Software-defined Quantum Communication Systems

Travis S. Humble\textsuperscript{a} and Ronald D. Sadlier\textsuperscript{a,b}

\textsuperscript{a}Quantum Computing Institute, Oak Ridge National Laboratory, Oak Ridge, TN, USA
\textsuperscript{b}University of Rhode Island, Kingston, RI, USA

ABSTRACT

We show how to extend the paradigm of software-defined communication to include quantum communication systems. We introduce the decomposition of a quantum communication terminal into layers separating the concerns of the hardware, software, and middleware. We provide detailed descriptions of how each component operates and we include results of an implementation of the super-dense coding protocol. We argue that the versatility of software-defined quantum communication test beds can be useful for exploring new regimes in communication and rapidly prototyping new systems.

Keywords: quantum communication, software-defined systems, cognitive radio

1. INTRODUCTION

Quantum communication (QC) is an active area of fundamental research and technology development that makes use of the quantum optical properties of light, for example, to transmit and receive quantum information.\textsuperscript{1} It enables novel capabilities such as quantum teleportation or quantum key distribution that cannot be provided by means of classical communication. The design of prototype QC systems is an important step toward realizing theoretical predictions and assessing experimental performance. Of course, similar issues face classical communication (CC) systems and we may expect QC research to leverage existing methods for system prototyping. In particular, software-defined implementations have proven useful for providing flexibility in the design and testing of conventional radio systems\textsuperscript{2} and in this contribution we extend the software-defined communication paradigm to the design and development of QC systems.

Software-defined communication (SDC) allocates signal processing tasks that nominally require specialized hardware to software implementations based on general-purpose computational power.\textsuperscript{2} Within traditional radio communications, the ideal SDC receiver would use an antenna and analog-to-digital converter (ADC) for signal sampling before handing off the remaining waveform processing tasks to software. These tasks, including mixing, filtering, and demodulation, are then tuned by simply reprogramming the radio. The ability for SDC to configure itself in real time affords the opportunity to adapt to transmission environment, i.e., to develop a cognitive radio.\textsuperscript{3} Similar techniques have been argued for use in classic optical communication systems.\textsuperscript{4,5}

Although much of the physics underlying quantum communication is very different from conventional radio, the SDC paradigm can apply to building quantum communication systems as well. This is because both domains employ many of the same processing primitives at the information (bit) level. This includes the de/modulation and de/coding techniques required for individual transmissions in addition to the handshaking exchanges needed to negotiate complete protocols. These common needs motivate our consideration of software-defined quantum communication systems and the evaluation of its feasibility with state of the art quantum optical hardware.

Of course, there are notable differences between quantum communication (QC) and classical communication (CC). These differences manifest from how information is encoded into the photonic carrier. In particular, QC encodes information into the quantum state of a photon using any number of degrees of freedom, e.g., polarization, quadrature phase, spatial mode, angular momentum, frequency, etc. By comparison, CC uses macroscopic amounts of photons to encode the classical state of the same degrees of freedom. This difference leads to unique capabilities for each physical domain, as has been well established.\textsuperscript{6}

Notwithstanding differences at the physical layer, quantum and classical communication share a dependence on logical control data known as metadata. Both regimes require metadata to control, manage, organize and
annotate the transmitted payload. In a typical CC example, metadata is concatenated with the payload by the
transmitter and then extracted by the receiver. This information may, for example, identify the demodulation
needed to recover the payload or specify the destination address needed for routing.

