ARM - Active Reliable Multicast

Dmitri Kondratiev

dkondr@bigfoot.com

Table of Contents

Abstract

In this document we provide design overview of the ARM - Active Reliable Multicast family of transport protocols that use IP Multicast to provide reliable one-to-many or many-to-many data delivery.

ARM should provide reliable data delivery in terms of the variety of application service requirements. We describe some of the mechanisms that existing reliable multicast transports use, with a focus on their strengths and weaknesses. We examine ARM protocol requirements that emphasise the need for reliable multicast transport protocols and summarise the reasons why we are likely to have a variety of ARM multicast transports, rather than a single one suitable for all applications.

The description of reliable multicast transport mechanisms, as well multicast transport requirements given here is based on our research of existing multicast protocols and on the excellent survey written by Bob Quinn from Stardust.com [Bob Quin].

Introduction

ARM is based on IP Multicast which is essential to enable applications that can service millions of users simultaneously by sending a single data stream to any number of receivers -- one-to-many or many-to-many data delivery. In turn IP Multicast uses unreliable transport protocol - UDP - a connection-less datagram protocol [RFC 768]. Thus IP Multicast does not provide any guarantees of delivery or ensure that data arrives in sequence or without duplicates, it is considered an unreliable transport protocol. To say more any services based on IP level protocol provide "best effort" delivery only. Quality of Service (QoS) with RSVP [RFC 2205] or class of service (e.g. differentiated services using IP TOS bits [Nichols] still does not solve congestion problems or network failures. Different applications have different requirements in data loss, out-of-sequence, or duplicated data. For most applications transport protocols that run over UDP does not meet reliability requirements. In these cases application may chose to use other transport protocols and / or ensure these requirements in application logic. ARM set of protocols should provide different levels of reliable, sequenced and "in time" delivery according to varying application needs.

Application delivery requirements

Application delivery requirements may range according to particular application needs. These needs and thus delivery requirements will differ considerably from application to application, with unicast being a trivial case of multicast delivery. In other words all applications care about their data but in different ways.

Multicast delivery requirements

- Data Loss (In)tolerant. Data Loss Intolerant requires that no data can be lost. Correct sequence is not guaranteed.
- Wrong Sequence (In)tolerant. Wrong Sequence Intolerant requires that all data must be received in correct sequence.
- Delay (In)tolerant. Delay tolerant requires that data to be delivered in time. Sequencing and no data loss are not guaranteed.
- Wrong Synchronisation of Group Members (In)tolerant . Also called Virtual Synchrony requirements. All data sent to a group of receivers must be received by all group members in correct order and at the same time. There may be more or less strong requirements on delay (in)tolerance for group members.

Session and stream delivery requirements

Application should be able to join / leave existing delivery stream any time. Customisable session (file) delivery should allow application to:

- Join session late.
- Leave session any time.
- Specify how and when to retransmit old data (immediately or later).

Group management

- Support scalable groups.
- Support various types of group management by controlling access to the multicast group and its data (authenticate receivers).
- Support centralised and decentralised group management authorities.
- Provide notification events for group members (all receivers have received all data, join / leave group events, etc.).

Examples of application delivery requirements

- Real-time Transport Protocol [RFC 1889] provides some basic reliability services. Specifically, it adds sequencing and timing to multimedia data streams (audio and video) sent via UDP and IP Multicast. Although multimedia data streams have strict sequence and timing requirements, they are tolerant of data loss (to a varying degree, depending on the data encoding schemes used). Hence, RTP does not provide any guarantees of data delivery, and doesn't need to. RTP allows an application to detect lost data, but does not provide a mechanism to recover it.
- Delivery of some types of "real-time data" - such as stored video and audio - can be delayed any amount of time, without decaying its value.
- Real-time "conversations" or interactive simulations and games that use real-time data delivery must also be timely to be effective. Delays in delivery can interfere with bi-directional communications as occur in many-to-many applications.
- Some one-way delivery data types may be intolerant of delay also. For example, for stock ticker information and on-line auction bids, time (delay) is literally money. Delay intolerant applications can have time-bounded reliability or use QoS protocols and services (e.g. RSVP), but the simplest way to insure prompt delivery is to forego reliability.
- Real-time applications are data loss tolerant, since data that cannot be delivered promptly may not be delivered at all.
- Many other applications for which delays are more tolerable than data loss.

ARM protocols should address any of the described above application needs. To do so ARM should be easily configurable to support only those delivery features that are needed by the application.

