its transmission rate according to the network conditions, the problem remains that live streams often lose their value when they are slowed down and delivered late.

1.3 Guaranteed Delivery

It would be ideal if every subscriber would receive the data stream without loss or corruption. However, when the content is live and cannot be slowed down, while the network has limited capacity, packet loss is difficult to avoid. Even when there is sufficient bandwidth at all times, store-and-forward packet-switched networks usually do not guarantee the delivery of all packets.

The main reason for packet loss is congestion, but additionally packets can get lost when a router goes down that has unprocessed packets in its buffers. In networks that use dynamic routing, packets can also be dropped when the routing forwarding rules change.

In cases where packets are accidentally lost, an end-to-end mechanism for retransmissions may be applied that can compensate the loss, similar to retransmissions in TCP. However, since this requires a form of feedback information, it is again important for reasons of scalability that the overhead involved with retransmissions is not linearly related to the size of the audience. For TCP connections this is no issue, as they only support a single receiver.

Fortunately, end-to-end transmission feedback can be avoided in at least two ways. First, it is possible to let the network components keep a copy of the most recently forwarded packets and let them participate in retransmissions by intercepting the retransmission requests and servicing them locally. This approach has well-known applications in [SPEAK01] and [CHIU98], but may lead to aggressive storage and increased processing power requirements at the network components.

The second alternative to end-to-end retransmission requests is that of encoding redundant information in the data packets. If enough redundancy is encoded, a lost packet's content can be entirely recovered from the extra information in the other packets. This technique is commonly know as Forward Error Correction and is applied in several systems and the subject of various RFC's including [LUB02] and [VIC02]. An interesting commercial application is offered by [BY02] whose FEC implementation encodes content in a way that allows clients to reconstruct the original data from any combination of received packets, equal in length to the original data. This protocol was later commercialized under the name Digital Fountain [DF04]. The downside of FEC is that it comes with a constant level of bandwidth overhead that is related to the level of packet loss tolerance, regardless of whether packets are actually lost.

Whichever approach to packet loss is taken, all will fail in case of a live data stream that incessantly produces more bytes than the network can forward. When retransmission requests are used, the problem will even be exacerbated as the retransmission requests use extra bandwidth, causing more data packets to be lost, leading to even more retransmission requests.

Having identified the requirements for live multicasts over packet-switched networks, it is understandable why the largest publicly available network, the Internet, is less suited for this. IP natively recognizes multicast packets by their class-D destination IP address, but actual multicast routing is disabled on a large part of the network, making it unavailable to almost all

end users. As of 1992 an application-based overlay solution to IP multicast exists in the form of the MBone experiment [SAV96] that uses software based multicast routers to allow native multicast network segments to be connected to each other, even while the global carrier networks that physically connect those network segments do not support multicast. While the MBone is over 12 years old and still in use at the time of writing, its penetration is limited mainly to academic institutions and larger ISPs. Hence, native multicast routing is not widely available. However, even with a connection to the MBone, the issues of flow control and guaranteed delivery remain unsolved, seriously limiting the use of multicast traffic to a relatively small number of applications and content.

Given the requirements for live multicast and the lack of support for them by the Internet, it is concluded that the problem of large-scale multicast is better dealt with in the application layer, rather than on the network layer. This leads to the popular research area of application layer multicast where a virtual network is constructed from software based router nodes that employ their own routing algorithms and congestion control mechanisms to support efficient multicast distribution. Over the last decade or so, research in this area has lead to an impressive number of application layer multicast solutions. A modest and incomplete list of solutions includes Scattercast [CHA00], Narada [CHU00], NICE [BAN02], Bayeaux [ZHU01] and Overcast [JAN00]. The fact that new initiatives continue to emerge illustrates the difficulty of solving multicast on packet-switched networks in a generic way.

This text introduces yet another application layer multicast solution called *Terracast.* Although Terracast has many similarities with existing solutions, its specific focus on scalability and real-time data lead to a design that distinguishes itself from related work on a number of important points. As discussed in the project's background, Terracast was designed to offer an end-to-end solution for efficiently multicasting live stock market data to potentially anyone connected to the Internet. The fact that live market data is in many ways more demanding and much less tolerant to data loss than audio and video means that aside from multicast routing and flow control, Terracast must be capable of giving certain hard guarantees over what is delivered to receivers. If packets must be dropped due to congestion or other irrecoverable problems, it is crucial that this is done in a fully deterministic way that does not corrupt the financial data. Where a viewer of a film may accept the random loss of one or two video frames, this type of behaviour can wreak havoc in financial data where the missed packet could contain an important trade.

The need for Terracast to be able to deterministically deliver only certain parts of a data stream when the network lacks sufficient capacity lead to the research area of layered multicast. With layered multicast, the packets of a live data stream are sent as individual streams. This allows receivers to subscribe to only those layers that the network can handle so that random packet loss can largely be avoided. For Terracast, an enhanced form of layered multicast is proposed that guarantees complete delivery for certain layers to avoid random loss altogether, making it suitable for market data.

Two core activities of Terracast can be isolated. The first is that of running and managing a robust and scalable overlay network that uses its own routing algorithms and supports multicast. The other core activity is that of managing flow control and congestion when live streams overload the network and ensuring layered multicast can be offered with guarantees.

2. TERRACAST SOFTWARE ARCHITECTURE

This chapter will describe the software architecture of the Terracast router daemons and is organized as follows. Section 2.1. starts with a high-level overview of the software and introduces the different components. Section 2.2. explains addressing on the Terracast overlay network, while section 2.3. describes the various communication protocols a Terracast router daemon offers to its connected user applications. Among others, these include best-effort datagram unicast packets, best-effort multicast packet distribution and reliable, layered multicast distribution services. Section 2.4. will focus on the router daemon's core: its kernel. The kernel is the central part of each router and participates in all communication. Section 2.5. describes how the router's kernel is able to send and receive data packets from the overlay network through its interface modules, while section 2.6. offers a closer look at the core processing components in the interface modules: its packet interceptor pipelines. It shows how interceptors increase the flexibility of the overlay routers. Section 2.7. explains how user applications can program against the Terracast router through a client-side library.

2.1 High-Level Overview

The Terracast network is an overlay network consisting of many software router daemons that maintain long-lived TCP connections among each other. This is depicted in illustration 1. This illustration shows a Terracast overlay network of four software router daemons, connected by four TCP connections. Which connections each router will engage in is configured at router startup, making the network's topology somewhat static, as changing the overlay topology requires configuration changes to at least two router daemons.

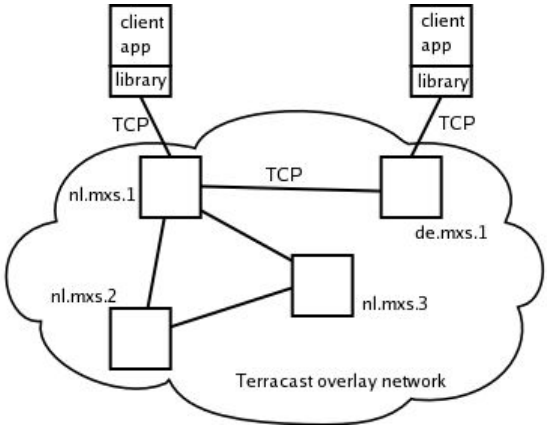


Illustration 1: A Terracast network is formed by software router daemons, interconnected by long-lived TCP connections.

The illustration also shows two connected user applications. User applications can use the overlay network for communication by programming against the Terracast client-side library which uses an RPC mechanism to relay the user's command and data packets to a single router instance. Router daemons and user applications have a one-to-many relationship where a single router can serve more than one user application at a time, but a user application can only be connected to one router.

2.2 Terracast Network Addresses

On Terracast, data packets can be addressed to a single communication endpoint, or to abstract multicast addresses. This section describes the role and representation of Terracast network addresses and introduces the concept of a *tport*.

Single communication endpoints are network addresses used by user applications. Access to them is exclusive, meaning that only one user application can use them for sending and receiving packets at a time. In that respect an analogy can be made between Terracast network addresses and those used on IP networks. An IP network address as used by user applications is the combination of a host IP address and a TCP or UDP port number. Together they form an address that is unique throughout the entire network[1]. They are suitable for sending packets to exactly one receiver.

Similar to IP, a unique network address on Terracast that can be used by a user application is the combination of the logical node address assigned to the overlay router and the port name chosen by the user application. If an application that is connected to a Terracast router with the logical node address "`nl.mxs.1`" wants to use port "`myport`" for receiving unicast data packets, the fully qualified network address becomes "`nl.mxs.1:myport`". Other applications connected to the Terracast overlay network that want to send packets to this application, will use this as destination address. To differentiate between ports in IP an Terracast networks, the remainder of this thesis will refer to Terracast port names as *tports*, rather than ports.

Just as with TCP ports, Terracast network addresses must be bound, prior to sending or receiving packets. When a data packet is sent from an endpoint address, it will contain this information as its source, allowing other routers or the receiving application to send a response.

1 Note that when IP masquerading or Network Address Translation (NAT) is used on IP, it becomes possible to share the same, masqueraded, network address at multiple locations.

Aside from network addresses that are bound and used by a single receiver application, Terracast allows packets to be sent to abstract multipoint destinations, or multicast addresses. A Terracast multicast address is a destination that everyone in the network can subscribe to. The network's software routers will make sure a copy of each data packet published to this multicast address is delivered to every user application that subscribed to it. Chapter 4 will discuss the algorithms used to keep track of subscribed receivers and multicast routing. At this point it is sufficient to say that multipoint destinations are *single-sourced.* This means that only one user application can publish data packets to a multicast address, making Terracast suitable for one-to-many, but not for many-to-many communication.

The single source restriction is a result of the way multicast addresses are formed. Just as a unicast address, multicast addresses consist of both the node address of a router daemon and an available tport chosen by the user application that wants to publish. Hence, "`nl.mxs.1:mygroup`" could be a valid multicast address, used by the user application that bound the tport "`mygroup`" while being connected to router "`nl.mxs.1`". The node part of a multicast destination address is that of the router daemon used by source application, while the second part is that of the tport that the application chose.

Because only the tport section of a multicast address can be freely chosen (as long as it is not already in use by another application that is also connected to the same router daemon), each multicast address explicitly contains the source's location on the overlay network. While this makes multicast communication less flexible because publishing is not anonymous, it greatly simplifies subscription management. This will be covered extensively in paragraph 4.2.6.

Both unicast and multicast addresses are syntactically equal. The address "`nl.mxs.1:timeservice`" could be a unicast addresses used and bound by an application that provides the local date and time in response to any data packet it receives, but it could also be a multicast addresses that anyone can subscribe to, to receive periodic date and time broadcasts from the user application that bound this address as a multicast address. Because of this syntactic equivalence, each packet contains a flag that indicates whether its destination address should be interpreted as a unicast or multicast address.

2.3 Protocol Endpoints

The TCP/IP protocol suite comes with different protocol implementations that run on top of the packet-oriented IP base protocol. These are UDP and TCP. Their main difference is in the fact that TCP is connection oriented and reliable, while UDP offers a best-effort, datagram oriented service. Terracast takes a similar approach where several higher layer protocols are implemented as services over the packet-oriented base layer. In Terracast these are referred to as protocol endpoints.

The current version of the platform comes with five standard protocol endpoints. Two offer unicast communication, while three provide various ways to do one-to-many data distribution. They are briefly summarized in table 1.

Protocol	***Description***
UUP	Unreliable Unicast Protocol. Offers best-effort unicast datagram services to user applications. This protocol is somewhat comparable in functionality to UDP.
RUP	Reliable Unicast Protocol. Offers reliable unicast communication between peers. The protocol is connection-oriented and functionally comparable to TCP. A large part of its state diagram was taken from TCP.
UMP	Unreliable Multicast Protocol Offers best-effort multicast datagram services to user applications on the Terracast overlay network. It is the functional equivalent of UDP multicast packets on IP networks. It does not guarantee packet delivery, nor in-order reception.
OLMP	Ordered Layered Multicast Protocol Offers multicast communication with receiver-driven rate control. Complete delivery is not guaranteed, but the packets that are received are guaranteed to be in their original order. This protocol has no equivalent in IP and is connection-oriented.

Protocol	*Description*
ROLMP	Reliable Ordered Layered Multicast Protocol Offers reliable multicast communication with receiver-driven rate control. Stream layering allows each subscriber to receive the data stream in the highest possible quality, while the source never has to slow down. This protocol has no equivalent in IP and is connection-oriented. Both OLMP and ROLMP were designed for real-time data distribution.

Table 1: The Terracast overlay network offers five different communication protocols to its user applications. Two offer unicast, while three provide different types of multicast communication.

2.4 Router Packet Switching Core or Kernel

The heart of each Terracast software router daemon is the packet switching core, or kernel. With the exception of a few specialized control packets, every packet that is received either from a static TCP connection to a neighbor router, or from a connected user application, passes through the switching core. Its task is to inspect the destination of each packet and use its routing tables to determine where to send it.

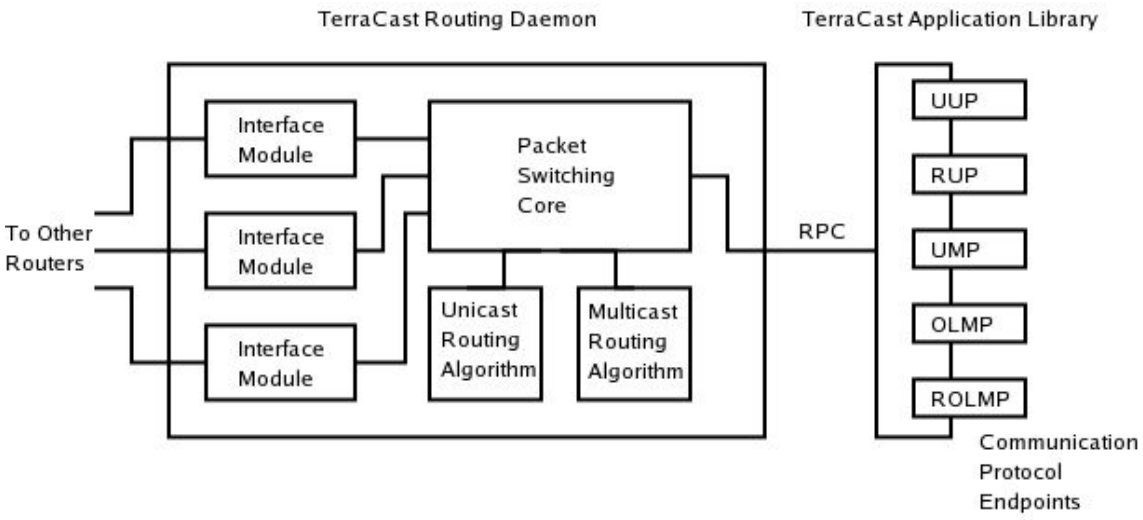


Illustration 2: Schematic view of a Terracast software router daemon. At the heart of the router is the packet switching core that is connected to the local user applications on one side and to the interface modules on the other.

The kernel's work runs in its own thread and is event driven. It remains idle until it is notified either by an interface module that maintains the long-lived TCP connection with a neighbor router, or by a user application that is sending a packet. Because the kernel uses only a single thread, packet switching in Terracast is a serial process that will not benefit from multiprocessor hardware.

Illustration 2 shows where in the router daemon the switching core is located and which other components it communicates with. It clearly shows the central position it has and how it uses its unicast and multicast packet routing modules when processing packets.

2.5 Interface Modules

Each router daemon in the network is connected to one or more neighbors. This is done by establishing long-lived TCP connection between each other. A router daemon runs one interface module instance for each configured neighbor. The responsibility of an interface is to establish the TCP connection and to pass packets from the kernel to the TCP connection and vice versa. Packets of the first kind are referred to as outbound packets, while the latter are inbound packets.

For establishing the TCP connection to its neighbor, the interface module implements a simple algorithm. It first tries to connect to the configured neighbor router by actively trying to connect to its TCP/IP network address. If the connection is aborted with a connection refused or other error, it is assumed that the neighbor is not yet running, and the interface module starts listening on its configured IP port so that the neighbor can connect to it as soon as it is started up.

The interface module will only wait for an incoming connection request for a brief period of time. After that, it goes back into the state of actively connecting to the peer. To avoid the situation where both peers continue to switch states exactly at the same time, the duration of the listening state is influenced by a random factor. Although this does not guarantee that the algorithm terminates, it works well in practice. The interface state transition diagram is depicted in illustration 3.

The advantage of letting each peer switch between the roles of both client and server when establishing a connection, makes it possible to connect routers even when one of them is on a masqueraded IP network and is therefore unable to accept incoming TCP connections. Of course

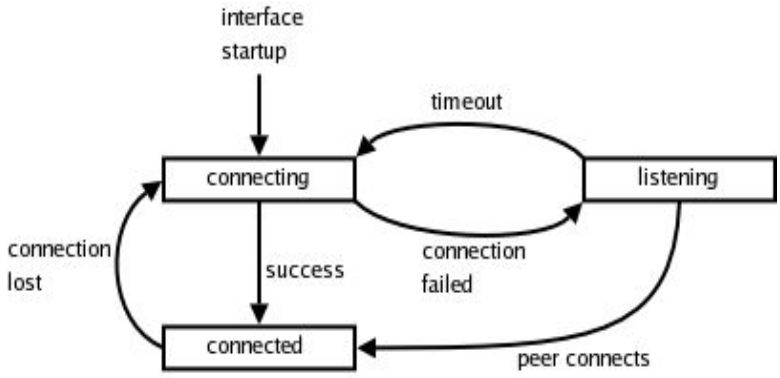


Illustration 3: State transition diagram for the interface module.

it is not possible to establish a connection between two routers that are both unable to accept incoming connections.

Interface module instances can be configured at router deployment time through configuration files. This will make the router daemon automatically create the specified interfaces at startup. If the configured neighbor routers are online, all long-lived TCP connections will automatically be established. It is also possible to add new neighbor connections and interface module instances at runtime. This way the overlay network topology can be changed flexibly and new routers can be added to the network.

When an interface module manages to establish its connection with the neighbor router, they exchange their Terracast node addresses to inform each other of their presence. When the interface receives the node address of its peer, it passes this information, together with the announcement that the connection was established, on to the router's switching core. This information allows the kernel's routing algorithms to build dynamic routing tables. How routing is done inside the Terracast router daemons is discussed in chapter 3.

2.6 Interceptor Pipelines

The previous section explained that the role of the interface modules is to establish the connection with a configured neighbor router and to send data packets from the router switching core to the neighbor connection and vice versa. However, the interface module comes with a framework that allows custom software plug-ins to influence the stream of packets that flows between the network and the router's kernel. This mechanism is referred to as the interceptor pipeline. Each software plug-in component that needs to control the packet flow is a class that

implements a simple programming interface. This interface allows interceptor instances to be chained together, forming a packet processing pipeline. The contract of an interceptor is that it receives a packet, applies its operations and then passes the modified packet to the next interceptor in the chain.

The interceptor pipeline sits between the router switching core and the network connection. When the router kernel delivers a packet to the interface module for transmission to the neighbor router, the interface module runs the packet through the interceptor pipeline, giving the interceptors the chance to modify the packet. Each packet that comes out of the pipeline is transmitted to the neighbor router.

Each interface module has two interceptor pipelines. One is used to process outbound packets, while the other is used for inbound packets. These pipelines are independent of one another, meaning that not only the ordering of interceptors, but also the number of processing steps can be different for inbound and outbound packets. Also, because each interface module operates its own interceptor pipelines, they can be configured uniquely. This is illustrated in figure 4.

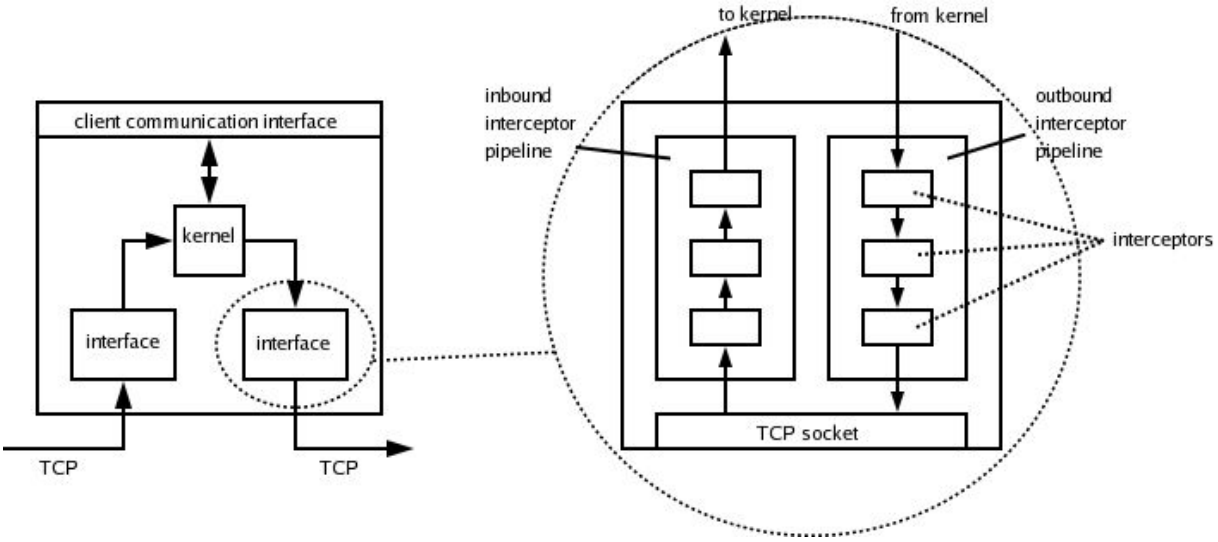


Illustration 4: A Terracast router daemon as shown on the left consists of a switching core or kernel and one or more interface modules that connect to neighbor routers. Using the unicast routing table, packets are forwarded from one interface to the other. The right hand side of the illustration shows the internals of an interface module. It shows how inbound and outbound packets are processed in separate interceptor pipelines.

