PRRT Implementation & Usage

In the course of this thesis the predictably reliable real-time protocol has been implemented into a runtime-loadable Linux kernel module[1]. The developed protocol stack fulfills the protocol specification defined in Chapter 2. The protocol implementation, conveniently integrates predictable reliability into networked applications as it wraps a standard BSD socket interface with few protocol-specific modifications and extensions. The source package includes small command line tools that implement PRRT's main features and allow for special configuration via command line options. Given a real-time media source, the example tools conveniently set up a networked pipe that transmits the source data between remote network sites with predictable reliability.

Protocol Interface

In addition to the host address and the source or destination port, which both define the Transport Service Access Point (TSAP) of a transport layer protocol, PRRT requires the exchange of additional information in order to enable the bidirectional error control. As the protocol itself is connectionless, session initiation must be implemented on higher layers in order to communicate the host addresses and ports for source data and feedback transmission. In order to provide the required communication and functionality, a C++ Socket library maps the PRRT socket API to the related system calls of the BSD socket interface and registers the required components in the PRRT protocol stack. The protocol stack implements the predictably reliable error control, the reliability control and the delay-based congestion control in a multi-threaded environment such that ordinary *send()* and *receive()* calls at the PRRT hosts entirely abstract the complex protocol functionality. As soon as a character buffer is written into the sender socket via *send()*, the protocol stack determines the delivery deadline of the datagram and initiates the error control operations.

the PRRT socket functionality that is implemented via a multi-threaded architecture within the network stack. receive() obtains a packet not earlier than packet timeout such that error control can is completed transparently.

use of connect and bind slightly modified since each PRRT session is bidirectional as it requires a feedback channel.

The protocol's programmer interface is defined in a C++ socket library as follows:

- **prrt_socket()** instantiates a new PRRT socket and allocates the required system resources. The socket is registered in a global register file ... the state of all

[1]Source code available for download under http://www.nt.uni-saarland.de/en/projects/running-projects/prrt.html

PRRT sockets. For each PRRT socket, the library creates an entry in the Linux */proc* file system that returns information and statistics about the socket state (current protocol parameters, number of source and repair packets sent/received), the observed network state (PLR, RTT, CC) and the protocol performance (P_r, RI).

- **connect()** sets up a sending PRRT socket. This function configures the *destination address*, the *destination port* as well as the *feedback address*. *destination address* and *destination port* might identify either a unicast receiver or a multicast group. *feedback address* identifies the local interface PRRT must listen at in order to obtain receiver feedback. In order to benefit from feedback suppression in multicast, *feedback address* should represent a multicast group. The feedback port is automatically set to *destination port + 1*. The function returns successfully if the required addresses and port numbers are valid and the automatically assigned feedback port is not already in use.

- **bind()** sets up a receiving PRRT socket. This function configures the *source address*, the *source port* as well as the *feedback address*. *source address* and *source port* might identify either a local network interface of the receiver or a multicast group. *feedback address* identifies the local interface PRRT must listen at in order to obtain receiver feedback. In order to benefit from feedback suppression in multicast, *feedback address* should represent a multicast group. The feedback port is automatically set to *destination port + 1*. The function returns successfully if the required addresses and port numbers are valid and the automatically assigned feedback port is not already in use.

- **close()** closes the PRRT socket and de-allocates all system resources. The socket is being deleted from the register file and the related entry in the */proc* file system is being removed.

- **send()** wraps a buffer of characters into a datagram of the required length and sends it to the receiver socket.

- **recv()** receives a datagram of specific length from the sender socket and returns the encapsulated buffer of characters.

- **setsockopt()** configure the PRRT socket via general UDP socket options and special PRRT socket options such as delay constraint, reliability constraint, rate constraint and manual settings for coding and protocol parameters.

- **getsockopt()** obtain the current settings for general UDP socket options and special PRRT socket options.

Additional socket functions beyond the BSD interface

- **getstats()** returns current socket statistics such as the socket state (current protocol parameters, number of source and repair packets sent/received), the observed network state (PLR, RTT, CC) and the protocol performance (P_r, RI).

- **getstreamrate()** returns the current goodput estimate for which the application's reliability requirement is satisfiable. If the application feeds data into the sender socket at a rate beyond the obtained goodput estimate, it might experience increased residual packet loss rate as PRRT's packet scheduler masks away excessive source and repair packets.

Example Tools

The following example tools implement a pair of simple PRRT sender and receiver. The tools presented in the following have been applied in order to obtain the experimental results in Chapter 6. They provide a simple self-test environment by transmitting an artificial character source but also a networked pipe functionality between the endpoints of a one-to-one or one-to-many real-time media streaming application. The tools export special protocol options such as fixed configuration or activation and deactivation of single protocol features.

