Stochastic Analysis of Delayed Mobile Offloading in Heterogeneous Networks

Matteo Del Vecchio

Corso di Simulazione di Sistemi

Informatica Magistrale - UniBo - A.A. 2018/19

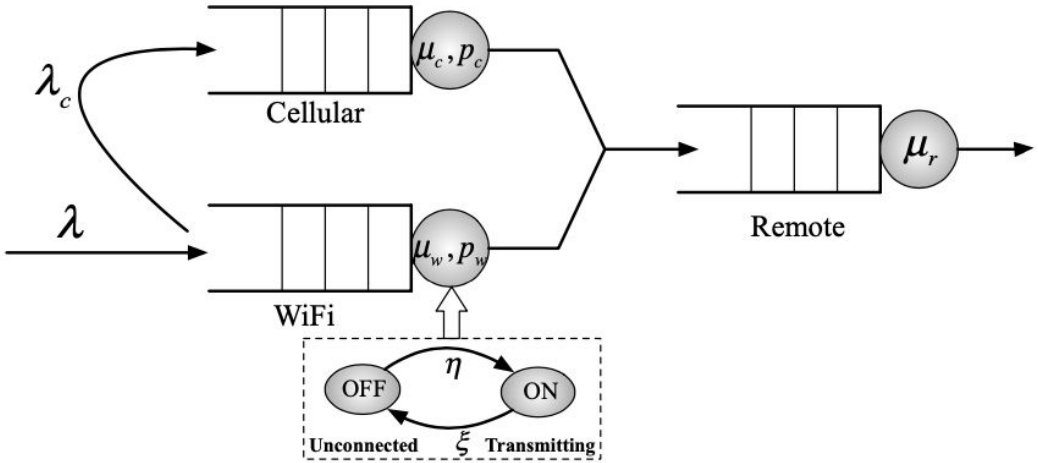
Introduzione

Un modello di offloading ha come obiettivo quello di delegare parte della computazione di un dispositivo al cloud, attraverso l'uso di network veloci (WiFi) o lenti (cellulare) per la trasmissione delle informazioni.

Caratteristiche fondamentali riguardano il consumo energetico ed i tempi di risposta.

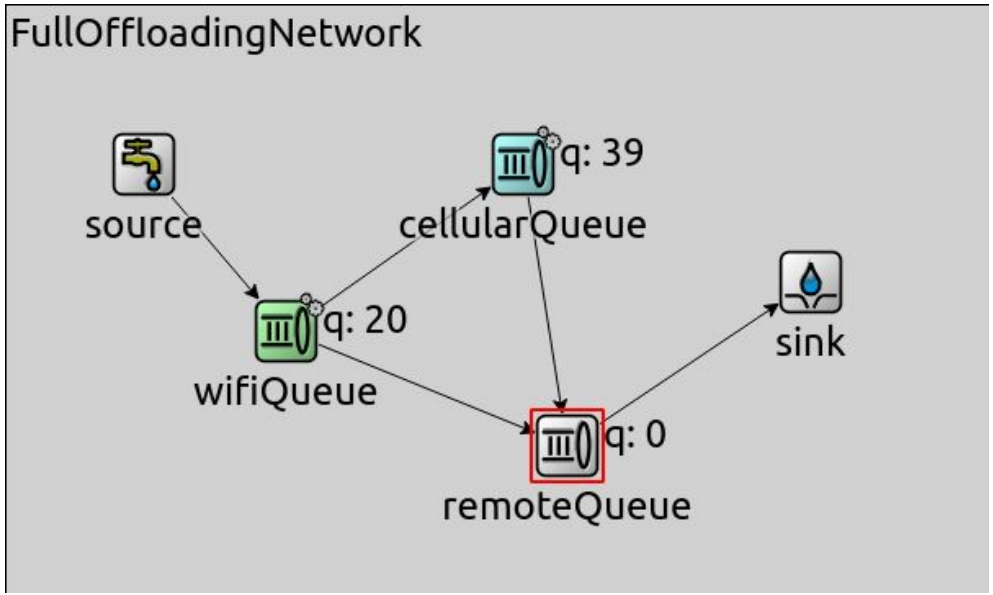
Il modello analizzato dal paper è il Full Offloading Model.

Full Offloading Model



I job sono inviati alla coda WiFi che, in caso di stato ON, li processa.
In caso di stato OFF non c'è alcun servizio ed i job possono decidere di andare in coda Cellular, allo scadere di una deadline assegnatagli all'arrivo.

