Zero Repair-Time for UDP Flows with Very Hard Delay Constraints

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

Abstract

Reliable packet delivery within stringent delay constraints is of primal importance to smart grids, high-frequency trading and online gaming networks, among others. Because retransmission and coding techniques counteract the delay requirements, reliability is achieved through replication over multiple fail-independent paths. Existing solutions such as parallel redundancy protocol (PRP) replicate all packets at the MAC layer over parallel paths. Although it achieves zero repair-time, a MAC layer design is faced with severe problems of scalability, security, and diagnostic inability, which render it unsuitable for present day networks. To address these problems, we present a transport-layer design: IP parallel redundancy protocol (iPRP). A migration to transport layer poses non-trivial challenges in the form of selective packet replication, soft-state and multicast support. Unlike existing solutions, iPRP works for routed networks and supports both unicast and multicast. It duplicates only time-critical UDP traffic. iPRP is application and network transparent, thereby no change in application software or network infrastructure is required. It has a set of diagnostic tools for network debugging. In addition to the design, we describe our IPv6 implementation of iPRP in Linux 3.6.11-rt29 that uses the NF_QUEUE framework. It is being installed in our campus smart grid network and is publicly available. We present performance results and show that it supports multiple flows with minimal processing and delay cost.

1 Introduction

1.1 Zero Repair-Time for UDP Flows

Time-critical applications, such as high-frequency trading [17], online gaming [27], and smart grids, have strict communication-delay constraints. Retransmissions can be both detrimental and superfluous, as they can introduce delays for successive, more recent data that supersede older ones. Hence, UDP is preferred (unicast and multicast) over TCP. However, increasing the reliability of such unidirectional UDP flows is a major challenge. In smart grids, for example, 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). The parallel redundancy protocol (PRP) IEC standard [15, 16] was proposed as a solution to this problem. It operates at the MAC layer and requires hosts to be connected to two cloned (disjoint) networks. The sender tags each MAC frame with a sequence number and replicates it over its two interfaces. The receiver discards, based on sequence numbers, redundant frames.

PRP has the drawbacks of layer-2 solutions: it is not scalable (limited to bridged networks); the allocated network resources are dictated by the spanning-tree protocol; network diagnostics are difficult. Also, PRP duplicates all the traffic unselectively; it has no security mechanisms; multicasting to a specific group of receivers is not natively supported. Perhaps the most limiting feature of PRP is that the two cloned networks need to be composed of devices with *identical* MAC addresses, which further exacerbates the network management difficulty.

We want to provide transparent packet replication for time-critical UDP flows (in Section 2 we discuss why solutions based on coding are not appropriate for this setting). However, we want to avoid the drawbacks of layer-2 solutions, thus adhering to the trend of migrating away from layer 2 (*e.g.*, VXLAN [19]). Further, in order for our solution to be easily deployed, we require it to be transparent to *both* the application layer and the IP network. Therefore it should be a transport-layer solution.

1.2 Challenges of a Transport Layer Solution

To reach the (multicast) destinations, a transport-layer solution for the packet replication of UDP flows should be able to have at its disposal two or more parallel paths; in principle, the paths should be disjoint. However, given the current technology, a host cannot easily control the path taken by an IP packet, beyond the choice of an 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 our transport-level solution. Hence, we assume a setting where sources and destinations of time-critical UDP flows have multiple communication interfaces. The job of a transport-layer solution is to transparently replicate packets over the different interfaces for the UDP flows that need it, match corresponding interfaces, and remove duplicates at the receiver. The paths between matching interfaces are calculated by the routing layer: by routing protocols or by topology management that runs over software-defined networking.

Note that we refuse the limiting requirement to have cloned networks. We exploit any available network redundancy, without imposing special requirements on the network. We do not require, nor guarantee end-to-end disjoint paths, though in most cases this is beneficial.

Figure 1 depicts a typical setting (inspired from the smart grid): hosts are multi-homed and each interface is assigned a different address (as usual). Each host interface is connected to a part of the IP network labelled the "A" or the "B" cloud, and some hosts have a third interface connected to the "LTE" cloud. Here, it is assumed that paths between host interfaces in the "A" cloud stay within it (and similarly with B), which automatically provides disjoint paths; however, the "A" and "B" clouds are interconnected. Such an arrangement can be achieved trivially with typically all routing protocols or with multi-topology routing [21]. We do *not* require physically separated networks, as this makes deployment and management more complicated. Supporting such a flexibility is essential for real deployments, but it poses significant challenges in protocol design.

Not all traffic requires replication, only certain hosts and certain UDP flows do (time-critical data). Hence, replication needs to be selective: a failure-proof mechanism is required for detecting and managing packet replication; it needs to be transparent to applications. 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.

Multicast is essential for applications such as PMU streaming or financial data dissemination. Supporting it increases the complexity of the soft-state maintenance - receivers should not flood the sender with keep-alives.

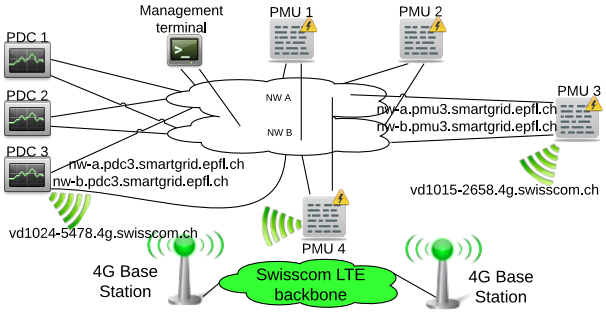


Figure 1: A typical iPRP use-case in the context of smart grids. Hosts (PDCs, PMUs) are connected to two overlapping networks (labeled *A* and *B*). Some devices use an additional LTE connection providing a low latency cellular service [3, 25]. Every PMU streams data to all PDCs, using UDP and IP multicast.

