iPRP: Parallel Redundancy Protocol for IP Networks

Miroslav Popovic*, Maaz Mohiuddin*, Dan-Cristian Tomozei* and Jean-Yves Le Boudec*
*I&C-LCA2, EPFL, Switzerland

***Abstract*—Reliable packet delivery within stringent delay-constraints is of paramount importance to industrial processes with hard real-time constraints, such as electrical grid monitoring. Because retransmission and coding techniques counteract the delay requirements, reliability is achieved through replication over multiple fail-independent paths. Existing solutions such as the parallel redundancy protocol (PRP) replicate all packets at the MAC layer over parallel paths. PRP works best in local area networks, e.g., sub-station networks. However, it is not viable for IP-layer wide-area networks, a key element of emerging smart grids. Such a limitation on scalability, coupled with lack of security, and diagnostic inability, renders it unsuitable for reliable data-delivery in smart grids. To address this issue, we present a transport-layer design: IP parallel redundancy protocol (iPRP). Designing iPRP poses non-trivial challenges in the form of selective packet-replication, soft-state and multicast support. In addition to unicast, iPRP supports multicast, widely used in smart-grid networks. It replicates only time-critical UDP traffic. iPRP only requires a simple software installation on the end-devices. There are no other modifications needed to the existing monitoring application, end-device operating system or to the intermediate network devices. iPRP has a set of diagnostic tools for network debugging. With our implementation of iPRP in Linux, we show that iPRP supports multiple flows with minimal processing-and-delay overhead. It is being installed in our campus smart-grid network and is publicly available.**

I. Introduction

Specific time-critical applications (found for example in electrical networks) have such strict communication-delay constraints that retransmissions following packet loss can be both detrimental and superfluous. In smart grids, critical control applications require reliable information about the network state in quasi-real time, within hard delay-constraints of the order of approximately 10 ms. Measurements are streamed periodically (every 20 ms for 50 Hz systems) by phasor measurement units (PMUs) to phasor data concentrators (PDCs). In such settings, retransmissions can introduce delays for successive, more recent data that in any case supersede older ones. Moreover, IP multicast is typically used for delivering the measurements to several PDCs. Hence, UDP is preferred over TCP, despite its best-effort delivery approach. Increasing the reliability of such unidirectional (multicast) UDP flows is a major challenge.

A. Problems with MAC-Layer Parallel Redundancy Protocol

The parallel redundancy protocol (PRP) IEC standard [1] was proposed as a solution for deployments inside a local area network (LAN) where there are no routers. Communicating devices need to be connected to two cloned (disjoint) bridged networks. The sender tags MAC frames with a sequence number and replicates it over its two interfaces. The receiver discards redundant frames based on sequence numbers.

PRP works well in controlled environments, such as a substation LAN, where network setup is entirely up to the substation operator, who ensures that the requirements of PRP are met (e.g., all network devices are duplicated). At a larger scale (for example, a typical smart grid communication network that spans an entire distribution network) routers are needed and PRP can no longer be used. Thus, a new solution is needed for IP wide area networks (WANs).

In addition to extending PRP functionality to WANs, the new design should also avoid the drawbacks of PRP. The most limiting feature of PRP is that the two cloned networks need to be composed of devices with *identical* MAC addresses. This contributes to making network management difficult. Furthermore, PRP duplicates all the traffic unselectively, which is acceptable for use in a LAN, but which cannot be done in a WAN, because links are expensive and unnecessary traffic should be avoided. Moreover, PRP has no security mechanisms, and multicasting to a specific group of receivers is not natively supported. As a layer-2 protocol, PRP supports multicast by way of broadcast, because multicast in layer 2 is implemented as broadcast. This is acceptable in a LAN, but not in a WAN. As modern smart grids use WAN for communication, supporting selective multicast, i.e. IP multicast, is a key requirement for a parallel redundancy protocol.

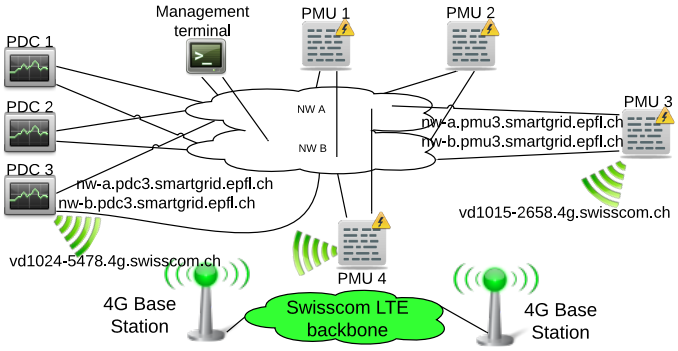

Fig. 1: A typical iPRP use-case in the context of smart grids. Devices (PDCs, PMUs) are connected to two overlapping network subclouds (labeled A and B). Some devices use an additional LTE connection providing a low latency cellular service [2]. Every PMU streams data to all PDCs, using UDP and IP multicast.

Concretely, Fig. 1 depicts a smart grid WAN where PRP cannot be directly deployed: devices are multi-homed and each interface is assigned a different IP address. Most devices have two interfaces connected to a main network cloud made of two fail-independent network subclouds labeled “A” and “B”, while some have a third interface connected to a 4G cellular wireless service (labeled “Swisscom LTE backbone” in the figure). It is assumed that paths between interfaces connected to the “A” network subcloud stay within it (and similarly with “B”). The “A” and “B” network subclouds could be physically separated, however in practice they are most likely interconnected for network management reasons.

A simple way to achieve the arrangement described before

is to divide the network into two logical subclouds, *A* and *B*. Then, by adjusting the routing weights of the links interconnecting the *A* and *B* subclouds, we can ensure that $A \rightarrow A$ and $B \rightarrow B$ traffic stays within *A* and *B* subclouds, respectively, thereby giving rise to fail-independent paths. In such a setting, the interconnections will be used only for $A \leftrightarrow B$ traffic.

We need a solution that, similarly to PRP, takes advantage of network redundancy for increasing the reliability of UDP flows, and that works in scenarios such as the one in Fig. 1. The existence of fail-independent paths is fundamental for the optimal operation of such a solution. However, in the event of a network-component failure, the paths can partially overlap. Then, the solution should reap the maximum possible benefits by operating in a degraded-redundancy mode. In other words, if complete end-to-end redundancy is no longer possible, the solution should continue to work.

In order for our solution to be easily deployed, we also require it to be transparent to *both* the application and network layers: it should only require installation at end-devices and no modifications to running application software or to intermediary network devices (routers or bridges).

In this paper we present the design and implementation of iPRP (IP parallel redundancy protocol), a transport layer solution for transparent replication of unidirectional unicast or multicast UDP flows on multihomed devices.

B. iPRP

An iPRP host has to send different copies of the same packet over different paths. With the current technology, a device cannot control the path taken by an IP packet, beyond the choice of a destination address, exit interface and a type-of-service value. Other fields, such as the IPv6 flow label or source routing header extensions, are either ignored or rejected by routers. Also, the type-of-service field is used by applications and should not be tampered with by iPRP. Hence, we assume that a choice of the path is done at the sources by choosing communication interface and the destination address. The job of iPRP is then to transparently replicate packets over the different interfaces for the UDP flows that need it, match corresponding interfaces, remove duplicates at the receiver, and do this in a way that is resilient to crashes (see Section IV-G).

Not all traffic requires replication, only certain devices and certain UDP flows do (time-critical data). Hence, replication needs to be selective: a failure-proof mechanism, transparent to applications, is required for detecting and managing packet replication. It needs to match well the interfaces, so that independent paths are used whenever they exist. However, the solution should continue to work if some paths are not disjoint.

The iPRP protocol design is such that it does not interfere with the existing security mechanisms and does not introduce any new security weaknesses (see Section V).

iPRP assumes that the network is traffic-engineered; the critical UDP data streams receive enough resources and are not subject to congestion. iPRP instantly repairs packet losses due to failures or transient problems such as transmission losses. It *does not* solve congestion problems due to under-dimensioned network links. TCP flows are not affected.

Our iPRP implementation is for IPv6, as it is being installed in our smart-grid communication network (smartgrid.epfl.ch), that uses IPv6 (following the argument that new network environments should avoid future transition problems and embrace IPv6 from the start). Our implementation is available at http://goo.gl/N5wFNt. Adaptation to IPv4 is straightforward.

II. Related Work

As mentioned in Section I, iPRP overcomes the limitations of PRP [1]. The authors of [3] are aware of the fact that PRP is limited to LANs and suggest a direction for developing PRP in an IP environment. Their suggestion is neither fully designed nor implemented. Also, it requires that the intermediate routers preserve the PRP trailers at the MAC layer, which in turn requires changes in all of the routers in the networks. It does not address all the shortcomings of PRP (diagnostic tools, lack of multicast support, need of special hardware). In contrast, our transport layer approach does not have these drawbacks.

MPTCP [4] is used in multi-homed hosts. It allows TCP flows to exploit the host's multiple interfaces, thus increasing the available bandwidth for the application. Like MPTCP, iPRP is a transport layer solution and is transparent to network and application. Unlike MPTCP, iPRP replicates the UDP packets on the parallel paths, while MPTCP sends one TCP segment on only one of them. In a case of loss, MPTCP resends the segment on the same path until enough evidence is gathered that this path is broken. So, a lost packet is repaired after several RTTs (not good for time-critical flows).