Full Offloading Model



Implementazione del sistema in OMNeT++

La variazione del colore della wifiQueue (verde o rosso) indica lo stato del relativo server

Tutti i moduli sono stati implementati da zero

Implementazione

Impostazione delle deadline e logica di esecuzione allo scadere della stessa

```
void OffloadingQueue::arrival(Job *job) {
    job->setTimestamp();
    job->setQueueCount(job->getQueueCount() + 1);
    EV << job << " queue count: " << job->getQueueCount() << endl;

    // WIFI is OFF so add deadline to jobs
    if (!wifiAvailable) {
        cMessage *deadlineMsg = new cMessage("deadline_reached");
        deadlineMsg->setContextPointer(job);
        job->setContextPointer(deadlineMsg);

        simtime_t deadlineLength = par("deadlineDistribution").doubleValue();
        emit(deadlineDistrib, deadlineLength);
        simtime_t deadlineTime = simTime() + deadlineLength;
        EV << "Deadline set for job " << job << "; firing time: " << deadlineTime << endl;
        scheduleAt(deadlineTime, deadlineMsg);
    }
}
```

```
std::string jobName = msg->getName();
if (jobName.std::string::compare("deadline_reached") == 0) {
    if (msg->getContextPointer()) {
        Job *job = (Job *)msg->getContextPointer();
        msg->setContextPointer(nullptr);
        EV << "DEADLINE REACHED! WIFI status: " << wifiAvailable << " - Associated job: " << job << endl;

        if (hasGUI()) {
            std::string text = std::string("Deadline for ") + std::string(job->getName());
            bubble(text.c_str());
        }

        if (job == servicedJob) {
            if (endServiceMsg->isScheduled())
                cancelEvent(endServiceMsg);

            prepareNextJobIfAny();
        }
        else if (job == suspendedJob) {
            suspendedJob = nullptr;
            if (endServiceMsg->isScheduled())
                cancelEvent(endServiceMsg);
        }
        else
            queue.remove(job);

        emit(queueLengthSignal, length());
        send(job, "out", 0);
        delete msg;
    }
    else
        EV << "DEADLINE REACHED!" << endl;
}
```

Implementazione

Cambio stato del server per la coda WiFi (con annessa sospensione/ripresa del servizio) e relativo reschedule dello stato successivo

```
else if (msg == wifiStatusMsg) {
    wifiAvailable = !wifiAvailable;
    EV << "WIFI STATUS CHANGED! Now is " << (wifiAvailable ? "ON" : "OFF") << "\n";

    // wifi OFF -> ON
    if (wifiAvailable) {
        if (suspendedJob) resumeService(suspendedJob);
        else prepareNextJobIfAny();
        updateNextStatusChangeTime();
    }
    // wifi ON -> OFF
    else {
        updateNextStatusChangeTime();
        if (servicedJob)
            suspendService(servicedJob);
    }
}
```

```
void OffloadingQueue::updateNextStatusChangeTime() {
    simtime_t nextChange = (wifiAvailable) ? par("wifiStateDistribution").doubleValue() : par("cellularStateDistribution").doubleValue();
    if (wifiAvailable) emit(wifiActiveTime, nextChange);
    else emit(cellActiveTime, nextChange);
    nextStatusChangeTime = simTime() + nextChange;
    scheduleAt(nextStatusChangeTime, wifiStatusMsg);
    EV << "Next WIFI status change time: " << nextStatusChangeTime << endl;
}
```

