論文

横浜キャンパス内 高品質映像配信に関する検討

浅田 裕紀　志藤 大海　藤井 哲郎

横浜キャンパスでは，様々な大型ディスプレイが設置され活用されている．それらは大きく分けて，講義に用いられる大型スクリーンとデジタルサイネージ的に設置された情報掲示用液晶ディスプレイである．本稿では，これらの映像スクリーンに，キャンパスネットを用いて効率的にフル HD 映像を配信できる映像配信システムの構築手法について検討した結果を報告する．グラフィックボードを搭載した汎用 PC により最新の映像符号化方式 H.265/HEVC を用いたリアルタイム映像伝送が実現できること．さらに，レイヤー 2 方式の安価なデジタルサイネージ用フル HD 映像配信装置と OpenFlow を用いたネットワーク装置を組み合わせてフル HD 映像をマルチキャスト配信する 2 方式について報告する．

キーワード：フル HD，H.265/HEVC，ソフトウェアベース，デジタルサイネージ，OpenFlow

1 まえがき

近年，屋外・店頭・公共空間・交通機関など，あらゆる場所での情報発信にディスプレイなどの電子的な表示機器を用いる「デジタルサイネージ」が普及し始めている．本学の横浜キャンパスにおいても，様々な映像表示装置が増加している．しかし，現状は個別の活用に限定されており，キャンパスネットワークを活用した統合化された映像配信は行われていない．本稿では，横浜キャンパスに数多く設置された様々なディスプレイへの効率的な映像配信を実現するために，キャンパスネットに適した映像配信方法を検討した結果を報告する．

横浜キャンパスでは，情報メディアルームに映像配信用のエンコーダ（映像符号化装置）が設置され，大講義室のスクリーンや塩尻高校などへの遠隔講義に利用されてきた．この装置は映像符号化に H.264/AVCHD 方式を用いている．これに対して，新しい映像符号化規格として H.265/HEVC 方式が登場してきた．従来の H.264 方式と比較して 2 倍近くの圧縮性能を誇る．しかし，エンコーダは非常に高価であり，100 万円を超える．本稿では，汎用 PC を使用することで，高価な映像伝送装置に頼ることなく，ソフトウェアベースにて H.265/HEVC 圧縮映像のリアルタイム伝送システムを実現することを目指す［1］．さらに，構築したシステムでの映像品質評価を行い，その有用性を明確にする．

大教室の大スクリーン以外に横浜キャンパス内で増加しているのが講義などの学内情報を掲示する液晶モニタである．これを活用することにより，デジタルサイネージのコンセプトに基づく積極的な利用が可能となる．デジタルサイネージでは同時に多くの画面に同一の映像を配信できる．これを実現する装置として，イーサネットを介して HDMI 映像を伝送できる HDMI Extender と呼ばれる装置が登場し，比較的安価にデジタルサイネージが構築できる．ところが，この映像配信装置は同一サブネットワーク内での使用を想定されており，映像配信にレイヤー 2（イーサネット層）のマルチキャストを用いている．その為に，IP を基本とするレイヤー 3（IP 層）で構成されている東京都市大学横浜キャンパスの学内ネットワークでは，そのままの状態では映像配信を行えない．レイヤー 2 のマルチキャストをレイヤー 3 で通すには VPN ルータを用いれば解決するが，送信機及び受信機毎にルータを設置するのは不効率である［2］．本稿では，ソフトウェアによるネットワークの一元制御・管理技術である SDN（Software-Defined-Network）の実現方式の一つである OpenFlow 方式を用いることにより，映像配信装置より送られるパケットを自在に制御し学内ネットワークでの映像配信を実現する．そのために，デジタルサイネージ用の映像配信装置の特性を調べ，OpenFlow 装置用のプログラムを開発する．これにより，横浜キャンパスネットワークでの映像配信実験を実現する．以上の 2 方式による高品質なフル HD 映像の横浜キャンパス内での配信を目指す．