In the case of QC, classical metadata may either be shared through a synchronized side-channel or gener-
ated by measurement of the transmitted quantum state. An example of the latter is found in quantum key
distribution (QKD) in which the transmitter and receiver share measurements to determine the next steps in
the key generation protocol.\textsuperscript{5} By contrast, quantum teleportation and entanglement swapping typically require
a side channel through which to share the classical measurements recorded by the transmitter and needed by
the receiver to recover or relay the quantum state.\textsuperscript{5} Similar examples include the cases of quantum memory
modules or quantum routers that use dynamic addresses to store and route information, respectively. These
latter examples serve to emphasize that a quantum receiver need only operate on the transmitted states and not
necessarily measure them. It is also possible to process metadata within the quantum receiver hardware. This
approach has been taken previously in some QKD and quantum teleportation testbeds.\textsuperscript{6–13}

The ubiquity and importance of metadata in QC motivates consideration of how the SDC paradigm may be
implemented to build communication testbeds. A typical QC transceiver can be decomposed into components
that separate the physical encoding layer from the metadata control layer. We identify these layers as separating
the hardware and software domains while interpolating between these domains is a middleware layer. We describe
implementations of all three domains that maintain a natural separation of concerns while also providing a
tunable interface for QC developers. Because QC is a relatively young field with a large design space, the
ability to explore design parameters rapidly using prototyped systems will support both testing new theories and
assessing existing communication strategies.

In this paper, we present a framework for defining a software-defined QC system with respect to hardware,
middleware, and software layers. We elaborate on the abstraction of these different layers and provide a concrete
example for the case of point-to-point super-dense coding communication. We include details of how the complete
system can be constructed and emphasize how the software and middleware layers should interact in order to
make the physics oblivious to an end user.

2. FRAMEWORK

We formalize the software-defined quantum communication (SDQC) framework by considering a single transmitter-
receiver pair with a quantum transmitter (TX) and quantum receiver (RX). A functional decomposition of each
terminal is shown in Fig. 1 with respect to the functional domain layers. These layers serve to separate
development concerns in constructing each transceiver with respect to the hardware physics, the software pro-
tocol, and a middleware that mitigates between the other two domains. Similar decompositions can be applied
to previously developed QC systems. Our objective is to show how to deliberately identify these domains at an
abstract level and subsequently develop them into concrete realizations.

A concrete representation of the SDQC framework is shown in Fig. 2, in which the TX hardware layer is
expressed as a quantum light source (QLS) for preparing quantum states and accessing the quantum channel,
the middleware is represented as a hardware device driver (HDD), and the software layer is represented by a
general purpose processor (GPP) running a user-defined QC program. The classical channel is assumed to be a
local area network (LAN) while the quantum channel is represented by some quantum optical modes.

In the TX of Fig. 2 the prepared states are encoded into the Hilbert (sub)space of some photonic degrees
of freedom, e.g., polarization, orbital angular momentum, or field quadrature variables. The hardware layer is
modeled to include all components necessary for state preparation, such as polarization or phase modulators,
with the physical encoding driven by the HDD. As middleware, the HDD implements an interface to the QLS for
use by the GPP software. It is the GPP that issues controls and manages the TX behavior by signaling to the
QLS which states to prepare. As a simple example, the GPP can send a bit to the HDD specifying which basis
to use for state preparation that the HDD then parses into the appropriate sequence of QLS control signals. Of
course, more elaborate protocols will require more elaborate interactions between the two layers.

\textsuperscript{*}In QKD, some measurements serve the role of metadata while others represent the payload. These distinctions are
not known at the time of transmission but are derived using an agreed upon CC protocol.
Figure 1. Functional decomposition of an SDQC system consisting of a single transmitter-receiver pair. Each terminal is composed from hardware, software, and middleware layers. Hardware layers interact via a quantum channel while software layers interact over the classical channel. The middleware serves to translate between the languages serving the hardware and software domains.

The RX in Fig. 2 is modeled similar to the TX, except that the RX GPP controls an HDD that drives a quantum light detector (QLD). The QLD measures received photons and outputs measurement information. The HDD samples the measurement information and relays it to the GPP. It is the presence of the QLD which distinguishes the RX from TX. A transceiver (TRX) combining both QLS and QLD components would need only one HDD and GPP to implement this design.

For both the TX and RX, the GPP also serves to communicate required metadata over the LAN. This includes, for example, negotiating the key protocol inherent to QKD or relaying feed-forward measurement information. Because the GPP is assumed to be software programmable, the techniques used in sharing metadata can be modified by the end users as needed. As an example, classical error correction steps are important to deriving keys in QKD but the error codes used may require tuning to the channel and observed bit error rates. These types of modifications are easily made using software-defined implementations expressed by the GPP.