Finally, the proposed solution needs to be secure, *i.e.*, to ensure that replication does not compromise secure UDP streams (encrypted, or authenticated).

1.3 iPRP

We design and implement iPRP (IP parallel redundancy protocol), a transport layer solution for transparent replication of unidirectional unicast or multicast UDP flows on multihomed hosts. iPRP targets unidirectional UDP flows with hard delay constraints (*e.g.*, critical monitoring and control). iPRP replicates the time critical flows on several interfaces and removes duplicates at the destination. It matches corresponding source and destination interfaces to ensure disjoint paths are used if possible.

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

Our iPRP implementation is for IPv6, as it is being installed in our smart-grid communication network (`smartgrid.epfl.ch`), which is required to use IPv6 (on the argument that new network environments should avoid future transition problems and embrace IPv6 from the start [4, 8]). Adaptation to IPv4 should be easy.

Contributions

We design an **application-** and **network-transparent** protocol for multi-homed hosts that takes advantage of any available network redundancy.

We design iPRP to **selectively replicate** target UDP flows. The destination address and UDP port number are used to distinguish between flows that require duplication and those that do not. The targeted flows are unidi-

rectional, *e.g.*, measurements. Hence, by design iPRP is triggered by the receiving host upon reception of a packet on any of its custom-defined set of monitored UDP ports. The receiving host periodically announces its iPRP capabilities to the sender via capability messages. If the sender is iPRP-enabled, the reception of the capability messages triggers packet replication.

We design a soft-state mechanism that guarantees **automatic session restart**. Both ends maintain a soft-state iPRP session that is automatically refreshed by the reception of capability messages (on the sender side) and by reception of data (on the receiver side). Thus after a failure recovery (in a link, or in a host), iPRP resumes automatically. Our duplicate removal algorithm is robust to such events and guarantees that a single copy of any received packet is delivered through the socket.

We natively **support IP multicast** (Section 3). The soft state approach enables receivers to join and leave transparently. Furthermore, we carefully tune a backoff algorithm (Section 4.4) to avoid iPRP senders from being flooded by receivers' capability messages.

iPRP is **secure**. iPRP authenticates and encrypts control messages and iPRP headers in the duplicated packets. It can be safely used for secured UDP flows and does not interfere with the encrypted payload (Section 5).

We provide **diagnostic tools**. iPRP configuration can be monitored and tested using standard tools such as ping and traceroute; we also provide specific iPRP tools (Section 6) to diagnose the joint-operation of parallel paths.

As a proof-of-concept, we developed an **IPv6 implementation** of iPRP using the NF_QUEUE framework. Our implementation (Section 7) is being installed in a real smart-grid communication network and is publicly available (`http://goo.gl/N5wFNt`) for deployment. On evaluating its performance (Section 8), we find that it induces a minor processing overhead of 0.4 μs while supporting multiple sessions.

2 Related Work

As mentioned in Section 1, iPRP overcomes the limitations of PRP [13]. The authors of [22] are aware of these limitations 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 [6] 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 duplicates 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, LACP [14] and ECMP [11] require seconds for failover.

Network coding exploits network redundancy for increasing throughput [7], and requires intermediary nodes to recode packets (hence needs specialized network equipment). Moreover, it is not suitable for time-critical applications because decoding delays are introduced, as typically packets are coded across "generations". Source coding (e.g. Fountain codes [18]) can be beneficial for the bursty transmissions of several packets. However, it adds delay, as encoding and decoding are performed across several packets which is not suitable for UDP flows with very hard-delay constraints.

MPLS-TP $1 + 1$ protection feature [24] 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. [21]) and can be used to create disjoint paths without building physically separated networks. It does not solve the problem of transparent packet replication, but serves as a complement to iPRP.

3 Operating conditions of iPRP

This section describes the conditions on hosts and network, required for iPRP operation.

iPRP provides $1 + n$ redundancy. It increases, by packet replication, the reliability of UDP flows. It does not impact TCP flows. In this version iPRP supports unicast and source-specific multicast (SSM), see below.

iPRP-enabled hosts configure a set of UDP ports as *monitored*. Only UDP flows that target such ports are affected by iPRP. When a host receives UDP packets on any of the monitored ports, it triggers a one-way *iPRP session* between the sender and the receiver: soft-state is maintained at the receiver and at the sender, via exchange of control messages. Soft-state is essential for automatic operation and crash recovery.

An iPRP session is unicast (single sender, single receiver), or multicast (single sender, group of receivers). If several hosts transmit to the same receiver (or group of them), then each sender creates a separate iPRP session.

Traffic in the reverse path is not duplicated. To establish an iPRP session in the reverse direction, the new receiver must monitor appropriate ports. This design is chosen because the targeted applications typically need reliable *unidirectional* communication. All operations, such as establishing a new iPRP session, terminating inactive ones, and intelligent matching of interfaces to be used (see Section 3.2), are performed automatically after a packet is received on a monitored UDP port.

Once an iPRP session is established, packet replication is started: the sender clones outgoing packets, rewrites the innermost UDP header, inserts a tag called iPRP header (Section 4.2). The resulting packets are sent to the matching receiver interfaces. All copies of the same original packet have the same *sequence number* in this header. At the receiver, duplicate packets with the same sequence number in the iPRP header are discarded. The original packet is reconstructed and forwarded to the application from the first received copy.