Implementazione

Sospensione del servizio con relativi update temporali e funzione di comparazione per il comportamento a priority queue

```
int OffloadingQueue::compareFunction(cObject *a, cObject *b) {
    Job *jobA = check_and_cast<Job *>(a);
    Job *jobB = check_and_cast<Job *>(b);
    cMessage *aDeadline = (cMessage *)jobA->getContextPointer();
    cMessage *bDeadline = (cMessage *)jobB->getContextPointer();
    if (aDeadline && bDeadline) {
        simtime_t now = simTime();
        simtime_t timeLeftA = aDeadline->getArrivalTime() - now;
        simtime_t timeLeftB = bDeadline->getArrivalTime() - now;
        if (timeLeftA < timeLeftB) return -1;
        else if (timeLeftA > timeLeftB) return 1;
        else return 0;
    }
    else if (aDeadline || bDeadline) {
        if (aDeadline) return -1;
        else return 1;
    }
    else {
        simtime_t creationTimeA = jobA->getCreationTime();
        simtime_t creationTimeB = jobB->getCreationTime();
        if (creationTimeA < creationTimeB) return -1;
        else if (creationTimeA > creationTimeB) return 1;
        else return 0;
    }
}
```

```
void OffloadingQueue::suspendService(Job *job) {
    cancelEvent(endServiceMsg);
    simtime_t startTime = job->getTimestamp();
    simtime_t elapsedTime = simTime() - startTime;
    simtime_t remainingTime = curJobServiceTime - elapsedTime;
    scheduleAt(nextStatusChangeTime + remainingTime, endServiceMsg);
    job->setTotalServiceTime(job->getTotalServiceTime() + elapsedTime);

    job->setTimestamp();
    suspendedJob = job;
    servicedJob = nullptr;
    EV << "Service SUSPENDED! Received: " << job << " - Suspended: " << suspendedJob << " - Serviced: " << servicedJob << endl;
    EV << "Elapsed service time for " << job << ": " << elapsedTime << endl;
    EV << "Remaining service time for " << job << ": " << remainingTime << endl;
    EV << "New scheduleAt time: " << nextStatusChangeTime + remainingTime << endl;
}
```

Simulazione

- Job arrival rate: λ = 0,5 pkt/min quindi 0,008333 pkt/s
- Service rate coda Remote: μ = 1
- Coefficiente energetico coda WiFi: p_w = 0,7 W, coda Cellular: p_c = 2,5 W
- Durata media della disponibilità WiFi: ξ = 52 min = 3120 s
- Durata media della disponibilità solo Cellular: η = 25,4 min = 1524 s
- Dimensione media di un job: E[X] = 10 MB
- Velocità media rete WiFi: s_w = 2 Mbps
- Velocità media rete Cellular: s_c = 200 Kbps

Metriche

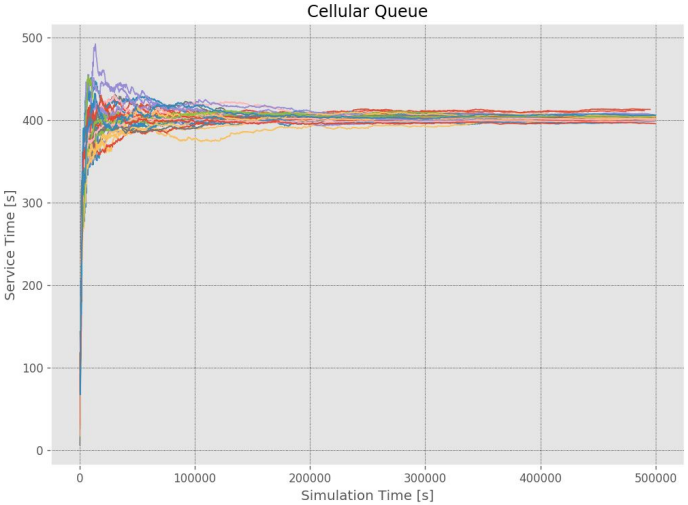
- Mean Response Time (MRT)
- Mean Energy Consumption (MCE)
- Energy-Response Weighted Product (ERWP):

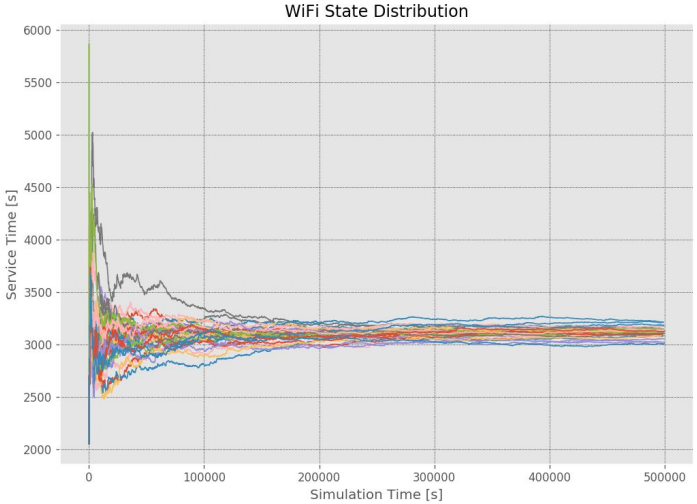
$$ERWP = \mathbb{E}[\xi]^{\omega} \cdot \mathbb{E}[T]^{1-\omega}$$