Figure 2. A component representation of the SDQC system shown in Fig. 1: a transmitter (TX) consists of a quantum light source (QLS) driven by a hardware device driver (HDD) that is controlled by a general purpose processor (GPP). The TX GPP communicates over a wide/local area network (W/LAN) with a receiver RX. The RX GPP manages an HDD that monitors a quantum light detector (QLD). The QLS/QLD link defines the quantum channel.

The particular representation of the SDQC framework presented in Fig. 2 is applicable to describing QC systems based on photon pairs, coherent pulses, or multiplexed quantum light sources. In the following sections, we provide additional technical comments on the feasible implementation of each layer using currently available technology. We subsequently present an experimental implementation of these ideas.

2.1 Hardware: Quantum Light Sources and Detectors

The hardware layer expresses components that are fundamental to the physical encoding of quantum information into the transmitted signal. Many quantum light sources and detectors are available as off-the-shelf components. For example, single-photon detectors are sufficiently advanced and wide-spread in their application as to be stand-alone items from optical suppliers. Similarly, weak-coherent pulses generated from attenuated output of photodiodes are easily setup for transmission. There is a significant variety in these elements with respect to
wavelength, bandwidth, stability, and cost so as to warrant their consideration as a replaceable element in the QLS/D design. Individual applications will require suitable pairing between the wavelengths of the source and detector and the modularity of the TRX can easily accommodate this change. Similar arguments also hold for research-grade hardware that may be tailored for specific experimental questions. The essential similarity is that both require externally accessible interfaces for the actively controlled elements and generated metadata.

The engineering challenging to the development of the QLS/D hardware within the SDQC framework is implementing the controls within the hardware layer. Nominal, this design requirement implies that the hardware consists of programmable elements that may be driven explicitly by the middleware. It is possible that pre-programmed hardware behaviors triggered from the upper layer must also be present. Device drivers supplied with most actively controlled components, e.g., translation stages, piezo-electric controllers, phase modulators etc., satisfy this requirement. Collectively, these device drivers and other control wires define the hardware interface. The remaining challenge, therefore, is the integration and mapping of hardware control implementations into defined interface. For most lab-based QC experiments, this is traditionally accomplished in an ad hoc manner that is sufficient for proof of principle but not robust to updates or modifications. Within SDQC, it is the role of the middleware to ease the hardware management by abstracting the interface required by software layer.

2.2 Middleware: Hardware Device Driver

The middleware parses metadata within the TX and RX. This includes translating metadata generated by the TX GPP specifying which qubits (states) to prepare within the hardware as well as tagging raw measurement data generated by the QLD. An HDD interface is defined to separate the concerns between the structure of the QLS/QLD and its expected behaviors required by the GPP.

Implementing the HDD requires knowledge of what hardware components are available and the means by which they are controlled, e.g., via drives. Several controlled components may be synthesized to implement selected software behavior, for example, state preparation or measurement in a specific basis. The particular methods implemented by the middleware to manage control of the hardware are, however, hidden from the other layers, i.e., it maintains separation of concerns. In addition, the HDD need only provide a library of elementary functions that the higher-level software layer can call upon. This separates the HDD from the particular protocol being implementing. Finally, the HDD passes information back to the software layer in a representation appropriate for that domain.

The design of the middleware interface is determined by the level of abstraction provided. The middleware interface can and should vary with the intended use of the terminal. For example, a terminal may be designed such that the user-defined GPP program explicitly requests that the HDD “rotate waveplate 1 to angle \( \theta = \pi/4 \)”. The resulting HDD implementation would then simply relay the appropriately parsed signal to the QLS in order to prepare the specified configuration. Alternatively, the HDD may be designed to accept more abstract commands from the GPP, e.g., “prepare a qubit in the \( X \) basis”, in which case translation into low-level actions would be determined by the HDD implementation to include rotation of the necessary waveplates. These cases are distinguished by how much they abstract away the hardware components from the software protocols. Either approach may be a useful implementation and the best choice must be driven by needs expected of the calling software.