Sender "prrtcat"

prrtcat is a PRRT-based sender application that implements connection-less streaming of real-time source data to a unicast PRRT receiver or a multicast group of PRRT receivers. Name and functionality are adopted from the POSIX tool netcat[2], which reads character data from the standard input and sends it to a remote socket via UDP or TCP. Similarly, prrtcat reads data from the standard input and encapsulates them into PRRT packets while performing the predictably reliable error control. Alternatively, the tool sends out an artificial character stream with a specified real-time source rate. This option is particularly interesting for a self-test or a performance measurement under evaluation of the socket statistics at the receiver.

The software is configured via the following command line options:

```
Sender options:
  -sp <port>                  destination port [default: 5004]
  -suc <unicast address>      unicast destination address
  -smc <multicast address>    multicast destination address
  -fb <address>               feedback address
  -code <coding parameters>   [default: 200,2,20,40,0,1,0,1,2]
                                i.e.: D_T(delay constraint)=200ms, D_C=2ms, D_FEC=20ms,
                                D_REQ=40ms, D_FB=0ms, k=1, N_P=[0 1 2]
  -g <target PLR>             [default: 0.000001]
  -j <data rate>               Sends from an artificial data source with specified data
                                rate (in Kbps) instead of reading from stdin (default)
  -ad <0|1|2>                 Enable adaptation [default: 0]
                                0: no adaptation
                                1: table lookup
                                2: online optimization
  -i <update interval>         update interval of the parameter adaptation in
                                milliseconds [default: 200]
```

[2]http://linux.die.net/man/1/nc

Receiver "prrtrecv"

prrtrecv offers basic PRRT receiver functionality. The tool instantiates a PRRT receiver socket and writes the received character buffers to the standard output. Except for the congestion control and the feedback scheduling, the PRRT receiver is a purely passive component that obtains all parameter settings via the corresponding protocol headers. Therefore, prrtrecv does not provide any options for coding parameters and application constraints. Bandwidth estimation is either provided by TFRC or a delay-based congestion control equation.

```
Receiver options:
  -rp <port>                  destination port (listen) [default: 5004]
  -ruc <unicast address>      unicast destination address (listen)
  -rmc <multicast address>    multicast destination address (listen)
  -bw <0|1|2>                 enable bandwidth estimation [default: 0]
                                0: no bandwidth estimation
                                1: TCP-friendly rate control (TFRC)
                                2: delay-based congestion control
  -i <updateInterval>         update interval of the bandwidth estimation in
                                milliseconds [default: 200]
  -fs <D_SUP>                 enable feedback suppression with maximum suppression
                                timer D_SUP [default: 0]
```

Example Usage

The following command lines provide some typical example usages of the prrtcat and prrtrecv:

- Send from the artificial character source with a source rate of $5\,Mbps$ to the multicast group 224.0.1.2:5004 while specifying a delay constraint of $400\,ms$ and fixed coding parameters with $k = 1$ and $N_P = [0, 1, 1, 4]$:

  ```
  $> prrtcat -j 5000 -code 400,1,20,40,0,1,0,1,1,4 -suc 224.0.1.2 -fb 224.0.1.2
  ```

- Set up a multicast receiver joining and listening to the group 224.0.1.2:5004 (character stream from standard output might be redirected to /dev/null) and sending feedback with a maximum suppression timer of $50\,ms$:

  ```
  $> prrtrecv -fs 50 -rmc 224.0.1.2 224.0.1.2 [> /dev/null]
  ```

- Send from the artificial character source at sender 192.168.0.1 with adaptive source rate to the unicast receiver 192.168.0.17:5004 under a delay constraint of $400\,ms$ and enable online parameter adaptation with update interval $200\,ms$:

  ```
  $> prrtcat -j 0 -code 400,1,20,40,0,1,0,1,1,4 -ad 2 -i 200 -suc 192.168.0.17
     -fb 192.168.0.1
  ```

- Set up the corresponding unicast receiver 192.168.0.17 sending feedback to 192.168.0.1 and returning a bandwidth estimate every 100 ms

```
$> prrtrecv -bw 2 -i 100 -ruc 192.168.0.17 192.168.0.1 [> /dev/null]
```

- Set up an online-adaptive networked pipe with predictable reliability between *streaming_server* at the sender 192.168.0.1 and *media_renderer* at the receiver 192.168.0.17. *streaming_server* writes data to standard output and *media_renderer* reads from standard input.

```
$> media_server | prrtcat -code 400,1,20,40,0,1,0,1,1,4 -ad 2
   -i 200 -suc 192.168.0.17 -fb 192.168.0.1

$> prrtrecv -ruc 192.168.0.17 192.168.0.1 | media_renderer
```

- Obtain current socket state and protocol statistics from the */proc* file system of an PRRT socket identified via *socket_number*:

```
$> cat /proc/prrt/<socket_number>/stats
```