Multicast operation has a few specificities. Firstly, all hosts in the group of receivers need to run iPRP. If only part of the receivers support iPRP, these trigger the start of an iPRP session with the sender and benefit from iPRP; however, others stop receiving data correctly.

Secondly, we recommend the use of source-specific multicast (SSM). In SSM the receivers explicitly join a group identified by a multicast address *and* a source IP address; only this source IP address is allowed to transmit to the group and there are as many replicated multicast trees as there are sender interfaces. The sender host can thus send out the replicated packets over each of its interfaces to the same destination multicast address and UDP port. Other flavors of multicast (*e.g.*, any-source multicast – ASM) do not ensure different multicast trees when the different network clouds are inter-connected and when the same multicast address is used. With ASM it could thus happen that, if there are n interconnected network clouds, the number of received packets is equal to n^2 (instead of n); further the trees are not necessarily disjoint and there could be single points of failure. In order to fully exploit iPRP and have disjoint paths, a solution when using ASM would be to use different multicast addresses for different networks, which in turn would impose additional management difficulties. All of these problems are avoided with SSM. Note that the protocol itself works with any flavor of multicast.

To guard against security weaknesses, all iPRP-related information is encrypted and authenticated. We rely on existing mechanisms for cryptographic key exchange. We discuss security considerations in Section 5.

3.1 UDP Ports Affected by iPRP

iPRP requires two system UDP ports for its exclusive 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 hosts always listen, on the respective ports, for iPRP control and data messages.

Also, a receiver host needs to configure the set (say **P**) of monitored UDP ports, over which iPRP replication is desired (such ports are *not* reserved by iPRP and can be any UDP ports). UDP ports can be added to/removed from **P** anytime during the iPRP operation. The UDP packet reception on a port in **P** causes 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.

3.2 Matching the Interconnected Interfaces of Different Hosts

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* clouds are interconnected. However, the routing is designed such that, a flow originating at an interface connected to cloud *A* with a destination in *A*, will stay in cloud *A*. PRP achieves this by requiring two physically separated cloned networks. iPRP does not impose these restrictions. Hence, iPRP needs a mechanism to match interfaces connected to the same cloud.

To facilitate appropriate matching, each interface is associated with a 4-bit identifier called *iPRP network discriminator (IND)*, which qualifies the cloud it is connected to. Hosts learn each of their interfaces' INDs via simple rules in a local configuration file. Of course, the 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 file 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 where the packet is supposed to transit (see Section 4.2). 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.

4 Protocol description

4.1 Protocol Walkthrough

Here, we discuss the series of events on a sender and a receiver during typical iPRP operation.

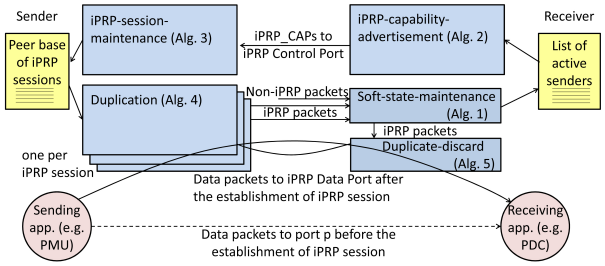


Figure 2: Overview of the functional blocks.

There are five iPRP functional blocks (Fig. 2): at the sender host – *iPRP-session-maintenance* (one per host) and *duplication* (one per iPRP session) – at the receiver – *iPRP-capability-advertisement*, *soft-state-maintenance* and *duplicate-discard* (one of each per host). They are described in the Algorithms 1-5.

iPRP is not explicitly started with a specific application. Rather, it is triggered by the reception of a UDP packet from the sender application by the receiver application on some *monitored port p* (it is made as monitored at time t_1 in Figure 3). The *iPRP session* is initiated by the receiver host: it adds the sender to the list of active senders and updates the associated last-seen timer (Algorithm 1). The same algorithm removes inactive hosts with expired last-seen timers from the list. iPRP functional blocks at the receiver start processing packets received on port p after t_1. At this point, the sender is unaware of the receiver's iPRP capabilities; the packets sent by the application are not yet affected.

The receiver periodically advertises its iPRP capabilities to the sender, who listens for such messages on its iPRP control port (t_2 in Figure 3, Algorithm 2). The `iPRP_CAP` message informs the sender that the receiver is iPRP enabled and provides information required for selective replication over alternative paths. It contains:

- The iPRP version.
- INDs of the networks to which the receiver is connected, to facilitate IND matching (see Section 3.2)