An example of an interceptor is one that filters packets coming from a specific Terracast router. When this interceptor is implemented as a manageable component that can be configured at runtime, it can be used to implement basic firewall functionality on the overlay network. When it receives a packet that matches its rejection pattern, it discards the packet by not passing it to the next interceptor in the pipeline. Another typical interceptor is a traffic monitor that counts the size of each packet that passes by and uses this to log traffic activity and compute bandwidth statistics.

The use of interceptor pipelines in the interface modules is a powerful mechanism that was somewhat inspired by the concept of aspect oriented programming. AOP allows the functionality of objects to be extended dynamically [KIC97]. This is different from OOP that takes a static approach through inheritance. With this mechanism a Terracast router can be extended with additional functionality without modifications to the software.

As well as offering flexible extensibility, the interceptor pipelines have another important, secondary responsibility: they act as a packet buffer between the router kernel and the network. It was shown that the router's switching core runs its own thread and is notified by the interface modules when a new packet was received over the network. The kernel thread is woken up, reads the packet from the interface module and processes it. If the kernel decides that the packet must be sent out through another interface, it passes it to that interface and waits for the next notification. This sequence requires that passing a packet to an interface module is a non-blocking operation. However, writing data to the long-lived TCP connection can block when the TCP send-window is full. Therefore, the interface module must be able to temporarily buffer outbound packets to guarantee that the kernel thread is never blocked when it delivers a packet to the interface. Additionally, the interface module needs its own thread that continuously dequeues and serializes packets from the buffer and writes them to the TCP connection.

Thus, the secondary task of the interceptor pipeline is to provide that temporary packet buffer and offer the necessary inter-thread communication between the kernel thread and the interface thread that writes to the blocking TCP socket. To this end, interceptors are divided in two categories: those whose intercept method can block, and those that always return immediately.

The first category is referred to as blocking or synchronous interceptors. They use the calling thread to do their packet processing, as well as the delegation of the packet to the next interceptor. Control will not be returned to the caller until the packet was successfully delivered to the next interceptor. While the synchronous operation keeps the complexity of this interceptor low, the disadvantage is that it can keep the caller blocked.

Illustration 5a shows the use of blocking interceptors in the outbound packet pipeline of an interface module. It shows the kernel passing a packet to the interceptor pipeline. The pipeline in this illustration contains only a single interceptor and it terminated by the “socket interceptor”. This is the pipeline's endpoint which does implement the interceptor programming interface, but does not delegate its packets to another interceptor. Instead, it writes them to the interface module's TCP socket. The illustration shows that the control is only returned to the router's kernel after the packet was entirely written to the TCP socket. How long a write operation on a TCP socket will block is unpredictable and has no upper limit. When the peer does not read bytes from the connection, the TCP send-window will fill up entirely, causing the operating system's kernel to block further write operations to the socket.

To avoid situations where the router's kernel is blocked for an arbitrarily long time, every interceptor pipeline should contain one non-blocking or asynchronous interceptor. A non-blocking interceptor guarantees immediate return of control to the caller by storing the packet in

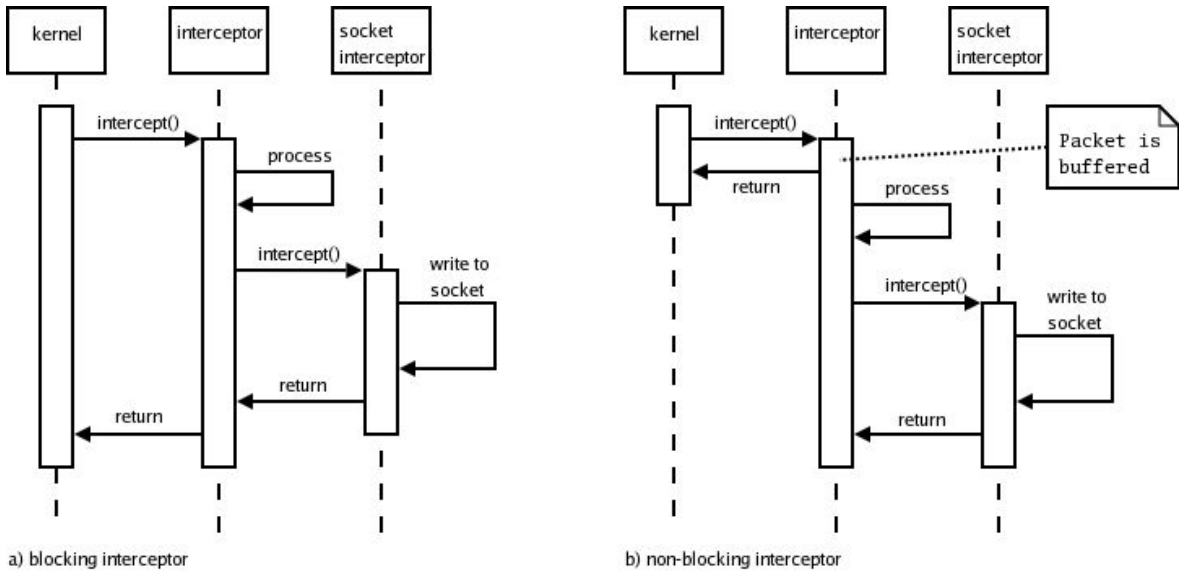


Illustration 5: Interceptor sequence diagrams. The left diagram shows the sequence diagram of a blocking interceptor. Because it uses the calling thread to do the packet processing as well as the delegation, the caller can be blocked. The diagram on the right shows how a non-blocking interceptor immediately returns control to the caller.

an internal buffer. These interceptors normally run their own thread to do the packet processing and delegation independent of the caller. This is depicted in illustration 5b.

A problem with blocking interceptors is that their packet buffer may grow indefinitely when the kernel delivers packets faster than the interface module's TCP connection can transmit. This is problematic as it requires infinite storage capacity. Also, it adds significant delay to the queued packets. Paragraph 4.4.6 will discuss several ways to handle situations where packets are delivered faster than they can be processed. At this point it is sufficient to say that each non-blocking buffer interceptor will discard packets when the size of its buffer exceeds a certain threshold.

Normally, an interceptor pipeline consists of zero or more blocking interceptors and one non-blocking interceptor. Because the non-blocking interceptor has a packet buffer that may discard packets, interceptors that measure actual traffic throughput would typically be situated after the non-blocking interceptor, while a firewall interceptor would usually sit in front of it.

2.7 Client Programming Library

The Terracast overlay network is accessible to user applications through the use of the client-side programming library. This library was already depicted in illustration 2 on page 11 and offers a local programming interface to applications.
The library connects to a router daemon at application startup and communicates with it using RPC. While the router daemon only provides basic functions for sending and receiving packets and for binding tports, the client-side library offers much richer functionality through the Terracast protocol endpoints that were introduced in section 2.3.

The RPC communication between the library and the router daemon is driven by the client. This means that only the client invokes functions in the router, but not vice versa. When the router's switching core receives packets addressed to a tport that is bound by the client application, it stores them until the client actively picks them up. Receiving packets is realized by letting the client continuously poll the router for packets. To minimize the overhead of the RPC polling mechanism, the router's poll function does not return until there is at least one packet delivered by the switching core. When more than one packet is waiting, the poll function will return them all as soon as it is invoked.

The reason behind the polling nature of the client library is network related. An alternative would have been to let the client library expose a notification function for receiving packets.

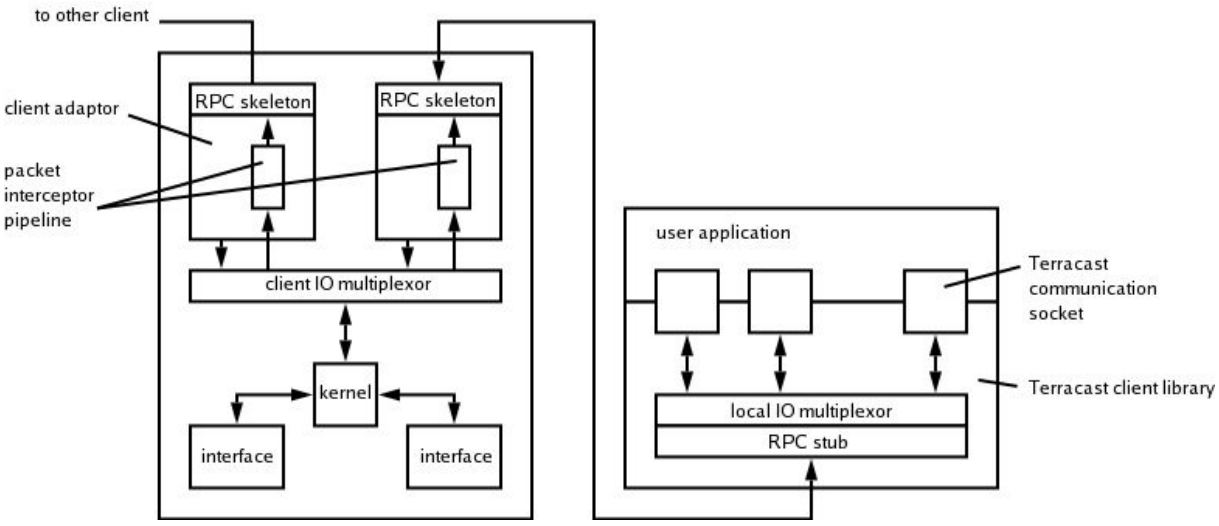


Illustration 6: User applications use the Terracast network through RPC using a client library. For each connected client, the router runs a client adaptor. When packets arrive addressed to a connected client, the router's client IO multiplexor passes it to the corresponding client adaptor where the packet is run through a packet interceptor pipeline and temporarily stored in a non-blocking buffer interceptor until the client library actively fetches the pending packets.

The client would then send a marshaled stub to the router daemon when it connects. This stub can later be invoked by the router daemon when packets are received for the client. In most RPC implementations the service provider's object skeleton listens on a TCP port for incoming invocations. Because a Terracast router daemon will not always be running on a client's local machine, it connects to a nearby router daemon that is willing to accept client connections. It is possible that this leads to a client application that runs on a masqueraded IP network segment, but connects to a router daemon that is not on this segment. In this situation, the nature of IP masquerading prevents the router daemon from initiating TCP connections to the client. Giving the router daemon a stub for delivering packets will therefore not work. Because it is difficult for the client to detect the difference between this problem and the situation where the router simply has no packets to deliver, Terracast avoids it by letting the client take the initiative in all communication between router and client.

The router daemon uses a packet buffer to temporarily store packets for each connected client application until they are picked up. To realize this, the router uses an interceptor pipeline similar to those used in the interface modules. This is illustrated in figure 6.

3. ROUTING ON AN OVERLAY NETWORK

This chapter discusses some of the properties overlay network topologies. In section 3.1. it will be shown that the topology chosen for the network influences performance and resilience and how a proper topology can be found. It will also show that the overlay network requires an adaptive shortest path routing algorithm to meet the Terracast requirements.

Section 3.2. zooms in on this routing algorithm. It explains the required properties and compares these to existing routing algorithms. When an existing algorithm is selected, it will be extended with functionality that allows it to scale to large networks.

3.1 Overlay Network Topologies

The previous chapter explained that the Terracast router daemons form an overlay network. The topology of this network is defined by which router daemons are configured to be neighbors by sharing a TCP connection. Changing the neighbor configuration changes the overlay topology. This makes the overlay topology independent of the topology of the underlying IP network.

3.1.1 Robustness

Changing the topology of the overlay network by adding or removing links (edges) is easy and cheap, however these changes have impact on the network's performance and robustness. For example, an overlay network with a minimum number of edges, where the nodes are connected in a long chain, is vulnerable to node failure. If any node other than the chain's head or tail fail, the overlay network becomes partitioned. Hence, to make an overlay network robust, it should have enough redundant edges to survive node failure. On a network with redundant links, there is more than one path (sequence of edges) between two nodes.

When redundant links are required to make the overlay network robust and when adding links is cheap because it only involves configuration changes, an overlay network could be configured as a fully connected graph where each node is a direct neighbor of all other nodes.
This may be a possibility for some networks, but it is impractical on networks where new nodes frequently join and leave. Also, on large networks with thousands of nodes, maintaining a fully

connected graph may be prohibitory expensive for the operating systems used on the machines. Because Terracast must be able to support networks of more than thousands of nodes, a fully connected graph is not a suitable overlay topology.

3.1.2 Dynamic Routing Algorithm

In a robust overlay network that is not configured as a fully connected graph, the nodes do not have a direct TCP connection to every other node. Therefore they need routing information to decide how to reach nodes that are not directly connected to them. In simple overlay networks, this information could be a statically configured table that says which TCP connection should be used to reach each individual node in the network. In a robust network however, that is able to survive the crash of one of more nodes, the information in this routing table must be dynamic. When a TCP connection is lost because the neighbor crashes, the table must automatically find an alternative route for all destination addresses that became unreachable after the crash.

What is necessary in a large and robust overlay network is a routing algorithm that has enough knowledge of the overlay topology to select an alternative path when a destination becomes unreachable after the crash of a neighbor.

When it is important that data packets are routed through the network as quickly as possible, it would be beneficial if the routing algorithm is able to always find the TCP connection that offer the fastest path to the destination. Because many existing routing algorithms search for the fastest or shortest path to each destination and because Terracast targets the distribution of real-time data, Terracast uses a shortest path routing algorithm to maintain its routing tables.

3.1.3 Shortest Path Routing

In many algorithms, the definition of the shortest path between any two nodes is the path that contains the least number of edges. This technique is referred to as minimum hop routing. In networks where not all edges offer the same capacity and performance, minimum hop routing is inadequate. To cater for this, more sophisticated routing algorithms measure the actual propagation time, or end-to-end delay of each edge and use this to find the sequence of edges between any two nodes that offers the least total end-to-end delay.

The use of the latter technique, rather than minimum hop routing, is particularly useful on overlay networks where the capacity of the edges can greatly vary and even fluctuate over time. The TCP connections of an overlay network must compete for bandwidth with all other networked applications, so the actual performance of any overlay edge changes all the time.

Because of this dynamic nature of overlay edges, Terracast uses a shortest path algorithm, based on minimum delay routing, that constantly measures the capacity of each edge and adjusts the routing tables accordingly.

3.1.4 Efficient Multicast Distribution

Although multicasting is discussed in detail in later chapters, it is briefly introduced here. The reason for this is that efficient multicasting on an overlay network puts some constraints on the overlay topology. To achieve optimal performance, the overlay topology should match the topology of the underlying IP network as closely as possible. This is illustrated in figure 7 which shows how a mismatch between physical (IP) and overlay topology results in poor overlay performance for multicast distribution.

The illustration shows five locations (buildings or departments) of a company network that are connected by five physical wide-area links. This physical topology is illustrated in figure 7c. An overlay network is now run on top of this company network and an overlay router daemon is deployed in each department. The router daemons are connected according to the overlay topology depicted in 7a.

With the physical network topology of figure 7c and the overlay topology of 7a on top of it, it is possible that the shortest path routing algorithm on the router daemon in department A measures that the other nodes in the overlay network are best reached through the paths that are represented by the dotted arrows of figure 7b. Hence, data packets addressed to nodes B, D and E are sent directly over A's TCP connections, while packets for node C are sent via B.

If at this point it is assumed that the multicast distribution tree is derived from the shortest path information in the routing tables (as many multicast routing algorithms including Terracast do), it is likely that multicast traffic from A to all other nodes follows the dotted arrows of figure 7b. If all nodes are subscribed to the multicast session, A transmits three copies of each data packet: one packet to each of its neighbors. Node B sends a copy of each packet it receives to C.

Because the physical topology does not have a direct link between departments A and E, the TCP connection between the overlay router daemons at A and E runs via department B. Department A only has a single physical link, so all three copies of a multicast data packet will

traverse this link. This is illustrated in 7d. This figure shows that the physical link between A and B carries three copies of the same multicast data packet and the physical link connecting B and E carries the same packet twice.

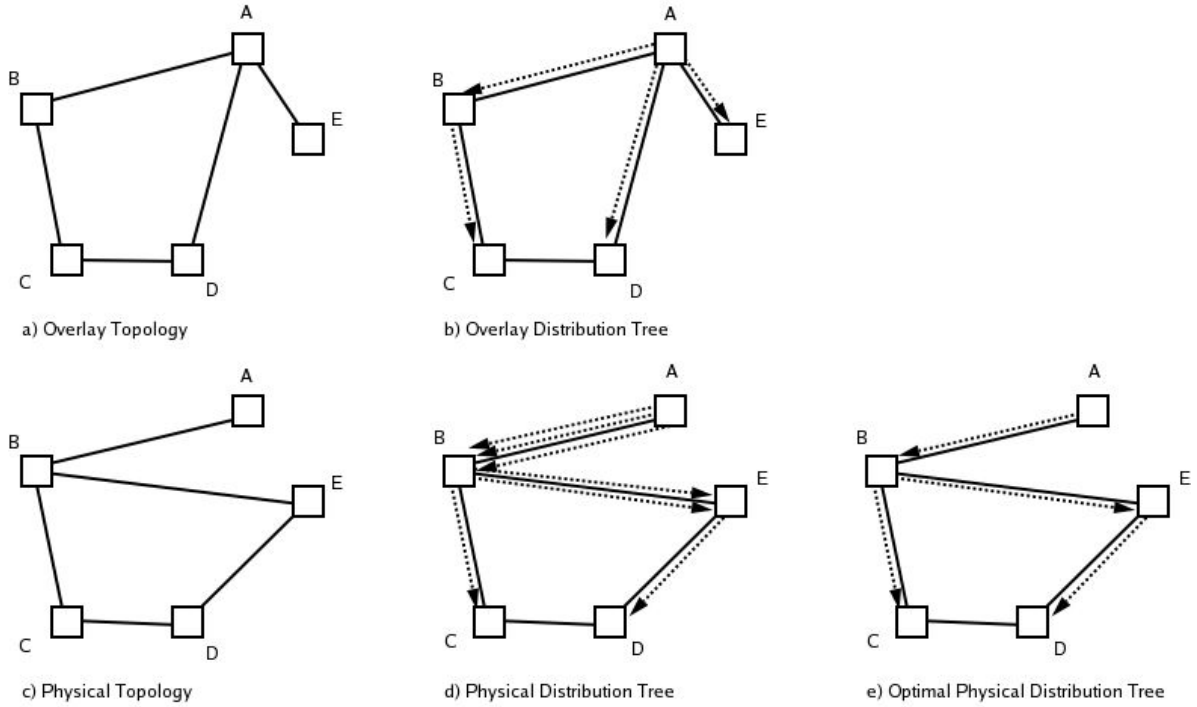


Illustration 7: Finding the best network topology for an overlay network that is used for multicasting requires some knowledge of the underlying physical network.

The redundant packets are an indirect result of the mismatch between the overlay topology and the topology of the underlying physical network and have a negative impact on the overall performance. In contrast, if the overlay network between the departments was configured to resemble the topology of the physical links, this would result in the multicast distribution tree that is depicted in illustration 7e. In this network, no physical link carries duplicate packets.

3.1.5 Autonomous Overlay Topology Construction

In the current Terracast implementation, configuration of the overlay topology is a manual process. To ease the administration of the network, the process of setting up and maintaining the topology could be implemented as an autonomous algorithm that is able to find the best router pairs. If this algorithm is run continuously in the background, it can even increase the resilience of the network by creating alternative edges after a the crash of a router daemon.

However, the overlay network runs on the application layer, independent of the underlying network. Without creating a coupling between the underlying network and the overlay router daemons, the independent nature of the application layer makes it difficult to determine whether a TCP connection to a certain router daemon will run over a unique physical path through the underlying network. Because of these difficulties, autonomous topology construction is not part of this thesis.

3.1.6 Conclusion

This section on overlay network topologies is concluded with a summarization of the properties an overlay topology should offer and what conditions must be taken into account when such topology is designed. These properties and conditions apply to a resilient overlay network that is targeted at the large-scale multicast distribution of real-time data. Removing and adding new edges to the overlay network is considered a manual process at this time. Also, the network must be able to scale to thousands of nodes and that propagation delay of packets should be minimized.

1. It was shown that in order to make an overlay topology resilient to the crash of one or more nodes, each node should be connected to more than two neighbors. Such a network offers more than one path (sequence of edges) between each pair of nodes.
2. The network must be able to scale to thousands of nodes and therefore the process of systematically connecting each node directly to all other nodes is not practical.
3. Because a node is not directly connected to all other nodes, it needs routing information to reach non-neighbor nodes.
4. As low propagation delay for packets is important in a network that is used for real-time streaming data, a routing algorithm should use performance measurements to determine the shortest path to any destination.
5. To allow for automatic re-routing of packets whose data path disappeared after the crash of a

node, the routing algorithm should constantly evaluate and adjust the routing information. This is especially useful on overlay networks where edge performance fluctuates.
6. The overlay topology should closely match the topology of the underlying physical network to minimize the number of duplicate multicast packets on each physical link and to maximize multicast performance.

3.2 Dynamic Routing

The problem of finding the shortest path between any two nodes in a network of arbitrary topology without prior knowledge of the network's layout has been solved long ago.

When packet-switching was first introduced by Paul Baran and then independently a few years later by Donald Davies, its focus lay on military applications where it allowed networks with many redundant links and routers to survive destruction of a significantly large part of the network, without the need for manual intervention and reconfiguration. This required routing algorithms running at the nodes and through ARPA, Baran's early work [BAR64] influenced today's protocols.

Currently two major classes of dynamic routing protocols exist that differ in the way shortest paths are computed and what routing updates are propagated to other routers. The first class of protocols is known as *distance-vector* protocols, the latter as *link-state* protocols.

This section discusses the merits and liabilities of both types of routing protocols. It then selects an existing protocol that best suits the Terracast requirements.