Similarly, link aggregation control protocol (LACP) [5] and equal-cost multi-path routing (ECMP) [6] require seconds for failover. LACP enables the bundling of several physical links together to form a single logical channel. The failure of a link is discovered through the absence of keep-alive messages that are sent every $1-30$ s. ECMP can be used together with most routing protocols in order to balance traffic over multiple best paths when there is a tie. In a case of failure, it relies on the reconfiguration of the underlying routing protocol, that is commonly detected by the absence of keep-alive messages.

Network coding exploits network redundancy for increasing throughput [7], and requires intermediary nodes to recode packets (specialized network equipment needed). Also, it is not suitable for time-critical applications as typically packets are coded across "generations" which introduces decoding delays. Source coding (e.g. Fountain codes [8]) can be useful for the bursty transmissions of several packets. However, it adds delay, as encoding and decoding are performed across several packets (not suitable for UDP flows with hard-delay constraints).

MPLS-TP $1 + 1$ protection feature [9] performs packet duplication and feeds identical copies of the packets in working and protection path. On the receiver side, there exists a selector between the two; it performs a switchover based on some predetermined criteria. However, some time is needed for fault detection and signaling to take place, after which the switchover occurs. Hence, a 0-ms repair cannot be achieved.

Multi-topology routing extends existing routing protocols (e.g. [10]) and can be used to create disjoint paths in a single network. It does not solve the problem of transparent packet replication, but can serve as a complement to iPRP in the following way. On top of the underlying network (base topology) additional class-specific topologies can be created as a subset of base topology. We can use this feature to define fail-independent A and B subclouds in order to ensure fail-independent paths between sources and destinations.

Another method to ensure the discovery of fail-independent paths is software-defined networking (SDN) [11]. Centralized controler is aware of the overall network topology and can impose routing rules in a way that guarantees independent paths/trees between all the hosts.

III. Operation of iPRP

A. How to Use iPRP

iPRP is installed on end-devices with multiple interfaces: on streaming devices (the ones that generate UDP flows with hard delay constraints) and on receiving devices (the destinations for such flows).

Streaming devices (such as PMUs) do not require special configuration. Streaming applications running on such devices benefit from the increased reliability of iPRP without being aware of its existence. iPRP operates as a modification to the UDP layer.

On receiving devices the only thing that needs to be configured is the set of UDP ports on which replication is required. For example, say that an application running on a PDC is listening on some UDP port for measurement data coming from PMUs. After iPRP is installed, this port needs to be added to the list of iPRP monitored ports in order to inform iPRP that any incoming flows targeting this port require replication. The application does not need to be stopped and is not aware of iPRP.

Nothing else needs to be done for iPRP to work. In particular, no special configuration is required for intermediary network equipment (routers, bridges).

B. General Operation: Requirements for Devices and Network

iPRP provides $1 + n$ redundancy. It increases, by packet replication, the reliability of UDP flows. It does not impact TCP flows.

iPRP-enabled receiving devices configure a set of UDP ports as *monitored*. When a UDP packet is received on any of the monitored ports, a one-way *soft-state iPRP session* is triggered between the sender and the receiver (or group of receivers, if multicast is used). *Soft-state* means that: (i) the state of the communication participants is refreshed periodically, (ii) the entire iPRP design is such that a state-refresh message received after a cold-start is sufficient to ensure proper operation. Consequently, the state is automatically restored after a crash, and devices can join or leave an iPRP session without impacting the other participants.

Within an iPRP session, each replicated packet is tagged with an iPRP header (Section IV-D). It contains the same *sequence number* in all the copies of the same original packet. At the receiver, duplicate packets with the same sequence number are discarded (Section IV-E). The original packet is reconstructed from the first received copy and forwarded to the application.

In multicast, all devices in the group of receivers need to run iPRP. If by mishap only part of the receivers support iPRP, these trigger the start of an iPRP session with the sender and benefit from iPRP; however, the others stop receiving data correctly. iPRP requires that the multicast communication uses the an IP address that supports source-specific multicast (SSM).

All iPRP-related information is encrypted and authenticated. Existing mechanisms for cryptographic key exchange are applied (security considerations in Section V) .

C. UDP Ports Affected by iPRP

iPRP requires two system UDP ports (transport layer) for its use: the *iPRP control port* and the *iPRP data port* (in our implementation 1000 and 1001, respectively). The iPRP control port is used for exchanging messages that are part of the soft-state maintenance. The iPRP data port receives data messages of the established iPRP sessions. iPRP-capable devices always listen for iPRP control and data messages.

The set of monitored UDP ports, over which iPRP replication is desired are *not* reserved by iPRP and can be any UDP ports. UDP ports can be added to/removed at any time from this set during the iPRP operation. Reception of a UDP packet on a monitored port triggers the receiver to initiate an iPRP session. If the sender is iPRP-capable, an iPRP session is started (replicated packets are sent to the iPRP Data Port), else regular communication continues.

D. Matching the Interconnected Interfaces of Different Devices

One of the design challenges of iPRP is determining an appropriate matching between the interfaces of senders and receivers, so that replication can occur over fail independent paths. To understand the problem, consider Figure 1 where the PMUs and PDCs have at least two interfaces. The A and B network subclouds are interconnected. However, the routing is designed such that, a flow originating at an interface connected to subcloud A with a destination in A, will stay in subcloud A. A potential problem can arise if a sender's interface, say SA, intended to be connected to the A subcloud, is mistakenly connected to the B subcloud, and vice-versa. Then one path from source to destination will go from SA (on subcloud B) to the destination interface DB (on subcloud B), and conversely on the other path. Following the routing rules, these flows will use interconnecting links between A and B subclouds. This is not desirable as these links can be of insufficient capacity because they are not intended to carry such traffic. Furthermore, it is no longer guaranteed that such paths are disjoint. PRP avoids this problem by requiring two physically separated and cloned networks. iPRP does not impose these restrictions. Hence, iPRP needs a mechanism to match interfaces connected to the same network subcloud.

To facilitate appropriate matching, each interface is associated with a 4-bit identifier called *iPRP Network subcloud Discriminator (IND)*, which qualifies the network subcloud it is connected to. The iPRP software in end-devices learns each of the interfaces' INDs automatically via simple preconfigured rules. Network routers have no notion of IND. A rule can use the interface's IP address or its DNS name. In our implementation, we compute each interface IND based on its fully qualified domain name. In Figure 1, the rule in the iPRP configuration maps the regular expression `nw-a*` to the IND value `0xa`, `nw-b*` to IND `0xb`, and `*swisscom.ch` to IND `0xf`, respectively.

The receiver periodically advertises the IP addresses of its interfaces, along with their INDs to the sender (via `iPRP_CAP` messages). The sender compares the received INDs with its own interface INDs. Only those interfaces with matching INDs are allowed to communicate in iPRP mode. In our example, IND matching prevents iPRP to send data from a PMU A interface to a PDC B interface. Moreover, each iPRP data packet contains the IND of the network subcloud where the packet is supposed to transit (see Section IV-D). This eases the

monitoring and debugging of the whole network. It allows us to detect misconfiguration errors that cause a packet expected on an A interface to arrive on a B interface.

IV. Protocol Description

The iPRP message exchange is divided into two planes: control plane and data plane. The control plane is responsible for exchange of messages required to establish and maintain an iPRP session. The data plane is responsible for replication and de-duplication of time-critical UDP flows. Note that, control plane messaging is non-time critical and far less frequent than data plane (data plane $\sim$ ms, control plane $\sim$ s).

The data plane operation is divided into two phases: replication phase and duplicate discard phase. Next, we discuss the operation of each plane and the description of key elements of iPRP protocol in detail.

A. Control Plane

The control plane is used for exchange of messages to establish and maintain an iPRP session. The iPRP session establishment is triggered when a UDP packet is received at some monitored UDP port p. In Fig. 3, UDP port p is made monitored at t_1 at the receiver, by adding it to the list of monitored ports. When the iPRP session establishment triggers, the receiver's *soft-state-maintenance* functional block (Fig. 2) adds the sender to the list of active senders (Alg. 1).

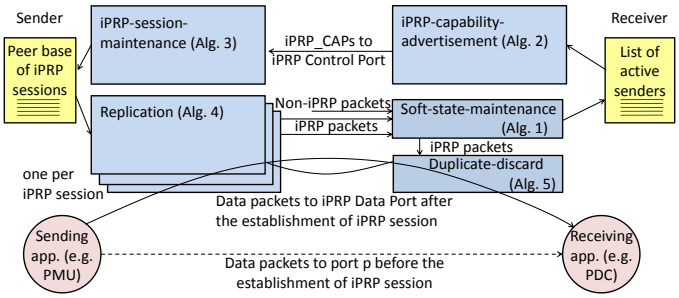

Fig. 2: Overview of the functional blocks.