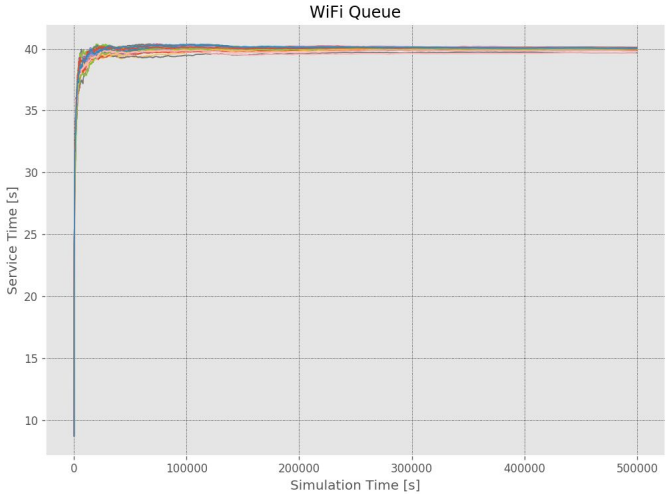
Parametri calcolati

- Job interarrival time: 1 / λ = 120 s
- Service rate WiFi: μ_w = s / E[X] = 0,025 pkt/s quindi service time: 1 / μ_w = 40 s
- Service rate Cellular: μ_c = s / E[X] = 0,0025 pkt/s quindi service time: 1 / μ_c = 400 s