3.2.1 Link-State Routing

Link-state protocols take the relatively simple approach of propagating the state of the entire network as a list of all known links in the network with their cost value. This way, every node in the network will have complete knowledge of the topology of the entire network. Changes to the topology can be propagated quickly to all nodes as the routing updates can be forwarded unchanged. Finding the next hop in the shortest path between any two nodes can then later be done by running Dijkstra's algorithm [COR90] on the network topology information.

To store the entire network topology in memory, the typical space complexity of a link-state protocol is $O(N^2)$ where N represents the number of nodes in the network. Successful link-

state protocols include OSPF (Open Shortest Path First) [MOY89] and IS-IS (Intermediate System to Intermediate System) [IEEE90].

3.2.2 Distance-Vector Routing

Distance-vector protocols are based on the Bellman-Ford protocol [FORD62]. They differ from link-state protocols in the kind of information they exchange with their neighbors, as well as the amount of information stored in routing tables.

The original Bellman-Ford protocol tells its neighbors how far away, in terms of intermediate hops, it is from all the other known nodes. Effectively it publishes the hop count of every shortest path it knows. This is done as a collection of tuples, where each tuple contains a destination address and a path length. These tuples are called *distance-vectors*.

The receiving node increments the length of every path by one and compares each path to its own shortest path information. When it turns out the neighbor offers a shorter path to a certain destination, it replaces its own path with the new path. From now on, data packets addressed to this destination will be routed via this neighbor. Every time the information in a node's routing table is changed, it will publish its shortest paths to its neighbors. The presence of new nodes is automatically learned when a set of distance vectors contains a new destination address. Some widely accepted implementations of distance-vector protocols include RIP [HED88], [MAL98] and Merlin-Segall [MS79].

An important consequence of the way routing information is exchanged is that the information is first processed and merged into the local routing table, at which point a new routing update is derived from it. Because of the processing time required at each node, propagation of routing changes through the network is typically slower than with a link-state protocol.

An advantage of distance-vector protocols is that they maintain relatively little state information. The space complexity for this class of protocols is typically $O(N \cdot e)$ where e is the average number of neighbors per node, while N represents the total number of nodes in the network.

Although the use of localized information and routing updates makes the protocol friendly in terms of storage requirements, it is susceptible to long-lived routing loops. This is probably the biggest disadvantage of the classical Bellman-Ford protocol. The loops occur when a route “backwashes” from a receiver to a sender after a link or node failure. This continuous exchange

of an invalid route leads to an explosion of routing updates whereby the length of the invalid path is constantly increased. Because of its nature, this problem is known as *counting-to-infinity*. The effects of the problem can be reduced by putting an upper limit on any path length. This approach is taken by RIP, which does not allow paths longer than 15 hops. Other well-known techniques include *poisoned-reverse* and *split-horizon*. The first explicitly advertises a destination as unreachable to the neighbor that provided the best path to the destination, while the latter omits this destination in its routing update to the neighbor. The counting-to-infinity problem and its solutions are further described by Bertsekas and others in [BERT87].

3.2.3 Distance-Vector Versus Link-State

Whether a link-state protocol is better than a distance-vector depends on the characteristics of the network. It is interesting to see that the Internet's predecessor, the ARPANET, initially used the Bellman-Ford-based RIP protocol, but was replaced by the SPF (Shortest Path First) link-state protocol in 1979 [MCQ79] because of RIP's slow convergence after almost a decade of research. In 1987, the protocol was given a revision that made its reaction to topology changes less aggressive by applying exponential averaging over link costs [KAHN89]. After SPF, OSPF was introduced and gone through several versions. In 1998, OSPF Version 2 [MOY98] was introduced and remains one of the most widely deployed protocols for intra-domain routing. In 1999 OSPF Version 3 [MOY99] also provided support for IPv6.

After RIP was replaced by SPF in 1979, many new distance-vector protocols have been designed that structurally solve the long-lived loops of Bellman-Ford. The Merlin-Segall protocol that was mentioned previously is one of them. Another important one is the Extended Bellman-Ford or ExBF protocol proposed by Cheng and others [CHE89]. Merlin-Segall avoids all routing loops, both short-lived and long-lived, while ExBF only prevents all long-lived loops.

A comparison by Shankar and others [SHAN92] between the SPF link-state protocol and the newer, loop-free distance-vectors Merlin-Segall and ExBF showed that ExBF performs as good as SPF under most circumstances with a much lower space complexity. Merlin-Segall generally performs worst. Following the outcomes of Shankar's comparisons between link-state protocols and the more modern, loop-free distance-vector protocols, Terracast adopts the Extended Bellman-Ford protocol as the basis for its overlay routing.

3.2.4 ExBF - The Extended Bellman-Ford Protocol

Because the ExBF distance-vector protocol is used as the base routing layer on the Terracast overlay network and because it will be extended with an address summarization mechanism to improve performance on large networks, it is important to understand how the protocol manages its routing information. The protocol is therefore briefly discussed in this section.

ExBF distinguishes itself from the traditional Bellman-Ford protocol by storing not only the length and destination of each path in its distance table, but also the pre-final node of every path. The pre-final node of a path is the last intermediate hop a packet visits before it reaches its destination node. Using the pre-final address of a path, a router can backtrack the entire path to any destination by recursively looking at the pre-final node of a destination and treating it as a new destination. When an ExBF router exchanges distance-vectors with a neighbor, these vectors also contain the address of the pre-final node of every path. This makes a distance-vector in ExBF a triple, rather than a tuple.

The process of recursively tracing an entire path is illustrated in figure 8 where the routing table of node A is given. Tracing the full path from this node to destination B goes by first looking up the pre-final node on the path to B which is E and then using E as a new destination. The pre-final node of the path to E is F, while the pre-final node to F is node A itself. This concludes the reverse trace of the path that leads to A, F, E, B.

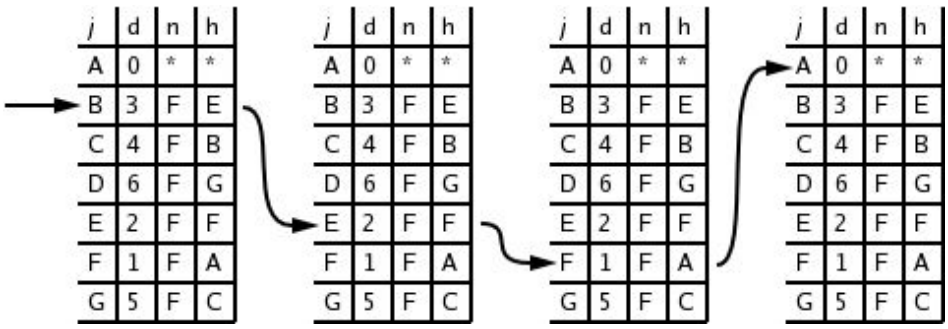


Illustration 8: ExBF uses its pre-final node information to backtrack each path through the network. The figure shows the routing table of node A, where the columns j, d, n and h represent the destination, distance, preferred neighbor and pre-final node respectively. By recursively following the pre-final nodes, the entire path to destination B is traced.

To avoid long-lived routing loops, ExBF uses its pre-final node information to apply the following trick: when node *i* advertises a routing entry to destination *j*, it refrains from sending it to any neighbors whose address is in the path $n_0, n_1, ..., n_r, j$, where n_0 is *i*'s next hop to *j* and n_r is the pre-final node of the path. For the example of illustration 8 that means that node A will not advertise its path to destination B, E or F to its neighbor F. Instead, it will explicitly report that A cannot reach those destinations.

3.3 Scalable ExBF Routing in Large Networks

Adaptive routing protocols such as distance-vector and link-state protocols maintain routing tables that contain a routing entry to each destination in the complete network. This way each packet can be forwarded directly to the neighbor that is in the shortest path toward the destination. When the network's topology or performance changes, this may trigger changes to certain routing entries. In link-state protocols these changes need to be propagated to all nodes in the network as every host maintains an image of the complete network topology. With distance vector protocols the situation is usually slightly better, but some changes still need to be propagated throughout the entire network. The obvious problem is the limited scalability of these routing tables. When the network grows in size, so do the routing tables. This requires more bandwidth to exchange the necessary routing data between hosts. Also, the more links there are, the more often something in the network will change. These factors severely limit the scalability of these protocols.

To overcome the problem of growing routing tables, computation time and excessive advertisement overhead, large networks are often partitioned into smaller domains, connected by gateways. While the gateway routers maintain routing information necessary to route packets to nodes in other network domains, nodes inside a domain only maintain information for those nodes inside the same domain. This mechanism introduces a form of hierarchy that allows the network as a whole to grow beyond the practical limits of standard distance-vector or link-state algorithms. Examples of routing algorithms that structurally divide the network in several levels of domains and exploit this hierarchy to make routing more scalable include [FRED88], [JAF86], [SAN85] and [LEE83].

In most hierarchical routing algorithms, the network's division in domains and sub-domains is reflected in its node addresses. This makes it possible for a routing table to substitute a set of destination addresses by a single wildcard entry. The further away groups of destination

addresses are, the more efficient aggregation of them can be applied. This approach does come with the consequence that connecting new nodes to the network requires the assignment of an address that is not only unique throughout the network, but also suits the range of addresses used by its neighbors. Node addresses now become location based.

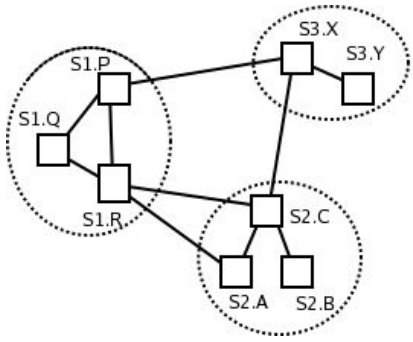
Hierarchical addressing schemes have been studied extensively in the past, including their applicability to distance-vector and link-state protocols. An important and well-known application of hierarchical addressing to allow for aggregation of routing tables on the Internet is Classless Inter-Domain Routing or CIDR [FUL93], which was introduced to support smaller sub-domains and more flexible and efficient aggregation than was possible with the classical, pre-defined network classes A, B and C. Other examples of applications of hierarchical routing include the ATM PNNI standard [ATM96] and the Internet Nimrod architecture [CAS96].

Address aggregation or summarization can often significantly increase the network's scalability, but because groups of nodes or even groups of entire sub-domains, are reached through a single routing entry, the path taken by data packets is often not the optimal path. The more summarization is applied, the less efficient the paths become on average. The factor by which the actual data paths on summarized networks differ from the optimal paths, is known as the *stretch-factor*: the maximum ratio between the length of a route computed by the routing algorithm and that of the shortest path connecting the same pair of nodes. The effect of the stretch-factor on routing performance was studied in [PEL89], which lead to a series of studies into algorithms that do take advantage of address summarization, but can guarantee that the stretch-factor always remains below a certain upper bound at all nodes. These algorithms include [AWER92], [EIL98] and [COW99].

3.3.1 Terracast Address Summarization Protocol

One of the requirements of the Terracast product is that is must be able to scale to deployments with many thousands of nodes. This makes it important to apply a form of address summarization. Here, Terracast takes a relatively straightforward approach. First, an administrator decides at deploy time which nodes form domains and which domains form aggregations of domains. This is done by encoding hierarchy in the Terracast node addresses using a dotted notation. As mentioned earlier, node addresses are ASCII strings of at most 127 characters. Only [a-z] and [0-9] are available. Addresses are case-insensitive.

An example of how a Terracast network is divided into logical clusters through hierarchical addresses is illustrated in figure 9. This figure shows a network of eight nodes, divided into three clusters. Assigning nodes to clusters is typically done on the basis of geographical properties, administrative boundaries or wide area links. All nodes inside a corporate network would typically be assigned the same logical domain, whereas a Terracast network that connects nodes from different corporations, would usually assign a separate domain to each corporate network part. Another criterion may be that nodes that often disconnect and reconnect again later – possibly because these Terracast software routers run on desktop PCs – are placed in a sub-domain, to avoid the routing updates triggered by their frequently changing state to propagate far into the network.



Routing table of node S2.C

dest	hop	cost
S2.C	-	0
S2.A	S2.A	1
S2.B	S2.B	1
S1.*	S1.R	1
S3.*	S3.X	1

Illustration 9: Example network with hierarchical, location-based node addresses that define logical domains. Each link is assumed to have cost 1.

Given the logical domains that now cluster groups of nearby nodes, each node can treat domains other than its own as a single entity and use a wildcard address that matches every host inside that domain. This reduces the size of the routing table, as well as the amount of routing information that needs to be exchanged between the nodes when the topology changes inside a domain. When a new host is added to domain S1 of illustration 9, there is no need to propagate that information to the other domains, as they already have a wildcard entry that will match the new address.

The right-hand side of illustration 9 shows the effect address summarization has on the size of the routing table of node S2.C. The cost value that is listed in the third column represents the cost of the shortest path to the nearest node inside that domain. The fourth routing entry contains the cost to reach node S1.R from S2.C. As all edges are assumed to have the same weight or cost, S2.C will route all traffic for domain S1 through neighbor S1.R.

When nodes exchange distance vectors, either because of a link change, or as a normal, periodic exchange, the receiving node first summarizes the destination address of each distance vector relative to its own address. Summarization is done by matching the advertised destination address with its own address field-by-field[2] and when a field does not match, the rest of the address is substituted by a wildcard. This can be illustrated by examining the first distance-vector exchange between nodes S1.R and S2.C after startup. For this and later examples, the following notation is introduced to describe a single distance-vector: (destination, pre-final node, distance). A periodic routing update between two nodes is a collection of distance vectors and is described as follows: $DV_{from,to}$:{(destination, pre-final node, distance), ...}. The contents of the routing table of a node *i* are described by: RT_i:{(destination, hop, pre-final node, distance), ...}. Note that in a distance-vector protocol, records in a routing table only contain the information of the best known path to a destination, whereas a distance table is a data structure that contains information on all paths to a certain destination – one path for every connected neighbor node.

Now when S2.C receives the first routing update from S1.R it will contain, among others, the distance-vectors (S1.R, *, 0) and (S1.P, S1.R, 1). The first vector says that node S1.R has a path to itself with cost 0 with no pre-final node, while the second vector advertises a path to S1.P with pre-final node S1.R and cost 1. Node S2.C processes all vectors sequentially. Like any distance-vector protocol, it first adds the cost of its edge to S1.R to the distance attribute of the first vector. As each edge in this example is assumed to have cost 1, it learns that it can reach S1.R with total cost 1. Then summarization is applied to the vector's destination address and S1.R is summarized to S1.*. This is because the first field differs from the first field of the local address which causes the remaining fields to be replaced by a wildcard. Because this vector does not have a pre-final node, S2.C puts its own address in the pre-final node attribute. The resulting information (node S1.* can be reached with cost 1 through the edge on which the vectors were received) is then merged into the node's routing table.

Then the second vector is processed. Its distance attribute is incremented and becomes 2 and its destination attribute (S1.P) is summarized into S1.*. What remains after summarization is a path to the same destination wildcard as the previous vector through the same hop, only with a higher cost. Because of the higher cost it is discarded and after all vectors have been processed, the single wildcard routing entry (S1.*, S1.R, S2.C, 1) remains as the best known path to all nodes in the S1.* domain.

2 Note that a network will typically have a hierarchy of multiple nested domains, rather than the 2 level of the example. This is much like the real world where a street can be seen as a domain inside a city and a city is a domain inside a country, etc.

Note that in the above routing entry the second attribute is the preferred hop field. It uniquely identifies which edge is associated with the entry's destination address. The original Bellman-Ford distance-vector protocol identifies the preferred hop by the address of the neighbor router at the other side of the edge. With Terracast however, all addresses in the distance vectors are summarized before they are processed. This may lead to hop attributes that contain wildcards and it may occur that the addresses of more than one neighbor summarize into the same wildcard. If this happens, the hop attribute in a routing entry can no longer uniquely identify a single edge. Because of this complication, Terracast can not identify a preferred hop through a node address. Instead, each Terracast router daemon internally assigns a unique number to each of its edges and stores these numbers in the preferred hop attribute in the routing table. Hence, assuming the edge that connects S2.C to S1.R was labeled “2”, the routing entry (S1.*, S1.R, S2.C, 1) from the above example would in fact be (S1.*, 2, S2.C, 1) .

```
# This function takes $destination_addr and summarizes it against
# $local_addr. The result is stored in $summarized_addr.

FUNCTION summarize ($destination_addr, $local_addr, $summarized_addr)
{
  FOR ($fieldnr = 0; $fieldnr < $local_addr.numfields; $fieldnr++)
  {
    IF ($destination_addr.numfields <= $fieldnr)
    {
      # all fields in the destination address are equal to
      # those in the local address - no summarization
      # necessary
      $summarized_addr = $destination_addr
      RETURN
    }

    $summarized_addr += $destination_addr[$fieldnr]

    IF ($local_addr[$fieldnr] != $destination_addr[$fieldnr])
    {
      # if destination has more fields than local, replace
      # them by a wildcard and exit
      IF ($destination_addr.numfields > $fieldnr + 1)
      {
        $summarized_addr += "*"
      }
      RETURN
    }
  }

  IF ($destination_addr.numfields != $local_addr.numfields AND
      $destination_addr.numfields != $local_addr.numfields + 1)
  {
    # local = A.B.C, destination = A.B.C.D.E.F
    $summarized_addr += $destination_addr[$local_addr.numfields]
    $summarized_addr += “*”
  }
  RETURN
}
```

Table 2: The algorithm used by Terracast daemons to summarize both the destination and pre-final addresses advertised in distance-vectors. Each address is individually summarized against the address of the local router node.

The algorithm used to summarize addresses relative to a node's local address is given as pseudo-code in table 2. This code is called for every Terracast host address that is received through the exchange of distance vectors. The parameters expected by the function are the address that needs to be summarized, the local address of the node and an address instance that is used to store the summarized address respectively. Additionally, table 3 contains a number of examples of addresses that are summarized against a certain local node address. The table shows that A.B.B is the result of summarizing address A.B.B against A.B.C.D and that A.B.B.* is the result of summarizing A.B.B.B. It is important to understand that the results are in fact two different addresses. The first (A.B.B) only matches the exact address A.B.B while the second is

a wildcard that matches everything that starts with A.B.B and has at least four address fields. This includes A.B.B.B, but also A.B.B.Z. However, it does not match A.B.B.

Address to be Summarized	***Local Node Address***	***Summarized Address***
A.B.C.Z	A.B.C.D	A.B.C.Z
A.B.C.D.E	A.B.C.D	A.B.C.D.E
A.B.C.D.E.F	A.B.C.D	A.B.C.D.E.*
A.B.C.D.E.F.G	A.B.C.D	A.B.C.D.E.*
A.B	A.B.C.D	A.B
A.B.B	A.B.C.D	A.B.B
A.B.B.B	A.B.C.D	A.B.B.*
E.F.G	A.B.C.D	E.*

Table 3: Examples of addresses that are summarized according to a certain local node address using the summarization algorithm of illustration.

3.3.2 Summarization Opacity and Border Gateways

Summarization of routing entries comes with the consequence that packets may not always be routed according to the shortest path. This was referred to as the stretch-factor. This paragraph describes how the negative effects of the stretch-factor can be reduced in a Terracast overlay network by using variable summarization levels.

On the Terracast network this path stretching can occur when packets are forwarded between nodes that are in different domains, because an entire sub-domain is treated as a single node with multiple edges. Consider the left part of illustration 10. Again it is assumed that all edges have cost a cost value of 1. Therefore, S1.P will learn that it can reach S2.* either through its own inter-domain link, as well as through neighbor S1.R. The first has cost 1, while the latter has cost 2. So when node S1.P needs to forward a packet for destination S2.F, it will send the packet directly down its inter-domain link to the S2 domain. The right side of the illustration shows the actual topology of the S2 domain. It is clear that the stretch factor is substantial for packets from S1.P to S2.F, as the shortest path runs through S1.R instead.

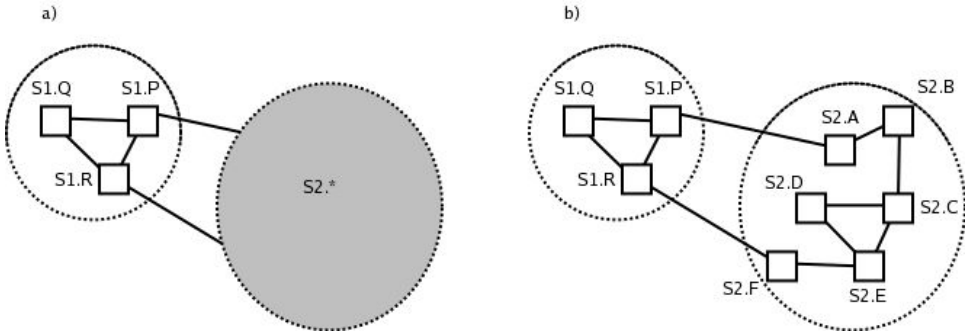


Illustration 10: This illustration shows the effect of decreasing the summarization opacity on the average stretch-factor. In illustration a) the default opacity is used, while illustration b) decreases the opacity level to get more detailed topology information.

Although Terracast's address summarization technique cannot guarantee a maximum upper bound on the stretch-factor like the previously mentioned protocols [AWER92], [EIL98] and [COW99], it can reduce the effect by decreasing the amount of summarization that is applied. The default summarization algorithm from table 2 tolerates only a single address field to differ from the local node address. All following fields are merged into an asterisk. This leads to compact routing tables, but a rather opaque view of the rest of the network. The network's opacity can be reduced by a summarization algorithm that tolerates more than one dissimilar field. Less aggressive summarization leads to less compact routing tables, but a lower average stretch-factor and more accurate routing. This inverse relation between scalable, compact routing tables and routing accuracy allows a Terracast network administrator to optimize the network for either property.