Multicast transport mechanisms

Reliable data delivery using acknowledgements (as in case of TCP) from receivers is inappropriate for use in one-to-many (or many-to-many) multicast transports, as it

may lead to "acknowledgement implosion" problem. Other reliable multicast mechanisms exist that try to avoid "acknowledgement implosion" problem.

Negative Acknowledgements

- Description.

  Receivers do not acknowledge received data at all. Instead receivers that detect missing data (by gaps in sequence numbers) respond negatively using negative acknowledgements (NAKs) to request retransmission.

- Advantages.

  Reduces the chance of NACK implosion significantly.

- Caveats.

  NAK implosions are still possible if many receivers simultaneously miss data. This implosion potential is much greater when a packet is lost near the sender (so the data loss affects a greater number of receivers).

- Conclusion

  Because of the NAK implosion potential, the use of NAKs alone is not sufficient to provide the robustness and scalability desirable for any reliable multicast transport.

Shared recovery with Multicast NAKs and Retransmissions

- Description

  To avoid the NAK implosion receivers share recovery responsibilities. When a receiver detects missing data and sends a NAK, another receiver that has received the data can potentially respond to the NAK by retransmitting the missing data.

  Multicast NAKs and Retransmissions Algorithm.

  NAKs and retransmissions are sent by receivers to a multicast group address. The algorithm resembles Host Membership Queries in IGMP.

  - 1. When a receiver detects missing data, it starts a (random) timer.
  - 2. When the timer expires and receiver hasn't yet received a NAK for the missing data from the group (multicast) address, it sends a NAK to the group (with a limited scope).
  - 3. All receivers within the scope see the NAK, start a (random) timer, and watch for the requested data to be retransmitted.
  - 4. If retransmitted data is seen, receivers stop the timer. Else, if their timer expires they retransmit the data to the multicast group address (with a limited scope) if they have it, or send another NAK if not.

- Advantages
  - Off-load the recovery burden from the sender, and provide faster recovery (since NAK and retransmitted data have a shorter distance to travel).

  - Other receivers that also missed the same data can see the NAKs, and don't need to send them to benefit from the retransmitted data.
  - Reduced recovery latency.
  - Reduced network traffic in the path between sender and receiver.

- Caveats.
  - May increase overall network traffic.
  - Receivers must be capable of gracefully handling redundant data.
  - Retransmission must be done in the limited scope.
  - Data delivery is delayed because of the time-outs to send NACKs.
  - Will not work on network topologies that don't support many-to-many multicast.

Local Shared recovery with Unicast NAKs and Retransmissions

- Description

  In this model individual receivers are identified as the local recovery designates. Receivers that miss data send the NAK via unicast directly to these designates. Recovery receivers may then retransmit data to the receiver via unicast (or multicast).

- Advantages.

  Recovery traffic is localised in the path between recovery designate and receiver. Reduced recovery latency (no time-outs).

- Caveats.
  - Needs a strategy to pick a recovery designate, which includes:
    - Identifying receivers
    - Initial setup must be done to establish a designates group using one of the following:
      - create tree or graph structure of designates;
      - passing a token among designates;
      - selecting designates probabilistically among receivers.
  - Needs mechanism to detect recovery designate that has gone away (crashed, stopped nodes).
  - Local NAK implosion may happen with the local recovery designate.

Redundant data

- Description

Sending redundant data may allow to avoid recovery altogether. Redundant data may completely reduce missed data. In this case receivers don't need to recover data and senders don't need to retransmit. This technique is the only method of providing reliable multicast delivery when there's no back-channel available, such as over satellite or through analog television transmission, via IP over video blanking interval, etc.

To satisfy different application requirements there exist two main methods for reliable multicast with redundant data:

- Forward Error Correction (FEC) [Perkins]. Redundant data is sent in parallel. Duplicate data is "striped" across multiple data streams or data is sent with parity.
- Data Carousel. Sends the data once, then sends it again, and again, in rounds as many times as required.

- Advantages
  - Effectively eliminates the latency involved with recovery.
  - FEC method addresses best of all real-time reliability requirements (e.g. stock ticker), although it can be used with any application.
  - Data Carousel method is only useful for applications that have a high-tolerance of data delivery latency.

  Caveats

  Increases the bandwidth usage. Higher reliability needs higher bandwidth.

- Conclusion

  Redundant data methods can be effectively combined in many cases with previously described reliable multicast implementation strategies.For example, when used in retransmissions, FEC can increase their reliability (to avoid having to retransmit a second time) [Macker].

Avoiding congestion and recovery with adaptive rate

- Description

  Avoiding aggravating network congestion is one of the most important requirements for any multicast transport. In case of reliable multicast this requirement is especially important for the design of retransmission protocol. Bad design may easily lead to network congestion.