Transiente Iniziale

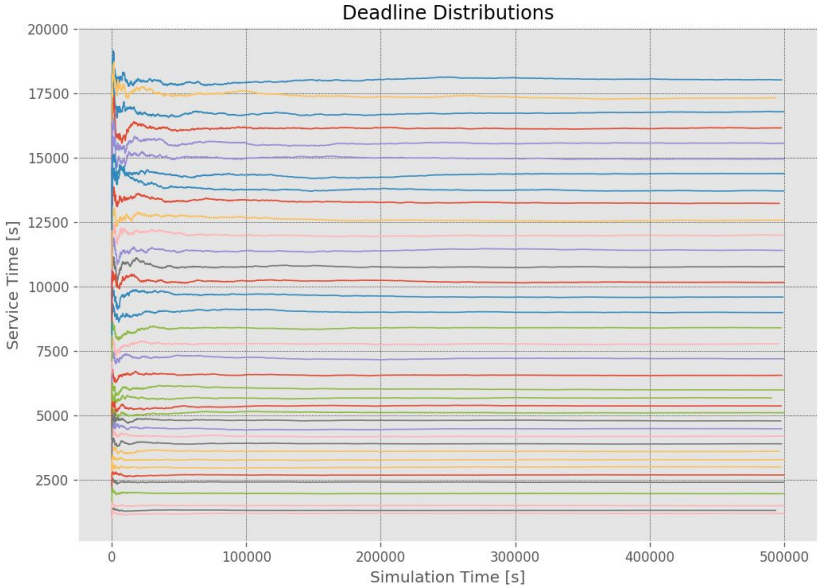


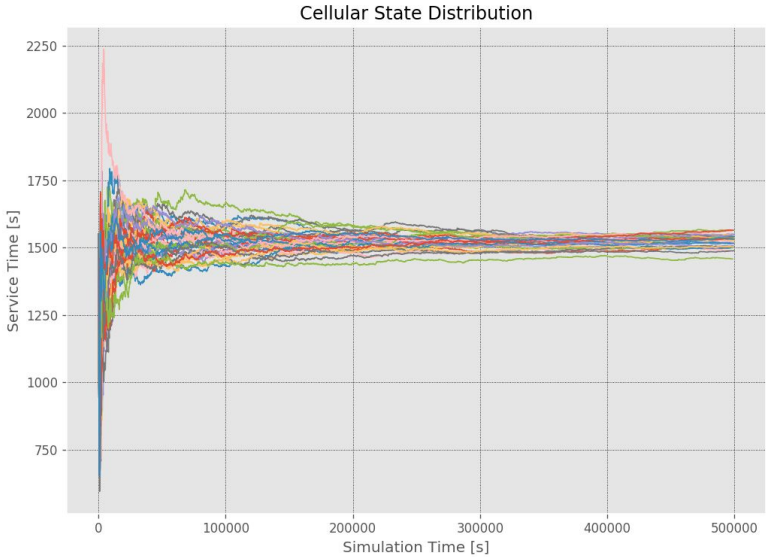




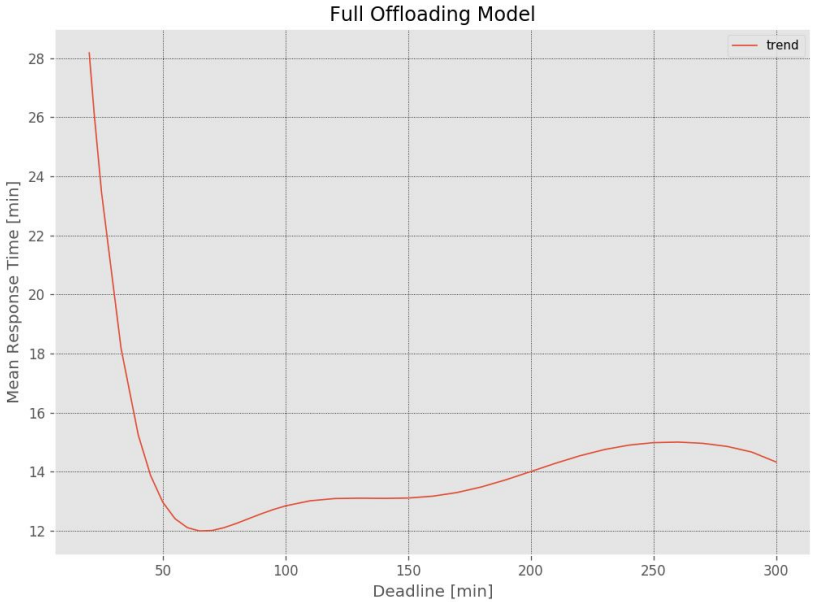
Transiente Iniziale

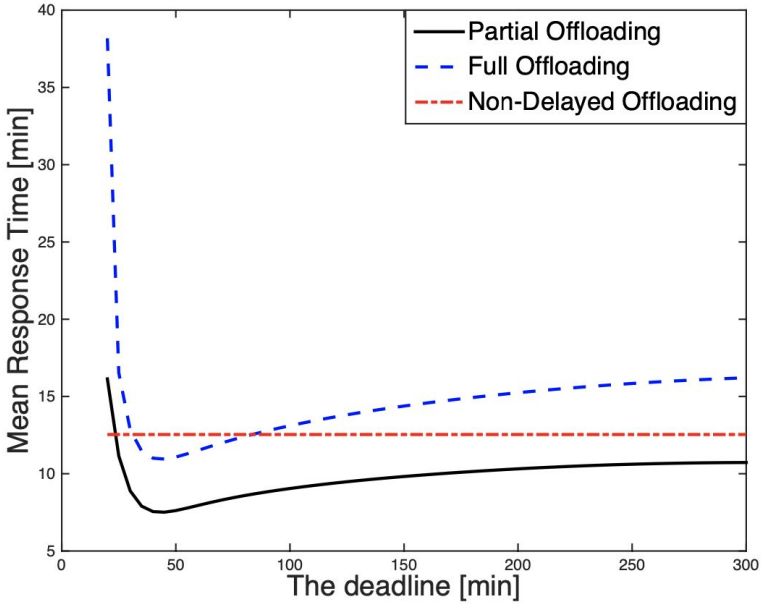
Dati risultati, il warmup-period è stato scelto di 200000 s