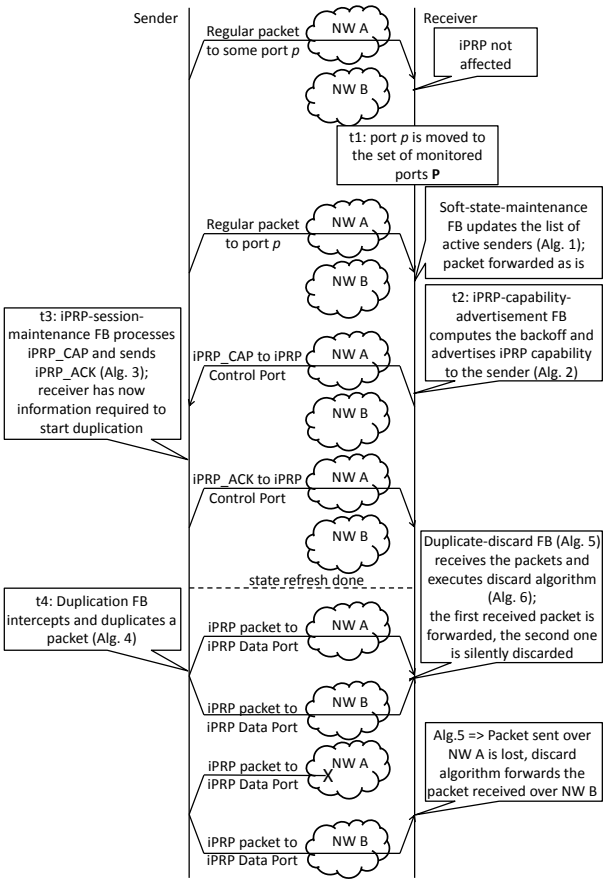


Figure 3: Message sequence chart for typical scenario when iPRP-capable hosts are starting iPRP operation.

- The source and destination UDP port numbers of the packet that triggered iPRP session establishment.
- In multicast, the multicast IP address of the group.
- In unicast, IP addresses of all receiver interfaces.
- A symmetric, short-lived cryptographic key for authentication and encryption of iPRP header (Section 5).

In multicast, receivers in the multicast group reply after a backoff period (Section 4.4) with the multicast group address and with the INDs of the receiver's interfaces. Using SSM allows the sender to transmit the packet on its interfaces using the same destination multicast address. However, it needs to advertise the other IP addresses of its interfaces to the group of receivers. It does so via an `iPRP_ACK` message, which also serves a terminating message for impending `iPRP_CAP`s thereby preventing a flood (Alg. 2). The receivers thus are informed of the source-specific multicast groups that they need to be subscribed to.

During iPRP operation, a new interface might be added or an existing one removed on a receiver. Such changes demand updates of the peer-base at the sender. In a multicast setting, the backoff algorithm in Sec-

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

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

Algorithm 2: iPRP capability advertisement

```
1  while true do
2      compute T_backoff (Section 4.4);
3      listen for iPRP_ACKs until T_backoff expires;
4      send iPRP_CAP messages to all hosts in the list
       of active senders from which no iPRP_ACKs are
       received;
5      sleep T_CAP − T_backoff;
6  end
```

tion 4.4 takes care of avoiding that the sender be flooded with updates. However, it can happen that all receivers do not support the same set of INDs. Our design goal is to serve the union of all INDs. 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 4.4).

Once the soft-state maintenance functional block learns about alternative networks, iPRP_CAP messages are sent over all of them. These are acknowledged with iPRP_ACK messages over the respective networks. Hence, iPRP control message exchange does not rely on any particular network, making our protocol robust. iPRP_CAP and iPRP_ACK messages are sent over a secured channel to the reserved iPRP control port (see Section 5). The security algorithm used for iPRP header protection can be chosen as part of the iPRP configuration. This determines the length of the key.

The iPRP-session-maintenance functional block on the sender host listens on the iPRP control port for the incoming iPRP_CAP control messages (as described in Alg. 3 and shown at time t_3 in Fig. 3) and updates the *peer-base* that contains information about the networks used for replication, when at least a pairs of interfaces are successfully matched based on their INDs. If no iPRP_CAP message is received at the sender after three consecutive firings of the T_{CAP} timer, the corresponding peer-base entry is deleted. So, iPRP_CAP messages constitute a keep-alive mechanism for iPRP sessions.

Algorithm 3: 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
```

When the iPRP session is thus initiated, the duplication functional block on the sender intercepts all outgoing packets destined to UDP port p of the receiver. These packets are subsequently replicated and iPRP headers (Section 4.2) 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. This procedure is described in Algorithm 4, see also time t_4 in Figure 3.

At the receiver, upon the reception of packets on the iPRP data port, Algorithm 1 is executed to update the list of active senders. Then, they are forwarded to the duplicate-discard functional block (Algorithm 5). It decrypts the number of bits that corresponds to the length of the iPRP header at the beginning of the payload. To this end, a receiver uses the symmetric key that was previously exchanged within a iPRP_CAP message. Then, function isFreshPacket (Section 4.3 - Algorithm 6) is called. It decides, based on the sequence-number-space ID and the sequence number, whether the packet is for-

Algorithm 4: Packet duplication

```
for every outgoing packet do
    check the peer-base;
    if there exists an iPRP session that corresponds
    to the destination socket then
        replicate the payload;
        append iPRP headers incl. seq. number;
        send packet copies;
    else
        forward the packet unchanged;
    end
end
```

warded to the application, or not: the first received copy should reach the application, subsequent copies are discarded. The duplication is thus rendered transparent to the sender and receiver applications.

Algorithm 5: Duplicate discard

