Abstract

Fast Payload Replication (FPR) is a technique for fanning out the payloads of individual packets to large numbers of recipients. It was developed as the central part of Ivip’s real-time mapping distribution system. However, as of late February 2010, Ivip has a new mapping system: Distributed Real Time Mapping (DRTM) which operates on different principles. Therefore, the material in this ID is no longer an essential part of Ivip. Some of it may be useful for other purposes and for two potential purposes in Ivip, both of which are entirely optional.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 7, 2010.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the BSD License.

Table of Contents

1. Introduction ................................................. 3
2. Goals ..................................................... 12
   2.1. Potentially high data volume .......................... 12
   2.2. Independent of routers and network structure .......... 12
   2.3. Simple UDP (DTLS) only operation ..................... 12
   2.4. Good but not perfect robustness ....................... 13
   2.5. Flexible trade-off of efficiency for robustness ...... 13
   2.6. Robustness against DoS may be achieved with private
        network links ........................................... 14
3. Non-goals .................................................. 15
   3.1. Not intended to provide end-to-end security .......... 15
   3.2. No autodiscovery or monitoring ........................ 15
   3.3. No attempt to automatically adapt to varying PMTUs .. 15
   3.4. Not intended for mass market consumer applications ... 16
4. Streams of packets for Replicators and QSDs .................. 18
5. Packet payloads and identification .......................... 22
6. The Fresh vs. Repeat Algorithm .............................. 25
7. RUAS functionality ......................................... 27
8. Replicator Functionality .................................... 28
9. QSD Functionality .......................................... 30
10. Further elaborations ........................................ 32
    10.1. Missing Payload Servers (MSPs) ......................... 32
    10.2. Delaying the output of Replicators .................... 34
    10.3. Private network links to avoid DoS attacks .......... 35
11. Security Considerations ..................................... 38
12. IANA Considerations ......................................... 39
13. Informative References ..................................... 40
Author’s Address .............................................. 41
1. Introduction

This document written quickly in an effort to support the RRG debate - so please excuse the lack of sub-headings and any rough spots.

Continuing from the Abstract:

Some of the following may be useful for two purposes in Ivip, both of which are entirely optional.

Firstly, to make a small system of Replicators to fan out real-time mapping information to Map Resolvers in ISP networks which, instead of caching mapping information, store the full database of mapping for one, multiple or perhaps all Mapped Address Blocks (MABs).

Secondly, some of these techniques might be useful within the systems of MABOCs (MAB Operating Companies) or whoever runs DITR (Default ITRs in the DFZ) sites - for real-time mapping distribution to the query servers at those sites.

By trading off efficiency for robustness, the FPR system can be made highly tolerant of random packet loss or loss of connection from some upstream Replicators. FPR is simpler and less efficient than Reliable Multicast or Secure Multicast, but can operate on a global scale over the DFZ. It is a host-to-host arrangement and is independent of routers and network topology. Packets are DTLS encrypted so spoofed packets cannot enter the Replicator system. Since it is not completely robust against packet or link loss, or secure against an attack which compromises a Replicator, the basic FPR should be supplemented with Missing Payload Servers and end-to-end authentication of received data in order to make an entirely robust and secure system. FPR should be able to fan out information to hundreds of thousands of recipients, worldwide, in less than a second.

Those parts of FPR which may be used for entirely optional purposes in Ivip, as noted above, are the Replicators, Missing Payload Servers and the basic concept of a full-database query server. With DRTM [I-D.whittle-ivip-drtm] there is no need for RUASes, the single inverted tree-structure of Replicators etc. Instead, smaller systems of Replicators, probably with Missing Payload Servers, may be used for the purposes mentioned in the Abstract. In the future, I plan to revise the following material accordingly.

The remainder of this ID is the same as version 00, but please remember that none of this is required for Ivip.

This ID explores a combination of techniques which as far as I know
is novel. It may be useful for various other applications than the one it was developed for: the central part of a fast-push mapping distribution system, on a global level, for the Ivip core-edge elimination architecture. [I-D.whittle-ivip-arch] [I-D.whittle-ivip-db-fast-push] Ivip is intended to solve the routing scaling problem for both IPv4 and IPv6 - whilst also being a good basis for the TTR Mobility architecture. A system such as Ivip could also support TTR mobility, irrespective of its support for routing scalability. [TTR Mobility]

FPR stands for "Fast Payload Replication", although "Directed Flooding Packet Payload Replication" would also be appropriate. (It seems that the acronym "FPR" is only used in the IETF for "Frame Policing Ratio" in [RFC3133].)

FPR is intended to be suitable for a business environment where multiple companies combine to create a shared infrastructure, with no single point of failure.

The units of FPR data replication are payloads contained within individual UDP packets. The simplest way to implement FPR will probably be to use DTLS [RFC4347] encryption and authentication between the sources of packets, the Replicators, and the destination devices which use these payloads. It would also be possible to use IPSec Authentication Header or a specifically written authentication arrangement to protect these devices from accepting spoofed packets. For simplicity, the use of DTLS is assumed in the following discussion.

While the following discussion of FPR does address other possible uses, the focus is on FPR as a component of the Ivip fast-push mapping distribution system. In this application, the final destination of the payloads are full mapping database query servers, known as QSDs (Query Server with a full Database) and the source of the packets are multiple devices known as RUASes (Root Update Authorization Servers). For simplicity, IPv4 is assumed in the examples, but FPR is equally suitable for use with IPv6.

In a later section I discuss another network element - a Missing Payload Server (MPS). This would extend the basic FRP system of Replicators to provide the QSDs with a distributed set of servers from which to obtain payloads which did not arrive in packets from the QSD’s multiple upstream Replicators.
One of 20 or so RUASes, each of which drives its packets to all three level 0 Replicators. Each level 0 Replicator receives a complete set of data to be sent to hundreds of thousands of QSDs. Three fully meshed level 0 Replicators drive each other and send streams to 20 level 1 Replicators. Each stream contains packets with payloads from all RUASes.

Figure 1: Five levels of Replicators drive hundreds of thousands of QSDs.
Figure 1 depicts a system with 3 level 0 Replicators, 30 level 1 Replicators, 300 level 2 Replicators etc. MPSes (Missing Payload Servers) are not shown. The Ivip system would probably involve 5 to 8 level 0 Replicators, but 3 makes for a clearer diagram.

The amplification factor is how many output streams a Replicator sends divided by how many it receives. This will vary depending on local choices, but I have shown all Replicators producing 20 streams and all level 1 and greater Replicators consuming two.

The amplification factor per level may be higher than this so fewer levels may be required to drive the required numbers of QSDs. In the long-term future, it is possible that hundreds of thousands of QSDs in ISP and larger end-user networks will receive mapping updates, in order to serve the ITRs in those networks. This ID explores the design of such a large-scale system. When Ivip is introduced, the whole system would be much simpler, such as with 3 Level 0 Replicators, higher amplification factors due to the initially low rate of updates and fewer levels due to this and the initially lower number of QSDs to be driven.

The principle is that the level 0 Replicators are fully meshed and (except for any lost packets, dead links or failure in one of these Replicators) receive the full set of packets to be sent out to all level 1 Replicators.