ASADA Yuki
SITOU Hiroumi
東京都市大学メディア情報学部情報システム学科
2016 年度卒業生
FUJII Tetsuro
東京都市大学メディア情報学部情報システム学科教授

2　ソフトウェアベース H.265 映像配信システム

H.265/HEVC は最新の動画圧縮規格であり H.264/MPEG-4 AVC の後継規格として登場した．4K，8K といった高解像度で情報量の大きい映像や，携帯端末向けの映像配信サービスにも利用されており，従来の H.264/MPEG-4 AVC と比較して同程度品質の映像を約半分のビットレートまで圧縮できる．ネットワーク運用の観点からも導入すべき映像符号化方式である．大スクリーンへの映像配信のみならず，携帯端末への映像配信も期待できる．しかし，最新の H.265/HEVC を組み込んだ映像伝送装置は非常に高価である．そこで汎用 PC を活用し，ソフトウェアベースの H.265/HEVC 映像伝送システムの構築を目指す．

映像配信の主たるターゲットは，情報メディアルームから大教室への映像ストリーンミングである．大教室には，プロジェクターを用いた大型スクリーンが設置されており，プロジェクターには講義用のパソコンが接続されている．横浜キャンパスのすべての講義用及び演習用パソコンには VideoLAN が提供するフリーの映像再生ソフトである VLC メディアプレーヤーが組み込まれている．このソフトは，H.264/AVCHD 映像の UDP によるストリーミング再生に用いてきた．なお，各教室に設置されたプロジェクターの最大解像度はフル HD であり，本稿でもフル HD 映像伝送を目指す．

H.265/HEVC による映像伝送システムは，送信用 PC を情報メディアルームのラックに設置し，キャンパスネットのコアルータを介して横浜キャンパス内に UDP により配信する．OS は Ubuntu 16.04 を用い，CPU は Core i7 3770 である．システムの基本構成を図 1 に示す．送信用 PC ではビデオカメラ映像を取り込む為に BlackMagic Design 社製キャプチャーカード DeckLink Studio 4K を用いる．HD カメラからの入力は，HDMI 及びプロ用の HD-SDI どちらでも入力できる．さらに，H.265/HEVC による映像符号化は処理が非常に重たいことが知られており，これを高速化する為に NVIDIA 社製グラフィックカード GeForce GTX 960 を組み込む．

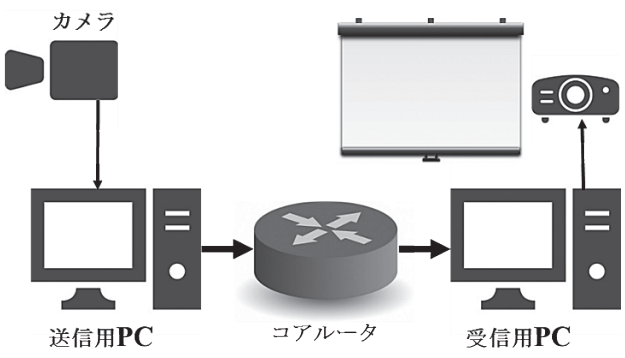

図 1　キャンパス内 H.265/HEVC 映像配信システムの構成

H.265/HEVC 方式での映像符号化を行う送信側システムには，オープンソースの動画変換ソフトである FFmpeg を組み込み用いる［1, 3］．映像を取り込む DeckLink Studio 4K の制御には DeckLink SDK を使用する．DeckLink SDK を FFmpeg に組み込み，コンパイルすることによって，キャプチャーカード DeckLink Studio 4K を入出力デバイスとして使用できる．また FFmpeg には NVIDIA 社により開発された Nvenc 機能が備わっている．これにより，NVIDIA 社製のグラフィックカードを用いて H.265/HEVC 符号化を高速に行うことができる．本稿では，GeForce GTX 960 を送信用 PC に組み込み，Nvenc 機能を活用して高速に符号化を行い，CPU の負荷を軽減する．