```
for every packet received on iPRP data port do
    get sequence number space ID (SNSID) ;
    get sequence number (SN);
    if isFreshPacket(SN, SNSID) then
        remove iPRP header;
        reconstruct original packet;
        forward to application;
    else
        silently discard the packet;
    end
end
```

In Figure 3 we show two scenarios after the time t_4; in one case both copies are delivered, in the other, one packet is lost. Note that the protocol remains unchanged if communication was started on network B.

Other Scenarios: When the receiver is not iPRP-capable, the functional blocks on the receiver are not executed. Consequently, an `iPRP_CAP` message is never sent and the sender does not start the duplication. Data packets are exchanged without any modifications. When the sender is not iPRP-capable, the Algorithm 3 is not executed. Hence, the `iPRP_CAP` messages sent to the iPRP control port by the receiver are not handled and regular data communication continues.

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 the capability advertisement procedure had been started by other receivers that listen to the same multicast groups. The new receiver will immediately be able to process correctly packets received over the iPRP data port.

4.2 The iPRP Header

In Fig. 4 we show the position and the fields of the iPRP header used in data packets. The Sequence-number-space ID 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 and the source UDP port number of the socket to which the application writes the packet. For the 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).

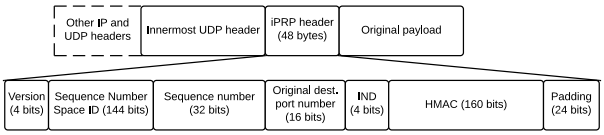


Figure 4: Location and fields of iPRP header.

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 [5]. In practice, typical MTU values are closer to the IPv6-recommended 1500 bytes. This leaves a comfortable margin for the inclusion of the iPRP and other tunneling protocol headers.

4.3 The Discard Algorithm

The redundant copies of a packet are eliminated by a discard algorithm running at the receiver. In some scenarios, the one proposed for PRP [26] 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. Also, it allows a quick session recovery by keeping track of the number of "very late" packets received in a row. We use 32-bit unsigned integer sequence numbers, large enough to avoid wrap-around.

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`". "`x` precedes `y` by more than `z`" is defined similarly.

```
1  function isFreshPacket(CurrSN, SNSID)
2  if CurrSN==SNSID.HighSN then
3  |   return false;
4  else if CurrSN follows SNSID.HighSN then
5  |   put missing packets' SNs in SNSID.ListSN (up
   |   to MaxLost entries);
6  |   SNSID.HighSN ← CurrSN;
7  |   SNSID.ResetCNT ← 0;
8  |   return true;
9  else
10 |   if CurrSN precedes SNSID.HighSN by more
   |   than MaxLost then
11 |   |   SNSID.ResetCNT++;
12 |   end
13 |   if SNSID.ResetCNT==MaxOld then
14 |   |   clear all variables;
15 |   |   return true;
16 |   end
17 |   if CurrSN is in SNSID.ListSN then
18 |   |   remove CurrSN from SNSID.ListSN;
19 |   |   return true;
20 |   else
21 |   |   return false;
22 |   end
23 end
```

• `ResetCNT` – number of "very late" packets in a row.

`ListSN` is bounded to a maximum of `MaxLost` entries. Thus, any received packet with a sequence number that precedes the `HighSN` sequence number of the "newest" seen packet by `MaxLost` is considered "very late" and is dropped. The reception of `MaxOld` "very late" packets in a row is symptomatic of a reboot of the sender and triggers a reset of all the counters. Hence, iPRP is able to recover after unexpected events (crashes and reboots). By default, `MaxOld` is set to 5.

The values of `MaxLost` and `MaxOld` are given in the iPRP configuration file and depend on the targeted application. For example, in our smart-grid setting, there is a hard delay constraint of 20 ms (any packet older than that can be safely discarded). To be conservative, we allow packets with the delays of up to 50 ms. Then, it should hold: `MaxLost` $> 50\text{ms} \times r$; r is the rate at which the application is streaming packets. We set `MaxLost` to 1024 which is enough to support rates that are way above any realistic PMU streaming rate.

As the iPRP header containing the sequence number is authenticated and encrypted, an attacker cannot introduce packets with fake sequence numbers. The algorithm keeps track of two variables and of one list per iPRP session. The most expensive operation is searching the list (line 17). However, in practice, `ListSN` is limited to few entries.

4.4 The Backoff Algorithm

In a multicast iPRP session between a sender and a group of receivers, the n group members advertise INDs via `iPRP_CAP` messages. We want to avoid message implosion at the source. We require the source to receive an `iPRP_CAP` within $D = 10$s after the start of the loop in Algorithm 2 (executed periodically every $T_{CAP} = 30$s).

Since we do not know the number of receivers in the multicast group, we need a backoff scheme that avoids message implosion for groups ranging from several hosts to millions (*e.g.*, online gaming).

We adopt the typical approach: each host performs a random backoff before transmitting an `iPRP_CAP`. At reception of an `iPRP_ACK`, its transmission is canceled.

The choice of the distribution of the backoff timer is important; it should be such that the expected number of `iPRP_CAP` messages received by the source remains small while at the same time guaranteeing with overwhelming probability that at least one message is received within time D. The classical choice is an exponential distribution, but it not very good as it tends to generate a large number of messages when the group is large. A distribution of the backoff time that ensures a much smaller number of redundant messages was proposed and studied by Nonnenmacher and Biersack [20]; it is the flipped truncated exponential distribution, defined by a PDF on $[0,D]$ that increases toward D, $f_D(x;\lambda) \stackrel{\text{def.}}{=} \lambda e^{\lambda x}(e^{\lambda D}-1)^{-1} \cdot \mathbb{1}_{\{x\in[0,D]\}}$ (it is inappropriately called "exponential" in [20]).

In lossless networks, when the backoff time is distributed as $f_D(x;\lambda)$, it is possible to determine a value λ that, for a wide range of RTT values (from milliseconds to hundreds of milliseconds) and for groups of receivers ranging from 1 to several millions, guarantees in expectation at most two redundant messages received at the source. Indeed, for a given λ, the expectation of the number of `iPRP_CAP` messages received by the source during an interval of length D when there are n is approximately $n(e^{\lambda R}-1)(e^{\lambda D}-1)^{-1}+e^{\lambda R}$ [20]. Thus, for fixed values of n and R, there exists an optimal value of the parameter λ that minimizes the previous expression, and that can be determined numerically. We show in Appendix A that, for a wide range of values of n and R, the previous expression is close to its minimum even

when the parameter λ is slightly larger than the optimal value. This suggests that a large enough value of λ is suitable. For example, for $\frac{R}{D} = 10^{-3}$ with $\lambda = 25/D$, we obtain in expectation $\approx 1.025 + 10^{-11}n$ redundant messages.

It is shown by simulation in [20] that this distribution is very robust to to lost iPRP_ACKs, namely the above numbers are only marginally affected by moderate loss rates. Further, we verify in Appendix A that this holds even in presence of a large number of consecutive packet losses.

In our implementation, each host computes the backoff by CDF inversion. It uses a random number generator to obtain a uniform random variable $U \in [0,1]$ and then sets $T_{\text{backoff}} = \lambda^{-1}\ln(1+(e^{\lambda D}-1)U)$ (Algorithm 2, line 2). We pick for our implementation $\lambda = 25/D$.

See Appendix A for more details.

5 Security considerations

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, we avoid replay attacks and the forged messages insertion.

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. Thus, the iPRP capabilities of the receiver are transmitted only to an authenticated source. iPRP_ACK messages are not required in unicast (one receiver that sends iPRP_CAPs so no risk of flooding).

In unicast mode, 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. Thus, 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 [1] can be used for group key management and for establishing a group security association multicast security, like in [12].

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, in multicast mode there is no need to include an additional key in the iPRP_CAP .

It is important to note that as it is transparent to the application, iPRP does not interfere with encryption/authentication at the application layer.

6 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.

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. Next, we describe the functioning of each tool:

• `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
```