2.3 Software: General Purpose Processor

In the SDQC framework, the software layer defines the abstracted behavior of the hardware but not the implementation details. The level of abstraction and therefore control that is provided to the software layer is determined by the overall design of the terminal and especially the limitations implied by the middleware interface. Depending on these design decisions, the software layer may explicitly define the type of information to be communicated as well as methods for validating transmission and negotiating classical metadata between the TX and RX. Alternatively, the middleware interface may only provide access to a more limited set of behaviors, for example, how many bits to exchange between users. The flexibility in assigning these responsibilities offers a natural way to control the terminal design space.
It seems necessary to justify that the demands of existing and near-term prototype QC systems can be satisfied using GPP’s and software-defined control. Current state of the art QC systems provide at most detection at rates of 1 Gbit/sec[13]. This upper bound on bit rate is due largely to operational limits of current QLD’s, which must employ trade-offs between quantum detector efficiency and response time. Additional losses arising from long-range communication only serve to reduce observed count rates and further limit QC systems to sub-GHz rates. By comparison, modern GPP’s containing multiple cores have theoretical clock rates well above 10 GHz. This represents a more than 10-fold increase in processing speed over data acquisition rates. Moreover, these clock rates correspond with $10^9$ floating-point operations per second (1 GFLOP) even for commodity GPP’s. Alongside gigabit per second (Gbps) communication links, the availability of more than 1 GFLOP suggest it is both possible and reasonable to carry out the computationally intensive part of many QC protocols within GPP’s. Of course, if transmission efficiency improves beyond 1 Gbps, GPP-based systems may require additional processing considerations, for example, the inclusion of specialized co-processors such as graphical processing units (GPU’s) or field-programmable array’s (FPGA’s). However, performance is not the primary intent of the SDQC systems; rather, the purpose is to provide an easily programmable method for prototyping new protocols and testing the limits of QC.

The design of the software layer requires a clear specification of the abstraction intended for the application programming infrastructure. This includes the application programming interface (API) exposed to the user as well as the supporting libraries providing the interface with the middleware. For a GPP implementation, this can be accomplished using standard system software programming and device drivers as well as more elaborate integrated programming environments.

### 3. SUPER-DENSE CODING SYSTEM

As a demonstration of the SDQC framework, we present an implementation of super-dense coding[13]. Super-dense coding is a protocol whereby two users, Alice and Bob, begin by sharing a pair of entangled two-level systems, i.e., qubits. The entangled qubits are initially prepared in the state

$$|\Phi^+(+)\rangle = \frac{1}{\sqrt{2}} (|0_A, 0_B\rangle + |1_A, 1_B\rangle),$$

where subscript $A$ denotes Alice’s qubit and $B$ denotes Bob’s qubit. Alice has a 2-bit message $b_1 b_2$ which she transmits to Bob by applying to her qubit one of the four unitary operators $O \in \{I, X, Z, XZ\}$. These operators have the distinction of mapping the original state within the complete set of Bell states,

$$|\Phi^{(\pm)}\rangle = \frac{1}{\sqrt{2}} (|0_A, 0_B\rangle \pm |1_A, 1_B\rangle)$$

$$|\Psi^{(\pm)}\rangle = \frac{1}{\sqrt{2}} (|0_A, 1_B\rangle \pm |1_A, 0_B\rangle)$$

The mapping between operators and bit pairs is established by Alice and Bob before beginning the protocol. We will use

| $b_1 b_2$ | $O$ | $|\psi_{A,B}\rangle$ |
|------------|-----|------------------|
| 00         | I   | $|\Phi^{(+)}\rangle$ |
| 01         | X   | $|\Psi^{(+)}\rangle$ |
| 10         | Z   | $|\Phi^{(-)}\rangle$ |
| 11         | XZ  | $|\Psi^{(-)}\rangle$ |

After applying the operator $O$ to her qubit, Alice transmits her qubit to Bob. Upon receiving Alice’s qubit, Bob performs a joint measurement that discriminates between the four Bell states. Based on the outcome of the measurement, Bob decodes the original two bits of message.
3.1 Software Layer