If a level 1 Replicator has a packet missing from one of its upstream level 0 Replicators, then it will usually be able to obtain the same payload from the equivalent packet from its other upstream level 0 Replicator.

A level 1 Replicator which is missing a packet from both its sources will not be able to send this packet’s payload to its 20 downstream devices. However, depending on how the cross-linking is structured, generally those Replicators will obtain the payload that packet carried from the equivalent packet from their other source.

This principle continues to the end - where the QSD is generally able to cope with a missing packet by using the payload of the equivalent packet from the other upstream level 4 Replicator.

The maximum length of the packets needs to be chosen so as not to violate any Path MTU from one Replicator to the next, and to the final recipient devices.

The data to be carried needs to be split into individual blocks, one for each payload, of around 1300 bytes. This suits Ivip reasonably well, since it will frequently be the case that an RUAS has either no
updates to send in a given period of time, such as 0.3 seconds, and
sometimes may have enough updates to fill several payloads.
Consequently, the RUAS only sends a packet when it needs to, and the
length and number of the packets reflect the amount of data to be sent.

Ivip mapping updates apply to a particular MAB (Mapped Address Block)
a DFZ-advertised prefix which encompasses a block of SPI (Scalable Provider Independent) address space. Each MAB is split into many (potentially hundreds of thousands) of arbitrary length ranges of address space called "micronets". Each micronet is mapped to a single ETR (Egress Tunnel Router) address. The most common mapping update is to change the ETR address of an existing micronet. Other updates join and split micronets, or announce that the RUAS which controls this MAB has made a snapshot of the full state of the MAB’s mapping - which a QSD can download for the purpose of initialising its copy of the mapping database, or to overcome any errors which have somehow accumulated in the section of it which concerns this MAB.

Within a short period of time, such as 100ms, there may be a series of mapping updates concerning a particular MAB. All mapping updates for a given MAB always arrive in payloads from one RUAS. If a complete series arrives in a single payload, then this is relatively simple. If this payload is missing from the streams of packets which arrive from the QSD’s two or more upstream Replicators, then the QSD will obtain the missing payload within a few seconds from a MPS. Then this series of changes will be applied to the MAB, and the slight delay will be of no consequence.

If the series of changes spans two or more payloads, and one or more of these is missing, then the situation is more complex. The QSD generally can’t apply any changes to the MAB until it has the full series which was transmitted effectively at the same time. This is because the series may involve zeroing the mapping of micronets, splitting and joining them and then setting the mapping of the resulting micronet to a new ETR address. The series of changes is best applied all at once, so the QSD needs to buffer this MAB’s changes and only apply them once it has the missing payloads.

In either case, where updates to the mapping of one MAB are delayed, the QSC must buffer subsequently received updates and apply them in order. This is for the same reason: that some updates alter the structure of the micronets and so cannot be applied out of order.

This illustrates that the Ivip system will work fine with all or most updates being applied within a second or so of them being sent by the RUAS, but that in the small number of cases where there is a few
seconds delay due to a missing payload, this is of no serious negative consequence. No end-user network absolutely relies on ITRs changing their tunneling behavior within a second or two of a mapping update being sent. It suffices that most will do this, and on rare occasions some ITRs will change a few seconds later.

Even if all changes to ITR tunneling behaviour were suspended for a minute or two, or some ITRs lagged behind the rest by a few minutes, no serious harm would occur. The worst outcome would be a consequent delay in multihoming service restoration (which for a minute or two is highly undesirable, but not disastrous) and likewise delays in traffic engineering changes or packets to mobile devices being sent to a new TTR, rather than the old one. In the mobile case, this simply means the mobile node needs to maintain its tunnel to the old TTR for a few minutes longer than it otherwise would. (Such changes in mapping only occur if the mobile node moves 1000km or so - not ever time it gains a new access network or IP address.)

The foregoing discussion illustrates that it is good enough for FPR to generally be very fast and robust, but for it to sometimes involve delays of a few seconds, or more rarely, a few minutes.

The total data-rate of the "complete stream" of packets handled by the FPR system needs to be carefully bounded in order to ensure that all devices in the system will not be overloaded. In Ivip, there will be multiple, asynchronous, independent sources of packets which drive the input to the system "level 0 Replicator" part of the system. In this ID, these are assumed to be a set of RUASes (Root Update Authorization Servers). The exact number of these is not particularly important, but in practice there might be a few dozen. These would need to coordinate their packet rate to ensure that at no instant was the FPR expected to carry more than some data rate of packets.

In the fully designed Fast-Push Mapping System, there may be an additional source of packets which is not an RUAS. Nonetheless, the discussion below anticipating a dozen or so 20 or so RUASes is sufficient to explore the current FPR design.

FPR is secure against an attacker sending spoofed packets, since these will not pass the DTLS software which accepts them into each Replicator and QSD.

FPR is not secure against an attacker gaining control of one or more Replicators. In order to achieve end-to-end integrity, the final recipient device (in Ivip, a QSD) will need to be able to authenticate each payload worth of mapping data, or larger bodies of mapping data assembled from multiple payloads. This will probably be
via a public key signature of that payload or larger body of data, which is included in the payload data stream – using the public key of the RUAS which sent the payloads.

Likewise, in order to be able to ensure confidentiality against an attacker who can snoop packets being sent by Replicators, the application data must be protected by encryption. In the present design, DTLS both encrypts and authenticates the payload of each packet. Confidentiality is not required for Ivip. (I have not yet determined if there is a DTLS mode which supplies only authentication.)

Since FPR is not absolutely robust against link loss or random packet loss, a complete system which delivers data entirely robustly must supplement FPR with some method by which the recipient can request missing packets. This could be external to the FPR system, but in a later section I explore the possibility of integrating Missing Payload Servers into the FPR system of Replicators. Forward Error Correction is another method of coping with a certain level of lost packets, but this involves considerable complexity and overhead. It also involves handling data in long block lengths which are not suitable for Ivip.

FPR’s use in Ivip is a critical part of the core-edge separation system. The FPR part of the mapping distribution system is a single global-scale system and it is intended to run reliably, continually, delivering data to potentially hundreds of thousands of QSDs all over the world. A halt in its operation for a few seconds or minutes would not be disastrous, but would delay multihoming service restoration, mapping changes for inbound TE, and the ability of mobile nodes to choose a closer TTR. The Ivip system won’t fail if the FPR system stops for minutes or even tens of minutes. Nonetheless, the FPR system is intended to operate continually, indefinitely, despite its individual component Replicators being taken in and out of service and the connections between them being changed from time to time.

FPR is most suited to an application which requires very fast replication of information, including perhaps on a global scale, where it is important that the data generally arrive quickly, but that occasional lost packets and consequent delays obtaining replacements will not be a problem.

While FPR is much less efficient than ordinary multicast – in which a single stream is replicated into multiple streams at one or more points in the distribution system. Efficiency is traded off directly for greater robustness against packet loss.
FPR does not rely on conventional multicast protocols or router capabilities. It should be possible to implement an FPR Replicator as a user space daemon on any server.