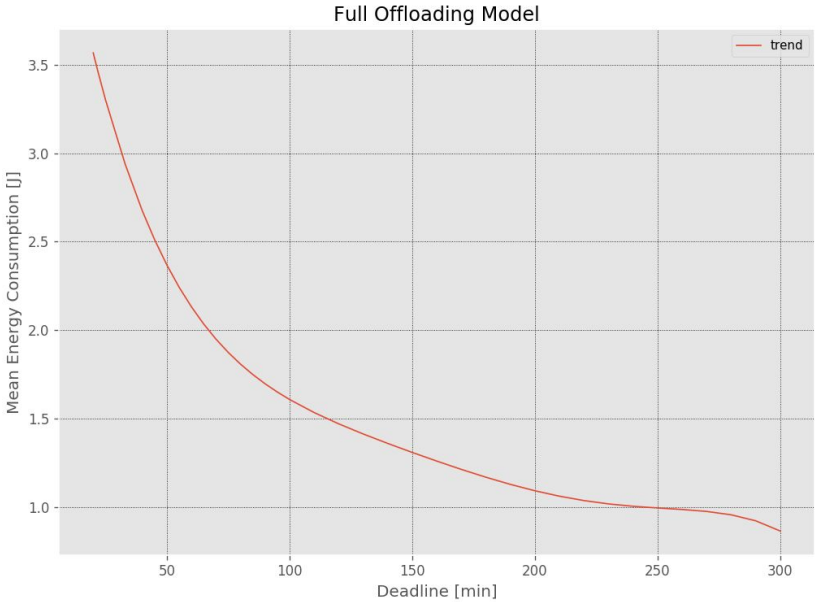
Risultati

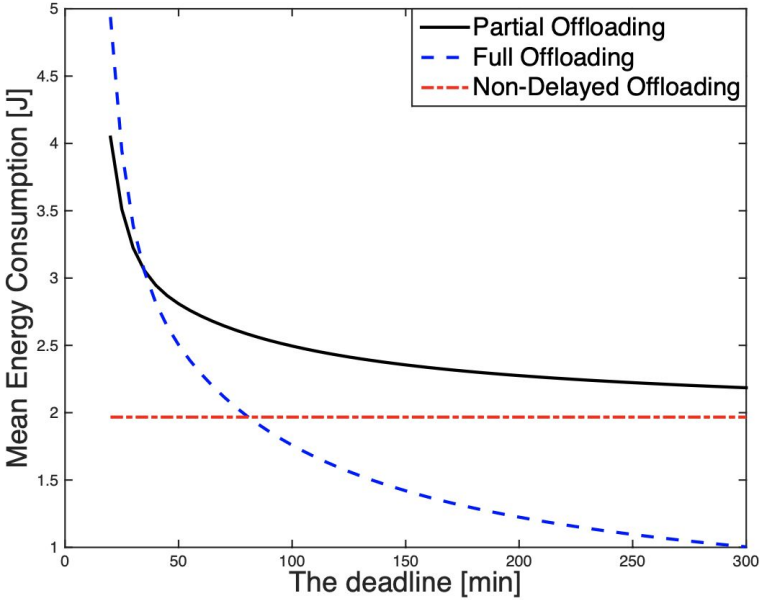




(a) Mean Response Time

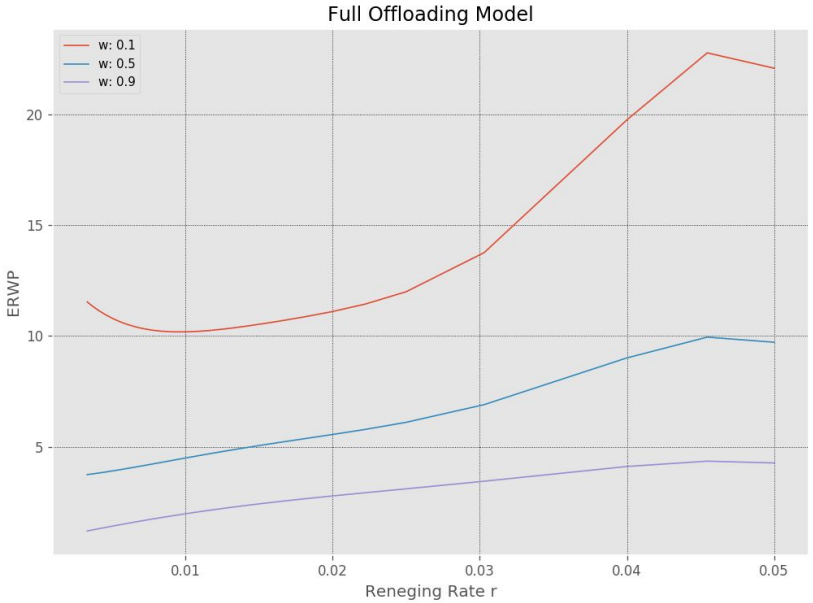
Risultati

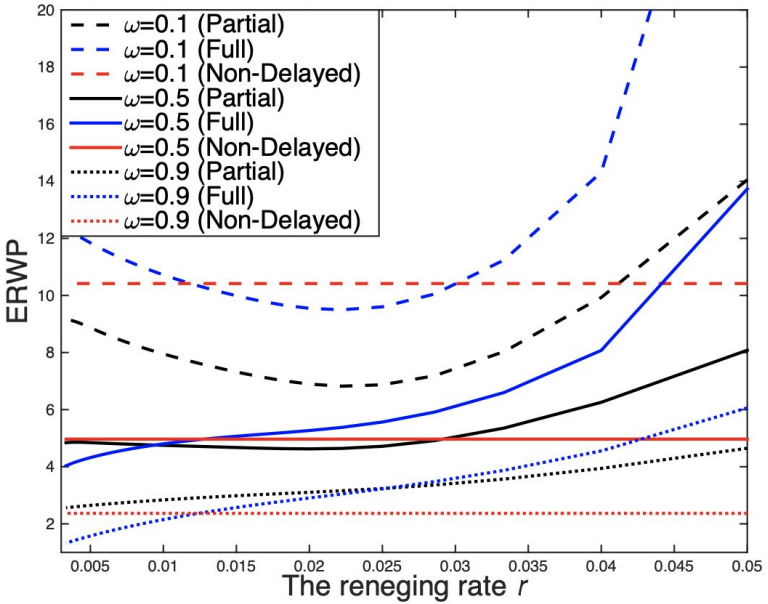




(b) Mean Energy Consumption

Risultati





(a) Reneging rate

Intervalli di Confidenza

Deadline [s]	Deadline [min]	Reneging Rate r	MEC	Mean	Variance	Left Value	Right Value	Included?
1200	20	0.0500	3.4760	3.5350	0.2047	3.1036	3.9664	OK
1320	22	0.0455	3.5576	3.6560	0.0426	3.4592	3.8528	OK
1500	25	0.0400	3.3820	3.3748	0.0473	3.1674	3.5821	OK
1980	33	0.0303	2.8416	2.8337	0.1763	2.4334	3.2339	OK
2400	40	0.0250	2.7338	2.7005	0.0662	2.4551	2.9459	OK
2700	45	0.0222	2.4517	2.4586	0.0443	2.2580	2.6592	OK
3000	50	0.0200	2.3477	2.3625	0.0331	2.1890	2.5359	OK
3300	55	0.0182	2.2009	2.1737	0.0365	1.9916	2.3558	OK
3600	60	0.0167	2.0865	2.0175	0.0103	1.9205	2.1144	OK
3900	65	0.0154	2.1074	2.1269	0.0831	1.8521	2.4018	OK
4200	70	0.0143	1.9652	1.9541	0.0425	1.7575	2.1506	OK
4500	75	0.0133	1.9371	1.9628	0.0218	1.8221	2.1034	OK
4800	80	0.0125	1.8386	1.7781	0.0148	1.6620	1.8942	OK
5100	85	0.0118	1.6860	1.7174	0.0153	1.5995	1.8354	OK
5400	90	0.0111	1.6827	1.6375	0.0265	1.4822	1.7928	OK