When a uniform level of summarization is used by every node, the entire network is either optimized for speed, or scalability. However, when this assumption is dropped, it becomes

possible to optimize each part of the network according to the local requirements. Depending on the usage or topology of the network, large, accurate routing tables may not always be a requirement. For example, if the network of figure 10 had only a single edge connecting the S1 and S2 domain, the summarized entry in the routing table of the border routers would never lead to less efficient inter-domain routing. By adjusting the level of summarization to each node's local conditions, a healthier balance between performance and scalability can be achieved.

In those parts of the network where the number of interconnections are few and expensive, it is important to make accurate routing decisions, so the level of summarization would be kept low. On the other hand, on those parts of the network where the nodes are interconnected by fast edges that run over LANs with plenty of bandwidth, summarization can be applied more generously to increase scalability.

The general observation is made that accurate routing information is most valuable to nodes that have a direct connection with a node that is inside another domain, because these are the nodes that aggregate address ranges into wildcards. Also, these are the nodes that deliver packets to a foreign domain and as figure 10 illustrates, accurate routing decisions made by these border routers can substantially decrease a path's stretch-factor. Once a packet has reached its destination domain, summarization no longer has any influence on the packet's path. This is because node addresses that lie at the same hierarchical level are never aggregated into a wildcard. Therefore, nodes that only share edges with nodes inside the same domain, can benefit from the highest level of summarization and compact routing tables.

If this strategy is applied to figure 10, the border routers (S1.R, S1.P, S2.A and S2.F) should tolerate two or more distinct address fields per routing entry, making the foreign domain transparent to them. All other nodes could still run the default, most aggressive level of summarization from illustration .

Whether or not different summarization levels should be used throughout a Terracast network can be a point of discussion. It can be used by individual network administrators to tune the performance of the network, but sufficient knowledge of the overall network topology is a prerequisite for the administrators. Only then can the adequate balance between performance and scalability be found. Without a detailed understanding of the network's topology, it may however still be a good idea to use more powerful machines at the border routers that apply less summarization than the nodes that are connected only to nodes that are in the same hierarchical domain, thereby making more accurate inter-domain routing decisions, while keeping the processing and storage capacity requirements at the nodes inside a domain modest.

4. MULTICAST ROUTING

This chapter covers the multicast features offered by the Terracast network. It discusses Terracast's supports multicast protocol and what features it supports. In particular, it is described how Terracast combines layered multicast functionality with reliable delivery to guarantee that a consistent representation of live data streams is delivered to all subscribers during fluctuating conditions on heterogeneous networks.

4.1 Introduction

The previous chapter explained the unicast routing layer operating on the Terracast overlay network. It allows networks of arbitrary topology to dynamically discover the shortest path to any destination node, while anticipating crashing nodes and varying link characteristics. The Internet itself already offers the same basic functionality, but because application-level networks that offer dynamic routing often derive their forwarding policies from their own network performance measurements, they may under certain circumstances adapt to changing network conditions quicker than the underlying Internet. The ability to quickly react to changing network conditions could in some cases already be enough reason to operate an overlay network, rather than directly using the Internet.

The reason that overlay networks can sometimes offer higher efficiency and lower end-to-end latency is because the Internet's inter-domain routing protocol does not always use the fastest paths between any pair of nodes, nor does it converge to new routes immediately when important carrier domains go offline. The latter is often caused by the Border Gateway Protocol BGP [REK95] that is run by backbone routers that connect different Internet domains and allows for dynamic routing between them. In many cases, BGP needs many minutes to reach a fully consistent view of the network after a fault [PAX97], [LAB99], [LAB01]. In contrast, switches in the public telephone network recover from link failure in the order of milliseconds. This makes the Internet sensitive to configuration errors, failure and abuse, which is reflected by the reports of the North American Network Operators' Group [NAN04] that shows that hardly any week is without serious disruptions among major ISPs.

The persistent instabilities of Internet routing were the motivation for a series of academic studies into overlay networks that would be able to circumvent these issues and use their own

application layer routing to recover from failures in the underlying IP network more quickly than BGP does. Among these studies is Resilient Overlay Networks or RON [AND01]. Aside from academic studies, many commercial offerings are available that essentially provide overlay services over the Internet with increased availability and improved end-to-end latency figures. These include Internap [IN04] as well as Virtual Private Network solutions offered by many Internet Service Providers.

Although Terracast could possibly also be used for this sole purpose, its main feature is to offer multicast support over existing IP networks. With this objective, it falls in the popular category of application layer multicast networks that include Narada [CHU00], YOID [FRAN00], Scattercast [CHA00] and many others. Terracast however, tries to go beyond offering plain multicast functionality and additionally focusses on supporting live multicast streams with limited jitter and delay through the combination of overlay multicast, multicast congestion control and layered multicast. In that sense Terracast distinguishes itself from most existing application layer studies.

The remainder of this chapter is structured as follows. Paragraph 4.2 briefly discusses the various methods for traditional multipoint packet distribution, as well as the mechanism used by Terracast. Paragraph 4.3 explains why standard, best-effort multicast alone is insufficient for most applications and how multicast can be made reliable. Paragraph 4.4 focusses on how reliable delivery can be combined with real-time delivery on Terracast by carefully pre-processing a data stream.

4.2 Algorithms for Multipoint Packet Distribution

Multicast routing in packet-switched networks has a long history but was added to the Internet Protocol relatively late. In a series of landmark papers, Deering introduced multicast extensions for the Internet Protocol [DEE85], [DEE88], [DEE90] that eventually lead to the birth of the MBone in 1992 [SAV96], [KUM95]. Deering's work is based on a strategy called *reverse path forwarding* or RPF. With RPF, data packets are not forwarded based on their destination address, but rather on their source address. A packet is forwarded by a router only if it was received over the link that would normally be used to send data packets to that source. RPF is therefore ideal for building implicit shortest path distribution trees from a source to all other nodes in the network. In [DEE90] Deering uses this technique to extend distance-vector protocols with multicast support.

4.2.1 Reverse Path Flooding

The simplest application of reverse path forwarding is flooding. This can be used to quickly flood packets originating at a particular source to all other nodes in the network. This is depicted in illustration 11a where node A floods a packet to rest of the nodes. At time $t = 1$ it sends its data packet to its neighbors. Upon receipt of the packet from A, each node checks if it was received on the link that offers the shortest path towards A and if this is the case, forwards the packet across all other links. These packets are marked as $t = 2$ in illustration 11a. The algorithm terminates at $t = 3$ after which at least one copy of the packet has been received by every node.

4.2.2 Reverse Path Broadcasting

Reverse path flooding is simple and effective, but its limited sophistication leads to unnecessary copies of packets. To avoid these duplicate packets, it is necessary for each router to know which of its links are *child links* in the shortest reverse path tree rooted at source A. Dalal and Metcalfe [MET78] proposed a technique for identifying child links which involves periodic notification packets sent by each node to its parent node (preferred hop towards A). Such notification informs the receiving node that it is the preferred hop for packets towards A. When a node receives a notification, it learns that this neighbor is a child in the shortest path tree rooted at A. Only those links will be used when forwarding multipoint packets.

This is shown in illustration 11b where nodes E, D and C do not forward the multipoint packet received from A because they know they are leaf nodes in the shortest path tree rooted at A. This technique is called reverse path broadcasting.

The use of explicit notification packets as described by Dalal and Metcalfe is unnecessary on Terracast. This is because the required child link information is already available in the unicast routing information as a by-product of the loop avoidance algorithm of ExBF. In ExBF, a node will inform its neighbors about the shortest path it offers towards destination A to all but the neighbor that is in the path towards A. Instead, an infinite distance is advertised to this neighbor. The receiving neighbor can use this information to conclude that it is a parent node in the shortest path tree of A, and it will immediately mark those links as child links for multipoint packets from A.

Note that this implicit child link information is also available in any distance-vector protocol that uses poisoned-reverse.

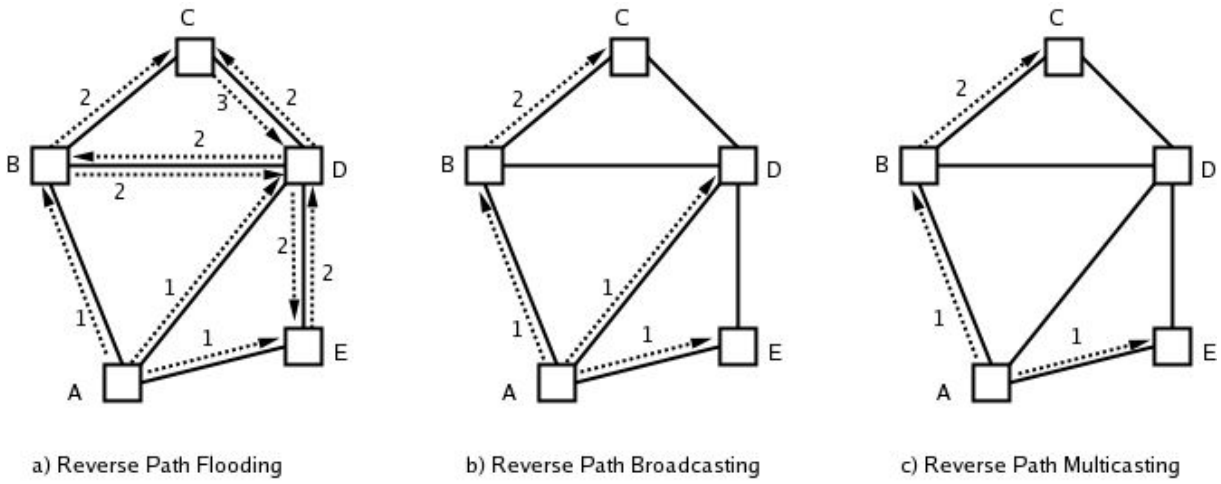


Illustration 11: Different applications of reverse path forwarding. Figure a) shows how straightforward reverse path flooding is used to deliver multipoint packets to all nodes in the network. In figure b) reverse path broadcasting is used to eliminate duplicate packets while figure c) adds explicit membership information to ensure packets are only sent to nodes that need it.

4.2.3 Reverse Path Multicasting

Reverse path broadcasting offers the most efficient way to get a single packet to all nodes in the network. Most applications however, do not have such a broad audience. Especially on large general purpose networks that carry many concurrent data streams, the nodes interested in receiving a particular multicast distribution usually represent only a subset of the entire network. Broadcasting every stream to all nodes could waste a significant amount of network capacity. It is therefore important the network uses subscription information to keep track of who is interested in receiving which streams.

The result of combining broadcasting with subscription information is shown in illustration 11c where A's data stream is delivered only to subscribed nodes C and E.

Tracking subscriptions can be done in two ways. The first technique is pruning. With pruning, each data stream starts as a broadcast. However leaf nodes that are not interested send an explicit prune message to their parent node, causing the parent to cease further forwarding of the stream. When a non leaf node receives prune messages on all its child links and has no interest in the stream itself, it will prune itself from the distribution tree. Pruning reports are

assigned a time-out value. When a time-out expires, forwarding is resumed. Leaf nodes therefore continually have to send new prune messages.

The alternative to pruning works in the opposite direction. Multicast packets are not forwarded across child links unless child nodes have explicitly expressed their interest. Hence, when a node becomes interested in a particular multicast stream, it sends a join message to its parent node. This causes the parent to become interested as well and send a join message itself. As with pruning, subscription information works with time-outs and needs to be renewed frequently.

4.2.4 Dense-Mode Vs Sparse-Mode

Multicast protocols that manage subscriptions through pruning are called dense-mode protocols whereas protocols that use explicit membership reports are known as sparse-mode. Which of these approaches is better often depends on the type of network and the multicast interest patterns.

In a network with few multicast sources where all nodes are interested, dense-mode forwarding is often convenient, while networks with few subscribers per multicast group are usually better off using a sparse-mode protocol. Sparse-mode forwarding is also important in networks with high volume streams that cannot afford to temporarily forward these streams over links that have insufficient capacity and no interested receivers.

Examples of dense-mode protocols include Protocol Independent Multicast – Dense Mode, or PIM-DM [ADAM04] and the Distance Vector Multicast Routing Protocol, or DVMRP [WAI88] which is the multicast protocol used on the MBone. It is based on the previously discussed distance-vector RIP protocol [HED88]. The most commonly used sparse-mode protocol is Protocol Independent Multicast – Sparse Mode, or PIM-SM [EST98].

4.2.5 Single-Source Vs Multi-Source Multicast

Multicast protocols not only distinguish themselves in the way they maintain subscription information, but also whether or not they allow more than one sender per multicast group. Protocols in which a multicast distribution is linked to a single source are generally technically easier to implement and less complex in nature. However, those that do allow multiple sources

are usually more powerful and versatile because they better facilitate distributed applications where participants both consume and produce data.

Well-known and established multi-source protocols are *Core Based Trees* or CBT [BAL93], PIM-SM, PIM-DM and DVMRP (the latter two actually being dense-mode versions of multi-source protocols). Examples of the less versatile, single-source protocols include *Source Specific Multicast* or SSM [BHA03] and *EXPlicitly REquested Single-Source* multicast, or Express [HOL99].

Which type of protocol is preferable depends on the requirements and usage patterns of the applications that will run on the network. A multi-source protocol can of course support applications that have only a single source per stream, but it will generally do that less efficiently than a single-source protocol. Multi-source protocols such as CBT and PIM-SM that use shared trees often also require additional administration regarding their rendezvous point. More information on the administration and overhead of rendezvous points in shared tree protocols is given in [WIL99]. An extensive performance comparison between single-source and multi-source protocols is given in [WEI95]. Given these properties, it is important to carefully consider the expected use and requirements of a network prior to choosing a multicast protocol.

Aside from their performance characteristics, single- and multi-source protocols have another interesting difference. In contrast to single-source protocols where only the receivers are anonymous, both the receivers and the senders are anonymous in multi-source multicast. This anonymity, for example, makes it possible to switch to a different source application in a single-source broadcast when the original source crashes. Because the actual sender address is irrelevant in multi-source multicast protocols, the switch can be made without the need to let the clients join a backup multicast group. This is different in most single-source multicast protocols because here the multicast group often explicitly contains the unicast address of the source in the form of S:G tuple (where S is the node address of the source and G is the group identifier used by the source application of node S). As such, switching to a secondary source application running under a different unicast node address is difficult if not impossible as it requires the receivers to join a new multicast group.

4.2.6 Terracast's Multicast Algorithm

The intended use on the Terracast network is to have many individual applications that each publish their own high volume market data streams to large numbers of receivers. It is not important for receivers or other nodes to be capable of publishing data packets to this multicast group. Instead, for security reasons, it is even good to know that only the real source can publish to the group[3].

Given these requirements, together with the fact that multi-source protocols are more complex than their single-source alternatives, a custom single-source, sparse-mode multicast protocol was built for Terracast. Although the protocol has many similarities with SSM, it is discussed in detail in the remainder of this section.

Joining a multicast group and constructing the distribution tree works similar to SSM. Initially there is a source application on node q that publishes data to group q:G, but since there are no active subscribers, no data is sent across the network. When the first node p_0 subscribes, it sends an explicit join message to its neighbor p_1 which offers the shortest path towards node q. Upon receipt of this message, p_1 marks its child link as an active subscriber for group q:G and subscribes to q:G itself by sending a join message further up the tree until the subscription reaches source q. The result is a data stream that starts flowing from q to subscriber p_0. This is depicted in illustration 12a.

A node that receives a join message for a group it is already subscribed to, does not forward the join message any further, but only marks the child link on which the message was received as another active subscriber that will receive a copy of each packet published to that group. This is illustrated by node u joining the active group q:G in illustration 12a.

The sparse-mode distribution tree generated when processes subscribe to an S:Q group initially equals a part (or all) of the optimal sink tree rooted at S. However, when the underlying unicast routing algorithm detects changes in the network performance and updates some of the routing table entries, the optimal sink tree changes accordingly and may no longer be matched by the multicast distribution tree, rendering the performance of the multicast tree sub-optimal.

An extreme example of this is when a link that is part of the multicast distribution fails entirely. In illustration 12b that happens when the link between u and w fails. At this point, the distribution tree becomes partitioned. When the underlying routing algorithm detects the failure,

3 The network's single-source properties can never be relied on for security. It will probably always be possible to inject malicious packets in a data stream by spoofing node addresses or compromising routers. Hence, for real security guarantees it will be necessary to use data encryption, electronically signed data packets, or a combination of both.

it updates the routing table and selects the adjacent link to *r* with the second-best distance to *q* to become the new preferred hop towards *q*. To reflect this change in the distribution tree, node *u* invalidates its subscription at *w* by sending a leave packet for group *q*:G. In this example the leave packet can be omitted, as the link to *w* is completely unavailable, rather than sub-optimal. It then sends a join packet for group *q*:G to the new preferred hop *r*.

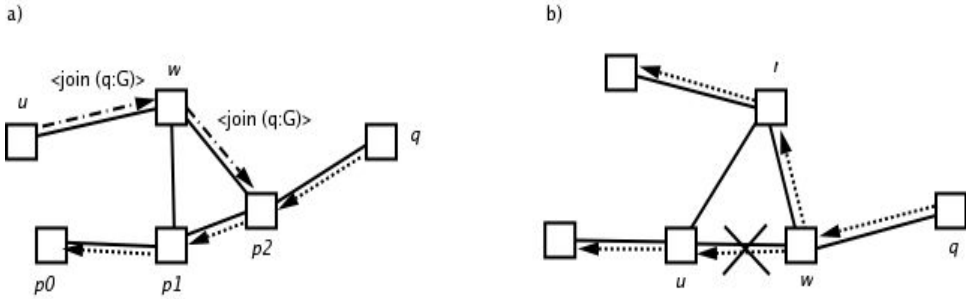


Illustration 12: Figure a) shows the process of joining an active multicast group. Figure b) is used to illustrate what happens when a multicast distribution tree becomes partitioned after a network link failure.

As an enhancement to minimize packet loss, node *u* would normally send the join packet first and optionally even wait for the first packet addressed to *q*:G to arrive before sending the leave packet. This step minimizes the chance of packet loss, but could have the consequence of packet duplication. However, occasional packet duplication, as well as loss are considered accepted behavior on the Terracast network. It is left to the receiving application or a specialized protocol layer on the receiving side to handle this.

To further illustrate the influence ExBF as the underlying unicast routing protocol has on recovery performance of a partitioned multicast distribution, consider figure 13a. Here it is assumed that all links have cost 1, except for the link between *p* and *w* which has cost 100. The result is that the expensive link connecting *p* and *w* will not be part of the optimal sink tree rooted at *q*. Assume that only *t* and *v* have a local application that is subscribed to group *q*:G, so that the sparse distribution tree equals the dotted arrows. The underlying unicast routing algorithm is ExBF, which leads to *p* advertising to *s* that *q* is unreachable, nodes *w*, *u*, *r* and *s* all have only a single route to *q*.

When the link between *u* and *w* fails, *u* becomes disconnected from *q* and invalidates the subscription information for child link *u*-*r*. The ExBF algorithm running at *u* informs neighbor *r* about the failed link as part of an immediate distance-vector exchange. This routing update

implicitly tells node *r* that its subscription for *q*:G through *u* is no longer valid and that it should look for an alternative neighbor to rejoin. An explicit leave packet to *u* is unnecessary.

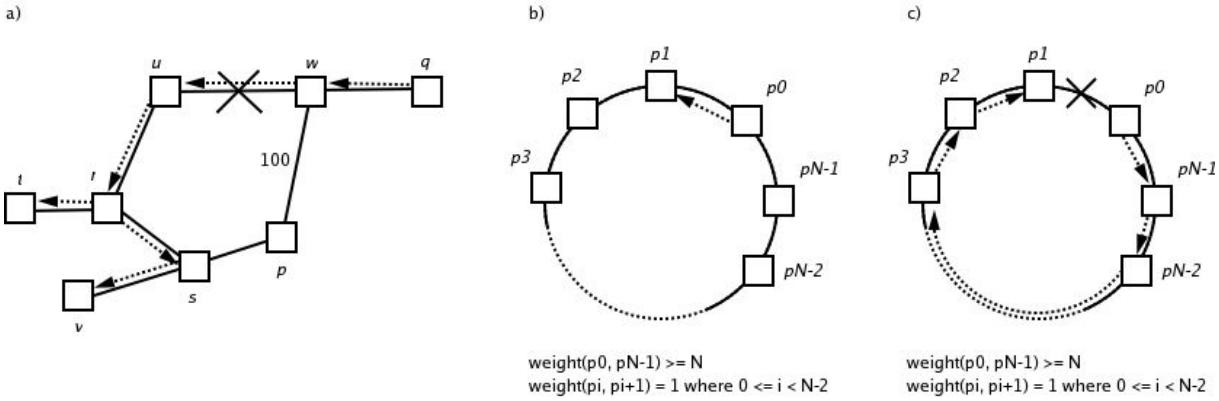


Illustration 13: Figure a: in some cases ExBF causes nodes to become disconnected to a finite period of time after a link failure, causing the recomputation of the distribution tree to become more complex. Figure b and c: when ExBF causes nodes to become temporarily disconnected while the unicast routing algorithm converges, recomputation of the distribution tree can take up to 3N time units. Hence, the worst-case time complexity for recovering from link failure is 3N.

Since *r* is also left with no alternative path to *q* and has no locally subscribed applications, it invalidates its *q*:G subscription and is no longer part of the distribution tree. Neighbors *t* and *s* are implicitly informed about this through the ExBF routing update sent by *r*. Eventually *w*, *u*, *r* and *s* all leave the distribution tree, while nodes *t* and *v* schedule a rejoin when a new path to *q* is found. At this point, Terracast informs their local subscriber applications about the broken subscription, allowing them to distinguish between an idle data stream and a broken subscription.