Our implementation of super-dense coding includes a software layer. The software layer is based on a library built within the GNU Radio signal processing framework. GNU Radio is a free software toolkit for deploying software-defined communications systems that offers primitive signal processing blocks for application development.\[16\] We have developed a Quantum Information ToolKit for Application Testing (QITKAT) library that provides C++ and Python based processing blocks to support prototyping stream-based quantum communication. The QITKAT library provides primitives for expressing communication protocols completely in software. This includes methods for encoding and decoding the SDC messages as well as interfaces exchanging network metadata between users. These blocks can then be connected using an interprocess communication system provided by the GNU Radio runtime environment. The runtime manager is responsible for maintaining the flow of data, while the block developer is responsible for ensuring each blocks consumes and processes samples in the desired way.

Using QITKAT and GNU Radio blocks, we have developed a TX and RX programs that permit Alice to encode binary data sending modulated entangled states and Bob to decode these modulations from measurements made on the entangled state. An instance of the flow graph for the SDC communication system is shown in Fig. 3, in which the block \textit{SDC Encode} accepts pairs of bits from a \textit{Message Source} block. The \textit{SDC Encode} identifies the appropriate operator based on the bit values according to the table in Eq. (3). The corresponding output flag is then sent to the \textit{QM Server} block, which represents a visible middleware component responsible for translating the modulation operators into the correct actions onto the fiducial Bell state. In the current implementation, \textit{QM Server} also manages a classical representation of the entangled states and does not trigger actual hardware commands, cf. below.

![Flow graph of the super-dense coding protocol using QITKAT blocks.](image)

The flow graph in Fig. 3 highlights how the individual processing blocks are connected by the flow of data from Alice to Bob. In the pictured implementation, the \textit{SDC Encode} block is sending a two-bit modulation code to the \textit{QM Server} using a TCP packet. When it is received, the server returns an identification number that uniquely labels the entangled state to which the modulation is applied. In addition to executing the control flags, \textit{QM Server} also transmits a classical notification message to \textit{SDC Decode} indicating that a modulated state has been transmitted. In practice, this message serves as the metadata indicating the expected time-of-arrival for a qubit or the storage location within a quantum memory cell. We use the \textit{QM Server} block twice, one for receiving and the other for transmitting messages, to simplify the network control. After accepting the \textit{QM Server} message, the \textit{SDC Decode} block queries for the results of the Bell-state measurement on that qubit. We use the \textit{QM Server} block twice, one for receiving and the other for transmitting messages, to simplify the network control. After accepting the \textit{QM Server} message, the \textit{SDC Decode} block queries for the results of the Bell-state measurement on that qubit. The \textit{QM Server} returns the results of the measurement, which are then interpreted by \textit{SDC Decode} according to the table in Eq. (3). The flow graph in Fig. 3 also includes a \textit{Bit Error Rate} block, which acts by computing the bit error rate between the message decoded by Bob and the original transmitted by Alice. The \textit{Scope Sink} block is a standard GNU Radio block that plots the output BER as a function of the sampled data.
3.2 Middleware Layer

The QM Server block serves as a visible middleware component. The encode and decode blocks issue control commands to modulate and measure the Bell state, respectively. The modulations are based on application of the operator $O$ in Eq. (3) while the measurements correspond with projections in the Bell basis of Eq. (2). This block is also responsible for the handshaking between the encode and decode blocks, which in our implementation is simply a classical transmission of packet counter to monitor the qubit sequence. This is in addition to the handshaking that underlies the classical network communications. In the current implementation, the server resides on a separate computer and communication is managed using TCP packets. The QM server may be running local on the same host as either TX or RX clients, or on a separate device as would be a more natural case when the server is managing separate hardware.

Our current QM Server does not manage a hardware layer. Instead the current server runs a software simulation of hardware behavior by maintaining a registry of requested and transmitted states as well as a history of the encoding applied to each state. This allows the server both to track the modulation sent by the encoder and to transmit the result of measuring the modulated states. The registry does not store complete state representations, but rather uses labels to encode the modulation of an entanglement resource.