Algorithm 1: (At the receiver) Soft-state maintenance (keeps the list of active senders up-to-date)

```
while true do
    remove inactive hosts from the list of active senders (last-seen timer expired);
    for every packet received on one of the monitored ports or on iPRP Data Port do
        check if the source is in the list of the active senders;
        if yes then
            update associated last-seen timer;
        else
            put sender in the list of active senders;
        end
    end
end
```

The *iPRP-capability-advertisement* functional block (Fig. 2) at the receiver, sends `iPRP_CAP` to the control port of the sender every T_{CAP} seconds (t_2 in Fig. 3, Alg. 2). This message informs the sender that the receiver is iPRP enabled and provides information required for selective replication over

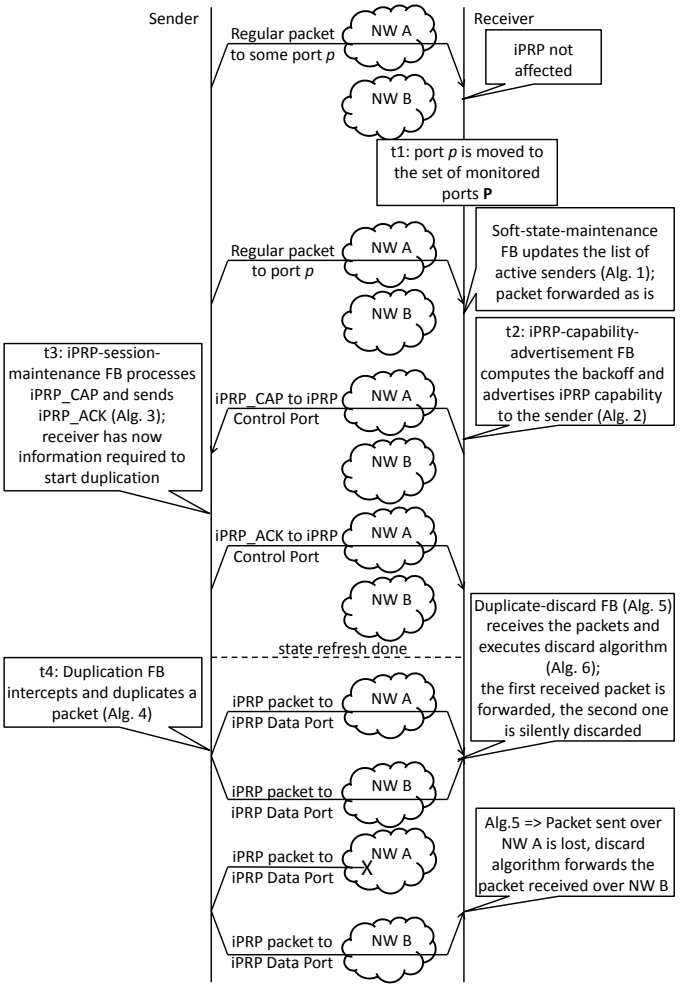

Fig. 3: Message sequence chart for typical scenario when iPRP-capable devices are starting iPRP operation.

Algorithm 2: (At the receiver) iPRP capability advertisement

```
while true do
    compute T_backoff (Section IV-F);
    listen for iPRP_ACKs until T_backoff expires;
    send iPRP_CAP messages to all hosts in the list of active senders from which no iPRP_ACKs are received;
    sleep T_CAP − T_backoff;
end
```

alternative paths. It contains: (1) the iPRP version; (2) INDs of the network subclouds to which the receiver is connected, to facilitate IND matching (see Section III-D); (3) the source and destination UDP port numbers of the packet that triggered iPRP session establishment; (4) in multicast, the multicast IP address of the group; (5) in unicast, IP addresses of all receiver interfaces; (6) a symmetric, short-lived cryptographic key for authentication and encryption of iPRP header (Section V)

On receiving the `iPRP_CAP`, the *iPRP-session-maintenance* functional block (Fig. 2) at the sender acknowledges it with an `iPRP_ACK`. The `iPRP_ACK` contains the list of sender IP addresses which are used by the receiver to subscribe to alternate network subclouds to

Algorithm 3: (At the sender) iPRP session maintenance

```
1  while true do
2      remove aged entries from the peer-base;
3      for every received iPRP_CAP message do
4          if there is no iPRP session established with the
           destination then
5              if IND matching is successful then
6                  establish iPRP session by creating new
                   entry in the peer-base;
7                  send iPRP_ACK message;
8              end
9          else
10             update the keep-alive timer;
11             update peer-base;
12         end
13     end
14 end
```

receive data through SSM. In multicast, the receivers send iPRP_CAP after a back-off period (Section IV-F) to avoid flooding. The iPRP_ACK message also serves a terminating message for impending iPRP_CAPs thereby preventing a flood (Alg. 2).

To complete the iPRP session establishment, the iPRP-session-maintenance functional block performs IND matching (section III-D) and creates a *peer-base* entry (t_3 in Fig. 3, Alg. 3). The *peer-base* contains all information needed by the sender for replication of data packets.

The second goal of control plane it to maintain an iPRP session. To this end, the iPRP_CAP messages are used as keep-alive messages (Alg. 3). iPRP session is terminated if no iPRP_CAP message is received for a period of $3T_{CAP}$. These messages are sent to a sender as long as it is present in the list of active senders. The list of active senders is maintained by the soft-state-maintenance functional block by updating the *last-seen timer* (Alg. 1) when a new data packet is received. Sessions that are inactive for more than $T_{inactivity}$ are terminated.

For each new iPRP session, a corresponding iPRP session establishment is triggered. If any of the required steps could not be completed due to message loss or iPRP incapability, an iPRP session is not established and packets are not replicated.

B. Data Plane: Replication Phase

The replication phase occurs at the sender to send out data plane messages once the iPRP session is established. The *replication* functional block (Fig. 2) on the sender intercepts all outgoing packets destined to UDP port p of the receiver. These packets are subsequently replicated and iPRP headers (section IV-D) are prepended to each copy of the payload. iPRP headers are populated with the iPRP version, a sequence-number-space ID, a sequence number, an original UDP destination port, and IND. The 32-bit sequence number is the same for all the copies of the same packet. The destination port number is set to iPRP data port for all the copies. An authentication hash is appended and the whole block is encrypted. Finally, the copies are transmitted as iPRP data messages over the different matched interfaces (see Alg. 4, t_4 in Fig. 3.

If a new interface is added or removed at the sender or receiver during the replication phase, the *peer-base* is updated by the iPRP session maintenance functional block. When the sender receives an IPRP_CAP with a new IND, it is instantly added to the peer-base if successfully matched. On the other hand, when it receives an IPRP_CAP without an IND currently in use, the missing IND is removed from the peer-base only after confirmation from multiple consecutive IPRP_CAPs (to handle the feedback suppression effect of the backoff algorithm in Section IV-F).

Algorithm 4: (At the sender) Packet replication

```
1  for every outgoing packet do
2      check the peer-base;
3      if there exists an iPRP session that corresponds to
       the destination socket then
4          replicate the payload;
5          append iPRP headers incl. seq. number;
6          send packet copies;
7      else
8          forward the packet unchanged;
9      end
10 end
```

C. Data Plane: Duplicate Discard Phase

The duplicate discard phase occurs at the receiver once an iPRP session is established to ensure that only one copy of replicated packets is forwarded to the application. Upon reception of packets on the iPRP data port, the associated last-seen timer is updated (see Alg. 1) and the packets are forwarded to the *duplicate-discard* functional block (Alg. 5). It decrypts the iPRP header at the beginning of the payload using the symmetric key used in iPRP_CAP message. Then, function isFreshPacket (Section IV-E - Alg. 6) is called. Based on the sequence-number-space ID and the sequence number, the packet is either forwarded to the application or discarded. The first received copy should reach the application, subsequent copies are discarded. The replication is thus rendered transparent to the sender and receiver applications. In Fig. 3 we show two scenarios after the time t_4; in one case both copies are delivered, in the other, one packet is lost.

Algorithm 5: (At the receiver) Duplicate discard

```
1  for every packet received on iPRP data port do
2      get sequence number space ID (SNSID);
3      get sequence number (SN);
4      if it is the first packet from this SNSID then
5          SNSID.HighSN← SN;          // Bootstrap
6          forward to application;
7      else
8          if isFreshPacket(SN, SNSID) then
9              remove iPRP header;
10             reconstruct original packet;
11             forward to application;
12         else
13             silently discard the packet;
14         end
15     end
16 end
```

D. The iPRP Header

Fig. 4 shows the position and the fields of the iPRP header used in data packets. The Sequence-number-space ID (SNSID)

is used to identify an iPRP session. This identifier is unique across all iPRP sessions terminating at the same receiver, thereby allowing multiple iPRP sessions on the same machine. In our implementation, it is chosen as a concatenation of the source IPv6 address, the source UDP port number of the socket to which the application writes the packet and a 16-bit reboot counter.

The SNSID is used by a receiver to tie the packets with different source IP addresses that belong to the same iPRP session together. When a new receiver joins a multicast group with already established iPRP session, it uses the source IP address in the SNSID to uniquely identify the sender of the packets and the source port number in the SNSID to uniquely identify the streaming application on the sender. However, in case of a crash and reboot of the sender, the sequence number is reset. Then, a new reboot counter in the iPRP header differentiates packets belonging to the new iPRP session from those of the old iPRP session, thereby ensuring a seamless recovery at the receiver.