There is nothing particularly surprising about the outcomes of the FPR arrangement. However, this combination of capabilities is a crucial component of Ivip. FPR’s capability to convey "mapping" information in essentially real-time from end-user networks (or entities they authorise to control their mapping) to hundreds of thousands of QSDs in ISP and large end-user networks all over the world enables Ivip to achieve at least two major benefits compared to other core-edge elimination systems, most prominently LISP [I-D.ietf-lisp].

Firstly, there is no need for ITRs (Ingress Tunnel Routers) to have to choose between multiple ETRs (Egress Tunnel Routers) – since the end-user network is able to control the ITR tunneling behavior in real-time. (ITRs receive mapping from QSDs in response to queries, and QSDs send updates to ITRs if they receive changed mapping via the FPR system.)

Secondly, this modularly separates control of ITR behavior from the core-edge separation scheme itself, enabling the end-user network to control the ITRs for whatever purposes they desire, and with whatever techniques and information they employ.

Without a real-time global mapping distribution system, the other core-edge elimination architectures to date cannot control ITRs directly, and so must build all the system’s reachability testing and decision-making capabilities into each ITRs and give end-users control via more complex mapping which includes multiple ETR addresses.

This discussion of core-edge separation architectures is of no direct relevance to FPR as a subsystem. However, it illustrates that FPR’s particular capabilities are crucial to being able to make some attractive architectural choices in a core-edge separation scheme.

Reliable Multicast [RFC2887] would not be as suitable, since it involves a single stream of packets, whereas in Ivip, FPR will fan out multiple independent streams, one from each of 20 or so RUASes. Reliable Multicast involves long blocks of data for its Forward Error Correction arrangement, which would introduce delays in the sending and receiving of application data. In Ivip, this would delay and very much complicate the reception of data, particularly when the data rate from each RUAS is low.

Neither Reliable Multicast or Secure Multicast [RFC3740] are robust
against lost packets and dead links, while FPR can be used in a way which gives a much higher degree of robustness against these.

As far as I know, most multicast protocols assume the use of routers at specific parts of the network. FPR is intended to operate without reference to routers or the address structures inherent in networks. FPR Replicators can be implemented in servers on arbitrary stable global unicast addresses. The structure of the links between Replicators has no reliance on network topology and can be arbitrarily chosen. FPR is intended to work reliably with links across the DFZ and so be able to scale well to a global distribution of recipient devices.

In summary, FPR is relatively simple and may complement established multicast protocols rather than exceed their performance in the applications they are best suited to. At least with Ivip, FPR’s apparently unique capabilities will enable a larger system to be designed in ways which would not be possible – or at least not as easy – with existing techniques.

I intend to write the requisite software for FPR – code for a Replicator – later in 2010.
2. Goals

2.1. Potentially high data volume

FPR should be able to handle relatively high data volumes.

The limiting factor with DTLS is likely to be the ability of Replicator software to send the output streams each with its own DTLS protection. If a customised authentication arrangement was used instead, then each Replicator could send essentially identical packets to all its downstream devices, saving on the separate cryptographic processing of each stream which would be inherent in DTLS or IPsec.

At present, I am unsure of the efficiency of using DTLS to produce large numbers, such as 20 or 50 output streams of packets. With 4 core 64 bit CPUs clocked at close to 3GHz, I would not be surprised if a modern COTS (Commercial Off The Shelf) server could fill a gigabit Ethernet link. However, this remains to be determined. The average data rates per stream with Ivip would be fractions of a megabit per second, even with the largest imaginable deployment.

2.2. Independent of routers and network structure

While FPR could be implemented in routers, it is intended to be implemented as software in a server. The use of DTLS means that a user-daemon with inbuilt DTLS capabilities can be operated on any server, since there are no special demands on the operating system.

In a global system, Replicators can be on any stable global unicast address. In a private network the addresses need only be stable. In all cases, the passage of packets between Replicators is controlled directly and in no way depends on the topology or addressing structure of the network. This makes FPR suitable for a global packet replication system with links across the DFZ.

2.3. Simple UDP (DTLS) only operation

Since DTLS sessions are set up via the same UDP ports which are used for data transfer, the entire Replicator could use a single UDP port.

This should facilitate recipients being behind NAT, since the recipient device makes the DTLS link to its upstream Replicators. Replicators themselves cannot be behind NAT, since the DTLS session could not be established to them. Replicators, at least for Ivip, are generally meant to be at well-connected data centers where the multiple links to other data centers can be used to ensure physical diversity of the streams being sent to any one Replicator.
2.4. Good but not perfect robustness

If a recipient receives streams of packets two upstream Replicators, and both of these feeds are disrupted in some way, then the recipient will not get some packet payloads. FPR has no NACK, but in a later section I discuss a system of "Missing Payload Servers".

The aim is to make FPR delivery of packets for any recipient with reasonably good network links (such as by the streams arriving via two physical links from Replicators with different topological locations) highly robust against individual packet losses or the failure or unreachability of an upstream Replicator. The purpose is to make missing payload recovery a rare enough event that the occasional delays and extra traffic it involves are not significant problems.

There can be no perfectly robust system, of course, in the event that all links from outside sources are disrupted at the same time.

2.5. Flexible trade-off of efficiency for robustness

By choosing how many input streams each Replicator or recipient device has, and by choosing these to arrive from Replicators near and far (geographically and topologically) it should be possible to achieve a wide-range of compromises between efficiency and robustness.

These choices can be made at a local level, for each particular Replicator or recipient. While the discussion below generally assumes each will receive two feeds, it will be possible to configure them to receive more than this number of feeds.

For instance, a Replicator or Recipient which can be given five feeds, each arriving over a different physical link, each from a Replicator whose location in the network is topologically different from the others. Each such upstream Replicator should, ideally, have feeds from other upstream Replicators are at least partially diverse with respect to each other. Then, the ability of the recipient to receive all packet payloads could be extremely robust against random packet losses and against outages in routers, data-links and other Replicators - at the cost of requiring outputs from more upstream Replicators and paying for the bandwidth of their multiple incoming streams.
2.6. Robustness against DoS may be achieved with private network links

No device on the open Internet can be reliably protected against a flood of packets generated by botnets. In order to minimise the damage such an attack could have on an FPR system, the higher layer Replicators (closer to layer 0, and including layer 0) of the inverted tree structure would need to be linked by private network links.

At some point in the Replication hierarchy, where Replicators are sufficiently numerous, the links to the next level (numerically higher, but lower in the inverted tree) could be carried by the public Internet. A DDoS attack with a given bandwidth capacity would only be able to affect a subset of the Replicators at that level, depending on how many there are at that level and whether their input capacity was 100Mbps or 1Gbps. Depending on all the factors, it would be possible to ensure that even the largest botnet DoS attacks has little impact on the delivery of data to recipients, if there are one or more layers of Replicators below this.

Costly private networks links between Replicators is a significant expense, but will probably be justified for Ivip in order to ensure this critical piece of Internet infrastructure can only be partly affected by the largest DoS attacks.
3.  Non-goals

3.1.  Not intended to provide end-to-end security

While Replicators only receive feeds from upstream replicators they are configured to use, and which accept their credentials, and while DTLS protects the payloads of packets between the Replicators and from the Replicators to the recipient devices, FPR does not provide end-to-end security against either alteration of the data or snooping of its contents.