When *p* receives the routing update from *s*, it loses its preferred hop for *q* and switches to neighbor *w*. The new cost to *q* is now 100 + distance(*w*, *q*) and neighbor *s* is informed. Upon receipt, *s* discovers the new path to *q* and informs its neighbors *r* and *v*. Node *v* which still has a subscribed application sends a join packet for group *q*:G to *s* to repair the subscription. When the new path reaches *t*, it sends a join packet to *r*. When the join operation finally reaches source *q* via *p* and *w*, the new, optimal distribution tree is complete, reconnecting all receivers to the data stream.

Effectively, recovering from failed link is divided in three steps. In the first step the underlying ExBF routing protocol starts a wave that propagates the link failure through the network. Depending on the actual topology, this can take up to N-1 packets, where N represents

the number of nodes in the network. To understand this, consider the ring network of illustration 13b. Assume p_1 is subscribed to group p_0:G. Due to the link costs (link p_0-p_{N-1} is more expensive than all other links combined), the sink tree rooted at p_0 runs anti-clockwise from p_0 to p_{N-1}. Because of ExBF's loop avoidance algorithm, nodes p_1 to p_{N-2} only have a single path to p_0 via their clockwise neighbor. Node p_{N-1} has both a path to p_0 via neighbor p_{N-2} (with cost N-1) and via neighbor p_0 (with cost N). Its preferred path therefore runs through p_{N-2}. The multicast distribution tree only contains the edge between p_0 and p_1 because only p_1 is interested in the data from p_0:G.

When the link between p_0 and p_1 fails, this information is propagated through the ring in anti-clockwise direction from p_1 to p_{N-1}. When p_{N-1} finally loses its preferred path, it switches to neighbor p_0 and tells p_{N-2} about the new route. This update is passed through the network in clockwise direction from p_{N-1} to p_1. When p_1 learns about the new path to p_0 after 2N time units, it sends a join packet for group p_0:G to p_2. This subscription is propagated through the network in N-1 time units and after it has been received by p_0, the new multicast distribution tree is computed as depicted in figure 13c. It is concluded that the worst case time complexity for recovering from a single link failure is 3N when ExBF is used as the underlying unicast routing algorithm. Note that this is the worst case complexity. For example when p_{N-1} was subscribed to p_0:G, rather than p_1, time complexity would have been down to 2N.

Until now, situations have been described in which a link that is part of the multicast distribution tree fails. More often however, links will not fail completely, but rather fluctuate in quality, causing the underlying routing algorithm to reassign preferred neighbors. In this case it is not a hard requirement that the distribution tree is changed to reflect these changes, as it is not partitioned. However, since the tree becomes sub-optimal, it could be preferable.

In general, the decision whether or not to recompute a distribution tree after a routing table update should be made according to the quality of the used path. When a node p in the distribution tree detects that the neighbor in the optimal path towards the multicast source has a cost that is only marginally lower than the neighbor that is currently p's parent in the actual distribution tree, the overhead of recomputation and the risk of packet loss or duplication may not weigh up to the increased performance of the new tree. Also, since the quality of each link is continuously re-evaluated, the cheaper sink tree may only have a temporal nature.

Two straightforward solutions to this recomputation problem are available. Either the algorithm could set a threshold on recomputation so that subscriptions are only moved to the neighbor with the currently optimal path towards the source if the decrease in cost is at least a

ratio x, where $x > 1$. A larger value of x will then postpone tree adjustments until a substantial improvement can be gained, while a smaller x makes the tree actively follow the changing characteristics of the underlying network. Another solution could be to postpone recomputation of the tree for a period inversely proportional to the cost decrease ratio x. The latter has the advantage that the optimal distribution tree is always guaranteed to be reached in finite time following the last routing table change. The current version of the Terracast software does not apply any of the above two recomputation strategies. After a distribution tree is constructed, it is not changed unless it becomes partitioned.

4.2.7 Asymmetrical Link Characteristics

Similar to most existing multicast protocols, Terracast's multicast algorithm is based on the reverse path forwarding technique as discussed in paragraphs 4.2.1 to 4.2.3. However, because reverse path forwarding builds shortest path distribution trees from source to receivers, based on distance figures that apply to the path in the opposite direction, reverse path forwarding only works well in networks with symmetric links whose bandwidth and delay characteristics are the same in both directions. In reality this symmetry cannot always be assumed, especially with the rise of asymmetric “last-mile” technology such as ADSL and cable modems. This is considered a known limitation of the Terracast network.

4.3 Reliable Multicast

The previous paragraphs have described how Terracast's single-source, sparse-mode multicast distribution algorithm that is similar to SSM offers multicast features to applications running on the Terracast network. For most types of applications however, offering best-effort multicast services alone is not sufficient. Like unicast applications, most multicast programs require some level of reliable communication. Examples of this are uploading files to several receivers simultaneously or replicating web server caches. Without guaranteed delivery, a multicast transport service has only limited applicability.

The intended use of the Terracast network is to multicast market data to large groups of receivers. Streaming market data has different requirements than a file transfer, but usually also requires some form of reliable transport to avoid corruption or otherwise rendering the data useless or even dangerous to use.

An example of when data loss leads to corruption is the calculation of VWAP. VWAP is a trading acronym for volume weighted average pricing, the ratio between the transaction volume of money against the share volume over a given trading period (usually a day). This indicator is often used as a performance benchmark for trades. When the market data stream that feeds the VWAP calculation looses packets, trade information is lost, causing the VWAP to become incorrect.

Over time, many mechanisms for reliable multicast have been proposed in the literature. Unfortunately, the multicast research community did not produce a single, widely adopted general purpose protocol in a way similar to how TCP became the generally accepted protocol for reliable unicast communication. Reliable multicast protocols often take different approaches to offering their services based on different requirements and targeted use. This has lead to an interesting collection of protocols that each have their own strengths and weaknesses. As is often the case, selecting a reliable multicast protocol requires careful examination of the network's characteristics, the needs of the anticipated applications and usage patterns.

Before Terracast's reliable multicast protocol is described, a number of existing protocols are discussed first in paragraphs 4.3.1 to 4.3.4. This illustrates the technical differences as well as the type of application and data they support best. Terracast's reliable multicast protocol was derived from these observations combined with the intended use of the product. It is described in paragraph 4.3.5.

4.3.1 XtremeCast

XtremeCast [MPULSE05] is a commercial protocol that constructs a data distribution tree that connects all receivers to the source and uses positive acknowledgments to acknowledge the receipt of packets by a child node in a way similar to TCP. A leaf node is responsible for acknowledging the packets received from its parent, restoring the ordering of packets that are received out of band and asking for retransmissions of lost packets. Non leaf nodes have the same responsibility but additionally also forward the received packets over their active child links. To this end the router stores all acknowledged packets in an in-memory data buffer. The packets in this buffer are relayed to the child nodes. When all child nodes have acknowledged a packet, it is removed from the buffer.

Because there is no feedback mechanism between different levels in the tree, it is possible a node receives and acknowledges packets faster than they can be forwarded to the attached

child nodes. This causes the packet buffer to fill up. When it reaches maximum capacity, the slowest child link that still has unacknowledged packets is disconnected. This causes a socket error in the application running at the disconnected child node. A bigger packet buffer thus allows the distribution to better deal with temporal network congestion. At the same time a mismatch between sender and receiver capacity and a large packet buffer also leads to increased end-to-end delivery delay.

The advantage of this algorithm is its simplicity. It is relatively resilient to temporal congestion and does not allow a single, slow receiver to stall the source. A disadvantage is that the lack of feedback from receivers to source makes it difficult to track receiver status and detect and anticipate congestion further down the distribution tree.

4.3.2 PGM

PGM, or Pragmatic General Multicast [SPEAK01] takes a more optimistic approach and uses negative acknowledgments (NAKs) to request retransmissions of lost packets. Intermediate PGM routers normally do not maintain a buffer of recently transmitted packets and will therefore forward retransmission requests upstream towards the source.

Relaying these requests to the source suffers from so-called NAK-implosions when a packet is lost close to the source which is detected by a large group of receivers that all send a NAK packet simultaneously. To avoid this, PGM uses a NAK suppression mechanism: when a receiver detects a lost packet, it unicasts a NAK to the upstream PGM router. Any additional NAKs from this subnet are suppressed because the router forestalls them by multicasting a NAK Confirmation packet (NCF) to the subnet. Once a receiver sees the NCF, it will not send a redundant NAK.

PGM comes with another optimization for packet retransmissions, called local recovery. In this process, any receiver that receives an NCF and is in possession of the missing data may multicast the missing data to the local subnet. Additionally, hosts may be configured as Designated Local Retransmitters (DLRs) for a number of multicast groups. In response to the NCFs they receive for these groups, the DLRs will multicast a copy of the missing data to receivers below it in the distribution tree. Local recovery greatly reduces the load on the core of the network due to retransmissions and reduces the latency of packet recovery for all receivers.

An important strength of PGM is its generally accepted nature. It is supported by many operating systems and commercial network components. A downside it that it lacks flow control

and does not offer an end-to-end, stream-oriented solution to reliable multicast. Applications using PGM must process the individual packets rather than reading from a byte stream.

4.3.3 MTCP

MTCP [RHEE98] uses a relatively complex control and feedback scheme to offer reliable multicast. In addition to offering local recovery and efficient retransmissions like PGM, MTCP also throttles the source application based on the feedback from the receivers so that the transmission rate is adjusted to the receivers' capacity. This is comparable to the flow control features offered by TCP. Another existing protocol that uses flow control is TRAM [CHIU98].

The flow control feature is very practical when multicasting a file to a group of receiver as it allows the network to send the file at the highest data rate supported by all receivers. Unfortunately this throttling is less useful when multicasting isochronous data such as audio or video.

4.3.4 Digital Fountain

One category of reliable multicast protocols uses forward error correction to allow receivers to recover from packet loss while completely eliminating the need for explicit or implicit retransmission requests that flow in the opposite direction of the data stream. Although there are many existing protocols in this category including RMDP [RIZ98] and the work of Nonnenmacher [NON97], one particularly interesting protocol is Digital Fountain [BY02], whose implementation encodes content in a way that allows clients to reconstruct the original data from any combination of received packets, that is equal in length to the original data.

Forward error correction has the advantage of being able to offer low latency data delivery because packet loss never results in NAK packets and retransmissions being propagated up and down the distribution tree. Another strength is that forward error correction does not require the use of specialized network components that assist in packet recovery. The weakness is that the redundant information that is added to the stream, sometimes called parity data, that forward error correction relies on causes additional overhead.

4.3.5 Terracast's Reliable Multicast Algorithm

Terracast's reliable multicast protocol runs over custom overlay network routers that can easily be modified to assist in packet recovery mechanisms. There is therefore no direct requirement to use forward error correction techniques. Instead, a tree-based NAK protocol is used with local recovery through packet buffers inside intermediate routers.

Detecting packet loss is done in the conventional way by sequencing each packet with an incrementing number. When the receiver detects a missing packet, it sends a retransmission request for this packet towards the source.

To avoid a cascade of retransmission requests when a packet is dropped close to the source, a tree-based negative acknowledgment is used, similar to those in PGM and TRAM. This technique provides localized repair without communication with the source, because an intermediate router that has a copy of the requested data packet will respond by sending it again and suppresses further propagation of the retransmission request. This is visualized in illustration 14a.

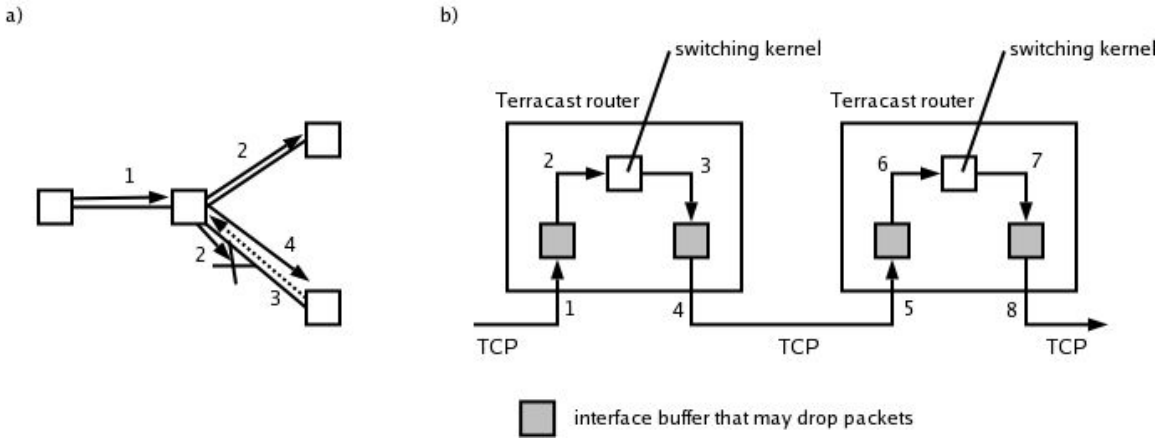


Illustration 14: Figure a) shows reliable multicast with tree-based negative acknowledgments. (1) Data packets are published by the source. (2) The branch point sends a copy of the packet to both neighbors, but one packet is lost. (3) The receiver detects the packet loss and sends a retransmission request to the source. (4) The branch point has a copy of the data packet buffered, responds to the request and suppresses the retransmission request. Figure b) shows possible places inside a Terracast router daemon where packets can be stored to service local retransmission requests. Because packets should be collected before they enter a component that may drop them, points 1 and 3 (5 and 7 respectively) are good, while 4 is not.

Localized repair works by having each router daemon store a window of transmitted packets. Packets are either stored for a fixed period of time after which they are discarded, or a fixed amount of buffer space is reserved to store the most recently forwarded packets. Since the

Terracast overlay network uses reliable TCP connections between router daemons, packets will never get lost while they are in transit between nodes. The only place where packets are discarded is in the interface buffer interceptors that were discussed in paragraph 2.6. Hence, there is no use in buffering packets that were actually transmitted over the TCP connection, as those packets are guaranteed to have reached the neighbor node, unless it has crashed. Instead, a Terracast router daemon should copy packets to its retransmission packet store before those packets reach a buffer interceptor that may discard packets.

Illustration 14b shows the router's buffer interceptors and strategic places to copy packets to the retransmission store. The gray components represent the interfaces with their interceptor pipelines. Because these are the only places where packets are discarded during congestion, the best places to copy packets to the packet store are 1 and 3 (and 5 and 7 respectively). Data packets stored at 4 are guaranteed to have reached 5 and are better stored by the neighbor router daemon.

A convenient way of implementing localized packet repair in Terracast is by adding an interceptor to both the inbound and outbound interceptor pipelines of each interface and letting those interceptors simply store a copy of each packet. When a retransmission request passes through the router on its way to the multicast source, the interceptor inspects the request packet. If it has the requested packet in its temporal packet store, it will not forward the retransmission request to the next interceptor, but inject the requested data packet into the network again.

To allow for greater efficiency, the interfaces do not maintain their own individual packet buffers, but instead share a central, in-memory storage buffer inside the router. Inside this buffer only a single copy of each packet is stored, even if the same packet was seen by multiple interfaces on its way through the router daemon. If an interface buffer stores a new packet in the central store that was already stored by another interface, the central packet store will merely add a pointer to that packet instance to the interface's packet reference table that records which packet was seen by which interface.

Another optimization that can substantially reduce the amount of required storage capacity is to only store packets that were explicitly discarded by the interceptors because of congestion. Storing all other packets has little value, as those are guaranteed to have arrived at the next router. No retransmissions will be requested for them unless some non deterministic packet loss occurs as a result of a crashing node.

Each entry in the packet reference tables has a time-out attached to it. This is to ensure packets are only temporarily stored. A packet will only be expunged from the central store when all references to it from the interfaces have timed out.

An interesting question is how long packets should be stored for local retransmissions. A short time-out minimizes storage requirements, but will also lead to more overhead towards the source as retransmission requests more frequently need to be propagated further upstream. Large time-outs are more tolerant to late retransmissions, but require more storage.

It is important to understand that storing copies of packets in the network can significantly increase the time a packet can survive in the network. However, it is crucial that all previously stored packets are purged by the time the sequence numbers and network addresses are reused, to avoid echoes of old packets ending up in new streams. For the same reason, TCP uses the TIME_WAIT interval to wait for all old packets to disappear before reusing network addresses [STEV93].

Another issue to consider is how to respond to retransmission requests for packets that have timed-out on all routers. This is especially important when a short time-out is used in the packet central packet stores. The smaller the time-out, the larger the chance for a NAK implosion when a router relatively close to the source drops a packet. Because it is important to keep the retransmission time-outs short and storage capacity minimal, each router remembers the source, destination and sequence number of all packets it has purged for a limited period of time. When a retransmission request is received for such a packet, the interface will respond by sending a special packet to the receiver that indicates that the requested packet has timed-out and cannot be recovered anymore. The retransmission request will not be propagated further upstream. The limited packet header information consumes substantially less space than the original packets, so they can be stored longer and avoid NAK implosions even after packets have timed-out.

4.4 Layered Multicast

The previous paragraphs described Terracast's ability to offer reliable multicast services over its overlay network. The implementation of the reliable multicast protocol was a custom one, inspired by many of PGM's features. Because Terracast will mainly be used for the distribution of real-time, isochronous streams, the protocol does not need the sender throttling capabilities of more sophisticated protocols such as MTCP. The result was a protocol that allows

applications on Terracast to publish their own data streams, knowing the subscribed receivers will either receive the stream intact, or not at all if the network lacks sufficient bandwidth.

The problem with real-time data streams is that, unlike file distributions, they cannot be slowed down to match the slowest receiver. Therefore, the source has to encode the stream in a bit rate that every possible receiver can handle. This is an unfeasible task as it requires knowledge of the receivers, as well as the available network capacity. This task is further complicated by the fact that the actual available bandwidth can differ greatly under the influence of other networked applications.

On unicast connections this can be solved by letting feedback information on the current throughput influence the quality of the encoding. When bandwidth is plentiful, the stream is encoded in the highest quality, while poor throughput figures cause the sender to choose a lower quality encoding that uses less bandwidth. This feature is available on most streaming media servers. Sophisticated servers can even adapt the encoding quality dynamically to respond to changing network conditions.

Letting throughput feedback determine the encoding quality for multicast streams is undesirable as it means that all subscribers would always receive the streams encoded in the quality that suits even the slowest receiver. Also, if adaptive encoding is used, the quality would constantly fluctuate even for subscribers close to the source on fast networks.

It is better if the encoded data stream is divided over a list of concurrent sub streams, or layers, where the content of each layer is complement to the content of the previous layer. The more of these layer are received by a subscriber, the higher the quality of the decoded material. Subscribers on fast networks can subscribe to all layers, while slow receivers subscribe to the most basic layers only. This allows each subscriber to receive the data stream in the highest possible quality, regardless of the state and performance of the other subscribers.

Applying layering to a data stream is not trivial. It requires a specialized encoder that knows over how many layers the data must be divided and how to scatter data across layers in such a way that a subset of the layers still offers a useful representation of the original data. On the receiver side, a specialized decoder is required to interpret the data.

Layered multicast protocols and layered data encoders go together. Having one without the other is useless. This coupling has lead to many solutions that combine a layered multicast protocol with a content specific encoder.

The concept of layered multicast streams is used by Terracast and will be described in detail in the following paragraphs. Before Terracast's solution is described, the history of layered

multicast, as well as some influential existing applications and algorithms are discussed in paragraph 4.4.1. Paragraphs 4.4.2 to 4.4.5 describe how layered data streams are treated by Terracast router daemons and how reliable multicast is applied to the individual layers.

Contrary to many existing solutions, Terracast does not combine the multicast protocol with the data encoder. This to allow third parties to develop encoders for new data types without re-implementing a multicast protocol. Terracast aims at offering a network service only, so it does not include layered encoders for various types of content. Nor are these encoders covered by this chapter. Instead, chapter 5 studies the real-world application of Terracast's layered multicast services to high volume real-time market data. The focus of that chapter is the design of a layered encoder and decoder for market data. Layered encoders for audio and video still continue to be the subject of many research papers and are not covered in this thesis.

4.4.1 Previous Work in Layered Multicast

The concept of layering a live multicast stream and allowing subscribers to receive as many layers as bandwidth capacities permit, is an active area of research. One of the pioneers of layered multicast schemes are McCanne and Jacobson, who in 1996 used their Receiver-Driven Layered Multicast protocol (RDLM) to divide a video stream in layers that were sent to separate IP multicast addresses [CAN96]. As long as bandwidth is available, McCanne's receivers would subscribe to more and more layers until packet loss is detected. Loss of packets is detected by appending incremental sequence numbers to the packets and is interpreted by receivers as a lack of sufficient bandwidth or network congestion. To safeguard the quality of the layers, the algorithm unsubscribes a layer as soon as packets are being missed.

The strength of this solution is that is can operate on any IP network that supports IP multicast traffic without the help of specialized network components. Unfortunately it also has some weaknesses that limit its applicability. Firstly the lack of reliability. RDLM offers a best-effort service that cannot recover lost packets. Another limitation is that it repeatedly overloads the network by subscribing to additional layers so that it can discover the current capacity of the network. Although this probing allows RDLM to determine how many layers can be received without congesting the network, it also introduces packet loss that cannot be recovered from. The protocol is further hindered by routers that do not immediately unsubscribe from a multicast group after the last subscriber left, causing packet loss to continue even after RDLM changed its subscriptions. This can negatively influence the protocol and cause it to overreact.

Other layered multicast algorithms are discussed in [WEN03]. Aside from layered protocols, there is also active research in content and applications that would benefit from network functionality such as proposed by McCanne. Examples are audio and video codecs that divide live multimedia content over layers to enhance the user experience over wide area networks [TAUB94]. In [BAB00] an audio codec is presented that can offer smooth audio degradation when network performance drops, while in [LI98] a relatively simple technique is proposed to divide any MPEG 2 stream into 3 layers.

4.4.2 Terracast's Layered Multicast Algorithm

Similar to RDLM, Terracast supports layered multicast by scattering a stream's packets across several layers. However, instead of publishing these layers as independent, individually addressable multicast streams, all packets are published to a single multicast group. Inside this stream packets are labeled with a layer number. A packet's layer number is also known as its priority level. Packets from the base layer have the highest priority and must be received by all receivers. Higher layers contain information that enhances the quality of the data stream and are labeled with decreasing priorities.