To maintain the format of the iPRP header for an IPv4 implementation, we suggest repeating source IPv4 address four times at the place of source IPv6 address. The original destination UDP port number is included to allow for the reconstruction of the original UDP header. The iPRP header is placed after the inner-most UDP header. So, iPRP works well, even when tunneling is used (e.g., 6to4).

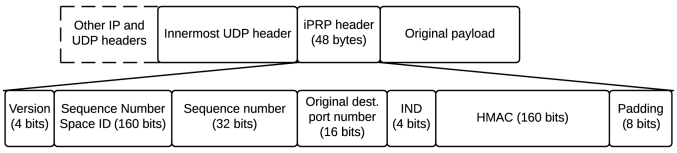

Fig. 4: Location and fields of iPRP header.

Like many protocols (such as DTLS, VPN, VXLAN, 4in6, etc.), iPRP adds its own header to the packet payload. In order to avoid packet fragmentation, we adopt the same solution as any tunneling protocol: at the sender, iPRP reduces the interface MTU size to the minimum of 1280 bytes required by IPv6. In practice, typical MTU values are closer to the IPv6-recommended 1500 bytes. This leaves a margin for the inclusion of the iPRP and other tunneling protocol headers.

E. The Discard Algorithm

The redundant copies of a packet are eliminated by a discard algorithm running at the receiver. In scenarios where the packets are received out-of-order, the discard algorithm proposed for PRP [12] delivers several copies of the same packet to the application. The function `isFreshPacket` (Alg. 6) avoids this issue. It is used by Alg. 5 to decide if a packet sequence number corresponds to a fresh packet. We use 32-bit unsigned integer sequence numbers, large enough to avoid the wrap-around problem.

Alg. 6 tracks the following variables per iPRP session, identified by a sequence number space ID (`SNSID`):

- `HighSN` – highest sequence number of a packet received before the current packet,
- `ListSN` – sequence-number list of delayed packets.

Algorithm 6: Function to determine whether a packet with sequence number `CurrSN` corresponds to a fresh packet in the sequence number space ID `SNSID`. The test "`x` follows `y`" is performed for 32-bit unsigned integers using subtraction without borrowing as "`(x-y)>>31==0`".

```
1  function isFreshPacket(CurrSN, SNSID)
2  if CurrSN==SNSID.HighSN then
3  |  return false ;                    // Duplicate packet
4  else if CurrSN follows SNSID.HighSN then
5  |  put SNs [SNSID.HighSN+1, CurrSN-1] in
   |  SNSID.ListSN;
6  |  remove the smallest SNs until SNSID.ListSN has
   |  MaxLost entries;
7  |  SNSID.HighSN ← CurrSN ;   // Fresh packet
8  |  return true;
9  else
10 |  if CurrSN is in SNSID.ListSN then
11 |  |  remove CurrSN from SNSID.ListSN;
12 |  |  return true ;                  // Late packet
13 |  else
14 |  |  return false; // Already seen or very late
15 |  end
16 end
```

`ListSN` is bounded to a maximum of `MaxLost` $< 2^{31}$ entries. `MaxLost` is the maximum sequence-number difference accepted by the application. In practice, we can take `MaxLost` $> R \times T_{late}$, where R is an upper bound on packet rate of the streaming application that corresponds to an iPRP session and T_{late} is the time after which packets are deemed out-of-date, thus irrelevant. Consequently, if a packet is received with a sequence number that precedes `HighSN` by more than `MaxLost`, it is deemed "very late" and dropped.

The value of `MaxLost` is configurable and depends on the targeted application. For example, in our smart-grid setting, there is a hard delay-constraint of 20 ms (any packet older than this can be safely discarded). To be conservative, we allow packets with the delays of up to $T_{late} = 50$ ms. We set `MaxLost` to 1024, high enough to support any realistic PMU streaming rate.

iPRP and its discard algorithm are able to recover after unexpected events (crashes and reboots). A problem can occur if, after a reboot of a sender, the same sequence numbers are reused. Then, fresh packets can be wrongly discarded as the receiver would be deceived into believing that it had already delivered such packets. This problem can be fixed by imposing handshakes between senders and receivers. However, such a solution is not appropriate if multicast is used and, furthermore, it would violate soft-state property. Our solution is to have a sender maintain a reboot counter that defines different sequence-number spaces within the same sender machine (see Section IV-D). Therefore, when a new reboot counter is encountered, the receiver creates a new `SNSID`, thereby resetting `HighSN`. Following a reboot of a receiver, all the receiver's counters are initialized upon the reception of the first iPRP data packet.

As mentioned earlier, the algorithm keeps track of one variable and of one list per iPRP session. The most expensive operation is searching the list (line 10). However, in practice,

ListSN is limited to few entries. The algorithm can be further optimized for a $\mathcal{O}(1)$ time complexity by using a hash table implementation for ListSN. Additionally, the algorithm is designed to have a fixed memory usage of 4×ListSN bytes.

Before stating the correctness of the algorithm, we need to introduce some definitions. We say that a received packet is *valid* if it arrives in order or if it is out-of-order but not later than T_{late}. Formally, this means that a packet received at time t with $\texttt{SN} = \alpha$ is not valid if some packet with $\texttt{SN} = \beta > \alpha + \texttt{MaxLost}$ was received before t.

Furthermore, let Δ be an upper bound on the delay jitter across all network subclouds. Formally, for any two packets i, j sent over any two network subclouds k, l: $\Delta \geq \left(\delta_i^k - \delta_j^l\right)$, where δ denotes the one-way network latency. Also, recall that $T_{inactivity}$ is used to terminate inactive sessions (Section IV-A).

Theorem 1 (Correctness of the discard algorithm)**.** *If $R \times \Delta < 2^{31}$ and $R \times (T_{inactivity} + \Delta) < 2^{31}$, then Alg. 6 guarantees that: (1) no duplicates are forwarded to the application and (2) the first received valid copy of any original packet is forwarded to the application.*

The proof is lengthy and is given in the Appendix A. To understand the practicality of the conditions in the theorem, note that $T_{inactivity}$ is in the order of seconds and is much larger than Δ. Therefore, the only condition to verify is $R \times (T_{inactivity} + \Delta) < 2^{31}$, which for, say $T_{inactivity} = 10s$ and $\Delta = 100ms$, requires $R < 2 \times 10^8$ packets per second – a rate much higher than ever expected.

F. The Backoff Algorithm

The soft-state in a multicast iPRP session is maintained by periodic advertisements (iPRP_CAP) sent to the source by each member in the multicast group of receivers. We want to prevent "message implosion" at the source for groups of receivers ranging from several hosts to millions. Failing to do so can have a similar effect as a denial-of-service attack. The source would be overwhelmed with processing iPRP_CAPs if all the multicast group members would send them. Nevertheless, if the source waits too long before receiving at least one iPRP_CAP, the start of the iPRP operation would be delayed. This is why we also require the source to receive an iPRP_CAP within at most $D = 10$s after the start of the loop in Alg. 2 (executed periodically every $T_{CAP} = 30$s).

A similar problem was studied in the literature on reliable multicast, where ACK implosion at the source needs to be avoided. To our knowledge, the solution that best fits our scenario was proposed by Nonnenmacher and Biersack [13]. We adopt it in our design: each receiver performs a random backoff before transmitting an iPRP_CAP. The source acknowledges each iPRP_CAP by an iPRP_ACK. The reception of an iPRP_ACK before the expiry of the backoff timer inhibits any receiver from sending its iPRP_CAP. The backoff timer follows a flipped truncated exponential distribution (inaptly called "exponential" in [13]), defined by a PDF on $[0, D]$ that increases toward D, $f_D(x; \lambda) \stackrel{\text{def.}}{=} \lambda e^{\lambda x}(e^{\lambda D} - 1)^{-1} \cdot \mathbb{1}_{\{x \in [0,D]\}}$.

We implement the backoff computation of [13] by CDF inversion. A uniform random variable $U \in [0, 1]$ is obtained via a random number generator. Next, the backoff is set to $T_{\text{backoff}} = \lambda^{-1} \ln(1 + (e^{\lambda D} - 1)U)$ (Alg. 2, line 2). We pick $\lambda = 25/D$. See Appendix B for a further discussion.

G. Robustness and Soft-state

iPRP is a soft-state protocol that is robust against host failures and supports joining or leaving the hosts from the network at any time, independently of each other. In a multicast case, it is expected that a new iPRP-capable receiver can show up (or simply crash and reboot) after an iPRP session with other receivers was established. Then, the new receiver will immediately be able to process packets received at the iPRP data port without the need to exchange control messages.

iPRP control message exchange does not rely on the availability of any particular network subcloud, making our protocol robust to network failures. Once the soft-state maintenance functional block learns about alternative network subclouds, iPRP_CAP messages are sent over all of them. Furthemore, the control plane communication to the reserved iPRP control port is secured (see Section V). The security algorithm for iPRP header protection can be chosen as part of the configuration.