This is because the recipient has no way of knowing that all the upstream Replicators it relies upon are not under the control of an attacker. A single such compromised Replicator could drive packets to most or all of its downstream Replicators by sending out packets with the identification numbers expected from the genuine source a little earlier than the genuine packets.

The use of DTLS to protect packets sent from Replicators to other Replicators and to recipient devices is intended primarily to prevent any of these accepting a spoofed packet generated by an attacker who does not control any Replicators. This protects against attackers injecting their own packets with bogus payloads.

3.2.  No autodiscovery or monitoring

The current description is for the basic functions of Replicators and later Missing Payload Server. In some applications it may be desirable for the Replicators to automatically choose their upstream and downstream Replicators. In almost any practical system, some kind of diagnostic functions would be needed in order to evaluate performance and debug problems. Such capabilities are for future work.

3.3.  No attempt to automatically adapt to varying PMTUs

To be deployed across today’s DFZ, all packets would need to be less than 1500 bytes long. I will assume 1470 bytes, for convenience, as a PMTU which can reasonably be expected in any DFZ path, because I have observed Google servers sending unfragmentable packets of this length. [DFZ-unfrag-1470]

The FPR system of Replicators has no PMTUD capabilities - and any PMTU problem encountered by a packet will not result in an RFC 1191 Packet Too Big message being sent beyond the upstream Replicator which sent the packet. Replicators would ignore such a message.

The Missing Payload Servers receive streams of packets just like
Replicators and QSDs, so they need to be located where there are no local PMTU restrictions which would prevent the reception of packets of the chosen maximum length. Missing Payload Servers communicate with each other, and handle requests from QSDs, via TCP - which does not involve any special MTU constraints.

In Ivip, is it likely that some Replicators, Missing Payload Servers and QSDs will be located in end-user networks which use SPI (Scalable Provider Independent) addresses. Packets addressed to SPI addresses will pass through an ITR and ETR. (Replicators may include an inbuilt ITR function so the packets it sends don’t have to go to any separate ITR.) If encapsulation is the method used for ITR to ETR tunneling then for IPv4, this involves a 20 byte IP-in-IP header. So the maximum length of a packet which could be handled by the FPR system in this scenario - a UDP packet with DTLS header and payload - is 1450 bytes.

In an ISP or end-user network today where gigabit Ethernet interfaces are always used and where all MTUs support ~9kbyte jumbo-frames, it would be possible to run an FPR network with ~9kbyte packets.

If a 1450 byte FPR system was successfully operating over the DFZ, at some time in the future, when all DFZ paths and likewise paths between all Replicators and recipients could support ~9kbyte packets, there could be a transition to using these larger packets.

Replicators will handle ~9kbyte packets and in principle the same Replicators could begin handling the larger packets without any need for reconfiguring the entire system. If the numbering systems by which the packet payloads are identified did not overlap, and if the Replicators had the capacity, the same system of Replicators could handle the 1460 byte packets and ~9kbyte packets simultaneously.

These larger packets would involve a different way of splitting up the data to be transmitted. Recipient devices (QSDs) may have software which copes automatically with different packet formats, but a more likely scenario is that the switch to jumbo-frames in the future would be accompanied by somewhat different ways of carrying the data - and so by the need for updated recipient software.

3.4. Not intended for mass market consumer applications

At each point - a Replicator, Missing Payload Server or QSD - redundancy is bought by increasing the incoming bandwidth, according to how many upstream Replicators are used. This is expensive for high data-rate applications and so FPR is not intended as a system for delivering audio or video material to mass-market end-users.
It is intended for recipients in ISP networks where the two or more feeds can be chosen to arrive via different physical links, different peering points and different border routers - so the physical diversity available in these settings can be directly employed to provide increased robustness. Since FPR lacks PMTUD capability, it is best used in scenarios where the location of Replicators and recipients is stable and carefully planned, with regard to any PMTU limitations which may affect them.
4. Streams of packets for Replicators and QSDs

All packets discussed below are those which pass the DTLS authentication process, and are presented to the FPR code as DTLS payloads, each consisting of an FPR header and FPR payload. (For simplicity, much of this discussion assumes that these packets are only received by Replicators and QSDs. However, Missing Payload Servers will also receive streams of packets, in exactly the same way.)

In all cases, the receiving device does not distinguish between packets which arrive from one incoming stream from those which arrive from another. This information is available from the DTLS software, but is not important to how the device processes the payloads of each incoming packet.

In this discussion the term QSD (Ivip full-database query server) is used to denote the devices which receive the packets and put their payloads to use, rather than sending the payloads to others, as Replicators do. This helps explain the FPR system’s role within Ivip. If FPR was used for another purpose, the packets would be received and used by some other device.

It would be possible for a single device to function as both a Replicator and a QSD. This may make sense during initial Ivip introduction. However, in a fully deployed Ivip system, with the QSD handling many requests from ITRs (directly and via caching QSCs) and with the QSD having a significant workload receiving the packets and processing them to update its database, separate servers for the QSD and Replicator functions would be the best approach.

The Replicator and QSD code would share some common elements - for the reception and processing of incoming packets. Replicators and QSDs are both required to receive multiple streams of packets. While they may operate with a single stream, two would be a typical number to receive and they may be required to receive many more. These statements apply also to the code for the Missing Payload Server.

A QSD or Missing Payload Server only receives streams from upstream Replicators. 2 streams would be a typical number, but perhaps as many 5 could be used to maximise robustness.

A level 1 or greater Replicator receives typically two or more streams from upstream Replicators in the numerically lower numbered level - which is "above" (upstream) in the inverted tree structure.

Level 0 Replicators receive streams from one or potentially many sources of packets. In Ivip, the sources are multiple RUASes. They
also receive a stream from every other level 0 Replicator.

The FPR system handles the sum of the unique payloads sent by all RUASes. For instance, in a given time period such as 100ms, RUAS-0 sends streams of packets to all five level 0 Replicators, with each stream containing 7 packets with a set of 7 unique DTLS payloads (FPR headers and FPR payloads). While the packets received by one level 0 Replicator are all different from those received by another, due to DTLS encryption, each level 0 Replicator receives 7 packets from RUAS-0, and the DTLS payloads of the packets received by one level 0 Replicator are identical to the DTLS payloads of received by each other level 0 Replicator.

The purpose of RUAS-0 sending five streams containing the same DTLS payloads, one stream to each level 0 Replicator, is to maximise the fault tolerance of the system. If one or two level 0 Replicators are down, or if they can’t be reached from RUAS-0, then there will be no loss of data being sent to the QSDs. Even if RUAS-0 was only able to send its 7 packets to a single level 0 Replicator, or if a single set of 7 was sent to various level 0 Replicators (such as packet 0 to R0-0, packet 1 and 2 to R0-3 and the rest to R0-5) then the system would still deliver all payloads to the QSDs. This is due to the level 0 Replicators being "fully meshed". Every one has an output stream to every other one. So as long as at least one packet with a given payload arrives at any level 0 Replicator, within a fraction of a second, all other level 0 Replicators will receive it as well.