When packets are labeled with priorities in the range 0 to 3, the stream is said to have 4 layers. Also, the convention is to treat 0 as the highest priority and 3 as the lowest. When a stream only contains a single layer, all packets are labeled with priority 0. To the software router daemons, a packet's priority value has no meaning other than that it is used as a criterion for discarding packets on congested links. When an outgoing link of a router has insufficient bandwidth to transmit all pending packets, it will forward only those with the highest priority. How this congestion control algorithm works is discussed in paragraph 4.4.6.

Terracast combines layered multicast with local packet recovery to realize a service that applies packet recovery only to those layers that can be delivered by the current bandwidth capacity. Packet loss is detected on a per-layer basis. All packets that are part of a layer are tagged with an incrementing sequence number. Each layer has its own sequence number, so all packets in one layer are sequenced independent of the other layers. This allows packet loss to be detected in every layer, regardless of the conditions and packet loss in other layers.

When the network has insufficient resources to deliver all layers of a stream, it is important that the receivers know this and will not attempt to repair all missed packets from all layers. Instead, the receivers should use a mechanism that provides them with feedback about

the current network capacity and uses that to decide for which layers it will recover missed packets.

Terracast's solution monitors the status of the entire stream and from this information derives how many layers of the stream can be reliably delivered to the user without stressing the network. It will then mark these layers as being *intact* and will send retransmission requests when an occasional packet is lost from these layers. In principle, the algorithm will only deliver packets from the reliable layers to the user application, but not before they have been put back into their original global order. The process of restoring the global packet order is described in paragraph 4.4.3.

In order for the algorithm to decide which layers can safely be marked as intact, it requires some status information about the reception of the stream as a whole as well as the network conditions. To provide this, every packet contains the sequence number of the most recent packet from all other layers. This information is known as a *sequence vector*. Depending on the type of content, it is said that the received packet is causally related to all packets from its list. This means that aside from carrying its own sequence number, each packet contains the current sequence number of all the other layers too.

By inspecting the sequence numbers each time a packet arrives, the receiver can determine whether it has missed any packets from the layers it marked as intact. If that is the case, these packets will have to be recovered. Each time a missed packet is detected, a countdown timer is started for it. If the packet is recovered before the time-out expired, the timer is canceled. If the timer expires before the packet was recovered, the layer is considered impossible to repair and is, together with all higher intact layers, removed from the list of intact layers and Terracast will no longer try to recover lost packets.

The appropriate time-out interval is application or content dependent. Essentially all it determines is how much delay is tolerated by the user. Analogous to the time-out used by the central packet retransmission store of the router interfaces, setting it to a low value means a temporary congestion can potentially corrupt a layer long enough to cause the receivers to drop it. Since packets can occasionally also get lost when unicast routing tables converge, causing changes to the shape of multicast distribution trees, or when a router crashes that had pending packets in its interface buffers, appropriate time-out values will probably have to be found through experimentation.

It is important to realize that normal out of order delivery effectively means that one or more packets are delayed. Based on the packet priority values, Terracast's packet prioritization

algorithm, that will be discussed in paragraph 4.4.6, explicitly introduces out of order transmission when a burst of packets queues in a router interface. Hence, the recovery time-out should be large enough to allow for this. Using a long time-out makes the stream more resilient to congestion, but also increases the time before a receiver discovers that a layer must be dropped due to bandwidth constraints. Until congestion is detected, delivery of the previously received packets that causally depend on missed packets from intact layers is postponed, causing delivery delay.

Starting timers when packet loss is detected allows for congestion detection, but it cannot be used to guarantee low end-to-end latency. When no packets are lost and all are received in their global order, the receiver cannot measure the total transmission delay. As such, a receiver cannot distinguish between an idle and a crashed source.

Aside from removing layers from the intact-list, the algorithm must also be able to detect when more bandwidth becomes available, allowing new higher layers to be added to this list, so that a higher quality stream is automatically delivered to the user. This is done by passively monitoring the layers that are not currently in the intact-list and thus not under repair. Every time a packet is received from these layers, this fact is stored for a period that is equal to the repair time-out discussed earlier. When a packet from a layer is received whose sequence numbers show that a packet was missed from a higher layer that is not in the intact-list, no retransmission request is sent, but the fact that this packet should have been received is recorded and stored for the same repair time-out period. If during the time-out period the packet is received after all, possibly because it was delayed by a router, the stored record is marked as received.

From the recorded arrival state of each packet, a packet arrival history per layer is built that is used to determine whether the reception quality is high enough to add a higher layer to the intact-list and repair any further packet loss. In Terracast, the reception history is used to calculate a moving average that describes the amount of packet loss over the last x seconds, where x is equal to the repair time-out. With a small moving average history, the receiver will quickly respond to increased network capacity, while a longer history will only add layers when the network's capacity is sufficiently high for a longer period of time.

The causality information recorded in the sequence vectors adds overhead to each packet, linearly related to the number of layers used in the stream. When sequence numbers are 32 bits and the publisher uses all 256 available layers, each packet comes with a kilobyte of causality information, contributing 12.5% overhead to a fully filled eight kilobyte packet. To ease layer

administration in the receiver socket, a Terracast multicast stream only supports a static number of layers. When a publisher binds a multicast group address for reliable, layered communication, it explicitly specifies the number of layers that will be used.

4.4.3 Restoring Global Packet Order

By default, packets are delivered to the user application in their original order. Not only are the packets inside the individual layers restored to their natural order using their sequence numbers, but the total ordering across the layers is also restored. If the source sends three packets of different priority (each in a different layer), they are all three received by the user application in the exact same order.

a) {0(9,2,6); 1(9,3,6); 2(9,3,7); 1(9,4,7); 2(9,4,8); 0(10,4,8); 2(10,4,9); 1(10,5,9)}

b) {0(9,2,6); 0(10,4,8); 2(9,3,7); 1(9,3,6); 1(10,5,9); 2(10,4,9)}

c)

0(9,2,6)	0(10,4,8)	
1(9,3,6)		1(10,5,9)
2(9,3,7)		2(10,4,9)

d)

0(9,2,6)	0(10,4,8)	
1(9,3,6)	1(9,4,7)	1(10,5,9)
2(9,3,7)		2(10,4,9)

e) {0(9,2,6); 1(9,3,6); 1(9,4,7); 0(10,4,8); 1(10,5,9)}

Illustration 15: The process of selective recovery and reordering:
a) A packet stream as published by a source.
b) The packet stream as received by a subscriber.
c) Packets as they are stored in the subscriber's receive buffers.
d) Packets after selective repair.
e) Reordered, partial packet stream as delivered to the application.

A prioritized, layered packet is described in the following notation: $P_{1(9,4,6)}$ where 1 represents the packet's priority or layer, 9 the sequence number of the last packet of layer 0 (the lowest, most important layer) that was sent at the time this packet was published, 4 represents the sequence number of this packet and 6 represents the sequence number of the most recent

packet of layer 2. Furthermore, the packet tells us that the stream uses three layers in total (layer 0 up to and including layer 2). Hence, this packet causally depends on packet 9 from layer 0 and packet 6 from layer 2 and will only be delivered to the user application after those packets have been delivered.

This mechanism of recording causality in sequence vectors and piggybacking them on the data packets was somewhat inspired by the application of logical clocks and vector time stamps used in communication of some distributed systems [RAY96].

Illustration 15 shows the process of selective packet repair and ordering. The notation {0 (9,2,6); 1(9,3,6), ...} is used to describe a sequence of packets where $P_{0(9,2,6)}$ was sent prior to $P_{1(9,3,6)}$, etc. The first line in the illustration, labeled a), shows the sequence of packets originally published by the source. It reads from left to right, so $P_{0(9,2,6)}$ was published first, followed by $P_{1(9,3,6)}$, etc.

In this example the source has published eight packets, divided over three priority levels or layers (0, 1 and 2, where 0 is the base layer with the highest priority). The second line b) shows the packet stream as received by one of the subscribers. It shows two lost packets and an incorrect ordering. Before the packets are delivered to the user application, they are stored in internal buffers during the repair and reordering process. This state is depicted in 15c. The packets are ordered in separate buffers, each representing a layer. The illustration shows the missing packets $P_{1(9,4,7)}$ and $P_{2(9,4,8)}$. If the receiver currently only has layers 0 and 1 in its intact-list, it will attempt to repair the hole in layer 1 by sending a retransmission request. If the given sequence was just a snapshot of a running stream, the receiver would have detected the missing packet $P_{1(9,4,7)}$ when $P_{0(10,4,8)}$ was received, because this packet says it depends on packet #4 from layer 1. So even before $P_{1(10,5,9)}$ from layer 1 was received in our example, the receiver already detected loss in layer 1 and immediately scheduled a repair time-out for the missing packet and sent a retransmission request. In fact, if it is assumed the receiver had received packet #2 from layer 1 prior to our snapshot of illustration 15, then the conclusion would be that packet #4 as well as packet #3 were lost.

Shortly after $P_{0(10,4,8)}$ was received, packet $P_{1(9,3,6)}$ is received. The receiver places the delayed packet in the appropriate receive buffer and cancels the repair time-out that it started earlier when it detected that the packet was missing. This is illustrated in 15d. Note that the hole in layer 2 is also detected when $P_{0(10,4,8)}$ is received, as that packet claims to be sent after packet #8 of layer 2 was published, so either packet #8 from layer 2 got lost in the network, or was

delayed. Since layer 2 is not in the intact-list, a retransmission request is not sent. However, as for all packets, a timer is started for packet #8 of layer 2.

When the user application is ready to read packets, the algorithm will return only packets from layers that are in the intact-list. Even though some of the layer 2 packets were received, they are discarded and not delivered. The resulting stream of packets that is delivered to the user is equal to the stream originally published by the source, with all layer 2 packets removed. Despite the fact that the network has insufficient capacity, dropped packets from every layer and delivered the packets out of band, the user received an uncorrupted, deterministic subset of the stream.

4.4.4 Ignoring Packet Reordering

Although restoring the original global packet ordering before delivering the data to the user application is assumed to be appropriate for most types of data, there is content for which each packet invalidates all previous packets. This is the case for stock quotes. When a new stock quote update is received for a financial ticker symbol, it renders the previous updates useless. For most applications that process real-time financial data, only the most recent information is interesting.

When global ordering is restored, Terracast will postpone the delivery of the most recent data until all prior packets have been received also. For a typical financial application, such as a market data terminal that displays quote updates to the screen, this postponing adds little value. When the tsocket finally delivers the burst of pending updates, the application will update the symbol's last value on the screen, leaving only the last and most recent update visible and overwriting all pending updates immediately.

Because the reordering process adds additional delay to the data delivery, it can be switched off by applications that do not benefit from it. Without reordering, global (causal relations between packets from different layers) and local ordering (order of individual packets inside a single layer) is ignored and packets are delivered to the application immediately after they have been received. Disabling reordering has no impact on the reliability. Lost or delayed packets are still recovered for all layers in the reliable-list, though the recovered packets will be delivered with additional delay. Whether or not this makes them useless is up to the application to decide. For market data it is important to know whether an update is older or newer than the one previously received. This is because a stock quote may be overwritten only by a newer one.

In this case, the source could add a (logical) time stamp to each stock quote, so the receiver can decide how to handle the update[4].

Aside from configuring a layered multicast tsocket to restore global ordering or no ordering at all, a receiver can also configure a tsocket to only restore local ordering. Whether this is useful will depend on the type of application and the content, but it offers the advantage that delivery of packets from lower layers (high priority) is not delayed during recovery of packets from higher, less important layers.

If the receiver does not require the network to restore the global packet order, it is interesting to see if the expensive sequence vectors can be omitted from the packets. Of course a packet still needs to carry at least a sequence number that represents it location inside its own layer to allow packet loss detection. Unfortunately this introduces a problem. Without sequence vectors, packets only maintain state information of their own layer. If no new packets are received for a certain layer, the receiver is unable to tell whether that layer is simply idle, or whether all its packets are getting lost for some reason. As a result, the receiver cannot decide whether the layer should be dropped because it is missing packets, or kept because there is nothing wrong. Hence, it is concluded that the sequence vectors are required, regardless of the receiver's need for ordered delivery.

4.4.5 Buffer Size Considerations

In most network protocols that restore ordering and guarantee packet delivery, packets are stored in a receive buffer that has a maximum size. In TCP, for example, the receiver no longer accepts packets if its receive buffer is full. This is not a problem, as the source is notified of this event and stops sending new packets.

A receive buffer can become full either when a packet retransmission takes long, while the set of pending, reordered packets builds up, or when the user application simply does not pick up the packets by reading the socket. In Terracast, the first problem does not really exist. Since Terracast sources publish live data streams that cannot slow down or pause and because real-time data should not be buffered by the source too long, receivers will only attempt to recover lost packets for a limited period of time. Whether a packet is received, recovered or lost, its buffer slot will be freed after this time-out, resulting in a natural size limit of the receive

4 Alternatively, the receiver could manually inspect the sequence vector of the received packets. However, as the sequence vectors are generated by a lower layer in the protocol stack, relying on them in a higher layer is generally not recommended.

buffer. How big a Terracast receive buffer can get is related to the average speed of the stream and the length of the repair time-out. In principle the algorithm's implementation does not enforce a hard size limit during packet repairs.

A more troublesome situation occurs when the user application does not read data from the tsocket fast enough. When this happens, the amount of pending data builds up in the tsocket and both storage requirements and the transmission delay increases. A safe way of handling this could be to remove a layer from the intact-list, causing those packets to be discarded from the buffers, while decreasing the amount of data that is delivered to the application. Unfortunately, Terracast currently does not have a clean feedback algorithm that can keep the number of layers in balance with the application's reading speed. Instead, Terracast throws a fatal exception to the user application and closes the tsocket when the receive buffer reaches a certain maximum size, or when the total time between arrival of packets and their actual delivery reaches a configurable threshold.

4.4.6 Packet Prioritization and Multicast Congestion Control

On Terracast receivers cannot unsubscribe from the individual layers of a multicast stream to save bandwidth, because they do not are not individually addressable. As a result, packets from higher layers that are not in the intact-list continue to arrive. By monitoring the continuing reception of higher layer packets the receiver is able to detect increased network performance. Unfortunately, this also leads to packet loss across all layers when the total available bandwidth is insufficient for the entire data stream. To recover this packet loss in the layers from the intact-list, the receiver must use retransmission requests that use additional bandwidth.

To break this vicious circle, the router daemons in the network inspect the layer numbers in the packets and only forwards the lower, more important layers of each stream during congestion. This is accomplished by a special congestion control interceptor, or CCI, that was briefly mentioned in paragraph 2.6.

Heterogeneous networks that are used for dissemination of live data streams are susceptible to unfair bandwidth division among data streams when a router that lacks sufficient outgoing bandwidth on one of its links needs to drop packets. If the CCI would drop random packets during a congestion, every data stream would statistically experience the same ratio of lost packets. This is inappropriate and instead bandwidth, rather than packet loss, should be divided equally among all live data streams on a congested link, regardless of their data rate. For

instance, when two streams traverse the same congested link, both should be allowed to use 50% of the link's capacity. If one of those streams requires only 30% of that, none of its packets should be dropped. The remaining 70% is then available to the second stream. If that stream requires the total 100% of the link's capacity, 30% of its packets must be dropped, leading to a 30/70 percent resource division, where only the second, more volatile stream experiences packet loss.

The implementation of Terracast's congestion control mechanism that runs in both the inbound and the outbound packet processing pipelines of every router interface is split in two subsystems. The *stream resource allocation algorithm,* from hereon referred to as RAA, is the first and deals with equally dividing the available bandwidth over all active streams in a scalable way.

The second part of the system, the *packet prioritization algorithm*, from hereon referred to as PPA, is responsible for dropping the appropriate packets inside a stream during congestion. Once the first algorithm has selected from which stream a packet should be dropped, the second algorithm takes care of dropping the oldest packet with the lowest priority in that stream.

RAA sorts incoming packets according to their source address. For unicast packets that is the combination of router address and local tport. For multicast packets that only contain a multicast destination address – the multicast group – this group is used as criterion as it effectively represents the source address in Terracast's single-source multicast model. Each incoming packet is then added to the queue that stores messages for that data source address. If a packet was sent by a new source, a new queue is automatically created to store it.

As these queues grow, a background thread constantly dequeues packets from the queues and transmits them over the outgoing link. When the link is fast enough to handle the entire packet flow, the queues will stay empty. This is illustrated in figure 16.

Queuing and dequeuing is done independently by different threads and will be discussed separately. The responsibility of the dequeuing thread is to select the stream that has sent the least amount of bytes over the congested link so far and to dequeue the most urgent packet from that stream's packet queue. The queuing thread is in charge of creating queues and storing packets and it is this thread that drops packets when the queues have reached a configured size threshold.

An important requirement for the congestion control mechanism is that it should be able to work with very large numbers of parallel data streams. This prohibits the system from keeping state information on the history of every data stream. Instead, the algorithms rely on

probability when selecting packets for transmission or disposal.

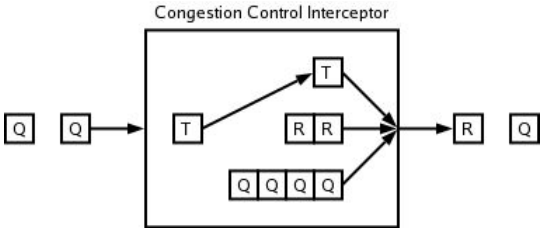


Illustration 16: The congestion control mechanism organizes pending packets according to their source address and buffers them in separate queues.

When the dequeuing thread selects a packet for transmission, the algorithm only looks at the first (most urgent) packet of each queue. To make sure each stream gets an equal share of bandwidth, it takes the individual packet sizes into account. When all packets – notice that only the first packet of each queue is observed – have equal sizes, the dequeuing thread simply selects a random queue and dequeues and transmits its first packet. Since each queue has the same probability of being selected, each stream will eventually forward an equal amount of bytes per second.

Since packets can have any size between 1 and MTU bytes (Terracast's current MTU is 8192 bytes), their size must be considered when selecting a stream for dequeuing. Queues with many large packets should have a smaller probability of being selected for transmission than queues with many small packets, in order to keep the bandwidth division fair. As the dequeuing thread only considers the first packet of each queue, its probability of being selected is inversely proportional to its size in bytes.

For instance, assume the first packet in queue T in illustration 16 is 100 bytes, the first packet from sender Q is 300 bytes and the packet in R is 500 bytes. To calculate the selection probabilities, first their selection weights, or importance, are calculated. The weight of the packet in P is computed by dividing the total size of T, Q and R by T's size: (100 + 300 + 500) / 100 = 9. Likewise, Q's weight is 900 / 300 = 3 and R's weight is 9/5. The weights are converted into selection probabilities by dividing them by the sum of all weights combined. This gives T a selection probability of 9 / (9 + 3 + 1.8) ≈ 0.65, Q a probability of 0.22 and R of 0.13. The formula for calculating the weight of a queue *m* is given in (1):

$$w_m = \frac{\sum_{i=0}^{n} s_i}{s_m} \tag{1}$$

where *s* represents the size of the first – most urgent – packet of queue *m* and *n* the total number of queues. The formula shows the inverse relation between packet size and selection weight.

The probability of queue *m* being selected to transmit its most urgent packet is then directly derived from the absolute selection weights:

$$P_m = \frac{w_m}{\sum_{i=0}^{n} w_i} \tag{2}$$

Equations (1) and (2) can be substituted for the formula that can directly calculate the weighted queue selection probabilities:

$$P_m = \frac{\sum_{i=0}^{n} s_i}{\sum_{j=0}^{n} \left(\frac{\sum_{h=0}^{n} s_h}{s_j} \right)} \tag{3}$$

To keep the delay under control that is inherently associated with temporarily buffering packets in queues, the system enforces a maximum total queue size. The sum of the sizes of all packets in all queues must never exceed this threshold. Enforcing this threshold is the responsibility of the queuing thread.

When a new packet is received, it is always accepted by the queuing thread. If necessary, a new queue is created to represents its source address, otherwise it is appended to its designated, existing queue. After the packet is added, the combined size of all queues is calculated. If it exceeds the configured maximum, the queuing thread runs a removal round in which it first selects a queue and tells that queue to drop its least urgent packet. If the queues are still too big after a packet was removed, another removal round is run until the threshold is met. Shrinking queues is always done after a packet was added, never preemptively. The reason for this is to let the new packet immediately participate in the removal selection process.

In contrast to dequeuing packets, where the algorithm attempts to select the most urgent queue, the packet removal process needs to find the least urgent one. This packet is found by considering the size of the individual queues. The largest queue should generally be shrunk to keep resource division among the streams fair.

Aside from the queue size, another selection criterion is how accurately a queue can match the required buffer threshold. This follows from the fact that individual packets can greatly differ in size, so removing a packet from queue T may free eight kilobytes, while removing a packet from Q may yield 40 bytes for example. If the queues in total exceed the maximum size by only a couple of bytes, it seems logical to remove the small packet from Q. This policy comes with the consequence that it implicitly encourages the use of larger packets. A queue with many small packets is more likely to accurately match the required amount of buffer space than a queue with few big packets. At the time of writing, insufficient testing was done to determine the true effect of this property, but encouraging application developers to use larger packets often has a positive effect on network efficiency and throughput. Therefore, when selecting a queue to shrink, the algorithm favors large queues that will expunge the least amount of bytes in order to match the maximum buffer capacity.

Before the weighted queue selection can be made in the removal round, the absolute weight factor for each queue is calculated. The weight factor combines the queue's size and ability to accurately match the total overcapacity. The latter is expressed in v. It represents the sum of the sizes of the packets that a queue must remove in order to eliminate the overcapacity.