  Detecting congestion and responding to it by adapting data transmission rate helps to prevent congestion problem. In protocol with a NACK-based retransmission network congestion may be determined by the number or rate of NAKs, which indicate data loss. With this feed-back information data transmission rate may be adapted accordingly.

- Advantages

  Allows to avoid network congestion.

- Caveats
  - For data loss tolerant real-time application, such as audio and video, the reduced data rate will result in reduced data resolution or "noise".
  - For loss intolerant applications reduced data rate will lead to increased latency, such as longer data transmission.
  - Group must adapt data rate to the lowest common denominator which can severely degrade data transmission or isolate slow receivers.

- Conclusion

  Adaptation of data transmission is a powerful mechanism for the effective congestion control. However to provide sensible rate adaptation there must be a mechanism to eliminate the "crying babies" - defective receivers in the group. One way to achieve this is a local (shared) recovery that can isolate a slow receiver.

Avoiding congestion and recovery with layered data

- Description

  Another way to cope with slow receivers is to send "layered data" [Speer]. Sender uses data encoder to produce lower resolution data when a low percentage of sent data is received, and higher resolution data when a higher percentage of data is received. For example, this approach is used in "Wave Video" encoding and scalable video transmission framework.

- Advantages
  - All group members receive at least some data.
  - Recovery may not be needed at all.

- Caveats
  - Works well for loss-tolerant applications.
  - Data coder dependent. Different data types need different encoding.

- Conclusion

  Layered data may be effectively used for specific applications and data types.

ARM transport requirements

The main requirements include scalability, congestion control, error recovery, and robustness. These in turn determine group management, late joining, and security.

- Scalability

- Service a large number of receivers. Scale to any number of receivers without impacting the sender. Contribute to network congestion in reasonable proportion to the number of receivers.
- Support different type of receivers.
  - Receivers with different bandwidth. Including those with asymmetry between downstream and upstream speeds.
  - Different receiver topologies: single hop, with asymmetric upstream and downstream paths, one-way transmission, etc.
- Provide group management functions for applications that need it.
- Be able to distribute group management functions among receivers.

- Congestion control - the main requirement influencing the design of all aspects of reliable multicast protocol. Congestion control must be at least validated by TCP benchmarks for congestion control mechanisms designed by Van Jacobson.

  Under a loaded network condition "for any link, the traffic arrival rate for a flow should respond to congestion in a way no more aggressive than multiplicative decrease and additive increase, with rates that give behaviour that is no more aggressive than current implementations." [Floyd]

- Error recovery
  - Detect error conditions such as data loss.
  - Report error condition to recovery designates or the sender.
  - Repair errors from retransmitted data and / or using other mechanisms (redundant, layered data, etc.)
  - Error recovery mechanism must scale.
  - Error recovery mechanism must avoid network congestion.

- Robustness

  Minimise the circumstances where protocol mechanisms employed will fail. Every mechanism applicability constraints must be clearly defined and used accordingly.

- Other protocol requirements
  - Support 'late joining' for file transfer protocols used by loss-intolerant applications. Late joining receiver should know how soon the missed data Will be retransmitted and how much data has missed to do pre-allocation.
  - Secure transmission. Provide IP Security (IPSec) compatibility which is a standard requirement for new protocols in the IETF. This includes implementation of :
    - Sender / receiver certification.
    - Key distribution and group key management.
    - Efficient security algorithms and processes.

ARM Design

ARM is designed as a set of protocol elements to be used for construction of reliable multicast protocols with given properties. This allows to build a protocol specifically tuned for one or another type of distributed applications. Protocols can dynamically adapt to the changing properties of packet flows in run-time. Adaption is achieved by symbiosis of protocol itself and Active Applications (AA) that may use sophisticated adaptation algorithms.

For example ARM elements may be chosen and tuned both for loss-tolerant streams and file-oriented transmissions. Different error recovery schemes can be selected and mixed together.

ARM protocols are built, customised, tuned and deployed by AA in Cambira VM. Protocols are constructed from the set of configurable elements attached to a flow processing channel - channel elements.

ARM programming with Channels

ARM protocol is implemented by one or more CVM channels controlled by one or more AAs.

Channel defines a packet processing program for one or more flows that one AA initially controls. AA may create several channels associated with different flows. Every channel has exactly one owner - AA that created it. Channels can be created by AA with permission to be shared by other AAs, or as private channels that nobody else can use.

Channel program is defined in terms of channel elements and can be specified by domain specific language (DSL). Channel Programming Language - CPL - is a declarative DSL that we plan to use for ARM protocol construction.