In all cases, the receiving device (Replicator or QSD) establishes the DTLS session with the source of the packets. To continue with the five level example of Figure 1, QSDs establish their DTLS sessions with level 4 Replicators. Typically two would be a good choice, but more could be used. The QSD is configured to use particular level 4 Replicators and the DTLS session can only be established if each level 4 Replicator accepts the username and password provided by the QSD.

Similarly, Replicators at levels 4, 3, 2 and 1 establish DTLS sessions with Replicators at the level above (numerically one less).

Each layer 0 Replicator establishes a DTLS session with each other layer 0 Replicator, and with each RUAS.

In this discussion, it is assumed that there is a strict layering of Replicators. While layer 0 is fully meshed, there is no meshing of other layers - no layer 3 Replicator receives a stream of packets from any other layer 3 Replicator. Also, no Replicator at levels 1 or greater is shown accepting a stream from Replicators at any level other than the one above. The diagram shows the Replicator system
ending at level 4, and with the next level being composed entirely of
QSDs.

There is nothing to prevent a QSD being driven partly or wholly by
streams from Replicators in levels other then 4.

Nor is there anything to prevent a Replicator getting some of its
streams from levels other than the one above. For instance, it would
be possible, in principle, to cross-connect all level 3 Replicators. However, due to their large number (3,000) this would be impractical
and inefficient. The strict layering of Replicators is not
absolutely required, and it may make sense to have a Replicator
driven by streams from a level 2 and a level 3 Replicator. It would
also be possible to take a stream from a level 4 Replicator and feed
it to a level 3 or 2 Replicator. This cannot result in the
equivalent of "routing loops", since most or all of the packets which
arrive from this link will contain payloads which the level 3 or 2
Replicator has already received - so those packets will not lead to
any further action. The advantage of doing this, from a level 4
Replicator which is dependent on different level 3 or 2 Replicators
than those streams are received from, is to provide diversity. If
those streams from the directly used level 3 or 2 Replicators are
disrupted, it is unlikely that there will be the same disruption in
the stream received from the topologically distant level 4
Replicator.

I have presented FPR in a strictly layered arrangement because this
is easier to depict and is theoretically the most efficient way of
fanning out information. However the details of connections between
Replicators and QSDs is not technically constrained by the FPR
system, and can be chosen freely to trade off bandwidth and computing
resources for robustness according to local conditions.

For instance, if a level 3 Replicator in Sydney Australia has
incoming streams from level 2 Replicators in Sydney and Singapore,
analysis of the connections might reveal that these two operate from
three or four level 1 Replicators which are not ideally diverse in a
topological sense, with their origins being mainly in the USA.
Assuming that the higher level Replicator outputs are more difficult
to obtain access to than those of the lower levels, it would be
possible to have a third stream feed this level 3 Replicator, from a
level 3 or 4 Replicator in Russia, which has most of its incoming
streams arriving from Europe. Typically, the packets arriving from
the Russian Replicator would arrive later than those from the higher
level Sydney and Singapore Replicators, and so would be ignored.
However, if there was a network outage which affected both the Sydney
and Singapore Replicators, even for a fraction of a second, the
payloads in the packets arriving from Russia would be used.
Each stream sent to a QSD, Missing Payload Server or a level 1 or
greater Replicator is, under ideal circumstances (no packet loss), a
"complete stream" in that its packets contains a complete set of DTLS
payloads which every QSD, ideally, will receive at least one of.

This is also true of the streams each level 0 Replicator receives
from other level 0 Replicators. If there is a single external source
of packets, then ideally, that source will send a separate "complete"
stream of packets to every level 0 Replicator. Due to the fully-
meshed flooding arrangement of the level 0 Replicators, then -
assuming there were no packet losses - it would suffice for the
single external source to send a single complete stream to just one
level 0 Replicator. Alternatively, the single source could send a
complete stream in various subsets, each to a different level 0
Replicator.

When the FPR system is used in Ivip, it is intended to receive
packets from multiple external sources - each an RUAS system.
Ideally, every RUAS will send its subset of the complete stream to
every level 0 Replicator. In this case, the "complete" stream is the
sum of all packets sent by all external sources. In fact, it would
suffice (assuming again no packet losses) for each external source to
send just a single set of packets to just one level 0 Replicator, or
scattered to various level 0 Replicators, because the payload of each
packet one will flood to all the other level 0 Replicators.

At the highest level - level 0 - the FPR system involves brute-force
flooding and fully-meshed redundancy to ensure that in ordinary
circumstances every level 0 Replicator receives the "complete stream"
- either directly from the one or more external sources, or from its
level 0 peers.

For a global, real-time, system such as Ivip, I anticipate that 4 to
8 level 0 Replicators would suffice. Each would be in a
geographically and topologically different location, and they would
all be meshed by private network links which would, ideally, be
geographically and topologically diverse. In a section below I
discuss the use of private networks to protect against DoS attacks.
5. Packet payloads and identification

Replicators and QSDs decipher each received packet from upstream
Replicators (or for the level 0 Replicators, from RUASes and other
level 0 Replicators) and use the first 32 bits of the DTLS payload to
decide what to do with the entire DTLS payload. The options are to
use it because it is deemed to be "Fresh" payload - a Replicator
replicating it, or a QSD using the payload to update its database -
or to "ignore" it, because the device has already received a packet
with the same payload, meaning it is deemed to be a "Repeat" payload.

Some or potentially all bits of the FPR header are used by the
Replicator or QSD to decide whether the Replicator or QSD has already
received a packet with the same payload. This section describes this
process in principle, and the next describes one way this process
could be implemented in Replicators, Missing Payload Servers and
QSDs.

FPR’s units of replication and flooding are payloads of packets. If
IPv4 packets are limited to 1450 bytes then there are 1422 bytes
available after the IP and UDP headers. If the DTLS header involves
an overhead is 50 bytes then 1372 bytes remain as the "DTLS payload".
(I have not yet researched DTLS in sufficient detail to determine
exactly what the overhead would be. This would depend on choice of
encryption algorithm and Message Authentication Code.)

At the start of the DTLS payload, a fixed number of bits must be
devoted to identifying the packet. I will refer to this as the "FPR
header" and for now assume it is 32 bits. The remainder of the DTLS
payload is the "FPR payload" - and is available for application data.

With these assumptions, the application data is contained in the 1368
byte FPR payload. The FPR software in the QSD (or other recipient
device, if FPR is used for a purpose other than Ivip) should make the
FPR header bits available along with the FPR payload, since it may be
helpful in processing the FPR payload.

In the current design, the only function of the "FPR header" is to
enable each Replicator and QSD to use this header to decide whether
each incoming packet, once deciphered from its DTLS form, is either a
"fresh" or a "repeat" packet with the timeframe T. The exact
algorithm for this decision is described in the following section
below. For now, the definitions are loosely:
Fresh:
No packet with FPR header bits identical to this packet’s FPR header bits has been received in the recent time period T. (Therefore, the packet is assumed to contain a fresh FPR payload and so must be replicated.)

Repeat:
One or more packets with FPR header bits identical to this packet’s FPR header bits value HAS been received in the recent time period T. (The first such packet was replicated, so subsequent packets, which presumably have the same FPR payload, are ignored.)