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 **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 **P** on the remote machine. If an iPRP session could not be established, an appropriate message is generated.

• `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.

• `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.

• `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.

• `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.

7 Implementation

To verify the operation of iPRP protocol and evaluate the performance overhead due to the addition of iPRP to TCP/IP stack, we implement iPRP for Linux based systems. For the proof-of-concept we opted for a user-space implementation in place of a kernel-space implementation trading the efficiency for the ease of debugging. A suitable framework for iPRP implementation should have the following properties:

• Allow for the selective filtering of IP packets so that the iPRP sequence of operation can be applied.

• Allows for packet mangling, so that 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.

• Minimal CPU footprint so that iPRP operation causes minimum CPU overhead.

We chose 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). Hence, our implementation supports all Linux kernel versions above 2.6.14. We use the the Linux kernel 3.11 with iptables-1.4.12.

In our implementation, we have four daemon, and the following is the mapping between them and the functional blocks introduced in Section 4:

• *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 **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 **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 **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 4.4) 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 3.2) 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 4.2). 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 4.3) 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 **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.

8 Performance evaluation

In order to evaluate the operation of iPRP protocol, we do two types of assessments. The first one is to evaluate the operation of the iPRP and its discard algorithm in different scenarios. The latter set of experiments is to assess the processing delay due to iPRP and the additional CPU usage used by iPRP software of our proof-of-concept implementation. Our test bed consists of two Lenovo ThinkPad T400 laptops with a 64-bit Ubuntu OS. The laptops (Table 1) are connected over two Ethernet based wired networks (one via USB adapter) and one ad-hoc Wi-Fi network. We label the interface eth0 as nwA, eth1 as nwB and wlan0 as nwC. To evaluate the real-time operation, we patched the Linux kernel 3.11.6 with the associated real-time Linux patch *rt29*.

Component	Version Specifications
CPU	Intel C2Duo, 2.53 Ghz
Ethernet Controller	Intel 82567LM Gigabit Card
Wireless Controller	Intel Wifi N d5300
Operating System	Ubuntu 12.04 LTS, 64-bit
Kernel	3.6.11-rt29 (RT-Linux Patch)

Table 1: Specifications of hosts used in the test bed

8.1 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. The packet losses and delays are emulated using the Linux `tc-netem` [10] tool on the test bed described in Table 1.

In Table 2 we summarize settings used in different scenarios. To mimic the traffic created by PMUs, we send a 280 byte UDP datagram every $20ms$, long enough to have stationary behavior. We emulate delays that are uniformly distributed within $10ms \pm 5ms$ (small differences in network topologies and/or loads), and within $1s \pm 0.2s$ (significant differences in network topologies or serious perturbations in network functioning). We emulate both independent and bursty losses. In both cases the overall packets loss rate is 5%. To produce bursty losses with `tc-netem` we use Gilbert-Elliot model [23] with $p = 0.01$, $r = 0.19$, $1-k = 0.01$, $1-h = 0.81$.

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 2: 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.

We use Scenario 0 to evaluate the operation of iPRP in the presence of more than two networks. In Scenarios 1-4, we test the discard algorithm by making asymmetric delays and losses, thus forcing it to keep track of delayed/missing packets. With Scenario 5, we test the expected iPRP side-benefit of having lower average one-way network latency, given that the iPRP duplicate-discard functional block always forwards the first packet delivered over any of the available networks. We measure delays and losses over individual networks and as experienced by an application after iPRP.

In Table 3, 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 3 shows that iPRP performs as expected in significantly reducing the effective packet losses.