3　H.265 による映像配信の検証

3．1　フル HD 映像配信実験

フル HD 映像伝送の検証として，本キャンパス 2 号館 1 階にある藤井研学生室から同 2 階にあるプレゼンテーションラボへ学内ネットワークを介したフル HD 映像の伝送実験を行った．フル HD 映像の撮影には，HDR-CX12 を用いた．伝送実験時には送信用 PC の GPU 負荷，CPU 負荷の計測を行う．GPU 負荷の計測には nvidia-smi コマンドを用いる．これは任意の秒数毎に GPU 負荷をパーセンテージで表示するコマンドである．CPU 負荷の計測には Ubuntu システムモニターを用いた．

最初に，パソコンでの H.265/HEVC 方式での映像符号化の負荷を把握する為に，CPU のみで映像符号化処理を行った．この時，演算の負荷が 100％に達し，処理が中断した．その時の CPU 負荷を図 2 に示す．これに対し，GPU として GeForce GTX960 を用いて H.265/HEVC の符号化を行った時の演算処理の負荷を図 3 に示す．CPU の負荷は常時 20％以下であり，余裕を持って処理が継続していることが解る．また，観測された GPU の負荷は 3 〜 7％程度であり，4K 映像の可能性を

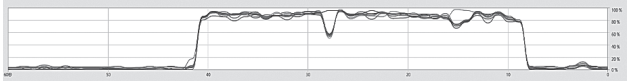

図 2　GPU 無しの H.265 符号化処理の CPU 負荷

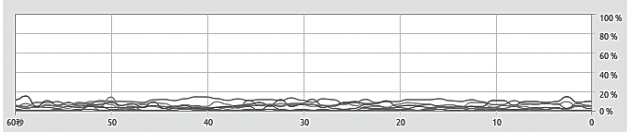

図 3　GPU 有りの H.265 符号化処理の CPU 負荷

図 4　プレラボへのフル HD 映像配信の様子

示唆する結果であった．この時，プレゼンテーションラボのスクリーンに映し出された実験映像の様子を図 4 に示す．同図より，伝送された映像が 270 インチのスクリーン上に綺麗に映し出されていることが確認できる．以上のことから，ソフトウェアベースの H.265/HEVC 圧縮の HD 映像の伝送が安定して行われていることが確認できた．

3．2　フル HD 映像の主観評価実験

本節では，フル HD 映像を用いた H.265/HEVC 圧縮映像伝送システムの映像品質に関する主観評価実験を行う．送信用 PC において映像を H265/HEVC で圧縮し伝送する際に，ビットレートを 1，3，6，12，18Mbps と変化させる．受信用 PC からディスプレイに表示した 5 種類のビットレートをランダムに視聴し，5 段階で主観評価を行う．素材となる映像には，人が喋る様子，水面に浮く鴨，交差点を走る車の 3 種類の映像を用いる．評価用の映像素材は，YUV422 非圧縮映像にて用意する．

この評価用の映像信号を送信用 PC に入力し，FFmpeg にて，H.265/HEVC 圧縮し予め設定したビットレートで受信用 PC へ UDP にて伝送する．受信用 PC は，映像データを同じく FFmpeg にて UDP 受信し DeckLink から録画装置へ出力する．この信号を BlackMagic Design 社製 Hyper-Deck Studio Pro を用いて YUV422 非圧縮映像フォーマットで録画する．3 種類の映像を 5 段階のビットレートでそれぞれ録画した．この非圧縮で録画した映像を評価対象として，55 インチの液晶ディスプレイに表示し 5 段階主観評価実験を行う．

被験者 10 人に対し，主観評価実験を行った結果を図 5 に示す．同図より，ビットレートが高い映像程 MOS 評価値が高いことが解る．また，非常に細かい映像である交差点に関しては，ビットレートが高くなっても，MOS 値が上がりにくいことも確認できる．但し，6Mbps 以上のビットレートでは，評価に差が殆ど無く，