V. Security Considerations

The iPRP protocol design is such that it does not interfere with upper-layer security protocols. However, in addition, we needed to provide security for the iPRP layer itself, as there are attacks that can stay undetected by upper-layer security protocols. Concretely, if an attacker manages to alter the sequence-number field of iPRP packets transmitted over one (compromised) network subcloud, the discard algorithm can be tricked in a way that the packets from both (compromised and non-compromised) network subclouds are discarded. Note that similar attacks exist for PRP, where an attacker, with access to one network, can force the discard of valid frames on another network. For example, say an attacker has access to network subcloud A. A PRP frame is represented as $A5$, where A is the network subcloud it belongs to and 5 is the sequence number. If $A5$ and $B5$ were received and the attacker retransmits the frame $A5$ by altering the sequence number as $A6$, then the actual $A6$ and $B6$ frames will both be discarded. In other words, an unsecured PRP or iPRP could weaken the network instead of making it more robust. Yet another argument for protecting the iPRP layer is that by doing so we minimize the exposure for prospective attacks in the future.

The iPRP control messages are encrypted and authenticated. This guarantees that the security of replicated UDP flows is not comprimised by iPRP and that it does not interfere with application layer encryption/authentication.

Specifically, iPRP_CAP messages and the corresponding iPRP_ACK messages are transmitted over a secure channel. The iPRP header inserted in the data packets is authenticated and encrypted with a pre-shared key. Thus, replay attacks and forged messages insertion are avoided.

We establish the secure channel for the transmission of iPRP_CAP messages depending on the type of communication, unicast or multicast. Details follow below.

Unicast: In unicast mode, a DTLS session is maintained between the sender and the receiver. It is initiated by the receiver upon the arrival of the first UDP datagram from the source. iPRP_CAP messages are transmitted within this session. So, the iPRP capabilities of the receiver are transmitted

only to an authenticated source. iPRP_ACKs are not required in unicast (since message implosion can occur in multicast only).

Unicast iPRP_CAP messages contain a symmetric key used to authenticate and encrypt the iPRP header. This key is updated periodically during a unicast iPRP session. Hosts keep a small fixed number of valid past keys to prevent losing the iPRP session because of delayed receiption of a new key. The oldest key is discarded upon reception of a new one.

Multicast: In multicast, iPRP relies on any primitive that establishes a secure channel with the multicast group. For example MSEC can be used for group key management and for establishing a group security association.

In this setting, both iPRP_CAP and iPRP_ACK messages, as well as the iPRP headers inserted in the replicated packets, are authenticated and encrypted with the group key. Thus, there is no need to include an additional key in the iPRP_CAP .

VI. iPRP Diagnostic Toolkit

As iPRP is designed to be IP friendly, it facilitates the exploitation of the diagnostic utilities associated with TCP/IP. The diagnostics include verification of connectivity between hosts and the evaluation of the corresponding RTTs (similar to ping), the discovery of routes to a host (similar to traceroute), etc. Furthermore, the toolkit also adds some more tools that are specific to iPRP and it gives iPRP a significant edge in network diagnostics and statistics collection over PRP. The toolkit comprises the following tools:

```
iPRPtest <Remote IP Address> <Port>
        <Number of packets> <Time period>
iPRPping <Remote IP Address>
iPRPtracert <Remote IP Address>
iPRPsenderStats <IP Address>
iPRPreceiverStats <IP Address>.
```

Imagine a typical scenario where an application on an iPRP enabled host that is subscribed to a particular multicast group (G) experiences packet losses. To troubleshoot this problem, the user at the receiving host would use the iPRPreceiverStats tools to consult the local list of active senders, to check for the presence of an iPRP session associated with any host sending multicast data to group G. If an iPRP session exists, then the tool returns the statistics of packets received over different networks in the iPRP session. Then, to understand if the problem is caused by multicast routing or lossy links, the user moves to the sending host.

First, with iPRPtest and by using the remote IP address of the receiver, the user establishes a temporary, unicast iPRP session with the host. If successful, the iPRPping tool is used to obtain the packet loss and RTT statistics over the multiple networks. Also, the iPRPtracert tool is used to verify the hop-by-hop UDP data delivery over multiple networks. For any iPRP session between two hosts, the iPRPsenderStats is used by the sending host to query the remote host about the statistics of the packets accepted and dropped by the duplicate discard functional block on that remote host. The operation of each tool is described in detail in Appendix C.

VII. Implementation and Performance Evaluation

We opted for a Linux-based user-space implementation that has the following properties: (1) Allow for the selective filtering of IP packets so that the iPRP sequence of operation can be applied; (2) Allow for packet mangling: iPRP header can be inserted and packets can be replicated at the sender, duplicates can be discarded and original packet can be restored at the receiver; (3) Minimal CPU overhead.

To this end, we use the libnetfilter_queue (NF_QUEUE) framework from the Linux *iptables* project. NF_QUEUE is a userspace library that provides a handle to packets queued by the kernel packet filter. It requires the libnfnetlink library and a kernel that includes the nfnetlink_queue subsystem (kernel 2.6.14 or later). It supports all Linux kernel versions above 2.6.14. We use the the Linux kernel 3.11 with iptables-1.4.12. More details on the implementation can be found in Appendix D.

We are deploying iPRP on our EPFL smart-grid communication network (smartgrid.epfl.ch). Also, we setup a lab test bed to evaluate iPRP performance. We use a sender and a receiver (Lenovo ThinkPad T400 laptops) interconnected with three networks (nwA, nwB and nwC below). We generate packet losses and delays in these networks to simulate different scenarios summarized in Table I. The packet losses and delays are emulated using the Linux tc-netem [14] tool.

Scenario	tc-netem delay : loss nature		
	nwA	nwB	nwC
0	S:IL	S:IL	S:IL
1	Z:IL	S:IL	not used
2	Z:BL	S:BL	not used
3	Z:IL	L:IL	not used
4	Z:BL	L:BL	not used
5	S:IL	S:IL	not used

TABLE I: Scenarios used for performance evaluation. tc-netem added delay : "Z" means 0 , "S" means small uniform $10ms \pm 5ms$, and "L" means large uniform $1s \pm 0.2s$. Loss nature: "IL" means 5% independent and "BL" means 5% bursty losses.

A. iPRP Behavior in the Presence of Asymmetric Delays and Packet Losses

Our goal here is to validate the design and implementation of iPRP by quantifying the packet losses and delays perceived by an application. We stress-test the discard algorithm with heavy losses and asymmetric delays and compare the performance with that in theory.

In Table II, we show the measurement results. We assume that the losses on different networks are independent. We compare the observed effective losses (iPRP column) with the expected effective loss percentage that is the product of observed loss percentages on different networks (theory column). A deviation would mean anomalies in the iPRP protocol and implementation. The accordance between the last two columns in Table II shows that iPRP performs as expected in significantly reducing the effective packet losses.

In Fig. 5 we show the CDF of one-way network latency (d_{IPRP}) for Scenario 5. In theory, it should be $d_{iPRP} = min(d_{nwA}, d_{nwB})$ and what is measured matches the theory very well. CDFs are not shown for Scenarios 1-4 as, by construction, it is almost deterministic which network has the shortest latency. For example, in Scenario 1 most of the times $d_{iPRP} = min(d_{nwA}, d_{nwB}) = d_{nwA}$.

Scen.	nwA	nwB	nwC	iPRP	theory
0	5.061	4.913	5.1537	0.0126	0.0128
1	5.057	5.002	not used	0.253	0.254
2	5.132	5.059	not used	0.259	0.254
3	5.014	5.013	not used	0.251	0.249
4	5.022	4.981	not used	0.247	0.249
5	5.051	5.002	not used	0.251	0.253

TABLE II: Loss percentages in various scenarios

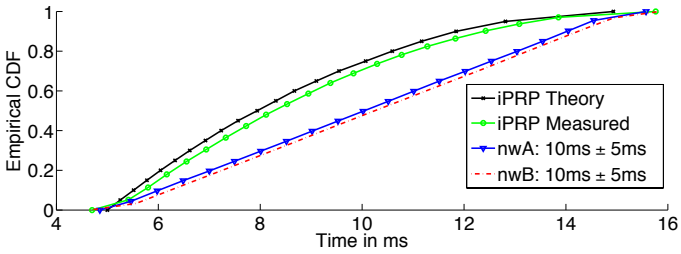

Fig. 5: iPRP side benefit: the delays perceived by the application are improved when iPRP is used, compared to those when only one of the individual networks is used.

B. Processing Overhead Caused by iPRP

In this subsection, we evaluate processing delays and the additional CPU load when iPRP is used. We conduct several runs of Scenario 1 (see Table I) and use *GNU gprof* to assess the average processing delay incurred by an iPRP data packet. In a sender, a data packet encounters only the *replicator* function which adds the iPRP header and replicates packets over multiple interfaces. This operation takes 0.8 μs on average. On the receiver side, a data packet encounters three functions. The *packet handler* copies a packet into user-space, verifies the fields of the iPRP header and prepares a packet for the *duplicate discard* function which indicates if a packet is to be dropped or forwarded. These operations take 0.8 μs and 0.4 μs on average respectively. Lastly, if a packet is to be forwarded, the iPRP header is removed and checksum is recomputed in 2.4 μs. On average, a data packet incurs a delay overhead of 4.4 μs due to iPRP.

In order to assess the additional CPU load when iPRP is used, we perform two experiments in which we record the CPU usage on the sender and on the receiver. The results are summarized in Table III and lead to the conclusion that the implementation presented in this paper is efficient.