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 3: Loss percentages in various scenarios

In Figure 5 we show the CDF of one-way network latency (d_{IPRP}) for Scenario 5. In theory we expect $d_{iPRP} = min(d_{nwA}, d_{nwB})$. What we measured matches the theory very well. This is a confirmation of the anticipated side-benefit of the iPRP: the delays perceived by the application are improved when iPRP is used, compared to those when only one of the individual networks is used.

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 $d_{iPRP} = min(d_{nwA}, d_{nwB}) = d_{nwA}$.

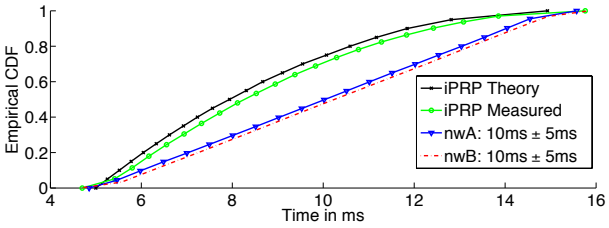


Figure 5: iPRP side benefit: reduced one-way network latency.

8.2 Processing Overhead Caused by iPRP

In this subsection, we evaluate processing delays and the additional CPU load when iPRP is used on the test bed described in Table 1. We conduct several runs of Scenario 1 (see Table 2) and use *GNU gprof* [9] to assess the average processing delay incurred by an iPRP data packet at ISD and IRD. In an ISD, 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. In an IRD, 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 by iPRP daemons on the sender and on the receiver. The results are summarized in Table 4. In Experiment 1 we keep constant aggregate packet rate of 1000 packets per second (pps) for all established sessions (1 session of 1000 pps, 2 sessions of 500 pps each, etc.). The CPU usage with iPRP is quasi-constant at 15 % for the sender and 12 % for the receiver. In Experiment 2 we keep constant packet rate of 10 pps for every individual session (1 session - 10 pps in total, 2 sessions - 20 pps in total, etc.). At the sender, the CPU usage increases, at first, at a rate of 0.9 % per session and the increase rate per session decreases to 0.32 % for larger number of sessions. At the receiver, the increase rate of CPU usage per each additional session goes from 0.8 % to 0.22 %.

Hence, we can say that with our user-space implementation we have a successful proof-of-concept. In future research, we will focus on making an even more efficient implementation in kernel-space or, even better, we would like to push the implementation of iPRP functionalities to the network adapter itself, similarly to the TCP segmentation offload technique [2].

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 4: CPU usage with iPRP and varying loads

9 Conclusion

We have designed iPRP, a transport layer solution for improving reliability of UDP flows in networks with very hard delay constraints. 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 and is thus suitable for networks such as smart-grid measurement dissemination, high-frequency trading and multiplayer online gaming. We have equipped iPRP with diverse monitoring and debugging tools, which is quasi impossible with existing MAC layer solutions. With our proof-of-concept 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.

References

[1] BAUGHER, M., CANETTI, R., DONDETI, L., AND LINDHOLM, F. Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, 2005.

[2] BOUCHER, L., BLIGHTMAN, S., CRAFT, P., HIGGEN, D., PHILBRICK, C., AND STARR, D. Tcp offload network interface device, Oct. 16 2007. US Patent 7,284,070.

[3] CIPRIANO, A. M., AGOSTINI, P., BLAD, A., AND KNOPP, R. Cooperative communications with HARQ in a wireless mesh network based on 3GPP LTE. In *EUSIPCO 2012, European Signal Processing Conference, August, 27-31, 2012, Bucharest, Romania* (Bucharest, ROMANIA, 08 2012).

[4] CISCO. Why IP IS the right foundation for the Smart Grid. *Whitepaper* (2010).

[5] DEERING, S., AND HINDEN, R. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[6] FORD, A., RAICIU, C., HANDLEY, M., AND BONAVENTURE, O. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), Jan. 2013.

[7] FRAGOULI, C., AND SOLJANIN, E. Network coding fundamentals. *Foundations and Trends in Networking 2*, 1 (2007), 1–133.

[8] GAO, J., XIAO, Y., LIU, J., LIANG, W., AND CHEN, C. P. A survey of communication/networking in smart grids. *Future Gener. Comput. Syst. 28*, 2 (Feb. 2012), 391–404.

[9] GRAHAM, S. L., KESSLER, P. B., AND MCKUSICK, M. K. Gprof: A call graph execution profiler. In *ACM Sigplan Notices* (1982), vol. 17, ACM, pp. 120–126.

[10] HEMMINGER, S., ET AL. Network emulation with netem. In *Linux Conf Au* (2005), Citeseer, pp. 18–23.

[11] HOPPS, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), Nov. 2000.

[12] HYUN, D., SUNG, W., PYON, D., KIM, T., HAN, W., LEE, J., AND JANG, J. IPv6 Secure Multicast Conferencing. In *Hybrid Information Technology, 2006. ICHIT '06. International Conference on* (Nov 2006), vol. 2, pp. 68–73.

[13] IEC 62439-3 STANDARD. Industrial communication networks: High availability automation networks, 2012.

[14] IEEE STANDARDS ASSOCIATION. IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks Link Aggregation., 2008.

[15] KIRRMANN, H., HANSSON, M., AND MURI, P. 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.

[16] KIRRMANN, H., WEBER, K., KLEINEBERG, O., AND WEIBEL, H. Hsr: Zero recovery time and low-cost redundancy for industrial ethernet (high availability seamless redundancy, iec 62439-3). In *Proceedings of the 14th IEEE International Conference on Emerging Technologies & Factory Automation* (Piscataway, NJ, USA, 2009), ETFA'09, IEEE Press, pp. 203–206.