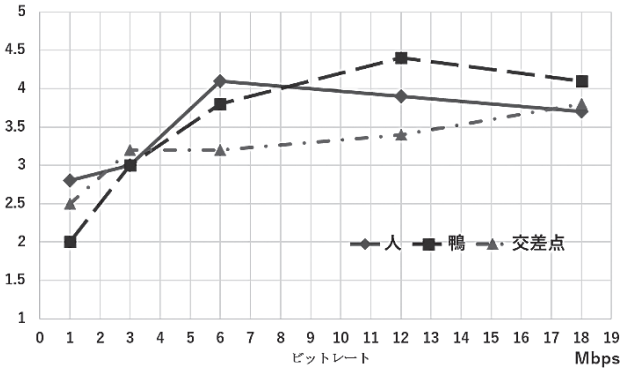

図 5　FFmpeg による H.265 符号化映像の主観評価実験（人物，水面を泳ぐ鴨，交差点の 3 種類の評価映像）

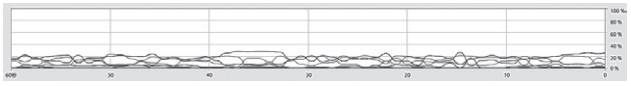

図 6　4K30P 映像における H.265 符号化処理の CPU 負荷（GPU: NVIDOA 社製 GeForce GTX960 有り）

MOS 値も最大 4.5 である．このことから，NVenc を用いた FFmpeg では，高ビットレートの映像品質に若干問題があることが推察できる．

3．3　4K30P の伝送実験

本節では，システムの性能確認の為，4K30P 映像伝送の検証を行う．学内の講義用及び演習用 PC には 4K モニタが接続されていないので，別途 4K モニタが接続された受信用 PC を用意した．OS は Ubuntu 16.04，CPU は Core i7 870，グラフィックボードは 4K30P 映像が扱える最も安価な MSI 社製 RADEON R7-240，4K モニタは LG 社製 27UD58-B である．送信システムは FFmpeg による UDP 映像伝送を行い，受信システムでは，受信用 PC 上での UDP 受信，再生に FFmpeg の動画再生機能 ffplay を使用する．受信システムに FFmpeg を使用する為，受信用 PC には送信用 PC と同様のソフトウェアインストールを行う．

本キャンパス 2 号館 1 階にある藤井研学生室から同フロアにあるメディアホールへ学内ネットワークを介した 4K30P 映像の伝送実験を行った．ffplay によって受信用 PC のモニタ上に H.265/HEVC 圧縮の 4K30P 映像が綺麗に映し出された．この時，送信用 PC の GPU 負荷は 5 〜 10% 程度，CPU 負荷は全体を通して 30% 程度であった．CPU 負荷を図 6 に示す．以上のことから，グラフィックボードを活用することにより，4K30P 映像も問題無く H.265/HEVC 方式で映像符号化し，送受信することが可能であることが示された．

4　デジタルサイネージ用映像伝送装置での配信

4．1　デジタルサイネージ用映像伝送システム

デジタルサイネージの普及とともに，安価な映像配信装置が普及し始めている．これは最低限の必要な機能のみを具備する装置にすることでコストを安く抑えている．この装置はHDMI Extenderと呼ばれており，HDMI映像信号をイーサネットのパケットに変換して受信機に送信する．配信にイーサネットを用いることにより，長いケーブルを新たに張り巡らす必要がなく，既存のイーサネットケーブルを利用できることを特徴とし，トータルの設置コストも抑えられる．