For example, if queue T has 10 packets of 100 bytes each, while the entire buffer currently has an overcapacity of 170 bytes (suppose the maximum tolerable size is 10000 bytes, while all queues combined add up to 10170 bytes), T would need to remove at least 2 packets to bring the total size down to a tolerable value. In this case, v_T is 200. The effect queue size and v have on the selection weight of a queue m is expressed in the following equation:

$$w_m = \frac{\left(\sum_{i=0}^{n} v_i\right) \cdot s_m}{v_m} \tag{4}$$

In this formula, n represents the total number of queues, v_m the number of bytes that would be removed if m was selected, while s_m represents the total size of m. The formula captures the linear relation between the weight and the queue size, as well as the inverse relation between weight and the minimum amount of bytes that would be expunged.

The absolute weight factors of all queues calculated with (4) can be converted into weighted selection probabilities by dividing the absolute weight factor by the sum of all weights. This is expressed in (5).

$$P_m = \frac{w_m}{\sum_{i=0}^{n} w_i} \tag{5}$$

Here P_m represents the selection probability of queue m. Now (4) and (5) can be merged into (6) that directly computes the weighted selection probability P for a queue m.

$$P_m = \frac{\frac{\left(\sum_{i=0}^{n} v_i\right) \cdot s_m}{v_m}}{\sum_{h=0}^{n} \left(\frac{\left(\sum_{j=0}^{n} v_j\right) \cdot s_h}{v_h} \right)} \tag{6}$$

When the selection probability for each queue has been computed, a roulette-wheel algorithm[5] is run to select a queue and let that queue remove its v bytes. If v is smaller than the current overcapacity (note that this implies that the selected queue purged all its packets and was automatically removed), another round is done until the queues are shrunk sufficiently. In Terracast's congestion control interceptor the queue selection process is implemented as an atomic operation. During selection rounds, no new packets may be added to or dequeued from the queues.

So far the queuing and dequeuing algorithms have been explained that together form the aforementioned stream resource allocation algorithm RAA for equally dividing available bandwidth among parallel streams. The remainder of this paragraph will focus on the packet prioritization algorithm PPA that works together with Terracast's layered multicast to ensure a stream's available bandwidth is only used to transmit packets from the most important (lower) layers unless the link allows all packets to be transmitted.

PPA works by sorting the pending packets inside a queue by their priority value and arrival order. This leads to a queue configuration as depicted

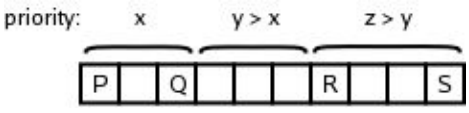


Illustration 17: Prioritized packet queue.

5 The roulette-wheel algorithm divides an imaginary wheel in slices (each representing one selection option) with sizes proportional to their probability value. The wheel is then virtually spun once to make a selection. The algorithm is discussed in [EIB03].

in illustration 17. In this illustration four packets that occupy interesting slots have been labeled. When the dequeuing thread has selected a queue, it will dequeue the packet whose size was used during the selection probability computation. This is the packet in slot S. This is the oldest packet with the highest priority. When a new packet is queued, it is appended to the leftmost slot of the sub-queue that represents its priority. For example, assume a new packet with priority z is placed in slot R. The packet that occupied the slot until then, together with all other packets on the right of slot R, are shifted one position to the right, causing the queue to grow one slot. When a packet is queued with a new priority value, a new sub-queue is automatically created and merged in the queue so that all priorities are sorted in incremental order.

While the individual sub-queues have no hard limit on their size, the queue as a whole, together with all other queues may not exceed the unit's maximum buffer size. When a queue is selected for shrinking by RAA, it will always purge packets from left to right, dropping the oldest packet with the lowest priority. When the a sub-queue becomes empty, it is automatically removed.

When packets with priority z are queued at the same rate as the queue is read by the dequeuing thread, it is possible that the packets with priority < z get stuck in their sub-queue for a significant period of time. This is detected by the downstream receivers through the sequence vectors and causes them to remove those layers from their reliable-list and not wait any longer for those packets to arrive. Because the congestion control interceptor does not inspect the sequence vectors of queues packets itself, it does not know when the receivers stop waiting for these pending packets. When the network's capacity finally increases, the old, obsolete packets will be dequeued and transmitted after all. Although this was not studied, the loitering packets occupy space in the queues and are expected to have a negative influence on the capacity of the other stream queues and their selection weights. When this is experimentally proved, it may be necessary to add a mechanism that can detect loitering packets and purge them. Notice that purging loitering packets after some configurable interval is also necessary to avoid echoes of old packets ending up in new streams when sequence numbers have wrapped. This issue was previously raised in paragraph 4.3.5.

5. LAYERING REAL-TIME MARKET DATA

The previous chapter discussed how layered multicast works and how it can be used to deliver a multicast data stream to a large number of subscribers through a heterogeneous network, where every receiver gets the data in the highest quality supported by its network connection. It showed that in order to use layered multicast, one must have the ability to packetize the data stream and scatter the data packets across multiple layers in such a way that the layers form an incrementing data set. While the base layer contains an uncorrupted, low quality representation of the original data, the information from each additional layer takes the base layer to a higher quality level.

The canonical example of a layered multicast stream is video. The server sends the base layer – the bare minimum of data that will allow a grainy picture and sound – at, say, 28.8 kilobits per second to all users. At the same time, the server sends enhancement layers of eight kilobits each, which provide additional video resolution and sound quality. The receivers subscribe to as many of these additional layers as they can get through their network connection.

This chapter discusses a layered data stream encoder for real-time market data. This encoder runs on the Reliable Ordered Layered Multicast Protocol, or ROLMP that was introduced in paragraph 2.3 and further discussed in paragraphs 4.4.1 to 4.4.4. The encoder divides a stream of real-time stock quotes over one or more layers in such a way that when a subscriber receives only a subset of the layers, it will still get a consistent and up to date view of the market without the risk of stale symbols for which the most recent update was missed. Layering techniques for audio and video will not be discussed as that has already been the subject of previously mentioned research, including [CAN97], [BAB00] and [TAUB94].

5.1 Encoding The Base Layer

Before the complete encoding algorithm is described, a leaky bucket algorithm is described that can trim an incoming stream of market data to a fixed bandwidth stream. This is useful for encoding the base layer of the layered stream. This algorithm will then be extended to produce layers containing incremental information that together contain the same data as the original stream.

Volatility of the symbol is an important criterion in deciding whether a certain quote update is important enough to be forwarded by the leaky bucket, or whether it should be discarded in favor of another update. When insufficient bandwidth is available to forward all quote updates, the updates for the most volatile stocks should be dropped in favor of the less volatile ones.

The leaky bucket algorithm works by arranging the individual symbols in a circular, linked list. Every slot in the list represents the most recent update for its symbol. Every time a new update is received, it is stored in the appropriate slot, overwriting the previous update. Thus, the linked list always contains the most recent trade for every symbol.

A virtual token traverses the list at a fixed speed. If the stream is to be thinned to one update per second, the token will visit one slot every second. When the token visits a slot, it removes the quote update and forwards it, leaving an empty slot. Empty slots will be skipped by the token without delay.

When one of the symbols has two updates per second in the incoming stream, the second incoming update after the last visit of the token will overwrite the update pending in the slot and the older pending update is dropped. This allows the leaky bucket to provide the clients of the thinned stream with the most recent, update of every symbol.

The algorithm is illustrated in figure 18a. While the virtual token visits the slots that have a recent quote update pending to be sent out, the left arrow represents a thread that receives the incoming market data stream and inserts new trades in the slots. To limit the outgoing transmission rate to one quote per second, the token thread (represented by the right arrow) sleeps for one second after visiting a slot. If the token is able to complete one full circle between two incoming updates of every symbol, there will be no data loss.

Instead of limiting the outgoing bandwidth to a fixed maximum, the dequeuing thread could also be configured to run as fast as possible, relying on external flow control such as a blocking write call to the network to reduce publishing speed. This could be particular useful when the algorithm is used on the server side of a point-to-point TCP connection to a client application. In principle the server could use the algorithm to send the data at full speed, until network saturation slows down the TCP connection.

The algorithm that encodes the base layer introduces two potential problems. The first is that the original order of the updates is lost. In fact, if every symbol updates one or more times while the token circles the list, the token will find every slot filled with a quote update. Then the order of the output stream is determined by the fixed order of the slots in the linked list. This is

usually not a problem for quote updates of different symbols as they are not directly related to each other.

A second, more problematic issue is that of additional delay, which is the result of the fixed period of time the token waits after every transmitted update. When a new incoming update is inserted in the list, it will have to wait for the rotating token to reach it, before it gets sent out. The time it takes the token to reach the new quote update depends on the number of updates waiting in the slots between the token and the new update, as well as on the token's rotational speed. In the worst case scenario this equals the total number of slots in the ring, divided by the update frequency or rotational speed.

$$t_{max} = \frac{N}{f_{update}} \tag{7}$$

Where t_{max} is the maximum delay in seconds that the algorithm can add to a quote update that is forwarded, f_{update} the update frequency, or rotational speed of the token in updates per second and N the total number of slots in the ring.

5.2 Encoding The Enhancement Layers

While the above algorithm provides a good way of thinning a stream of live market data with varying bandwidth to a fixed maximum rate, without the risk of stale symbols, a more advanced algorithm is needed to produce the enhancement layers that form a layered market data stream. A straightforward way of doing this is to have several instances of the algorithm running that each receive a copy of the input stream, letting their token run at different speeds and tagging their updates with a layer number. The layers could then be combined into a single stream. Unfortunately this introduces a lot of redundant information in the stream as the higher layers contain updates that are also available in the lower layers. Although wasteful, this technique of encoding an entire stream in different qualities and sending them to clients concurrently to meet their individual bandwidth capacities is widely used on the Internet for streaming video and is known as *simulcasting*. One overlay product that relies on simulcasting is [CHU00], which is the subject of paragraph 6.2.

A more efficient approach is to have each layer contain only information that is not already present in the lower layers, with no redundancy among them. To achieve this the leaky

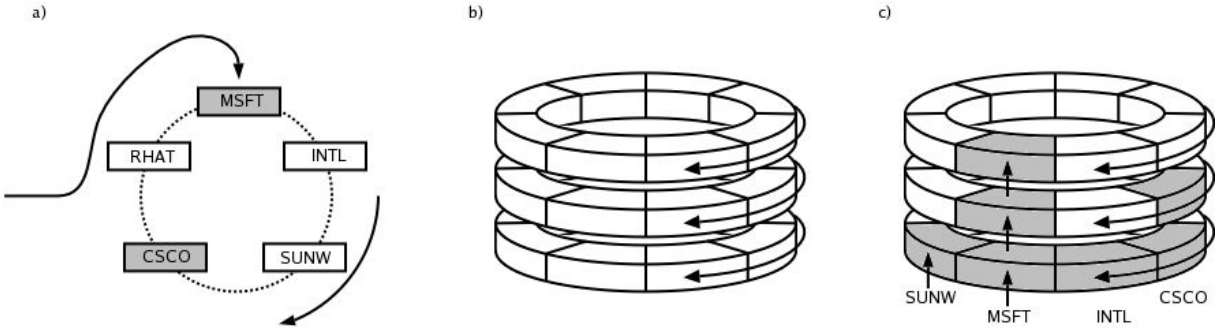


Illustration 18: Figure a: thinning a stream of live market data. The gray slots represent symbols for which a recent quote update was received, while the transparent slot are empty. Figure b: when the algorithm of figure a is vertically stacked, each instance represents a stream layer and each symbol has a column that can store more than one update. Figure c: the stacked algorithm of figure b at work. The gray slots contain a pending quote update, while the transparent slots are empty. The illustration shows how the updates of a volatile symbol are spread out over several higher layers.

bucket algorithm is extended to a multi-level ring as illustrated in figure 18b, creating a virtual cylinder of vertically stacked, circular lists, where each ring represents a layer of the output stream. Each ring has its own token. Although it is not compulsory, the tokens all circulate their ring at the same speed. The data structure also identifies columns. A column is the collection of vertically stacked slots that can all store an update for the same symbol.

The algorithm has a single input thread that receives the original stream. This thread is not illustrated in the figure. Newly received updates are always inserted in the lowest ring. When a new update arrives for a slot in the lowest ring that already contains an update for that symbol, this older update is vertically shifted into the next ring, so that the lower slot can always store the latest update. Once an update is shifted upwards, it cannot return to a lower ring anymore, even if all other slots beneath it were emptied.

Symbols that have more than one trade during the time it takes the token to complete a full rotation of the ring will be spread out to two or more higher layers, while symbol that have very infrequent trades will usually only occupy a single slot in the bottom ring. The result of this is depicted in figure 18c. It shows the algorithm using three layers to thin a market data stream that contains updates for eight symbols. Four symbols are included in the figure and assigned to the four visible columns in the front. The figure also shows that all four symbols have pending updates in the bottom ring, while MSFT is so volatile that it shifted updates into all three layers before the tokens could dequeue any of the pending updates.

When a symbol is so actively traded that it manages to shift an update out of the top ring, that update is irrecoverably lost. If required, loss of updates can be avoided by letting the token

in the top ring run without dequeuing delay, letting it empty every slot immediately after an update was queued. The result of this is that the transmission rate of the upper layer will equal the transmission rate of the original stream, minus the combined rate of all lower layers. Because the transmission rate of the lower layers have a fixed maximum, the upper layer's rate will fluctuate and follow the variations of the original stream.

Note that when the tokens all rotate at the same speed, this does not imply that they visit the same symbol slots at the same time. Instead, they will usually be at different positions in their ring. This is is a result of empty slots that are skipped without waiting.

Since the layered algorithm is based on the leaky bucket algorithm from figure 18a, it has the same drawbacks, namely out of order transmission and additional delay. Every ring can contain an update for the same symbol at a certain time and because the tokens can have different column locations inside their ring, it is undefined which ring will dequeue and transmit its update first, who will be second and so on. This means that not only the order of symbols is shuffled, but even the updates for a single symbol are no longer guaranteed to be transmitted chronologically.

While new updates for a symbol are queued, shifted and sent out, in some cases an update can survive in the rings substantially longer than its younger successors. This can occur when an update that is just about to be dequeued by the token, gets shifted to the next layer by a more recent update that entered through the lower ring. In this case it is likely that the more recent update is dequeued sooner than the older update that now has to wait for the token of its new ring to reach its slot. To address this problem, the system needs to provide a way of putting the outgoing, layered updates back in the original, chronological order. This is done by labeling incoming updates with sequential numbers and have the tokens pass their dequeued, labeled updates to a transmit window that uses the sequence numbers to restore the chronological order. When updates are shifted out of the upper ring – and therefore considered lost – the transmit window is notified not to wait for their sequence numbers.

When applying order reconstruction across all rings, the transmit window must have the same capacity as all rings combined: $size_{tw} = N \cdot j$, where $size_{tw}$ is the maximum capacity of the transmit window, N the number of symbols in the original stream (and therefore the number of slots per ring) and j the number of layers used. The potential size of the transmit window, together with its bursty behavior when a delayed quote update releases a substantial amount of pending updates from the window, may justify a mechanism that applies order reconstruction to

achieve chronologic transmission per symbol only, similar to the output of the base layer encoder.

Like the algorithm from figure 18a, the layered system introduces delay. Since the model is essentially a collection of instances of the first algorithm, its worst case delay is the sum of each ring's worst case delay:

$$t_{max} = \sum_{i=0}^{j} \frac{N}{f_{update}(i)} \tag{8}$$

where t_{max} is the maximum delay in seconds that the algorithm could introduce for a single quote update, N the number of symbols in the original stream, j the number of used layers and $f_{update}(i)$ the update frequency of the token in layer i.

Thinned market data streams are not suitable for all applications. The calculation of VWAP for example, which was already mentioned in paragraph 4.3 requires the reception of all trades over a particular interval. Another example is a trading application that monitors the order book of an instrument. Still, under some circumstances, Terracast's reliable layered multicast can be used to support these demanding applications. One way is to encode all updates of a particular instrument in the same layer, rather than scattering them across all layers as done in figure 18c. Because Terracast guarantees that stream layers are delivered without packet loss, the subscriber can rely on the information to compute derived information such as WVAP. The consequence of course is that when there is not enough bandwidth available to receive all layers, one or more higher layers are automatically dropped and no single update is delivered for the instruments encoded in these layers. When this happens, the subscriber is notified to avoid stale or incorrect computations, while the impact is limited to the symbols from these higher layers.

For VWAP calculations this is better than the default layered encoding that decreases the quality of all instruments with every layer that is dropped. Which of these two encoding schemes is best, again depends on the type of application and content.

6. RELATED WORK

Terracast is a relatively large project that covers a number of network research areas and combines several algorithms and ideas to offer an end-to-end solution to live multicast over the Internet. Although there are more products and solutions that offer end-to-end solutions, it is often difficult to compare them, as each has its own unique focus. This also holds for Terracast. While it studies solutions for live multicast, it does not focus on distribution of audio or video, but market data. The distinct properties of real-time market data influenced the design and application of algorithms for Terracast, making it difficult to compare it to existing work.

It would be possible to take multipoint video distribution and see how Terracast compares to solutions that were tailored to video, or instead do this with live market data. However, such comparison would not be very useful. There are many types of multicast data that all benefit from different qualities of a distribution platform and so far no single product has emerged that can handle all types equally well. Therefore this chapter will not attempt to compare Terracast as a whole to other products, but instead discuss a number of solutions that look similar to Terracast. These discussions attempt to emphasize the strengths and weaknesses of the products and further understate that no single product fits all needs.

It is interesting to take a close look at the key algorithms and techniques used inside Terracast's components and compare these to similar existing algorithms. In most cases this will be more practical than comparing the product as a whole. Candidates for such comparisons are Terracast's dynamic unicast routing layer with address summarization, or Terracast's combination of reliable multicast and stream layering. However, because this thesis has constantly discussed previous work immediately in the text when a particular algorithm or component of Terracast was described, an additional, separate set of comparisons would be redundant. Hence, the remainder of this chapter is limited to discussing a number of overlay multicast solutions with a similar scope.

6.1 Scattercast

Scattercast [CHA00] is probably the overlay solution that comes closest to Terracast, except for the fact that it does not use layered multicast. Like Terracast, it offer a complete overlay network based on software router daemons that run their own dynamic routing

algorithm. Scattercast focusses on the multicast distribution of live content and also acknowledges the problems of network heterogeneity. To solve this, Scattercast takes very different approach than Terracast. While Terracast uses data layering to allow individual subscribers to receive the content in the highest possible quality, Scattercast uses content specific code to re-encode, or transcode the data stream in a lower quality during congestion. When Scattercast is used to disseminate a live MPEG video stream from one source to many receivers, the overlay network constructs a distribution tree that connects the receivers to the source and then multicasts the data packets over the tree. A source will typically encode the MPEG stream in a high quality to suit subscribers with fast network connections. As the video stream traverses the distribution tree, it may encounter routers that are in state of temporal or permanent congestion and have insufficient resources to forward the data stream over its child links. To circumvent packet buffers that keep growing while the congestion lasts, Scattercast has an MPEG transcoder embedded in each router that will transcode the incoming video stream to a bit rate that the outgoing child links can support. Because each router only transcodes data streams when necessary, each subscriber will receive the video stream in the highest possible quality.

To ensure that Scattercast is not limited to MPEG streams alone, the router daemons were designed to support the addition of new transcoders through a flexible plug-in architecture. This allows administrators to add transcoders for various audio and video formats, as well as custom data streams used by software for online collaboration. An example of the latter, which is also used in the Scattercast paper, is a shared whiteboard application that applies progressive JPEG encoding to the images drawn on the whiteboard. The transcoder in this case is a JPEG converter that can send only a selective number of JPEG scans during congestion. The net result is that all participants receive timely feedback of whiteboard modifications even when network capacity fluctuates. This is very similar to what Terracast attempts to do with layered multicast.

Although the Scattercast paper does not cover live market data, it is possible to build a transcoder for market data and plug it in the router daemons. Building such transcoder based on the algorithms discussed in chapter 5 is relatively simple. Scattercast does not use layered streams, so an instance of the leaky bucket algorithm that is used to encode the base layer in paragraph 5.1 would suffice.

Little is known about the actual performance of Scattercast, so it is difficult to say if Terracast's reliable layered multicast is a better approach. What is known however is that the need for each router daemon to understand the encoding of every stream through a set of plug-in

modules makes the network inflexible and unable to support obscure encoding formats that are only used and understood by the specialized software that use them, unless the users of these data formats write their own Scattercast transcoders and have sufficient privileges to add them to the router daemons. Market data would fall in that category.

There are more pitfalls when using these application specific transcoders. For example, when a transcoder contains a bug, it could have substantial complications for the health of the router. Another potential problem is that transcoding a multimedia data stream is relatively hard work that requires powerful computers. This is worse for router daemons, that may need to transcode many concurrent streams at the same time. When this task is done by the routers, it may significantly increase the hardware requirements.

A Terracast router does not understand the contents of a data stream. The actions it has to take during periods of congestion are the same for every type of data stream and limited to a number of simple prioritization computations. This makes the Terracast router daemons less complex than their Scattercast counterparties with more modest hardware requirements.

6.2 End System Multicast

End System Multicast, described in [CHU00], [CHU01] and [CHU04], is an overlay network targeted to supporting the multicast of mostly multimedia content across the Internet. To this end it creates a virtual network of unicast connections between the participating nodes, creating a mesh of software routing daemons. This is similar to Terracast, yet the mesh management and routing protocol used in End System Multicast, called Narada, autonomously constructs the mesh without the assistance of an administrator. Also, Narada is capable of adjusting the topology of the mesh to fit the changing set of participants and fluctuating network capacity. It is worth noting that, in contrast to Terracast, Scattercast's mesh management and routing protocol, called Gossamer, also offers this powerful feature.