[17] LITZ, H., LEBER, C., AND GEIB, B. Dsl programmable engine for high frequency trading acceleration. In *Proceedings of the Fourth Workshop on High Performance Computational Finance* (New York, NY, USA, 2011), WHPCF '11, ACM, pp. 31–38.

[18] MACKAY, D. J. C. Fountain codes. *Communications, IEE Proceedings- 152*, 6 (Dec 2005), 1062–1068.

[19] MAHALINGAM, M., DUTT, D., DUDA, K., AGARWAL, P., KREEGER, L., SRIDHAR, T., BURSELL, M., AND WRIGHT, C. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational), Aug. 2014.

[20] NONNENMACHER, J., AND BIERSACK, E. W. Scalable feedback for large groups. *IEEE/ACM Transactions on Networking (ToN) 7*, 3 (1999), 375–386.

[21] PSENAK, P., MIRTORABI, S., ROY, A., NGUYEN, L., AND PILLAY-ESNAULT, P. Multi-Topology (MT) Routing in OSPF. RFC 4915 (Proposed Standard), June 2007.

[22] RENTSCHLER, M., AND HEINE, H. The parallel redundancy protocol for industrial ip networks. In *Industrial Technology (ICIT), 2013 IEEE International Conference on* (Feb 2013), pp. 1404–1409.

[23] SALSANO, S., LUDOVICI, F., AND ORDINE, A. Definition of a general and intuitive loss model for packet networks and its implementation in the netem module in the linux kernel. Tech. rep., Technical report, Technical report, University of Rome, 2009.

[24] SPRECHER, N., AND FARREL, A. MPLS Transport Profile (MPLS-TP) Survivability Framework. RFC 6372 (Informational), Sept. 2011.

[25] VILLA, T., MERZ, R., KNOPP, R., AND TAKYAR, U. Adaptive modulation and coding with Hybrid-ARQ for latency-constrained networks. In *EW 2012, 18th European Wireless Conference, April 18-20, Poznan, Poland* (Poznan, POLAND, 04 2012).

[26] WEIBEL, H. Tutorial on parallel redundancy protocol. Tech. rep., Zurich University of Applied Sciences.

[27] ZHANG, C., HUANG, C., CHOU, P. A., LI, J., MEHROTRA, S., ROSS, K. W., CHEN, H., LIVNI, F., AND THALER, J. Pangolin: speeding up concurrent messaging for cloud-based social gaming. In *CoNEXT* (2011), K. Cho and M. Crovella, Eds., ACM, p. 23.

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 $n \lesssim 10^6$.

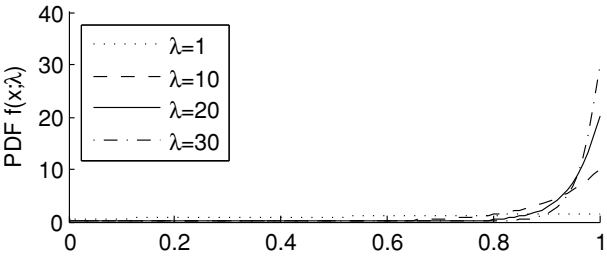


Figure 6: The PDF of flipped truncated exponential distributions for various values of λ.

A.1 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 [20]:

Theorem 1 *In a lossless network, for a multicast group of size n, the expected number of messages received by the source is*

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

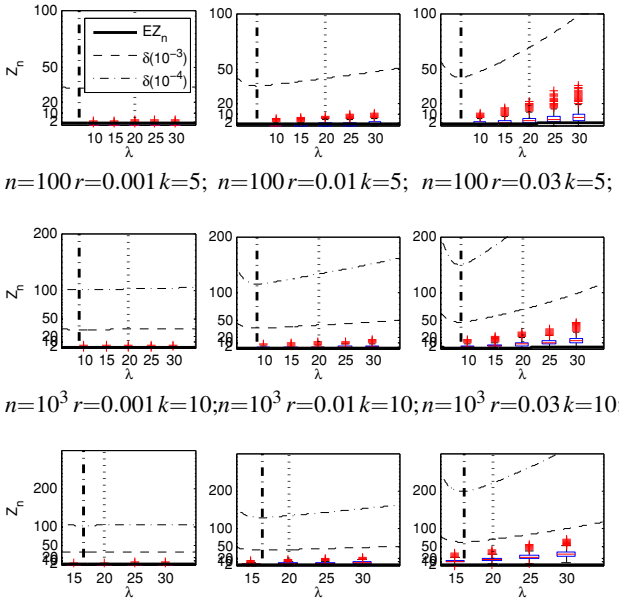
In particular, for a fixed choice of λ, the probability that Z_n exceeds a number of messages $\delta > 0$ is upper bounded by $\mathbb{P}[Z_n > \delta] \le \Psi_n(\delta, r;\lambda)$, where

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

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, $\mathbb{E}Z_n = n\mathbb{E}Y_{1n}$:

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



n=100 r=0.001 k=5; n=100 r=0.01 k=5; n=100 r=0.03 k=5;

n=10^3 r=0.001 k=10; n=10^3 r=0.01 k=10; n=10^3 r=0.03 k=10;

n=10^6 r=0.001 k=20; n=10^6 r=0.01 k=20; n=10^6 r=0.03 k=20;

Figure 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 1 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.

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

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

□

A.2 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 $r = \mathrm{RTT}/D = 0.03$.

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.