本稿ではアイシル社製のaHDMI-EX150mを用いる．この装置は送受信装置一組で２万円と非常に安価に手に入り，約60Mbpsにて高品質なフルHD映像の配信ができる．デジタルサイネージは同じ映像を近距離間の複数の画面で映し出すため，遠距離への映像伝送は想定されていない．その為，映像配信はレイヤー２のマルチキャストを用いて行っている．aHDMI-EX150mにおける装置間の接続手順は，送信機がマルチキャストプロトコルを用いて受信機を探し，受信機はマルチキャストプロトコルに対してARPパケットを飛ばし，送信機がARPに応答すると接続が確立されUDPによる映像配信が始まる．

レイヤー２のパケット伝送では，宛先MACアドレスを用いてパケットのルーティングを行うので送信機と受信機が同一ネットワーク内にいることが前提となっている．宛先MACアドレスを用いるのでルータを超えた配信はできない．この限定的な通信機能は，デジタルサイネージで使用するには十分だがレイヤー３のコアルータを核とする横浜キャンパス内で使用するには問題がある．

本学横浜キャンパスの学内ネットワークは，２号館情報基盤センターのコアルータを中心とした複数のサブネットワークで構成されるレイヤー３での管理を基本に構成されている．レイヤー３ではIPパケットによるルーティングを行うので，レイヤー２での制御機能しかないaHDMI-EX150mではレイヤー３で構成される学内ネットワークで映像を配信することができない．また，学内ネットワークのルータはマルチキャスによる運用を行っていないので，マルチキャストを用いた配信も行えない．

この問題を解決する方法として，VPNを用いて送信機と受信機の間を中継させる方法が考えられる．この方法を用いれば送信機と受信機のコネクション確立を妨げるルータを気にせず配信を行える．しかし，この方法だと映像配信装置を使用する度にVPNルータを設置しなければならいので運用するにあたり非常に手間がかかる．さらに，デジタルサイネージ向けの装置の特徴である１対多方式の配信を行う際には使用する受信機毎にVPNルータを設置しなければならないので，コスト面でも非効率である．

そこで，スイッチ内の伝送設定やポート，IPアドレスの設定をプログラムで変更することが可能なSDNの一つであるOpenFlowを用いてレイヤー２を基本とする映像パケットをレイヤー３のIPネットワークでも配信できると考えられる［4］．レイヤー２ではイーサネットプロトコルに従い，そこでOpenFlowプロトコルを用いたパケット書き換えを導入し，ルータを超えた映像配信を実現できる．OpenFlowを用いることにより，従来のネットワークを再構築せずに柔軟なネットワークの構築が期待できる．

4．2　OpenFlowについて

OpenFlowとは，インターネットの再構築を目的としてスタンフォード大学を中心に研究開発されているオープンソースの技術である［5］．Open Networking Foundation（OFN）によって規格化及び標準化が進められている．現在広く使われているネットワークは，ネットワーク機器に組み込まれた経路制御プロトコルを用いてパケット転送を行っている．ネットワーク構成を変更する際は，管理者が個別の装置の設定をマニュアル操作で変更している．ところが，サーバ仮想化やクラウドの登場による事業形態の急激な変化に伴い，マニュアル操作より迅速かつ柔軟にネットワークの構成変更を行うことが求められるようになった．この問題解決のために，ソフトウェアにネットワークを制御・管理させ，ネットワーク構成をプログラミングで可能にしようというコンセプトの元登場したのがSDN（Software-Defined-Network）である．OpenFlowとはこのSDNの標準化規格の一つである．

OpenFlowスイッチは主に物理スイッチと仮想スイッチに分類される．物理スイッチはハードウェアによる専用スイッチであり，様々なメーカから製品が提供されている．仮想スイッチはソフトウェアによるスイッチで，OS上で構築されており，実装は比較的容易である．本稿では，実装が容易で，利用例が多いOpenFlow対応の仮想スイッチであるOpen vSwitchを使用する［6］．

OpenFlowコントローラは，OpenFlowプロトコルを用いてOpenFlowスイッチと通信し，OpenFlowスイッチが持つフローテーブルの追加・修正を行いパケットの転送を指示する．このコントローラに実装するプログラムを独自に開発することにより，従来のネットワーク機器では実現できなかった柔軟な転送ルールや経路制御を定義できる．コントローラは商用以外にもオープンソースで活発に開発が進められている．