The idea is that in any given scenario, there may be some mechanism by which a packet could be so delayed by the routing system and Replicators that it arrives some time D later than it otherwise would. The algorithm needs to identify any delayed packet with the same payload as one already received as a "Repeat" which can be ignored.

There are various ways of achieving these goals. One approach is to use the 32 bit FPR header in the following manner. This is purely for Ivip. Other applications would choose to identify the payloads differently. (This is a preliminary exploration, to demonstrate one way of performing the algorithm.)

10 bits: epoch in 1 sec increments (epochsec):
When the RUAS sends out packets with this payload, it sets these 10 bits according to the current time (epoch), quantized to seconds units. RUASes should agree on a common timebase, so all the packets sent at a particular time by all RUASes have the same, or +/-1, values for these bits. This wraps around every 17 minutes 4 secs. Maybe it would be better to have 20 or 32 bits here. This value is not currently used for the Fresh / Repeat algorithm, but it will be used by QSDs and Missing Payload Servers for identifying recent payloads.

7 bits: RUAS identifier (ruas):
This identifies the RUAS which sent the payload, from 128 possible RUASes.

1 bit: Normal / Jumbo (nj)
0 means normal ~1500 byte packet size. 1 means ~9kbyte packet size. To support simultaneous reception of both types of packet, the RUAS will maintain separate sequence number counters for each set of packets.
14 bits: Sequence number (seq):

The RUAS sends out each payload with a sequence number which is one more than that used for the previously sent payload. The Fresh / Repeat algorithm doesn’t rely on this sequential order, but it will help with the retrieval of missing payloads from Missing Payload Servers. These numbers wrap around every 16,384 payloads. The RUAS should not send more than 1000 packets a second, which is about 1.3 megabytes a second, assuming the packets are ~1500 bytes. So “seq” can’t wrap around in less than 16.384 seconds.

The next section explains how these bits are used.
6. The Fresh vs. Repeat Algorithm

This section describes an algorithm for deciding whether DTLS payload (FPR header and FRP payload) is "Fresh" or a "Repeat". I tried using sliding windows and ran into problems. This approach maintains a timer for each sequence number, for each RUAS. This would have been prohibitive in the past, but a quad core ~3GHz CPU would use only a small fraction of its power running these timers.

There could be other ways of implementing this algorithm. The aim here is to show a practical approach, which may not be optimal.

For each of the 128 RUASes, the software maintains an array of $2^{14}$ timer (down-counter) variables for ~1500 byte packets another such array for ~9kbyte packets. (It will be many years before the DFZ supports ~9k byte packets, but the code should be ready to support a separate stream of such packets.)

In the implementation below, the timer variables are 4 bits each, but I use only 3 bits. The 4 bit timer variables are in a multidimensional array, indexed on "ruas" ($2^7$), "nj" ($2^1$) and "seq" ($2^{14}$). So there are $2^{22}$ 4 bit timer variables, occupying 2 megabytes of RAM. This is a few cent’s worth of DRAM. The whole array fits well within the L2 cache of modern multi-core CPUs, which is typically 8 megabytes.

When a packet arrives and is successfully deciphered, the software looks at the four fields: "ruas", "nj" and "seq" in the FPR header. The software uses these to index into the array and read a particular timer variable.

If the timer value is zero, then the payload is deemed to be "Fresh". This is because this payload is the first to be received from this RUAS with this "seq" number in the last 10 or so seconds. The software then sets the timer variable to 5.

Later, after this RUAS has sent another 16,384 payloads, it will send another payload with the same "seq" value. But by then, more than 10 seconds will have elapsed and this timer value will have reached zero - so that new payload will be recognised as Fresh too.

If the timer variable is non-zero, this payload is deemed to be a "Repeat" and no further action is taken on it, or the timer variable. This would occur if another packet with the same payload arrives less than 10 to 12 seconds after the first one.

Meanwhile, a background process steps through all the timer values every 2 seconds - a million a second, which is a fraction of a CPU’s
worth of work, and modern chips have four CPUs. If the value is non-zero, it is decremented. If it is zero (which most of them will be) the variable is not changed. There is no need for locking these timer variables, since these two types of access are thread-safe. The payload handling code only writes the variable if it was zero and the timer code only writes to it if it was non-zero.

Since the down-counting operation is asynchronous with respect to the payload handling code, it could be 0.0 to 2.0 seconds before the first decrement operation. The actual time required for the counter to reach zero after a Fresh packet is recognised will be between 10.0 and 12.0 seconds.

This arrangement will reject as a "Repeat" any second occurrence of a payload which arrives up to 10 seconds after the first ("Fresh") one. It may reject one which arrives as much as 12 seconds later. I assume that the Replicators themselves do not delay packets and that the routing system would never deliver a packet with such delays which would amount to 10 seconds. If a longer time is required, this algorithm could be modified.
7. RUAS functionality

With the assumptions from the previous section, there can be up to 128 RUASes. Each can generate up to 1000 DTLS payloads per second. However, the total FPR system will have a specified maximum data rate, probably at a granularity of a short time such as a few milliseconds to a few tens of milliseconds. Therefore, there needs to be some arrangement by which the RUASes cooperate so the rate at which packets (really DTLS payloads) are replicated by the level 0 Replicators does not exceed this maximum.

Each RUAS has its own section of the $2^{22}$ bit numbering range to use as sequence numbers for its DTLS payloads for - the value it writes to the "ruas", "nj" and "seq" fields in the FPR header. For the ~1500 byte stream of payloads, the RUAS must cycle through the $2^{14}$ range of "seq" sequentially. If it is also sending jumboframe packets, it will maintain an independent counter to set the "seq" bits in those payloads.

As part of this cycling, the RUAS should not, within 12.0 seconds, generate two DTLS payloads with the same particular value for "seq" in their FPR headers, but with different FPR payloads.

All RUASes should use a common timebase for setting the "epochsec" field in the payloads they generate.

At a bare minimum, for the RUAS to successfully launch a DTLS payload, it must deliver a packet containing that payload to at least one level 0 Replicator. This is assuming all the level 0 Replicators are operating and that they are fully meshed - with each receiving a stream from the others. If the RUAS only delivered the payload to a single level 0 Replicator, which was not sending a stream to any other level 0 Replicator, but was sending streams to all its downstream level 1 Replicator, then depending on the interconnections at the various levels, this may not result in the payload being delivered to all QSDs.

Therefore, the RUAS should ideally have a stream to each level 0 Replicator, to maximise the chance that most or all of these Replicators receive the payload directly, or from another such Replicator.

The question of how RUASes format the data in the FPR payload, for the purposes of reassembly in the QSDs and so that QSDs can use end-to-end encryption to check its authenticity, is outside the scope of the FPR system.
8. Replicator Functionality

Most Replicators will need to receive two or perhaps a few more streams from upstream Replicators. Level 0 Replicators will receive many more streams. Firstly, they will receive a stream from each other level 0 Replicator. Secondly they will receive a stream from each RUAS.

The same Replicator code should be usable at all levels, so Replicators in general should be capable of receiving over 100 input streams. This does not mean the total volume of packets would be 100 times the complete set of payloads the FPR system is replicating.