QM Server also simulates noise in the transmission channel by incorporating an anisotropic depolarizing noise model. This model is parametrized by probabilities $p_x$, $p_z$, and $p_{xz}$ for the $X$, $Z$, and $XZ$ operators specified independently and it acts by introducing statistical bias in the measurement outcomes. That is to say, the relative probabilities of the measurement outcomes are biased according to the parameters of the noise model and sampled using the output from a pseudo-random number generator. Again, the server does not track the actual state vectors, but rather maintains the consistent behavior of the noise model, modulations, and observed measurements.

3.3 Hardware Layer

For SDC, the necessary hardware includes a source of entangled particles, a modulation mechanism, and a measurement apparatus. Assuming the use of polarization-entangled photon pair state, a non-deterministic source can be constructed using the process of spontaneous parametric down conversion (SPDC) pumped by an external laser. This approach, however, lacks a means of announcing the photon’s presence. Heralded pair production offers a slight more complicated alternative but with the advantage that each photon is tagged in a known time slot.

For polarization entangled biphoton states, the modulation operators are implemented using an optical wave plate for implementing the $X$, $Z$, and $XZ$ transformations. Because the orientation determines the operator being implemented, we can mount the waveplate(s) on an electronically driven rotator. The state of the rotator, and the photon polarization, can then be driven using computer-controlled electrical signals. The measurement of the photon pair state at the RX can be implemented partially using linear-optical Bell-state measurement device. In this setup, a static beam splitter interferes the two photons and polarization analyzers measurement the resulting state. The observed measurements can then identify 3 of the 4 possible Bell states, but cannot detect all of them. Alternative approaches, using ancilla or hyper-entanglement, can measure all four states but at the cost of additional complexity. In our design, we assume a static optical network precedes a bank of detectors, which output a unique signature for each encoded state.

In order to interface with the layout described above, we have developed a hardware interface based on the combined use of an FPGA and ARM processor. Our particular implementation uses the Xilinx Zynq board with a custom daughter board that accepts input from the detector bank. An example of the hardware is shown in Fig. 4. We have not yet implemented the outgoing control signals needed by the TX to drive the waveplate rotators, but instead have focused on refining the timestamping capabilities at the RX terminal. In our design, the FPGA accepts TTL signals from the connected detectors and generates timestamps for photon arrivals based on edge detection and an on-board clock. We use multiple input channels to to detect coincidence arrivals and store the resulting timestamp(s) as well as the excited channel in the on-board memory. The ARM processor uses read/write access to the same memory region and, therefore, can run user-defined code to process the generated timestamps. In our client-server model, the ARM-based server monitors the local memory buffers for data and
responds to request from a network-connected client process. Raw timestamps can either be processed on board, using a QITKAT program, or transmitted over the network to a host computer. We are currently testing the use of both UDP and TCP packets for managing the networking.

Figure 4. A physical representation of the SDQC architecture for the SDC RX implementation. The GPP on the left runs the QITKAT program while the customized Zynq board in the middle defines the HDD interface, and, on the right, the pair of silicon photodetectors represent the QLD.

4. CONCLUSIONS

We have extended the paradigm of software-defined communication to include quantum communication systems. We defined an SDQC framework based on the decomposition of QC terminal into three layers, which separate the concerns of the hardware, software, and middleware layers. We have also provided a detailed description of how each components should operate and we have provided experimental results of an SDQC implementation of the super-dense coding protocol. Our experimental design has emphasized the role of middleware for abstracting the high-level, software control and managing the low-level (hardware execution. Ultimately, we expect the use of a common software layer to provide a robust family of functions that can be used for rapidly prototyping new applications.