VIII. Conclusion & Future Work

We have designed iPRP, a transport layer solution for improving reliability of UDP flows with hard-delay constraints, such as smart grid communication. iPRP is application- and network-transparent, which makes it plug-and-play with existing applications and network infrastructure. Furthermore, our soft-state design makes it resilient to software crashes. Besides unicast, iPRP supports IP multicast, making it a suitable solution for low-latency industrial automation applications requiring reliable data delivery. We have equipped iPRP with diverse monitoring and debugging tools, which is quasi impossible with existing MAC layer solutions. With our implementation, we have shown that iPRP can support several sessions between hosts without any significant delay or processing overhead. We have made our implementation publicly available and are currently installing it in our campus smart-grid [15]. In the future, we intend to do extensive measurements on our smart-grid and study the performance of iPRP in real networks.

Number of sessions	Exper. 1: Aggregate of pps for all sessions kept constant to 1000		Exper. 2: pps per session kept constant to 10	
	Send. [%]	Rec. [%]	Send. [%]	Rec. [%]
0 (Idle)	3.7	0.9	3.7	0.9
1	14.5	11.8	4.5	2.2
2	14.1	11.9	5.6	2.4
4	15	11.3	5.7	2.3
10	15	12	7.3	3.2
20	15	12	10	5.2

TABLE III: CPU usage with iPRP and varying loads

References

[1] H. Kirrmann, M. Hansson, and P. Muri, "Iec 62439 prp: Bumpless recovery for highly available, hard real-time industrial networks," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, Sept 2007, pp. 1396–1399.

[2] M. Elattar and al., "Using lte as an access network for internet-based cyber-physical systems," in *IEEE World Conference on Factory Communication Systems*, 2015.

[3] M. Rentschler and H. Heine, "The parallel redundancy protocol for industrial ip networks," in *Industrial Technology (ICIT), 2013 IEEE International Conference on*, Feb 2013, pp. 1404–1409.

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013.

[5] IEEE Standards Association., "IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks Link Aggregation." 2008.

[6] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Internet Engineering Task Force, Nov. 2000.

[7] C. Fragouli and E. Soljanin, "Network coding fundamentals," *Foundations and Trends in Networking*, vol. 2, no. 1, pp. 1–133, 2007.

[8] D. J. C. MacKay, "Fountain codes," *Communications, IEEE Proceedings*, vol. 152, no. 6, pp. 1062–1068, Dec 2005.

[9] N. Sprecher and A. Farrel, "MPLS Transport Profile (MPLS-TP) Survivability Framework," RFC 6372 (Informational), Internet Engineering Task Force, Sep. 2011.

[10] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "Multi-Topology (MT) Routing in OSPF," RFC 4915 (Proposed Standard), Internet Engineering Task Force, Jun. 2007.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[12] H. Weibel, "Tutorial on parallel redundancy protocol," Zurich University of Applied Sciences, Tech. Rep.

[13] J. Nonnenmacher and E. W. Biersack, "Scalable feedback for large groups," *IEEE/ACM Transactions on Networking (ToN)*, vol. 7, no. 3.

[14] S. Hemminger *et al.*, "Network emulation with netem," in *Linux Conf Au*. Citeseer, 2005, pp. 18–23.

[15] M. Pignati and al., "Real-time state estimation of the epfl-campus medium-voltage grid by using pmus," in *Innovative Smart Grid Technologies Conference (ISGT), 2015 IEEE PES*, 2015.

Appendix A
Proof of Theorem 1

To prove the statement of Theorem 1, we need the following lemmas.

Lemma 1. *If $R \times \Delta < 2^{31}$ and $R \times (T_{inactivity} + \Delta) < 2^{31}$, then the wrap-around problem does not exist.*

Proof: The wrap-around problem can arise in two scenarios.

Case 1: A late packet arrives with $\texttt{CurrSN} < \texttt{HighSN} - 2^{31}$. As $R \times \Delta < 2^{31}$, the time required by the source to emit 2^{31} packets is longer than Δ. Hence, HighSN cannot precede CurrSN for more than 2^{31} and this scenario is not possible.

Case 2: A fresh packet is received with $\texttt{CurrSN} > \texttt{HighSN} + 2^{31}$. This means that from the point of view of the receiver, there were more than 2^{31} iPRP packets lost in succession. As $R \times (T_{inactivity} + \Delta) < 2^{31}$, the time for more than 2^{31} consecutive packets to be sent is greater than $(T_{inactivity} + \Delta)$. Hence, the time between reception of any two packets differing by SNs more than 2^{31} is greater than $(T_{inactivity})$. Therefore, during this time the iPRP session would be terminated and a new session will be initiated when the fresh packet is received. Hence, this scenario is also not possible. ∎

Therefore, in the rest of the proof, we can ignore the wrap-around problem and do as if SNs of received packets were integers of infinite precision. Also, a notation such as $\texttt{HighSN}_{t^-}$ [resp. $\texttt{HighSN}_{t^+}$] denotes the value of HighSN just before [resp. after] time t.

Lemma 2 (Monotonicity of HighSN)**.** *If at time t, a packet with $\texttt{SN} = \alpha$ is received, then $\texttt{HighSN}_{t^+} = \max(\texttt{HighSN}_{t^-}, \alpha)$. Therefore,* HighSN *increases monotonically with time.*

Proof: From Alg. 6, when $\alpha > \texttt{HighSN}_{t^-}$ (line 4) then the value of HighSN is changed to α (line 7). Otherwise, when $\texttt{HighSN}_{t^-} \geq \alpha$ (lines 2 and 9), HighSN is unchanged, i.e., $\texttt{HighSN}_{t^+} = \texttt{HighSN}_{t^-}$. The two cases combined together give $\texttt{HighSN}_{t^+} = \max(\texttt{HighSN}_{t^-}, \alpha)$. ∎

Lemma 3 (Fresh packet is never put in ListSN)**.** *If at time t, a packet with $\texttt{SN} = \alpha$ is forwarded to the application then $\alpha \notin \texttt{ListSN}_{t'^+} \forall\ t' \geq t$.*

Proof: Let us prove by contradiction. Assume that $\exists\ t' > t$ such that $\alpha \in \texttt{ListSN}_{t'}$. Hence, $\exists\ t_1 \in (t, t']$ when α was added to ListSN. As $t_1 > t$, from Lemma 2, we conclude that $\texttt{HighSN}_{t_1^-} \geq \texttt{HighSN}_{t^+} \geq \alpha$. Now, from Alg. 6, we know that only SNs $> \texttt{HighSN}_{t_1^-}$ can be added to ListSN. Hence, α cannot be added to ListSN at time t_1. Therefore, we have a contradiction. ∎

Lemma 4. *At any time t, $\texttt{HighSN}_{t^-}$ is equal to* SN *of a packet received at some time $t_0 < t$ or no packet has been received yet.*

Proof: HighSN is modified only at line 7, where it takes the value of the SN received. Hence, HighSN cannot have a value of a SN that has not been seen yet. ∎

Now, we proceed with the proof of the theorem. First, we prove statement (1). Assume we receive a duplicate packet with $\texttt{SN} = \alpha$ at time t. It means that a packet with $\texttt{SN} = \alpha$ was already seen at time $t_0 < t$. Then, from Lemma 2 it follows that $\alpha \leq \texttt{HighSN}_{t^-}$. Then, either $\alpha = \texttt{HighSN}_{t^-}$ (line 2) or $\alpha < \texttt{HighSN}_{t^-}$ (line 10).

Case 1: When $\alpha = \texttt{HighSN}_{t^-}$, the packet is discarded according to line 3.

Case 2: When $\alpha < \texttt{HighSN}_{t^-}$, line 10 is evaluated as false due to Lemma 3. Hence, the packet is discarded by line 14.

Next, we prove statement (2) by contradiction. Assume we receive a first copy of a valid packet with $\texttt{SN} = \alpha$ at time t but we do not forward it. This can happen either due to line 3 (case 1) or due to line 14 (case 2).

Case 1: Statement from line 2 was evaluated as true, which means that $\alpha = \texttt{HighSN}_{t^-}$. As $\texttt{SN} = \alpha$ is seen for the first time, Lemma 4 is contradicted. Hence, this case is not possible.

Case 2: Statement from line 10 was evaluated as false, which means that $\alpha < \texttt{HighSN}_{t^-}$ and $\alpha \notin \texttt{ListSN}_{t^-}$. We show by contradiction that this is not possible, i.e., we now assume that $\alpha < \texttt{HighSN}_{t^-}$ and $\alpha \notin \texttt{ListSN}_{t^-}$. Now, there are three cases when $\alpha \notin \texttt{ListSN}_{t^-}$ can be true:

(i) $\texttt{SN} = \alpha$ was added to and removed from ListSN before time t because it was seen (line 11) which is impossible as the packet is fresh.

(ii) $\texttt{SN} = \alpha$ was added to ListSN and later removed at time $t_0 < t$ because the size of ListSN is limited to MaxLost entries (line 6). This means that at time $t_0 < t$ a packet with $\texttt{SN} = \beta$ was forwarded and $\beta - \alpha > \texttt{MaxLost}$ (line 6). However, this means that the packet with $\texttt{SN} = \alpha$ was not valid at time t_0 and therefore is also not valid at time $t > t_0$.