本稿では，多くの稼働実績がある Python 言語の Ryu を使用する［7, 8］．Ryu は NTT 研究所によって開発された OpenFlow コントローラである．OpenFLow 1.0, 1.2, 1.3, 1.4 に対応しており，Apache 2.0 ライセンスにて提供される．開発言語は Python2.7 および Python3.4 に対応している．他のコントローラと比較すると OpenFlow プロトコルに幅広く対応しており，多くの SDN 研究で使用実績がある．

OpenFlow プロトコルは，OpenFlow スイッチと OpneFlow コントローラのやりとりに使う通信規約のことである．OpenFlow の仕様では OpenFlow スイッチの仕様は定められているが，OpenFlow コントローラ自体の仕様は特に定められていない．しかし，コントローラとスイッチのやりとりが OpenFlow プロトコルによって使用が定められているので，スイッチやコントローラの組み合わせを変えても OpenFlow を実現することができる．現在 OpenFlow プロトコルの最新バージョンは 1.6 である．本研究では，使用例が多く安定性が高いバージョン 1.3 を用いる．

4．3　デジタルサイネージ用伝送装置の検証

送信機と受信機の通信に OpenFlow プロトコルを用いることにより，レイヤー 3 のルータを超えるパケット転送が可能となり，横浜キャンパスネットワークでの動画配信が可能となる．本節ではオープンソフトウェアである Open vSwitch と Ryu による OpenFlow ネットワークを構築する．使用言語は Python で，OpenFlow によるパケット書き換えによるマスク超えを利用し，動画配信装置の学内ネットワークへの適用を目指す．

まず基本検証として，aHDMI-EX150m の受信機と送信機の間に OpenFlow スイッチを置き OpenFlow による動画配信が行えるかを確認した．構成を図 7 に示す．Open vSwitch を組み込み，OpenFlow 仮想スイッチとなる PC は，富士通製 PRIMERGY TX100 S3p，CPU は Pentium G640，OS は Ubuntu 16.04 である．ギガビットのポートは Intel 82574L，Intel 82579LM，及び増設した Intel Pro/1000 PT Dual Port の 4 ポートである．使用した Open vSwitch のバージョンは 2.5.0, Ryu はバージョン 4.6 を用いる．この時 OpenFlow プロトコルとして 1.3 を用いる．基本検証用のプログラムとして，MAC アドレスを学習しフローテーブルに追加するプログラムを開発した．Python で開発し，ステップ数は 89 行である．検証の結果，プログラムは正常に動作し，aHDMI-EX150m の映像を Open vSwitch と Ryu による OpenFlow で伝送可能と判断した．