If we assume an upper limit of 8 level 1 Replicators, then the worst case quantity of packets any Replicator must handle is 8 times the total actually being replicated. This would be when a level 0 replicator receives the total set collectively from the 100 or so RUASes and then receives the same set from each of the 7 streams from the other level 0 Replicators. So this provides a reasonable definition of how many DTLS sessions a Replicator may need to create to "upstream" devices - and the total volume of data it should be able to receive via these sessions.

Except for monitoring purposes, the Replicator makes no distinction between DTLS payloads which arrive from any of its upstream sources. Each such payload is handled, as described above, by the Fresh / Delayed algorithm. Only payloads deemed Fresh require any further action.

Each Fresh payload is replicated to all the downstream devices, each with its own DTLS protection, due to each such session having different session keys and states. Just as the incoming streams are unidirectional, so are the output streams. Apart from DTLS handshakes, each Replicator does not send packets upstream, or receive them from downstream, devices.

The replication process does not alter the DTLS payload. There is no hop-count or checksum to check or update. The same DTLS payload is simply sent out via all downstream DTLS sessions. It would be best if this was scheduled to even out the flow of packets for each such session. So a DTLS payload would be sent out on session 0, then on session 1, etc. rather than sending two or more different DTLS payloads on any one session one after the other.

When a Replicator is handling a jumboframe stream as well as the ordinary ~1500 byte stream, it maintains separate input and output sessions for the jumboframe packets. So the structure of links between Replicators for jumboframe packets could be identical to that
for ~1500 byte packets, could be similar or could be entirely different. Therefore, a Replicator which handles both will need approximately double the DTLS sessions and of course bandwidth and CPU power to handle both.
9. QSD Functionality

The QSD receives incoming streams as just described for Replicators. However, a QSD would only receive all ~1500 byte streams, or all jumboframe streams. Therefore, its Fresh / Repeat algorithm only needs half the number of timer variables as a Replicator.

QSDs don’t receive streams from the numerous RUASes, and it is probably safe to assume that no-one would run a QSD with more than 8 input streams. So while a QSD is only required to handle up to 8 or so DTLS sessions, each of these streams would be a complete stream, so the incoming data rate requirement is the same as that of a Replicator – 8 times the total data rate of the complete stream.

When Fresh DTLS payloads are received, their contents – the 32 bit FPR header and FPR payload is passed to the rest of the QSD software, and the mapping information in these payloads will be interpreted as will be described in [I-D.whittle-ivip-db-fast-push].

This processing will involve some kind of end-to-end integrity checking, involving the public key of the RUAS which sent the payload. With the above arrangement, the RUAS of the packet can easily be determined from the "ruas" field in the FPR header.

Perhaps it will be possible to individually authenticate every payload – but I am concerned about devoting too much space in every payload to the required MAC bits. This concern would not apply to jumboframe payloads which are much longer. Checking each payload would be simpler, but more costly in terms of CPU resources and space used in each payload. Assembling information from multiple payloads into a larger block for authentication would be more efficient, but more complex. It also means that a missing payload will delay the use of information in other payloads.

Exactly how the end-to-end authentication will be done is for future work. It depends more on the Ivip mapping system than on the FPR system itself, so I intend to explore this in the future in [I-D.whittle-ivip-db-fast-push].

QSDs will also need to recognise any missing packets and to download a replacement. The algorithm for this is for further work, but the 10 bit "epochsec" field will also be useful for this. Missing packets could be detected, after a second or two, by a gap in the sequence numbers of payloads from a given RUAS.

Perhaps one form of request for missing packets might be to send two 32 bit values, containing the FPR headers of the successfully received payloads which bracket the assumed missing packets. The
full 32 bits uniquely identifies each payload, and the 1 second resolution "epochsec" field will enable the Missing Payload Server to narrow down its search through its cache.
10. Further elaborations

The above is a reasonably exhaustive exposition on the early design phase of a simple, but flexible, data replication system. Here are some elaborations to be more fully developed in the future.

10.1. Missing Payload Servers (MSPs)

I originally planned for QSDs to request payloads they did not receive from a handful of HTTP servers run by each RUAS. This could have scaling problems, so I have developed an alternative which is closely integrated with the basic FPR system of Replicators.

In Ivip, the RUAS will be making snapshots of the mapping information for each MAB (Mapped Address Block) on a regular basis, such as every 5 minutes or so. It will make these snapshots (in a compressed form) available via several HTTP servers so QSDs all over the world can download them during initialization. If a QSD was more than a few minutes behind with missing payloads, then it would be better for it to download the most recent snapshot instead and apply the updates it has received since that snapshot was made. So the missing packet server probably only needs to handle packets in the last 5 or so minutes. This fits well with the 10 bit "epochsec" field in the FPR header.

I haven’t yet decided how a QSD can specify which missing payload(s) it wants. One method may be to send the 32 bit FPR headers of the last payload received before the missing payloads and of the first payload received afterwards.

I considered using a UDP protocol for requesting and receiving missing payloads from MSPs, but chose TCP instead, probably HTTP or HTTPS over TCP. This avoids any PMTU problems and removes the need for acks, resending queries and responses. TCP also avoids difficulties inherent in lightweight UDP protocols where the MSP could be used to amplify small query packets with spoofed source addresses into larger responses to DoS a victim.

An MPS (Missing Packet Server) is a COTS server running software with an input stage identical to that of a Replicator or QSD. That is, it uses DTLS to receive two or more streams from Replicators and it uses the Fresh / Repeat algorithm to ignore all but the first appearance of a new payload.

An MPS at a particular location would receive a stream from one or more physically and topologically close Replicators and ideally from some physically and topologically distant Replicators. "Topology" in this case means not just the underlying DFZ topology, but also that
distant Replicator’s location in the "topology" of upstream Replicators.

The aim is to receive at least one local stream, which is inexpensive - probably from a Replicator in the same data center - and one or a few streams from distant Replicators. This is so that in the event of the local Replicator suffering an outage and so missing some packets, it is likely that the distant one will not be missing the same packets. If the local outage, such a complete loss of connectivity for a few seconds, or significant packet loss due to congestion, also affects the ability of the MPS to receive packets from distant Replicators, then the same packets may be lost. A simple workaround for this is to have the distant Replicator delay its stream by ten seconds or so. Such delayed outputs from a Replicator should only be used to drive QSDs and MSPs - never another Replicator.

MPSes do not need to interpret the packets in order to update a mapping database, as does a QSD. The MPS does not need to interpret the payloads at all, or perform end-to-end authentication on their contents. The MPS only needs to store complete DTLS payloads for ten minutes or so and be able to provide them to requesters. The requesters will be either QSDs or other MPSes. So an MPS is a relatively light-weight network element. It may be quite busy at times responding to queries and sending out payloads, but most of the time, it is storing payloads in a simple fashion, and is not required to do any work on their contents.

The request protocol does not need to be secure, since Ivip mapping information is public information. However, each MPS may wish to restrict its queriers to those which match an ACL.

By some means TBD, each QSD and MPS could be configured to use several MPSes - including perhaps a distant one which is unlikely to be affected by any brief local outage which caused this QSD to be missing some packets. (Folks in North America, Siberia and Africa would have reason to give each other access to their MPSes!)