Channel element (CE) is the main programming unit of CPL. Channel program is defined by CEs and their interconnection, or channel topology. CE is built on similar principles as Click Router [Click] element is. However CPL goes much further then Click and provides programmer with first-class objects CEs. User-defined types of CE with class inheritance is supported. The real power of channel program is the ability to adapt to changing flow conditions. To achieve this new AAs coming to the router must be able to reason about existing channel programs already running in CVM. Thus channel program must have all necessary mechanisms for AA to find out in run-time channel topology and element types that channel consist of. For this run-time introspection is supported by running instance of CE itself.

CE type is defined by it its class. CE class has the following attributes:

- Class name.
- Input ports.
- Output ports.
- Channel Element Constraints. Constraints here define types and number of input/output ports that this element may have in run-time.

Port type is defined by its class. Port class has the following attributes:

- Class name.
- Direction: input or output.
- Connection: push, pull or agnostic.
- Port constraints. Constraints here define types of ports that this port can be connected to. These are additional to general agreement for legal connections among ports (see below).

Connected Channel Elements define Channel program. Elements are connected by means of connections established between their ports. Connections can be established between input and output port of the same connection type. Valid configurations are: push output --> push input, pull output --> pull input, agnostic output --> agnostic input.

Ports and connections can be created and deleted in run-time. When created, port is not active first. Connection can be created between inactive ports. To activate connection both ports must be explicitly made active.

The only way CE elements can communicate with each other is by sending and receiving packets on existing connections. Packets can be of two different major types: network packet and information packet. These are the base types from which all other packet types are derived. Every packet type consist of the following distinct sections:

- Reference to payload buffer.
- Reference to packet header.
- Annotation hash containing attribute value pairs. Annotation may be used by CE to pass results of their work upstream to other CEs.

Any of these sections may be empty (null references). However, at least one section must be non-zero in the packet.

Protocol examples

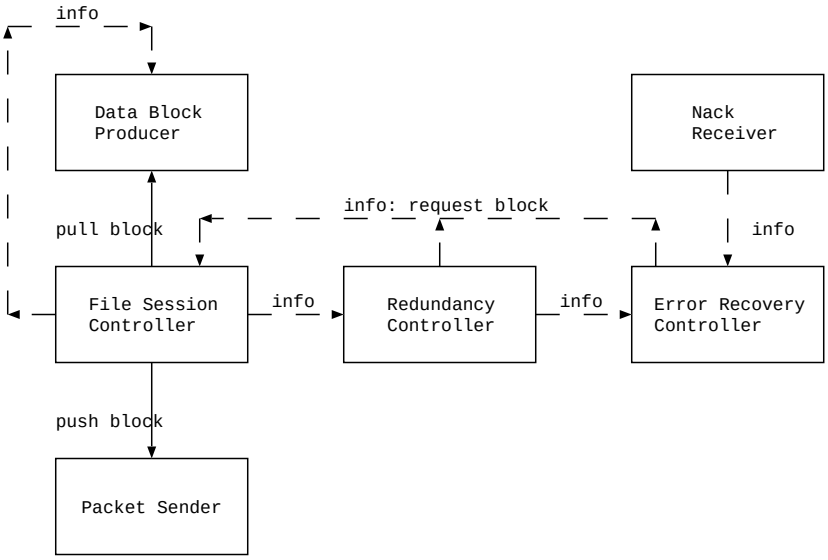

Figure 1. Simple ARM file transfer protocol

In fig.1 simple file transfer protocol is described. The protocol program consists of the following elements:

- Data Block Producer - reads specified data block from file storage. Has one pull output port that has requested data block ready for any consumer. Another port is input push port which is used for block requests.

- File Session Controller - creates new file session and maintains its state. Main function: create session packet from data block.

  Has two info ports and two data ports. One info port is used to get requests from other parties to send session data. Second info port is used to request the block from Data Block Producer.

  One data port is used to "pull" data blocks from producer. Another data port is used to "push" session packet to Packet Sender.

- Packet Sender - sends packets to network.
- Redundancy Controller - keeps redundancy count and retransmits session accordingly.
- Error Recovery Controller - accumulates NACKs and requests retransmissions when session is over.
- Nack Receiver - receives and filters redundant NACKs from network and informs Error Recovery Controller about outstanding retransmissions.

This example shows a simple channel program that most probably will be created and controlled by one AA. Another application may come later and do some research on amount of packets that are getting lost. It may then decide to that sending redundant data is an overkill in this particular case (current network topology and available bandwidth). Then, to make multicast more efficient this application may decide to remove Redundancy Controller CE and create direct connection between File Session Controller and Error Recovery Controller.