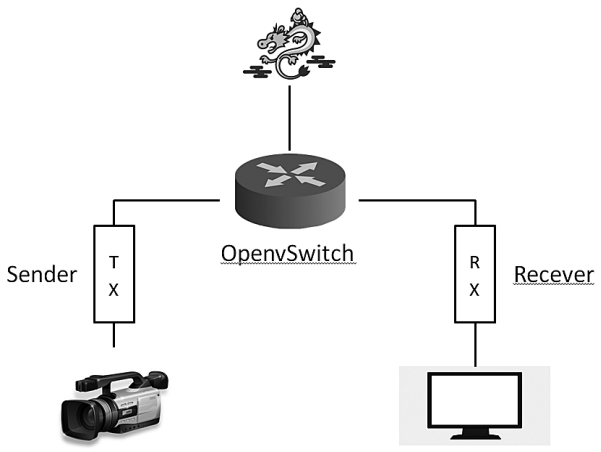

図 7　OpenvSwitch 検証用ネットワークの構成

aHDMI-EX150m を学内ネットワークに適応させるために，送信機と受信機のコネクションの確立方法を確認する．検証には Allied Telesis 社製の CentreCOM GS908M を用いて確認した．この装置はレイヤー 2 の HUB であり，ポートごとに細かな設定を行うことができる．その一つに任意のポートから出入りするパケットを指定したポートにミラーリングするポートにミラーリング機能がある．この装置に送信機・受信機を接続し送信機から出るパケットを Wireshark にてキャプチャした．パケットキャプチャした結果，送信機からはまずブロードキャストアドレスが出されて受信機を探す．受信機が見つかると受信機宛に ARP パケットが飛ばされる．ARP パケットが返答されると，受信機から送信機宛に UDP パケットが飛ばされる．この UDP が送信機に入ると映像配信が始まる．映像配信には 226.2.2.2 のマルチキャストアドレスが用いられていた．

本学のキャンパスネットワークはマルチキャストでの配信は行えない．なので，このマルチキャストアドレスを書き換えることにする．また，コネクションの確立に ARP が使われているので ARP に対する処理も加える．但し，この動 aHDMI-EX150m は基本的に同一サブネットワーク内での動作を想定されているので，OpenFlow によって送信パケットの宛先 MAC アドレスが受信機の MAC アドレス以外に書き換わった状態だと，コネクションの確立に必要な受信機から送信機へ送られる UDP が正常に送られないことが確認された．コアルータを超えるために宛先アドレスは受信機の MAC アドレスにすることはできない．そこで，aHDMI-EX150m が送受信機間で接続が確立されていなくとも送信機からの映像パケットが受信機に入ると映像が映る性質を利用する．これは，デジタルサイネージのような複数画面に同時に映し出すために送信機 1 に対し受信機が複数個でも映る仕組みである．この仕組を利用しネットワークを構築する．

4．4　OpenFlow を用いたネットワークの構築

OpenFlow プロトコルを用いたマルチキャスト動画

配信システムを構築する．aHDMI-EX150m の送信機を 2 号館藤井研学生室に設置し，受信機を 4 号館学生ホールに設置し，2 号館から学術センター内コアルータを越えて 4 号館の学生ホールに映像を送る．この 2 号館から 4 号館への配信制御を OpenFlow にて実行する．構成図を図 8 に示す．

以下に示す処理手順を実行する OpenFlow コントローラ用のプログラムを開発する．パケット処理の流れは，HUB を介して送信機と受信機を接続し一旦映像配信を確立させる．HUB に OpenFlow スイッチをつなぎ映像パケットを OpenFlow スイッチに取り込む．入力された映像パケットの IP アドレスと MAC アドレスを書き換え，学術センター内コアルータに送る．ルータを超えるためには，宛先 MAC アドレスを学術センタールータの 133.78.113.254 ポートのアドレスを用いる．宛先 IP アドレスをマルチキャストアドレスから受信機に設定した IP アドレスへと書き換える．マッチフィールドとそれに従うインストラクションに ARP や ICMP 等の処理を記述したコントローラ用プログラムのステップ数は合計 180 行となった．図 9 及び 10 に実際に OpenFlow コントローラを動かした時の OpenFlow スイッチに入るパケットと出るパケットを Wireshark でキャプチャしたものを示す．設定したとおりに書き換わった事が解る．

226.2.2.2 が使われていたマルチキャストアドレスを書き換えることにより，学内ネットワークを流れる aHDMI-EX150m から送信された映像パケットはユニキャスト用の UDP パケットとなり，レイヤー 3 の IP ネットワークの制御手順に従い受信機まで送られる．この時の 4 号館に設置した受信機の画面を図 11 に示す．同図から明らかなように OpenFlow スイッチ によって書き換えられたパケットが受信機に届き，映像が映し出されている．同図において，液晶 TV の左下にある黒い箱が aHDMI-EX150m の受信装置である．この小さな安価な装置を学内の任意の場所に設置するだけで，高品質な映像の配信が実現できることが示された．なお，4 号館のネットワークは通信速度が 100Mbps の回線であったが，安定した映像配信が実現できることを確認した．