On this mesh, Narada creates shortest path trees that act as distribution paths for multicast traffic. Narada meshes limit themselves to participants of the same multicast group. This is different from Terracast that creates a single overlay mesh that is used to support independent, concurrent multicast transmissions. The difference between these two approaches is that software routers in Terracast often relay data streams for which they have no direct interest themselves. Yet they forward these streams because they are in the shortest path between the source and one or more subscribers. When an overlay network is built entirely from end user

workstations on relatively slow connections, it is better to let workstations participate only in the distribution of data they need themselves. Terracast's focus has been on offering a managed overlay network, whose software router daemons are hosted on dedicated machines at strategic places across the Internet. In such set-up it is better to build a shared mesh where all streams can benefit from the dedicated routers. Again, this difference in focus understates the difficulty of comparing one system to the other.

End System Multicast addresses the problem of network heterogeneity in a simple but pragmatic way through simulcasting. By default, it always publishes both a high quality and a low quality version of the same stream. Aside from the audio, which exists only in a single quality that is used by both video streams, the video streams are completely independent. New participants on the overlay initially subscribe to the high quality video stream, as well as the audio stream. When the receiver detects too much packet loss, it automatically subscribes to the low quality stream, without unsubscribing from the high quality stream. Further, to ensure the delivery of the audio stream and the low quality video stream, those streams are prioritized over the high quality stream. Therefore, this ensures that bandwidth-limited nodes are still able to participate in the broadcast without sacrificing video quality for all other nodes.

Narada does not come with packet recovery mechanisms that are used to repair occasional packet loss. This makes the system less suitable for use with applications that are sensitive to data loss such as trading applications and statistical analysis of market data. To minimize the problem, Narada employs a similar technique as Terracast's multicast protocol: during changes made in the distribution tree of a multicast stream, router daemons continue to forward data packets over the old links for some time until the routing tables have fully converged. Of course this cannot avoid packet loss altogether, as packet loss can also be the result of a crashing router daemon.

7. EVALUATION

The Terracast project has been a research project at MarketXS to determine if the public Internet can be a feasible medium for the professional distribution of real-time market data. This has lead to the design of the application level multicast solution described in this thesis. To conclude whether or not Terracast lives up to its expectations and proves a useful tool for cheap distribution of market data, the system must be implemented completely and deployed in a realistic environment. Unfortunately, at the time of writing, the system was still under construction and not fit for extensive testing and performance analysis. As a result, there are no performance figures to either confirm or contradict the expectations.

During the development of the routing algorithms, the system was tested on a cluster of workstations in a controlled laboratory environment. The main purpose of these tests were to verify the correct behavior of the unicast routing algorithm and its address summarization mechanism. The unicast routing layer is based on the ExBF protocol which uses recursive backtracking techniques to avoid routing loops by inspecting the entire path to each node prior to advertising them. However, Terracast's address summarization that is used to keep the routing tables small and the number of exchanged routing updates minimal, even when the network scales to tens of thousands of nodes or more, makes it impossible to track the entire path to any given destination by inspecting only the local routing table. Yet, it is crucial that ExBF's functionality is not lost and the routing layer's implementation was therefore tested with many different topologies and failure scenarios. With the help of dedicated machines from Telerate, the University of Kent at Canterbury and various colleagues at MarketXS, it was possible to build a limited, wide area overlay topology with nodes in various continents that experience realistic network performance fluctuations. This network was run day and night to look for routing inconsistencies, but none were found.

The reliable layered multicast algorithms, as well as the application programming library that interfaces with user applications were not stable enough to provide a serious load on the wide area overlay network. Instead, the network has been used for multimedia multicasts without data layering. This was done by connecting the output of popular MP3 streaming software to Terracast, packetizing the stream and publish the data packets to a Terracast multicast group. Getting the stream off Terracast and into a music player was done by writing a small application that subscribes to the multicast group on Terracast, while forwarding the audio

stream to the client through an HTTP connection. HTTP streaming is supported by most media player software. A similar approach was taken for MPEG video. At the source, an MPEG file was streamed with the open source vlc program from VideoLAN [VLC05] that was configured to service clients through HTTP. A small program was written that connects to vlc's HTTP port, packetizes the video stream and published the packets on Terracast. On the client side, the same program that converted the MP3 stream from Terracast to HTTP was used to relay the MPEG stream to a local vlc instance.

As these tests were done without data layering and the packet recovery features it offers, the result was a video stream that came in bursts during congestion and missed packets while the router daemons were changing the routing tables, or when congestion caused a router's packet buffer to overflow.

8. CONCLUSIONS

The system presented in this thesis belongs to the popular research category of multicast platforms for wide area IP networks where many systems have attempted to alleviate some of the issues of the plain multicasting primitives that currently hinder serious application development. Some have been more successful than others, but so far no single system managed to offer a solution generic enough to support all applications.

Many people see IP multicast networks as a way to use the cheap, publicly accessible and versatile Internet for distributing content that is traditionally handled by specialized networks only. However, there is no such thing as a free lunch. One of the reasons why the use of IP networks can be relatively cheap, is because they use packet switching and packet switching was developed specifically to allow communication networks to be concurrently shared by many users. Many applications do not require a dedicated amount of resources the entire time and a system that does not allow resources to be shared looses a lot of valuable capacity to applications that leave the communication medium idle for long intervals. The traditional telephone system that was mentioned in chapter 1 is a good example of this. Before the introduction of digital techniques, the telephone system reserved a dedicated wire for each conversation, wasting precious bandwidth if the wire was idle when nothing was being said for some time.

The introduction of packet switching in digital networks allowed communication infrastructures to be used much more efficiently. While this makes communication cheap and accessible, it inherently meant the loss of guaranteed capacity, causing communication to slow down when the network is busy. For typical applications such as file distribution, fetching a web page or sending email, this is slightly inconvenient at worst. As the Internet's bandwidth, reach and accessibility kept increasing, people started to look for ways to use it for new applications like video broadcasts that still use their specialized dedicated infrastructures.

Unfortunately the shared, heterogeneous nature of the Internet does not suit the live streaming data of video broadcasts, which means that additional protocols are necessary to avoid these streams from slowing down when the network is busy.

Terracast has been designed for the distribution of real-time market data in a professional environment and market data, like video, is high-volume, live content. However, because market data has more stringent requirements regarding reliability and latency than video, and

because most existing systems implicitly or explicitly focus on video distribution, these systems were considered unusable and a new platform was built instead.

Like most overlay solutions, Terracast contains a dynamic routing algorithm that allows it to find and constantly optimize the shortest routes in terms of end-to-end latency on the overlay topology that was chosen by administrators. Because the Terracast network is expected to be run as a large collection of decentralized, separately administered, interconnected sub networks similar to the Internet itself, it comes with a hierarchical addressing scheme that allows the network to grow to very large numbers of nodes, without giving up local administrative control. Unfortunately, as the software had not yet reached a functional version, it was impossible to verify the overlay's expected behavior.

When it comes to multicast protocols, Terracast uses a combination of data layering and reliable multicast techniques to offer an end-to-end service for distributing live data streams to a large number of subscribers over a heterogeneous network. This mechanism that cooperates with the congestion control and packet recovery components inside the router daemons, ensures that live data is not delayed and that each receiver is guaranteed to receive a consistent representation of the data that is free of corruption at all time. The latter is rarely seen in other overlay solutions that deal with video that has a certain level of resilience to data corruption. Again, at the time of writing the software had not reached a state that allowed extensive testing for verification.

A system that uses layered multicast requires applications to pre-process their data stream with a specialized layered encoder. Although layered multicast has not yet been generally accepted and used, layered encoders have already been developed for common digital audio and video formats. As Terracast focusses on market data, it needs its own special layered encoder. This encoder was introduced in chapter 5. Evaluating the encoder was difficult as Terracast does not yet allow for a realistic end-to-end test with real applications. However, as the part that encodes the base layer can also be used in unicast transmissions, it was built early on in the project and used in several mission critical parts of MarketXS' Finance Engine product where it continues to be an indispensable component.

Terracast is not yet in production. It currently does not reliably ship thousands of stock quotes per second around the Internet where they are delivered to large numbers of customers free of data corruption and with minimal transmission delay. A lot of work still needs to be done to reach this state. Nevertheless it is already possible to point out interesting areas where the product could be improved in the future.

One known problem of Terracast's multicast routing algorithm is that it is based on reverse path forwarding techniques where distribution trees are constructed according to link cost information in the opposite direction. This leads to sub-optimal distribution trees on networks that use asymmetrical links. This is particularly problematic on networks with a high number of ADSL and cable modem connections. Hence, it may be useful to see if Terracast can be changed to use an alternative tree construction algorithm.

Terracast's interface modules constantly re-evaluate the status and performance of each virtual network link. They do this by timing a small packet that is being echoed back by the peer. The consequence of this approach is that only latency is measured while bandwidth is ignored, while links that have a very low latency are not guaranteed to offer high bandwidth and vice versa. Solving this properly is not a triviality as different types of content may have different performance requirements. Market data may prefer low latency over bandwidth, while video may do the opposite and accept some latency in exchange for more bandwidth.

An important distinction between Terracast and existing overlay solutions such as Scattercast and End System Multicast, is that Terracast is not capable of autonomously building an efficient overlay mesh. Instead, this task is left to administrators. If an incorrect mesh topology is chosen, it will impact the performance that is achieved by the routing algorithms that run on top of the overlay. Currently, developing an autonomous overlay construction algorithm for Terracast is the subject of a research project by Ming Au at the Technische Universiteit Delft under the supervision of professor Dick Epema.

BIBLIOGRAPHY

[ADAM04]: A. Adams, J. Nicholas and W. Siadak, *Protocol Independent Multicast - Dense Mode (PIM-DM)*, Internet Engineering Task Force, 2004

[AND01]: D. Andersen, H. Balakrishnan, F. Kaashoek and R. morris, *Resilient overlay networks*, Proceedings of the 18th ACM symposium on Operating Systems Principles, 2001

[ATM96]: The ATM Forum technical committee, *Private Network-Network Interface Specification Version 1.0*, The ATM Forum, 1996

[AWER92]: B. Awerbuch and D. Peleg, *Routing with polynomial communication - space trade-off*, SIAM Journal on Discrete Mathematics, Volume 5, 1992

[BAB00]: F. Babich and M. Vitez, *A novel wide-band audio transmission scheme over the Internet with a smooth quality degradation*, ACM SIGCOMM Computer Communication Review, Volume 30, Issue 1, 2000

[BAL93]: T. Ballardie, P. Francis and J. Crowcroft, *Core based trees (CBT)*, ACM SIGCOMM Computer Communication Review, Vol 23, 1993

[BAN02]: S. Banerjee, B. Bhattacharjee and C. Kommareddy, *Scalable application layer multicast*, ACM SIGCOMM Computer Communication Review, Volume 32, Issue 4, 2002

[BAR64]: P. Baran, *On Distributed Communications*, RAND Corporation, 1964

[BERT87]: D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, ISBN 0132009161, 1987

[BHA03]: S. Bhattacharyya, Ed., *RFC 3569 - An Overview of Source-Specific Multicast (SSM)*, Network Working Group, 2003

[BY02]: J. W. Byers, M. Luby and M. Mitzenmacher, *A Digital Fountain Approach to Asynchronous Reliable Multicast*, IEEE Journal on Selected Areas in Communications, Volume 20, 2002

[CAN96]: S. McCanne, V. Jacobson and M. Vetterli, *Receiver-driven layered multicast*, ACM SIGCOMM Computer Communication Review, Volume 26, Issue 4, 1996

[CAN97]: S. McCanne, M. Vetterli and V. Jacobson, *Low-Complexity Video Coding for Receiver-Driven Layered Multicast*, IEEE Journal on Selected Areas in Communications, vol. 15, no. 6, 1997

[CAS96]: I. Castineyra, J.N. Chiappa and M. Steenstrup, *The nimrod routing architecture*, Nimrod Working Group, 1996

[CHA00]: Y. D. Chawathe, *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*, Ph.D. Thesis, 2000

[CHE89]: C. Cheng, R. Riley and S.P.R. Kumar, *A Loop-Free Extended Bellman-Ford Routing Protocol Without Bouncing Effect*, In Proceedings of the ACM SIGCOMM, 1989

[CHIU98]: D.M. Chiu, S. Hurst, J. Kadansky and J. Wesley, *TRAM: A Tree-based Reliable Multicast Protocol*, Sun Microsystems Laboratories Technical Report Series, TR-98-66, 1998

[CHU00]: Y. Chu, S.G. Rao, and H. Zhang, *A Case for End System Multicast*, Proceedings of ACM SIGMETRICS, 2000

[CHU01]: Y. Chu, S. G. Rao, S. Seshan and H. Zhang, *Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture*, Proceedings of ACM SIGCOMM, San Diego, CA, 2001

[CHU04]: Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan and H. Zhang, *Early Experience with an Internet Broadcast System Based on Overlay Multicast*, USENIX Annual Technical Conference, Boston, MA, 2004

[COR90]: T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms (MIT Electrical Engineering and Computer Science*, MIT Press, ISBN 0262031418, 1990

[COW99]: L.J. Cowen, *Compact routing with minimum stretch*, Proceedings of the tenth annual ACM-SIAM symposium on discrete algorithms, 1999

[DEE85]: S.E. Deering and D.R. Cheriton, *Host groups: a multicast extension for datagram internetworks*, Proceedings of the 9th symposium on Data communications, 1985

[DEE88]: S.E. Deering, *RFC 1112 - Host extensions for IP multicasting*, SRI Network Information Center, 1988

[DEE90]: S.E. Deering and D.R. Cheriton, *Multicast routing in datagram internetworks and extended LANs*, ACM Transactions on Computer Systems (TOCS), 1990

[DF04]: Digital Fountain, *Perfect Communications over Imperfect Networks*, http://www.digitalfountain.com, 2004

[EIB03]: A.E. Eiben and J.E. Smith, *Introduction to Evolutionary Computing*, Springer, ISBN 3-540-40184-9, 2003

[EIL98]: T. Eilam, C. Gavoille and D. Peleg, *Compact routing schemes with low stretch factor*, Annual ACM Symposium on Principles of Distributed Computing, 1998

[EST98]: D. Estrin, D. Farinacci and others, *RFC 2362 - Protocol Independent Multicast-Sparse Mode (PIM-SM)*, Network Working Group, 1998

[FORD62]: L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, ISBN 0691079625, 1962

[FRAN00]: P. Francis, *Yoid: Extending the Internet Multicast Architecture*, Unrefereed report, 2000

[FRED88]: G.N. Frederickson and R. Janardan, *Designing networks with compact routing tables*, Algorithmica 3, 1988

[FUL93]: V. Fuller, T. Li, J. Yu and K. Varadhan, *RFC 1519 - Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*, Network Working Group, 1993

HED88: C. Hedrick, RFC 1058 - Routing Information Protocol, 1988

[HED88]: C. Hedrick, *RFC 1058 - Routing Information Protocol*, Network Working Group, 1988

[HOL99]: H.W. Holbrook and D.R. Cheriton, *IP multicast channels: EXPRESS support for large-scale single-source applications*, ACM SIGCOMM Computer Communication Review, Vol 29, 1999

[IEEE90]: IEEE, *Intermediate System to Intermediate System Intra-Domain Routeing Exchange Protocol for use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)*, ISO DP 10589, 1990

[IN04]: Internap, *Bringing reliability, performance and security to the Internet.*, http://www.internap.com, 2004

[JAF86]: A.E. Baratz and J.M. Jaffe, *Establishing vitual circuits in large computer networks*, Computer Networks and ISDN Systems, Volume 12, 1986

[JAN00]: J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek and J.W. O'Toole Jr., *Overcast: Reliable Multicasting with an Overlay Network*, In Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI), 2000

[JI03]: L. Ji and M. Scott Corson, *Explicit multicasting for mobile ad hoc networks*, Mobile Networks and Applications, Volume 8, Issue 5, 2003

[KAHN89]: A. Khanna and J. Zinky, *A revised ARPANET routing metric*, ACM SIGCOMM '89, 1989

[KIC97]: G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier and J. Irwing, *Aspect-oriented programming*, In Proceedings of the European Conference on Object-Oriented Programming, 1997

[KUM95]: V. Kumar, *MBONE: Interactive Multimedia on the Internet, 1st edition*, New Riders Publishing, Indianapolis Indiana, ISBN 1562053973, 1995

[LAB01]: C. Labovitz, A. Ahuja, A. Bose and F. Jahanian, *Delayed Internet routing convergence*, IEEE/ACM Transactions on Networking (TON), 2001

[LAB99]: C. Labovitz, A. Ahuja and F. Jahanian, *Experimental Study of Internet Stability and Backbone Failures*, Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing, 1999

[LEE83]: J. van Leeuwen and R. Tan, *Routing with compact routing tables*, RUU-CS (Ext. r. no. 83-16), Utrecht, 1983

[LI98]: X. Li, S. Paul, and M. Ammar, *Layered Video Multicast with Retransmissions (LVRM): Evaluation of Hierarchical Rate Control*, In Proceedings of the INFOCOMM '98 Conference, 1998

[LUB02]: M. Luby, et al., *RFC 3452 - Forward Error Correction (FEC) Building Block*, Network Working Group, 2002

[MAL98]: G. Malkin, *RFC 2453 - RIP Version 2*, Network Working Group, 1998

[MAR04]: K. Maruyama, *System and Method for Amicable Small Group Multicast in a Packet-Switched Network*, United States Patent No. 6,757,294 B1, 2004

[MCQ79]: J.M. McQuillan, I. Richer and E.C. Rosen, *An overview of the new routing algorithm for the ARPANET*, Proceedings of the sixth symposium on Data communication, 1979

[MET78]: Y.K. Dalal and R.M. Metcalfe, *Reverse path forwarding of broadcast packets*, Communications of the ACM, Vol. 21, Issue 12, 1978

[MOY89]: J. Moy, *RFC 1131 - The OSPF Specification*, Network Working Group, 1989

[MOY98]: J. Moy, *RFC 2328 - OSPF Version 2*, Network Working Group, 1998

[MOY99]: J. Moy, R. Coltun and D. Ferguson, *RFC 2740 - OSPF for IPv6*, Network Working Group, 1999

[MPULSE05]: MPulse Technologies, *The Pulse of Enterprise Messaging*, http://www.mpulsetech.com/prod/xcast.htm, 2005

[MS79]: P.M. Merlin and A. Segall, *A failsafe distributed routing protocol*, IEEE Transactions on Communications COM-27, 1979

[NAN04]: The North American Network Operators' Group, *mailing list archive*, http://www.cctec.com/maillists/nanog/, 2004

[NON97]: J. Nonnenmacher, E. Biersack and D. Towsley, *Parity-based loss recovery for reliable multicast transmission*, ACM SIGCOMM Computer Communication Review, Volume 27, Issue 4, 1997

[PAX97]: V. Paxson, *End-to-End Routing Behaviour in the Internet*, IEEE Transactions on Networking, Vol. 5, No. 5, 1997

[PEL89]: D. Peleg and E. Upfal, *A Trade-Off between Space and Efficiency for Routing Tables*, Journal of the ACM, Volume 36, Issue 3, 1989

[RAY96]: M. Raynal and M. Singhal, *Logical Time: Capturing Causality in Distributed Systems*, IEEE Computer, Volume 29, Issue 2, 1996

[REK95]: Y. Rekhter and T. Li, *RFC 1771 - A Border Gateway Protocol 4 (BGP-4)*, Network Working Group, 1995

[RHEE98]: I. Rhee, N. Ballaguru and G.N. Rouskas, *MTCP: Scalable TCP-like Congestion Control for Reliable Multicast*, Technical Report: TR-98-01, 1998

[RIZ98]: L. Vicisano, L. Rizzo and J. Crowcroft, *TCP-like Congestion Control for Layered Multicast Data Transfer*, In Proceedings of the IEEE INFOCOM, 1998

[SAN85]: N. Santoro and R. Khatib, *Implicit routing in networks*, The Computer Science Journal, 1985

[SAV96]: K. Savetz, N. Randall and Y. Lepage, *MBONE: Multicasting Tomorrow's Internet*, IDG, ISBN 1-56884-723-8, 1996

[SHAN92]: A.U. Shankar, C. Alaettinoglu, I. Matta and K. Dussa-Zieger, *Performance comparison of routing protocols using MaRS: distance-vector versus link-state*, Proceedings of the 1992 ACM SIGMETRICS, 1992

[SPEAK01]: T. Speakman et. al., *RFC 3208 - PGM Reliable Transport Protocol Specification*, Network Working Group, 2001

[STEV93]: W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley Professional, ISBN 0201633469, 1993

[TAUB94]: D.Taubman and A.Zakhor, *Multirate 3-D Subband Coding of Video*, IEEE Transactions on Image Processing, Volume 3, Number 5, 1994

[VIC02]: L. Vicisano, M. Luby, et al., *RFC 3453 - The Use of Forward Error Correction (FEC) in Reliable Multicast*, Network Working Group, 2002

[VLC05]: VideoLAN, *VLC free cross-platform media player*, http://www.videolan.org, 2005

[WAI88]: D. Waitzman, C. Partridge and S. Deering, *RFC 1075 - Distance Vector Multicast Routing Protocol*, Network Working Group, 1988

[WEI95]: Liming Wei and Deborah Estrin, *The Trade-offs of Multicast Trees and Algorithms*, IEEE/ACM Transactions on Networking, Volume 4, Issue 2, 1995

[WEN03]: W. Wen, B. Mukherjee, S.-H. Gary Chan and D. Ghosal, *LVMSR: an efficient algorithm to multicast layered video*, Computer Networks: The International Journal of Computer and Telecommunications Networking, Volume 41, Issue 4, 2003

[WIL99]: B. Williamson, *Developing IP Multicast Networks, Volume 1*, Cisco Press, ISBN 1578700779, 1999

[ZHU01]: S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz and J. Kubiatowicz, *Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination*, Proceedings of the 11th International Workshop on NOSSDAV, 2001