(iii) $\texttt{SN} = \alpha$ was never added to ListSN. Consider the set $\mathbb{T} = \{\tau \geq 0 : \texttt{HighSN}_{\tau^+} > \alpha\}$. $\mathbb{T}$ is non-empty because $t \in \mathbb{T}$, by hypothesis of our contradiction. Let $t_0 = \inf \mathbb{T}$. Then, necessarily $\texttt{HighSN}_{t_0^-} \leq \alpha < \texttt{HighSN}_{t_0^+}$ (say, $= \beta$). β is the SN of a packet received at time t_0. Since α is valid, $\beta - \alpha < \texttt{MaxLost}$. Otherwise, α would be invalid at time t_0, therefore at time t, which is excluded. Then we have two subcases possible:

a) $\texttt{HighSN}_{t_0^-} < \alpha$. Then, by line 5, α is added to ListSN, which is a contradiction.

b) $\texttt{HighSN}_{t_0^-} = \alpha$. But, by Lemma 4 a packet with $\texttt{SN} = \alpha$ must have been received before t_0 which is a contradiction because α is a fresh packet at $t \geq t_0$.

APPENDIX B
DETAILS OF THE BACKOFF ALGORITHM

A. Backoff Evaluation

In this section we consider that $D = 1$, for the sake of simplifying the presentation.

The *flipped truncated exponential distribution* with parameter $\lambda > 0$ is then a distribution with density $f{:}[0,1] \to \mathbb{R}$ and CDF $F{:}[0,1] \to [0,1]$:

$$f(x;\lambda) \stackrel{\text{def.}}{=} \frac{\lambda e^{\lambda x}}{e^{\lambda} - 1};\ F(x;\lambda) = \int_0^x f(x;\lambda)\, dx = \frac{e^{\lambda x} - 1}{e^{\lambda} - 1}.$$

In Figure 6 we plot PDFs of the flipped truncated exponential for various values of λ. This distribution is designed to ensure that the few transmissions that occur in the beginning of the interval silence with high probability the majority of the transmissions scheduled in the end of the interval. In what

follows, we show that $\lambda = 20$ or $\lambda = 25$ are suitable when $n \lesssim 10^6$.

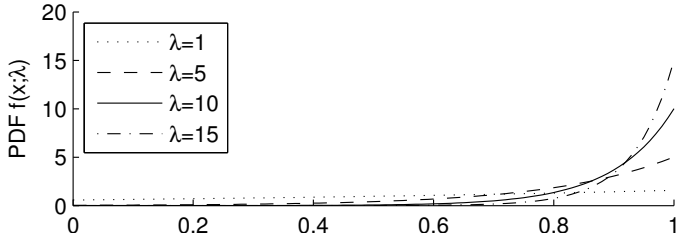

Fig. 6: The PDF of flipped truncated exponential distributions for various values of λ.

B. Backoff Analysis

For the sake of analysis, we make the simplifying assumption that the RTT from the source to any member of the group is the same: RTT $= r$, $r \in [0,1]$. $Z_n(r)$ denotes how many `IPRP_CAP` messages are received by the source during an interval of length T_{CAP}. We want to choose a parameter λ that ensures a small $Z_n(r)$ in most realistic scenarios. Hence, we show the following result that overlaps with the one in [13]:

Theorem 2. *In a lossless network, for a multicast group of size n, the expected number of messages received by the source is*

$$\mathbb{E}Z_n = \Phi_n(r;\lambda) = n\frac{e^{\lambda r}-1}{e^{\lambda}-1} + e^{\lambda r}\left[1 - \left(\frac{1-e^{-\lambda r}}{1-e^{-\lambda}}\right)^n\right].$$

In particular, for a fixed choice of λ, the probability that Z_n exceeds a number of messages $\delta > 0$ is upper bounded by $\mathbb{P}[Z_n > \delta] \le \Psi_n(\delta, r; \lambda)$, where

$$\Psi_n(\delta,r;\lambda) = \frac{1}{\delta^2}\frac{e^{\lambda r}-1}{e^{\lambda}-1}\left[n + 2ne^{\lambda r} - n(n-1)\frac{e^{\lambda r}-1}{e^{\lambda}-1}\right] + \frac{e^{\lambda r}}{\delta^2}\left\{2e^{\lambda r} - 1 - (2e^{\lambda r}+2n-1)\left(\frac{1-e^{-\lambda r}}{1-e^{-\lambda}}\right)^n\right\}.$$

a) Proof: Denote the backoff drawn by receiver i by X_i and denote the smallest one by $\hat{X}_n := \min_{i=1}^n X_i$. For a fixed value of r denote $Y_{in}(r) = \mathbb{1}_{\{X_i \le \hat{X}_n + r\}}$. Then the source receives $Z_n(r) = \sum_{i=1}^n Y_{in}(r)$ messages before the receivers' transmissions are canceled.

Since the Y_{in} are exchangeable, $\mathbb{E}Z_n = n\mathbb{E}Y_{1n}$:

$$\begin{aligned}\mathbb{E}Y_{in}(r) &= \mathbb{P}(X_i \le \hat{X}_n + r) = \mathbb{P}(X_1 \le \hat{X}_n + r)\\ &= \mathbb{P}\left(\bigcap_n \{X_n \ge X_1 - r\}\right)\\ &= \int_0^1 [\mathbb{P}(X_2 \ge x_1 - r)]^{n-1} f(x_1)dx_1\\ &= \int_0^1 [1-F(x-r)]^{n-1} f(x)dx.\end{aligned}$$

Hence

$$\mathbb{E}Z_n = n\frac{e^{\lambda r}-1}{e^{\lambda}-1} + e^{\lambda r}\left[1 - \left(\frac{1-e^{-\lambda r}}{1-e^{-\lambda}}\right)^n\right] = \Phi_n(r;\lambda)$$

The second part is an application of Chebyshev:

$$\mathbb{P}[Z_n > \delta] \le \frac{\mathbb{E}Z_n^2}{\delta^2} = \frac{1}{\delta^2}\{n(n-1)\mathbb{E}[Y_{1n}Y_{2n}] + n\mathbb{E}Y_{1n}\}.$$

For this we need the second moment. We have that

$$\begin{aligned}\mathbb{E}[Y_{1n}Y_{2n}] &= \mathbb{P}[X_1 < \hat{X}_n + r \text{ and } X_2 < \hat{X}_n + r]\\ &= \int_{\substack{\{x,y\in[0,1]:\\ |x-y|<r\}}} f(x)f(y)[1-F(\max(x,y)-r)]^{n-2}\,dx\,dy\\ &= 2\int_0^1 dx\, f(x)\int_x^{x+r} dy\, f(y)[1-F(y-r)]^{n-2}.\end{aligned}$$

□

C. Parameter Selection

We explore values of r ranging from 0.001 to 0.03. For $D = 10$ seconds, this corresponds to the RTT ranging from 10 ms to 300 ms. For a given n, we compute numerically λ that guarantees the best average performance: $\lambda^* \in \arg\min_\lambda \Phi_n(r;\lambda)$. We find that λ=20 is an acceptable value for a wide parameter range that guarantees an expected number of messages below 3 when there are up to 1000000 members in the group. Since the optimum λ^* increases with n, and since the the expectation as a function of λ shows a slow increase toward the right of the optimal value λ^* (*i.e.*, for $\lambda > \lambda^*$), an even safer choice is $\lambda = 25$.

We now consider the case when the first k acknowledgements are lost in the network. This can lead to a dramatic increase in the number of received `IPRP_CAP` messages. We perform 10^6 independent runs for various values of λ, n, k, and r and we record the empirical distribution of the number or received capability messages (in addition to the first k). When $\lambda = 25$, the largest observed number of received capability messages is around 50 when the 20 first consecutive acknowledgements are lost for a group of 1000000 receivers in a network with $r = \text{RTT}/D = 0.03$.

We depict our findings in Figure 7 together with the theoretical upper bound in the lossless case for the 0.999 and 0.9999 quantiles. We conclude that $\lambda = 20/D$ or $\lambda = 25/D$ are judicious choices.