5　まとめ

グラフィックカードを用いたソフトウェアベースの H.265/HEVC 映像符号化装置を構築した．これにより，フル HD 映像を 5Mbps 程度に圧縮し，UDP により横浜キャンパス・ネットワークを介して伝送するリアルタイム映像伝送システムが実現できた．また，デジタルサイネージ用の HDMI Extender を OpenFlow によりマルチキャストにて動画配信できるシステムも構築した．これは，デジタルサイネージ向けの安価な動画配信装置から出されるマルチキャストの映像パケットを OpenFlow スイッチで書き換え，マルチキャスト未対応の学内ネットワークに映像を配信構成となっている．これらの 2 方式を活用することにより，横浜キャンパス内でフル HD の高品質な映像を簡単に配信でき，統合化さえた情報発信を行う基盤が構築できることを示した．今後，これらのシステムの活用が期待される．

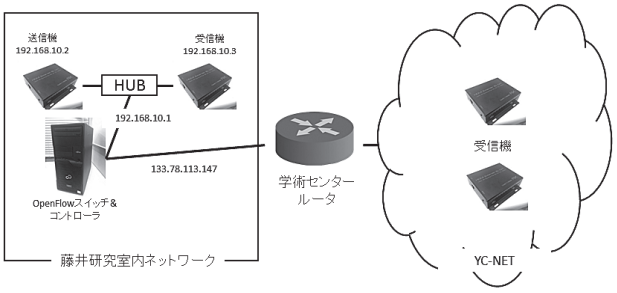

図 8　OpenFlow を用いた配信システムの構成（学術センターのコアルーター―を介した映像配信）

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
2	0.002596913	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
3	0.005173886	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
7	0.007759673	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
32	0.010363765	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
59	0.012952938	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
92	0.015798553	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
122	0.018442555	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
128	0.020644377	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
129	0.023249961	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
130	0.025835326	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
131	0.028403953	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
132	0.031002925	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
133	0.033576537	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008
134	0.036168899	192.168.10.2	226.2.2.2	UDP	1050	2065→2066 Len=1008

図 9　OpenFlow による書き換え前パケット（aHDMI-EX150m の送信器からのパケット）

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
2	0.000100417	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
3	0.000151360	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
4	0.000350282	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
5	0.000449654	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
6	0.000498465	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
7	0.000601228	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
8	0.000700508	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
9	0.000800141	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
10	0.000848660	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
11	0.000949972	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
12	0.001049189	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
13	0.001150032	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
14	0.001197538	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024
15	0.001300032	133.78.113.147	133.78.121.40	UDP	1066	2068→2068 Len=1024

図 10　OpenFlow による書き換え後のパケット

図 11　2 号館藤井研から 4 号館食堂への映像配信，画面下の HDMI 延長器（aHDMI-EX150m）の受信箱のみで受信可能

参考文献

［1］ 羽鳥「4K 映像を用いた自由視点映像伝送システムの開発」東京都市大学, 環境情報学部, 卒業論文, 2016
［2］ 三谷「マルチキャストを用いた学内映像配信システムの構築」東京都市大学，環境情報学部，卒業論文，2012
［3］「FFmpeg Documentation」，https://ffmpeg.org/ffmpeg.html
［4］ 大畑「OpenFlow プロトコルを用いたネットワーク制御システムに関する検討」東京都市大学，環境情報学研究科，環境情報学専攻，修士論文 , 2015
［5］ 馬場，大上，関山，高畑「OpenFlow 徹底入門 SDN を実現する技術と知識」翔泳社 , 2013
［6］ Open vSwitch, http://openvswitch.org/
［7］ Ryubook，http://osrg.github.io/ryu-book/ja/html/
［8］ Ryu SDN Framework, http://osrg.github.io/ryu/