In ordinary operation, each MPS would have a complete list of recent packets. If it was missing some packets, it would determine this by looking at the FPR headers and finding a gap in the "seq" numbers recently received for each RUAS.

It would be scalable for each MPS to maintain a TCP connection with another MPS so the two could use the one link to request and deliver missing packets in both directions. Therefore, the MPSes could be arranged in multiple partially meshed groups - or these could be connected and so form a single global network of MPSes. The request
protocol would probably need an option to cancel a request. For instance, an MPS in Los Angeles might first request one or more missing packets from an MPS in New York. But if the NY MPS replies that it too is missing these packets, the LA MPS might request them from an MPS in Beijing - which responds that it has them, and starts sending them. The LA MPS will then want to cancel the request to the NY MPS. HTTP or HTTPS is probably a good protocol for this purpose.

QSDs would use the same protocol for querying MSPs. Whether the QSD starts an HTTP(S)-TCP connection when it needs missing packets, or whether it maintains such a connection in readiness, would be a matter for local policy.

In this scenario, MPSes form an interdependent network, which will be highly robust. Most MPSes will have all the recent packets. Those which don’t will automatically obtain them from other MPSes within a few seconds.

An ISP which runs one or a few QSDs could run a missing packet server for all of them, with long-lasting sessions to a few other MSPs in nearby and distant ISPs. Alternatively, the QSDs could use one or more MSPs operated by other organisations, perhaps on a commercial basis.

Since an MSP is simply software running on a COTS server, they are not expensive or difficult to deploy. It would be possible to run an MPS on the same host as a QSD, but if they are using streams from the same Replicators, then there will be a high correlation between the sets of packets which each function misses. Therefore, it makes sense for each QSD to use a nearby MSP, and then a distant one, rather than to run an MSP at the same site which will need to make much the same queries of other MSPs as the QSC would.

10.2. Delaying the output of Replicators

If a QSD or MSP relied on a single upstream physical link, or a router or other device which might be subject to transient disruption, then having multiple streams from upstream Replicators will not necessarily ensure the QSD gets all the payloads which are sent. This is because the disruption will likely affect all such streams, which will be carrying much the same payloads at the same time.

A possible workaround is to have one or more of the streams delayed at its source - in the output function of its Replicator. If one such stream was delayed by 5 seconds, then it would typically be able to deliver every payload which was not delivered during a 4 second disruption.
So it may be desirable for delays such as this to be an option when a QSD or MSP requests a stream from a Replicator. A Replicator does not see a QSD request any differently from the request from another Replicator. So the question arises as to whether the stream from one Replicator to another should be delayed - and if so, by how much.

It should be reasonably safe for a QSD or MPS to receive a stream with a delay of a few seconds, since the QSD does not propagate the payloads any further. The time could be locally chosen so that when added to a reasonable estimate of the longest delay affecting packets going into that Replicator, that there is still a safety margin within the minimum timeout of the QSDs timers for the purposes of Fresh / Repeat detection.

To delay the packets received by a Replicator would be much more problematic. These delayed payloads could be propagated to other Replicators - and these delays could be added to by similar arrangements between other Replicators. Then, the total delay might exceed the limits of the Fresh / Repeat algorithm and QSDs and Replicators would mistake older payloads for ones which were actually sent 15 seconds or so later. (This could be prevented with a more elaborate algorithm, which also uses the 10 "epochsec" bits, but I think this raises further complications.) This would only occur with the highest allowed data-rates which might not occur in practice until the system was being used intensively - many years in the future.

I think that delaying a stream to a Replicator could in principle improve its robustness if its two streams were likely to be subject to the same brief disruptions. However, it would be better to locate Replicators at data centres with multiple physical links and to try to ensure that the streams are most likely to arrive over diverse links. QSDs will cope with missing packets, and the aim of the Replicator system is to minimise the number of packets they miss. However, Replicators should not contribute to delays which might disrupt the ability of QSDs and other Replicators to correctly distinguish a Fresh packet from a Repeat.

10.3. Private network links to avoid DoS attacks

The Replicator system as described above is a promising method of fanning out information to a very large number of recipient devices all over the world, in fractions of a second. While the system is distributed and has no single point of failure, if it was used for a purpose as important as distributing mapping for a core-edge separation system such as Ivip, it would no-doubt be threatened by DoS attacks in the form of gigabits per second of packets directed from large numbers of hacked botnet PCs.
Internet protocols are intended to operate on the open Internet. However, the use of FPR may be a partial exception. Some root nameservers are toughened against DoS by being distributed to multiple high-bandwidth sites using anycast. In principle, by having enough fully meshed level 0 Replicators, the same goal could be achieved — for an attack to succeed, it would need to overwhelm all, or almost all of the devices at the same time.

To some extent this can be achieved with Replicators, but it would probably be best to toughen the system against DoS attacks by linking the RUASes, the level 0 Replicators and at least the level 1 Replicators over private network links with assured bandwidth and no possibility of being affected by packets arriving from the Internet. In this case, the RUASes and level 0 Replicators may have private addresses. The level 1 Replicators may also have private addresses on their input side — the part which makes DTLS links to level 0 Replicators.

In this model, the output addresses of the level 1 Replicators would be on public addresses so level 2 Replicators could establish sessions with them. Probably these level 1 Replicators would have two separate gigabit Ethernet ports — one for the private address and upstream links and the other for the public addresses and downstream links.

The downstream public address of the level 1 Replicators might be the target of a DoS attack, but once the sessions have been established, Replicators do not need to receive any packets on those DTLS sessions. So a DoS attempt there would have little or no effect.

Instead, a DoS attack would need to focus on the level 2 Replicators. Ideally, depending on the capacity of the attackers, these would be so numerous that an attack could only disrupt a subset of them. Even then, due to the cross-linked nature of the Replicator system, the impact of that attack on QSDs may be greatly diluted due to level 3 and 4 Replicators working fine from streams arriving from level 2 Replicators which were not targeted.

Using private network links to fully mesh the level 0 Replicators, and for their streams to the level 1 Replicators, is a non-trivial matter. However, the benefits of a fast push mapping distribution system core-edge separation scheme for the Internet in general are immense — so this expense is therefore worth considering.

If the first two levels are carefully optimised so that there are, for instance, 5 to 8 level 0 Replicators (only two or three are needed for highly reliable operation) and 50 to 100 or level 1 Replicators (of which quite a few could be dead without significantly
disrupting streams to QSDs) then this system could drive 1000 to 2000 or perhaps more level 2 Replicators. This would probably make the system largely immune to DoS attacks - but of course the exact details would need to be considered at the time of deployment.
11. Security Considerations

For future work, but see notes above about the need for end-to-end authentication, and hardening against DoS attacks.
12. IANA Considerations

[To do.]
13. Informative References

[DFZ-unfrag-1470]

[I-D.ietf-lisp]

[I-D.whittle-ivip-arch]

[I-D.whittle-ivip-db-fast-push]

[I-D.whittle-ivip-drtm]
Whittle, R., "DRTM - Distributed Real Time Mapping for Ivip and LISP", draft-whittle-ivip-drtm-00 (work in progress), February 2010.


[TTR Mobility]
Author’s Address

Robin Whittle
First Principles

Email: rw@firstpr.com.au