APPENDIX C
IPRP DIAGNOSTIC TOOLS

A. IPRPtest

`iPRPtest` tests the unicast iPRP operation between the local and remote hosts. Firstly, it checks for the presence of an iPRP session between the two machines by querying the local peer-base and returning the peer-base entry corresponding to the iPRP session identified by the inputted IP address. This entry consists of a list of the interfaces (and their IP addresses) of the remote host connected to the networks identified by the INDs. Here is an example output if an iPRP session exists:

```
  $ iPRPtest aa::1 1234 10 5
Interface  Remote IP address  IND
  eth0          aa::1         0xa
  eth3          cc::1         0xc
```

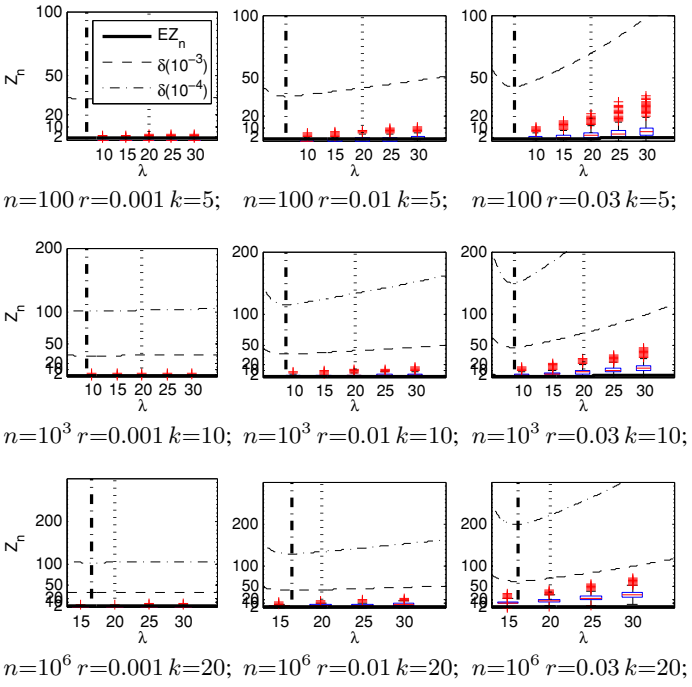

Fig. 7: The expected number of received capability messages $\mathbb{E}Z_n$ (close to 2 in all cases), an upper-bound on the 0.999 quantile $\delta(10^{-3})$, and an upper-bound on the 0.9999 quantile $\delta(10^{-4})$ computed using Ψ_n from Theorem 2 as a function of λ for 100, 1000, and 1000000 receivers when $\mathrm{RTT}/D = r$. We indicate the optimum λ^* for which $\mathbb{E}Z_n$ is minimized. We simulate a lossy environment where the first k acknowledgements are lost. We give boxplots of the empirical distribution (obtained after 10^6 runs) of the received capability messages in addition to the first k for all scenarios above and for values of λ ranging from 10 to 30.

If it does not exist, `iPRPtest` tries to establish one. It communicates the UDP port number to the iPRP-session-maintenance functional block on the remote machine, which is then added temporarily to the set $\mathbf{P}$. After the temporary-iPRP-session establishment, `iPRPtest` sends periodic probe packets to the remote host along multiple paths, depending on the parameters *number of packets* and *time period*. Finally, the iPRP session is closed and the corresponding UDP port is removed from set $\mathbf{P}$ on the remote machine. If an iPRP session could not be established, an appropriate message is generated.

B. IPRPping

`iPRPping` evaluates the end-to-end connectivity over multiple paths, to a remote host with an iPRP session with the local host. It exploits the ICMP `ping` and does not exercise iPRP that operates on UDP. `iPRPping` queries the local peer-base for the remote IP addresses associated with the the inputted IP address and uses the native ping to check connectivity over multiple paths in a round-robin fashion. `iPRPping` can also be used to obtain the path MTUs along all paths to a host by varying the size of the *ICMP echo request* packets. Finally, it reports the packet loss and RTT statistics for all the available paths. In the case of absence of an iPRP session, an appropriate message is generated.

C. PRPtracert

`iPRPtracert` enlists the routes taken by IP packets over all the paths to the remote host with the inputted IP address. It queries the local peer-base for the remote IP addresses used during an iPRP session. Then, it uses the `traceroute` from the TCP/IP suite to trace the routes over multiple paths in a sequential manner. If the remote host does not have any iPRP session with the local host, `iPRPtracert` does not attempt to establish an iPRP session and generates an appropriate message.

D. IPRPsenderStats

`iPRPsenderStats` queries the remote IP address for packet delivery statistics associated with its iPRP session. For unicast, the argument is the IP address of the remote host with an iPRP session. In multicast, the argument is a multicast group IP address. `iPRPsenderStats` queries the remote IP address of one of the subscribers of the multicast group, for its statistics. If iPRP session does not exist, an appropriate message is generated. The reported statistics are

- *PktCtrX*: Total number of packets successfully received over the network with IND X.
- *LastTimeSeenX*: UTC time stamp of last received packet over the network with IND X.
- *WrongINDX*: Number of non-IND X packets received over the IND X network. This can happen due to a common link between multiple networks or faulty cabling at the hosts. The iPRP self-configuring property makes it immune to such faults, thus enabling detection without disrupting the normal data delivery.
- *AccINDX*: Number of packets received over the IND X network and forwarded to the application. The highest AccINDX corresponds to the fastest network.

E. IPRPreceiverStats

This tool is used to locally obtain the statistics *PktCtrX*, *LastTimeSeenX*, *WrongINDX* and *AccINDX* at the receiver. In a unicast operation, the argument is the IP address used by the sender to establish the iPRP session with local machine. In multicast operation, it is the used multicast IP address. `iPRPreceiverStats` queries the locally stored statistics table to report the above mentioned fields.

Only `iPRPsenderStats` and `iPRPreceiverStats` can be used to diagnose or obtain information from multicast iPRP sessions. The dearth of diagnostic tools for the multicast iPRP operation is attributed to the low number of diagnostic tools for IP multicast.

Appendix D
Implementation Details

Our implementation comprises four daemons and the mapping between them and the functional blocks introduced in Section IV is as follows:

- *iPRP control daemon (ICD)*: Algorithms 2 and 3
- *iPRP monitoring daemon (IMD)*: Algorithm 1

- *iPRP sender daemon (ISD)*: Algorithm 4
- *iPRP receiver daemon (IRD)*: Algorithm 5.

We explain the structure and function of each daemon by giving a walk-through of normal iPRP operation. First, a host is configured as iPRP enabled by the initialization of the ICD. It comprises a client that generates control messages and a server which expects them from other ICDs on the iPRP control port (1000). ICD has to be started on two machines for an iPRP session to be established between them. The ICD itself does not use NF_QUEUE but creates an NF_QUEUE instance (IMD). This means that all packets filtered by iptables' rules are handled by the corresponding IMD. The IMD maintains the list of active senders on a receiving host and remains idle on a sending host.

In the absence of any iPRP sessions and the set $\mathbf{P}$ being initially empty, the ICD and IMD are idle on both the sender and the receiver. When port p_1 is put to set $\mathbf{P}$ on a receiving host, the local ICD creates a packet filtering rule in iptables to filter incoming UDP traffic destined to p_1. This is repeated for each additional port added to the set $\mathbf{P}$. When traffic is encountered at the port p_1, it enters queue q_{mon} and the IMD puts the source IP address into the list of active senders (Algorithm 2). Then, it creates an iptables rule to handle incoming UDP traffic destined to iPRP data port (1001), into the queue q_{recv}. Furthermore, this being the first entry in the list of active senders, the IMD creates an NF_QUEUE instance (IRD) to handle packets in q_{recv}.

Then, the ICD on the receiver does a backoff (Section IV-F) to establish a secure DTLS session with the ICD on the sender. It communicates unicast `iPRP_CAP` messages that advertise the available IP addresses, INDs, the multicast IP address and the cryptographic key for authenticating the iPRP header. The IND in our implementation is calculated from the interface names.

On receiving an `iPRP_CAP` message, the sending-machine ICD sends an `iPRP_ACK` (omitted in unicast) to avoid further `iPRP_CAPs`. Then, it performs IND matching (Section III-D) to create a peer-base entry and the cryptographic key is stored locally. Next, it creates an iptables rule to filter the outgoing traffic to the multicast group and port p_1 into the queue q_{send}. Also, it creates an NF_QUEUE instance (ISD) to inspect all in q_{send}.

For packets in q_{send}, the ISD uses information from the peer-base entry associated with the multicast group to create an iPRP header (Section IV-D). The HMAC trailer is formed using the locally stored pre-shared cryptographic key and the `openssl` *sha-1* cryptographic hash function. Finally, the newly formed packet is encrypted using `openssl aes` function and sent to the destination port 1001 over the available networks.

The IRD receives the packet with an iPRP header, authenticates it and updates the list of active senders (Alg. 1). Depending on the decision of the discard algorithm (Section IV-E) the packet is either dropped, or forwarded to the application in its original form. The `ListSN` is realized with an array of size `MaxLost` so that updation and deletion always occurs at the `(CurrSN % MaxLost)` location. This facilitates an $O(1)$ execution time.

When the IRD stops receiving data from a particular multicast group, the corresponding entry in the list of active senders is erased and the corresponding iptables filtering rule is removed. Hence, the ICD stops sending `iPRP_CAPs` to the sending host, ISD is terminated and the associated iptables filtering rule is deleted. The IRD is terminated when the list of active senders becomes empty. Also, for each deleted entry from the set $\mathbf{P}$, the associated iptables filtering rule is removed.

All state information is locally stored with a time stamp, and aged state information is periodically removed. The periodic exchange of `iPRP_CAPs` and `iPRP_ACKs` serves as a keep-alive mechanism for soft-state property. To reduce the operating footprint of iPRP on the operating system, system calls not directly in the path of a data packet are batched. For instance, `gettimeofday()` system call is scheduled every second instead of each packet arrival. As a consequence, the granulatity of time used for soft-state maintenence is increased to 1 s instead of 20 ms. This increase only delays the start and end of an iPRP session but does not effect the time-critical data packets.