ACKNOWLEDGMENTS

T. S. H. would like to thank Henry Humble for comments regarding the SDQC design in Sec. 2 and Toby Flynn and Laura Ann Anderson for help evaluating the Zynq middleware interface. R. D. S. thanks the Department of Energy Science Undergraduate Laboratory Internships (SULI) program for support. This work was supported by the Defense Threat Reduction Agency. This manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Kumar, P., Kwiat, P. G., Ralph, T., and Sasaki, M., “Special issue on quantum communications and information science,” IEEE Journal of Selected Topics in Quantum Electronics 15 (2009).
[2] Mitola, J., *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*, PhD thesis, Royal Institute of Technology, Kista, Sweden (2000).

[3] Mitola, J., “Cognitive radio for flexible mobile multimedia communication,” *Proc. IEEE Int. Workshop Mobile Multimedia Communications* (1999).

[4] Savory, S. J., “Electronic signal processing in optical communications,” *Proc. SPIE: Optical Transmission, Switching, and Subsystems VI* **7136**, 71362C (2008).

[5] Cox, W. C., Simpson, J. A., and Muth, J. F., “Underwater optical communication using software defined radio over led and laser based links,” *MILCOM 2011*, 2057–2062 (2011).

[6] Wilde, M. M., “From Classical to Quantum Shannon Theory,” *arXiv:1106.1445* (2011).

[7] Bennett, C. H. and Brassard, G., “Quantum cryptography: Public key distribution and coin tossing,” *Proc. IEEE Int. Conf. Computers, Systems and Signal Processing*, 175 (1984).

[8] Bennett, C. H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., and Wootters, W. K., “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels,” *Phys. Rev. Lett.* **70**, 1895–1899 (1993).

[9] Lodewyck, J., Bloch, M., Garcia-Patron, R., Fossier, S., Diamanti, E., Debuisschert, T., Cerf, N. J., Tualle-Bouri, R., McLaughlin, S. W., and Grangier, P., “Quantum key distribution over 25 km with an all-fiber continuous-variable system,” *Phys. Rev. A* **76**, 042305 (2007).

[10] Jouguet, P., Kunz-Jacques, S., Debuisschert, T., Fossier, S., Diamanti, E., Alléaume, R., Tualle-Bouri, R., Grangier, P., Leverrier, A., Pache, P., and Painchault, P., “Field test of classical symmetric encryption with continuous variables quantum key distribution,” *Optics Express* **20**(13), 14030–14041 (2012).

[11] Zhang, H.-F., Wang, J., Cui, K., Luo, C.-L., Lin, S.-Z., Zhou, L., Liang, H., Chen, T.-Y., Chen, K., and Pan, J.-W., “A real-time QKD system based on FPGA,” *Journal of Lightwave Technology* **30**(20), 3226–3234 (2012).

[12] Martinez-Mateo, J., Elkouss, D., and Martin, V., “Blind reconciliation,” *Quantum Information and Computation* **12**, 791–812 (2012).

[13] Cui, K., Wang, J., Zhang, H.-F., Luo, C.-L., Jin, G., and Chen, T.-Y., “A real-time design based on FPGA for expeditious error reconciliation in QKD system,” *IEEE Transactions on Information Forensics and Security* **8**, 184–190 (2013).

[14] Eisaman, M. D., Fan, J., Migdall, A., and Polyakov, S. V., “Invited review article: Single-photon sources and detectors,” *Rev. Sci. Instrum.* **82**(7), 071101 (2011).

[15] Bennett, C. H. and Wiesner, S. J., “Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states,” *Phys. Rev. Lett.* **69**, 2881–2884 (1992).

[16] “GNU Radio website,” (2013). http://www.gnuradio.org.

[17] Shelton, D. P., O’Donnell, W. M., and Norton, J. L., “Note: Fast, small, accurate 90[degree] rotator for a polarizer,” *Review of Scientific Instruments* **82**(3), 036103 (2011).

[18] Rakonjac, A., Roberts, K. O., Deb, A. B., and Kjærgaard, N., “Note: Computer controlled rotation mount for large diameter optics,” *Review of Scientific Instruments* **84**(2), 026107 (2013).

[19] Dušek, M., “Discrimination of the Bell states of qudits by means of linear optics,” *Optics Communications* **199**(14), 161 – 166 (2001).