5700	95	0.0105	1.6582	1.6871	0.0258	1.5340	1.8402	OK
6000	100	0.0100	1.6309	1.6343	0.0021	1.5906	1.6780	OK
6600	110	0.0091	1.5603	1.5991	0.0171	1.4744	1.7237	OK
7200	120	0.0083	1.4618	1.4633	0.0101	1.3675	1.5591	OK
7800	130	0.0077	1.3855	1.4775	0.0187	1.3472	1.6078	OK
8400	140	0.0071	1.3668	1.3778	0.0162	1.2565	1.4990	OK
9000	150	0.0067	1.3325	1.3176	0.0129	1.2091	1.4260	OK
9600	160	0.0063	1.2196	1.1446	0.0171	1.0201	1.2691	OK
10200	170	0.0059	1.2221	1.2080	0.0073	1.1266	1.2894	OK
10800	180	0.0056	1.1444	1.1855	0.0030	1.1336	1.2374	OK
11400	190	0.0053	1.0782	1.0830	0.0075	1.0004	1.1657	OK
12000	200	0.0050	1.0799	1.0866	0.0014	1.0508	1.1224	OK
12600	210	0.0048	1.0960	1.0823	0.0014	1.0470	1.1176	OK
13200	220	0.0045	1.0697	1.0296	0.0043	0.9669	1.0923	OK
13800	230	0.0043	1.0440	1.0609	0.0066	0.9836	1.1382	OK
14400	240	0.0042	1.0115	1.0219	0.0052	0.9528	1.0909	OK
15000	250	0.0040	1.0012	1.0030	0.0057	0.9308	1.0751	OK
15600	260	0.0038	0.9840	0.9821	0.0077	0.8983	1.0660	OK
16200	270	0.0037	0.9599	0.9681	0.0108	0.8692	1.0670	OK
16800	280	0.0036	0.9106	0.9097	0.0017	0.8704	0.9490	OK
17400	290	0.0034	0.9083	0.8695	0.0004	0.8498	0.8893	
18000	300	0.0033	0.8972	0.8759	0.0037	0.8175	0.9343	OK

Deadline [s]	Deadline [min]	Reneging Rate r	MEC	Mean	Variance	Left Value	Right Value	Included?
1200	20	0.0500	3.4760	3.4006	0.0985	3.1701	3.6311	OK
1320	22	0.0455	3.5576	3.5338	0.0722	3.3364	3.7312	OK
1500	25	0.0400	3.3820	3.3735	0.0132	3.2890	3.4580	OK
1980	33	0.0303	2.8416	2.8857	0.0966	2.6575	3.1140	OK
2400	40	0.0250	2.7338	2.6961	0.1048	2.4583	2.9338	OK
2700	45	0.0222	2.4517	2.4843	0.0108	2.4081	2.5604	OK
3000	50	0.0200	2.3477	2.3584	0.0142	2.2708	2.4460	OK
3300	55	0.0182	2.2009	2.2046	0.0161	2.1116	2.2977	OK
3600	60	0.0167	2.0865	2.0404	0.0340	1.9049	2.1758	OK
3900	65	0.0154	2.1074	2.1198	0.0296	1.9934	2.2462	OK
4200	70	0.0143	1.9652	1.9539	0.0474	1.7940	2.1137	OK
4500	75	0.0133	1.9371	1.9534	0.0253	1.8365	2.0703	OK
4800	80	0.0125	1.8386	1.8355	0.0180	1.7369	1.9342	OK
5100	85	0.0118	1.6860	1.7028	0.0081	1.6367	1.7688	OK
5400	90	0.0111	1.6827	1.7053	0.0175	1.6081	1.8025	OK
5700	95	0.0105	1.6582	1.6652	0.0064	1.6066	1.7239	OK
6000	100	0.0100	1.6309	1.6340	0.0031	1.5933	1.6747	OK
6600	110	0.0091	1.5603	1.5525	0.0289	1.4277	1.6774	OK
7200	120	0.0083	1.4618	1.4662	0.0078	1.4014	1.5310	OK
7800	130	0.0077	1.3855	1.3711	0.0337	1.2362	1.5060	OK
8400	140	0.0071	1.3668	1.3765	0.0071	1.3146	1.4385	OK
9000	150	0.0067	1.3325	1.3071	0.0115	1.2283	1.3860	OK
9600	160	0.0063	1.2196	1.1974	0.0172	1.1012	1.2937	OK
10200	170	0.0059	1.2221	1.2110	0.0094	1.1399	1.2821	OK
10800	180	0.0056	1.1444	1.1675	0.0058	1.1117	1.2233	OK
11400	190	0.0053	1.0782	1.0721	0.0036	1.0283	1.1159	OK
12000	200	0.0050	1.0799	1.0843	0.0013	1.0576	1.1109	OK
12600	210	0.0048	1.0960	1.0705	0.0040	1.0239	1.1171	OK
13200	220	0.0045	1.0697	1.0544	0.0038	1.0091	1.0997	OK
13800	230	0.0043	1.0440	1.0568	0.0021	1.0234	1.0903	OK
14400	240	0.0042	1.0115	1.0068	0.0018	0.9759	1.0377	OK
15000	250	0.0040	1.0012	0.9990	0.0056	0.9439	1.0542	OK
15600	260	0.0038	0.9840	0.9753	0.0025	0.9386	1.0121	OK
16200	270	0.0037	0.9599	0.9594	0.0060	0.9024	1.0164	OK
16800	280	0.0036	0.9106	0.9054	0.0018	0.8744	0.9365	OK
17400	290	0.0034	0.9083	0.8865	0.0011	0.8617	0.9114	OK
18000	300	0.0033	0.8972	0.8941	0.0041	0.8469	0.9413	OK

Configurazioni (num batch, num obs): (5, 10) e (7, 12); Distribuzione T-Student con confidenza del 90%

Grazie per l'attenzione!

Bibliografia

- Huaming Wu. 2018. Stochastic Analysis of Delayed Mobile Offloading in Heterogeneous Networks. IEEE Transactions on Mobile Computing 17, 2 (February 2018), 461-474. DOI: https://doi.org/10.1109/TMC.2017.2711014
- https://omnetpp.org
- https://github.com/matteodelv/SdSFullOffloading